Seamless Flow Migration on Smartphones without Network Support

Ahmad Rahmati[1], Clayton Shepard[1], Chad Tossell, Lin Zhong[2], Philip Kortum, Angela Nicoara[2], Jatinder Singh[2];   [1] *Student Member, IEEE*, [2] *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. (*i*) We report a comprehensive characterization of IP traffic on 27 iPhone 3GS users for three months. (*ii*) 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 pre-existing 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. (*iii*) 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

— — — — — — — — ◆ — — — — — — — —

I. 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, e.g. 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]. Switching between networks can significantly improve the performance [5, 6], energy efficiency [1, 7], and connectivity [8] of mobile Internet. In this work, we focus not on *policies*, 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, e.g. cellular handoff,

connection gateway, and Mobile IP, which incur extra operating expenses and additional latency [9]. 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 infrastructure support or protocol changes? 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 to migrate flows between networks without network support and without disruption to the user. The 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 pre-existing flows on the old network to terminate normally, up to a specific wait-time set by the migration policy. 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. Based on our traces, we show that using Wait-n-Migrate, we can successfully migrate web flows for 90% and 95% of cases, for

- *Ahmad Rahmati, Clayton Shepard, and Lin Zhong are with the Electrical and Cmputer Engineering Department, Rice University, Houston, TX. E-mail: {rahmati, cws, lzhong} @rice.edu.*
- *Chad Tossell and Phil Kortum are with the Psychology Department, Rice University, Houston, TX. E-mail: {chad.tossell, pkortum} @rice.edu.*
- *Angela Nicoara and Jatinder Singh are with the Deutsche Telekom R&D Laboratories USA, Los Altos, CA. E-mail: {angela.nicoara , j.singh } @telekom.com.*

wait-time values 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% to well under 10% 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 II, we present a motivational user study to show that brute force network switching is unacceptable to users, and then discuss related work. In Section III, 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 IV, we present the design and trace-based evaluation of Wait-n-Migrate and Resumption Agent. In Section V, we report their implementation on iPhone and evaluate their performance impact. In Section VI, 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 VII, and conclude in Section VIII.

II. BACKGROUND

A. 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 application flows on the old network and enable the new network. This behaviour 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 web pages 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 included an equal number of 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 web sites, 9 of 10 mentioned that they experience it more frequently with larger transfers, e.g. mentioning pages that are as "heavier" or "with lots of graphics". When asked whether they have experienced this phenomenon more often in specific conditions, 8 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 VI.

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% Agresti-Coull confidence interval [10] for 8 and 10 positive answers out of 10 are (0.52 , 0.91) and (0.66 , 1), respectively, i.e. there is a 90% 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 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.

B. Related Work

TCP/IP 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 [11] between different

---

[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.

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 [12], 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 [13, 14], to act as a proxy between a mobile device and the Internet. For example, such gateways have been employed for switching between interfaces (Wiffler [5]), for multihoming ([3, 4, 15]), and for striping ([13, 16-23]). 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, e.g. by adding explicit support, as in [24, 25], or through Mobile IP [26-28], 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 travelled by packets increases the connection latency. 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.

All three categories of solutions discussed above require additional infrastructure or network support, and thus are not immediately deployable. Those that have begun deployment suffer from limited or unsuccessful adoption. Furthermore, these solutions increase network latency, which is already known to be a major bottleneck in mobile Internet performance [29]. Our techniques allow the immediately deployment of system policies that leverage multiple networks without changes to applications or infrastructure. MultiNets [30] is one such example that utilizes Wait-and-Migrate to switch between cellular and WiFi networks, in order to effectively 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. Snoeren et al. [31] supported resumption through a client agent for the purpose of failover between replica servers, while keeping servers largely unchanged. In contrast, Resumption Agent is an application agnostic solution for network switching, providing automatic resuming capabilities for all pre-existing applications. Further, it can handle 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), e.g., as enabled through Virtual Wi-Fi [33]. They also retain pre-existing connections on their original AP, while assigning new connections to new APs. Yet, our work focuses on smartphones and presents mechanisms for switching between multiple, heterogeneous networks. We go well beyond retaining pre-existing connec-

tions by addressing long-lived flows and supporting pre-existing 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 particular for the purpose of migrating flows between networks, by providing detailed analysis of traffic protocols, flow length and concurrency, and the active application concurrent to the flows. These findings are crucial for designing and evaluating the feasibility of network migration.

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].

III. 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 IV.

A. iPhone Field Trace Collection

We gathered real-life network traces from 27 iPhone 3GS users over the course of 3 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 because it represents the cutting edge of smartphone design for usability, accounting for 55% 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 from the Apple App Store and numerous third-party repositories.

Whenever the phone's CPU is not asleep, LiveLab records TCP network connection statistics every two seconds using the *netstat* tool, also 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 two seconds and 15 minutes, respectively. Finally, it recorded the complete packet headers for three of the participants over one month, in order 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 is 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 provides an unprecedentedly detailed look into the connectivity on contemporary mobile devices.

B. Focus on TCP Flows

The packet-level logging data shows 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 to for diagnostics, device discovery and error messages specific to each network. Therefore, for the purpose of switching between networks, they can be
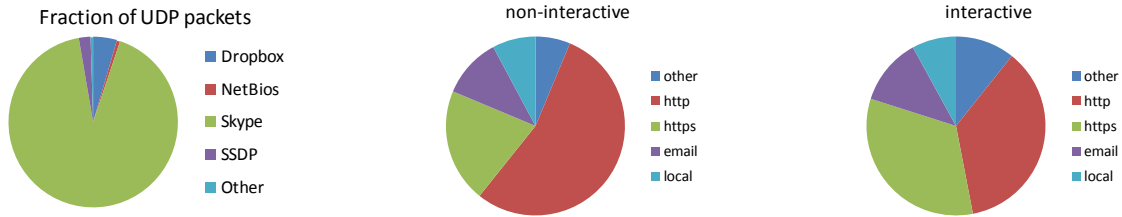
Fig 1. Fraction of packets according to application for UDP packets (Left). Fraction of TCP flows for each application type (Center: non-interactive sessions. Right: interactive sessions, i.e. phone display was on).
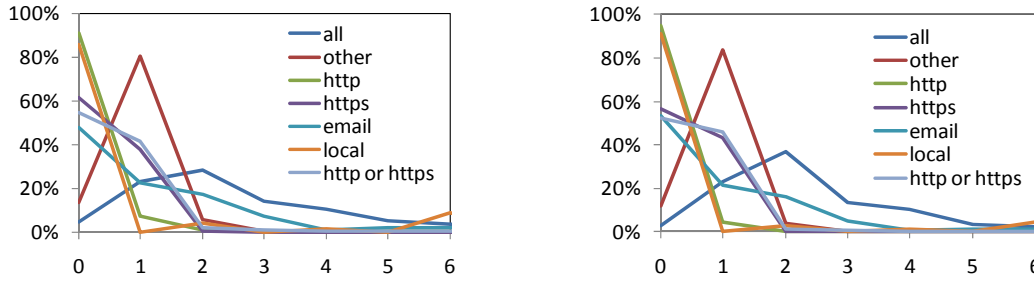

Fig. 2: We observed few concurrent TCP flows (Median = 2), and non-interactive and interactive usage were similar. Distribution of number of concurrent TCP flows for different TCP ports, average among all users. (Left: non-interactive sessions. Right: interactive sessions.)
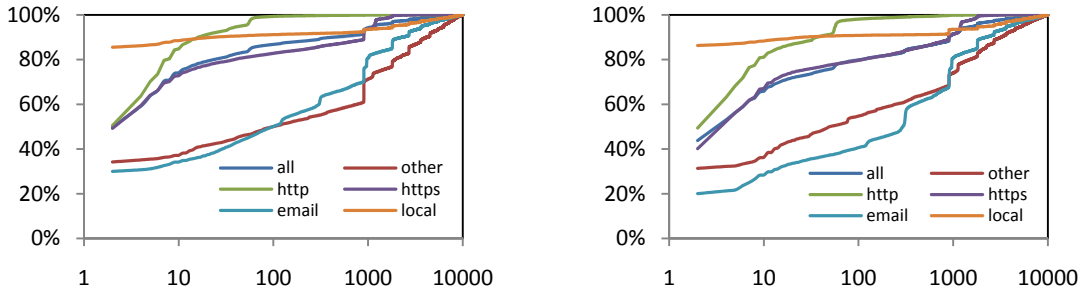

Fig. 3: Flows during interactive usage have slightly shorter lifetimes. CDF of TCP flow lifetimes in seconds, based on TCP port, average among all users. (Left: non-interactive sessions. Right: interactive sessions.)

safely ignored. TCP, and UDP account for 93% and 7% of all remaining packets, respectively. TCP flows present the main challenge towards flow migration. While we will examine TCP flows in details later, we will first discuss UDP flows. We analyze the UDP flows based on port numbers, and further corroborate this analysis with the applications being used. Notably, the phones were almost always listening to all UDP ports. We have found the following services and applications utilize UDP on the phones (Fig. 1):

- Skype (92%) uses UDP ports 12340 and 20515.
- Dropbox (4%) uses UDP broadcast on port 17500
- Simple Service Discovery Protocol (SSDP) (2%) is used to advertise and discover network services.
- NetBIOS (1%) for local area network device discovery and networking
- Other (<1%) 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. We will analyze how Skype can be migrated to a different network in Section III.D.1. For the remainder of this paper, we will focus on TCP flows, unless mentioned otherwise. Using the port number of the server, we divide external

TCP flows into three categories:

- Web (HTTP: 80, HTTPS: 443): These are used by not only the browser, but also by a number of native applications that utilize web services or a built-in browser.
- Email (IMAP: 143, 993, POP3: 110, 995, SMTP: 25, 465): These are used by the native email client, and will not include email 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 non-interactive 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.

Fig. 2 and Fig. 3 show the distribution of the number of flows and the CDF of flow lifetimes, respectively, according to TCP port for both interactive and non-interactive sessions whenever the phone's CPU was running. We can see that flows have similar characteristics during interactive and non-interactive 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.
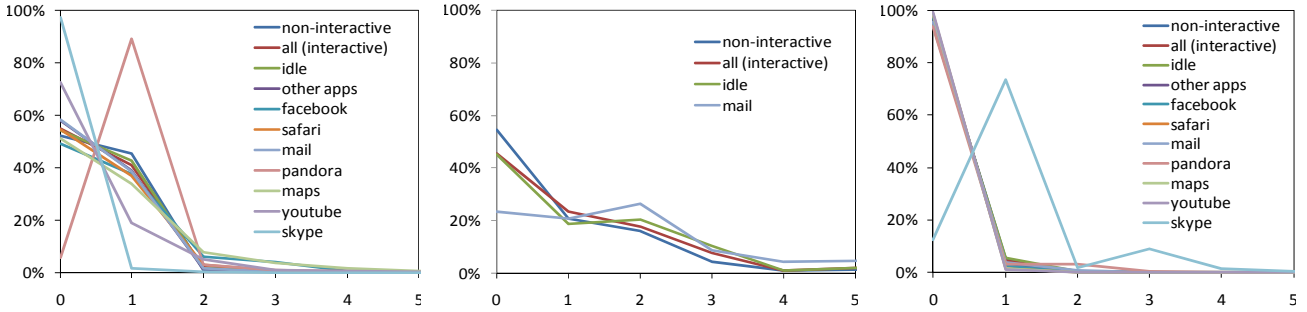
Fig. 4: Distribution for the number of TCP flows when running different Internet applications. (Left): Web. (Center): Email. (Right): other ports, excluding the Push Service.
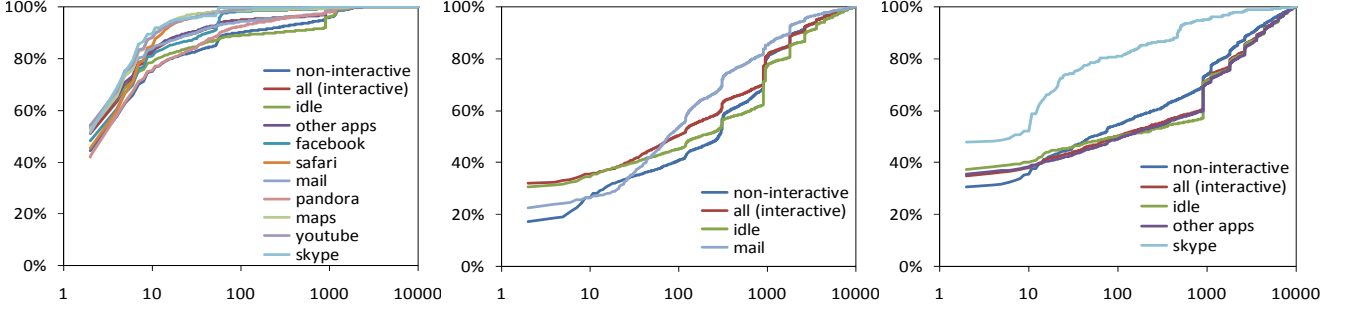

Fig. 5: CDF of TCP flow lifetimes (seconds), based on active application. (Left): web ports. (Center): email ports, (Right): other ports.

C. Flow Concurrency

While analyzing the LiveLab data, we were surprised to discover that there are few concurrent flows on the iPhone platform, with negligiable difference between interactive and non-interactive sessions. The median number of flows was 2. However, there almost always exists one particular flow, 97% 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 distribution 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 III.B.1 (web, email, 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% of interactive phone Internet use. Non-interactive usage, including when the display was off, idle time, when the home screen was displayed are presented separately. Other applications, including those without specifically requiring internet connectivity, are clustered together as others. For email 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. non-striping) are effective for at most 20% of flows*, as the other 80% of times when a web flow exists, it is a single flow. However, we expect this number to increase as more applications and services on mobile devices become available. The mail application, while not typically data intensive, presents

an exception, as it regularly uses multiple flows when active.

D. Flow Lifetime

We have found that most interactive flows on the phone were short lived, and it is often possible to automatically predict long-lived flows. We measure the flow lifetime without including the connection / teardown phase (e.g. wait_fin). Our logs show a wide variation in the lifetime of TCP flows on the experimental phones, in particular between interactive and non-interactive usage sessions.

Fig. 5 shows, on average among our participants, the CDF (cumulative distribution function) of TCP session lengths for different TCP ports and different active applications, similar to Section III.C.

Our first finding is that *most flows are short lived*. In fact, 50% and 44% of flows for non-interactive 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 email flows (i.e., IMAP, SMTP, POP3) is much lower: 30% and 20% for non-interactive 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.

1) Long-lived Non-Standard TCP Flows

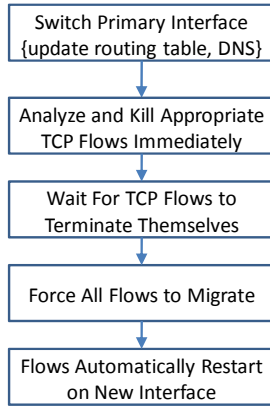We next consider long-lived flows that use non-standard

Fig. 6: Flowchart for Wait-n-Migrate

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

protocols, other than web, ftp, and email. 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, non-interactive 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 non-standard 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 re-establish 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. For yet another example, as long as the primary interface in the system routing table is correctly updated, e.g. 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 (~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 notifying the system of the network change for the benefit of applications that can handle disruptions gracefully, instead of simply losing connectivity, as performed by AutoSwitch in Section VI.

E. 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. Yet, increased multitasking will not reduce the usability and effectiveness of the general case of Wait-n-Migrate mechanism or the Resumption Agent mechanism. We note that Android and the newly released iPhone iOS 4.0 allow background appli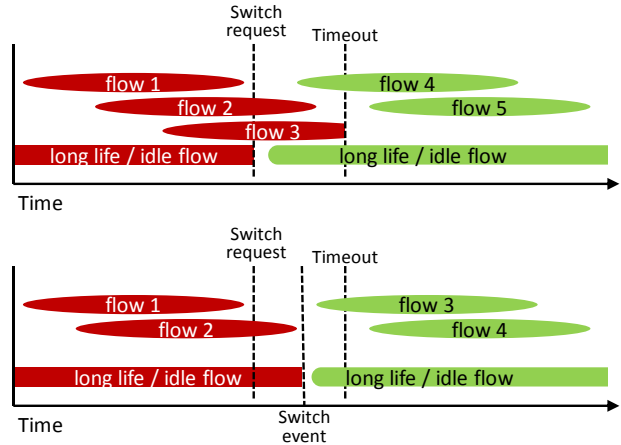cations, e.g. Skype and Pandora, to access data networks [41]. This, alongside the increasing processing power and memory of phones, suggests an increase in the usage of background capable applications (e.g. instant messaging, Twidroid). Therefore, we would expect to see an increase in the number of simultaneous network flows, from those shown in Fig. 2 and Fig. 4.

For both Wait-n-Migrate and Resumption Agent, 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, an increase in the number of multitasked applications will not affect the general performance of our mechanisms.

IV. MIGRATION WITHOUT NETWORK SUPPORT

Based on the findings from Section III, for the purpose 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 network support. 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.

A. Wait-n-Migrate

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

In order 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 III. Different wait-time values can be used for switching to different net-

works. For example, when the system policy requests a network switch to a slower or less efficient network, e.g. in order 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.

When the system cannot be connected to both networks simultaneously, a special case of Wait-n-Migrate can be used. This special case takes advantage of the fact that that most TCP flows are short lived. It monitors TCP flows and attempts to choose the best moment to switch within a specifically allowed time range, in order to minimize disruptions. This is possible through the statistical properties of TCP flows, as presented in Section III.

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 in order to avoid user disruption. However, for flows that are known to be highly likely to live beyond the wait time, e.g. based on the findings in Section III, Wait-n-Migrate can terminate them immediately. For example, we already know that several types of flows are long lived, e.g., Push notifications and idle email 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.

B. 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 III.B, 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 that allows resuming from a specified location in the 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 web page. The FTP standard also supports resuming via the *rest* command. Standard email protocols (e.g. IMAP, POP, and SMTP) can also be restarted from the beginning of any email, 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. Finally, 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*. If Resumption Agent is used in conjunction with Wait-n-Migrate, the content length and bandwidth can further assist in determining whether to kill flows immediately or wait for them to terminate normally.

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 in 4.2.1. 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 pre-existing applications. Moreover, Resumption Agent can handle network migration and two non-trivial challenges to Resumption Agent for web flows, posed by unsupported content and encrypted HTTPS flows. We next discuss them and present our solutions.

1) Unsupported Content

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

(*i*) 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.

(*ii*) The second group is content uploads, usually using HTTP POST, in which there is always the risk of repeating an action, e.g. 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 behaviour and will avoid automatically resuming such a transfer if it is disconnected.

(*iii*) 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, in order 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, e.g. 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
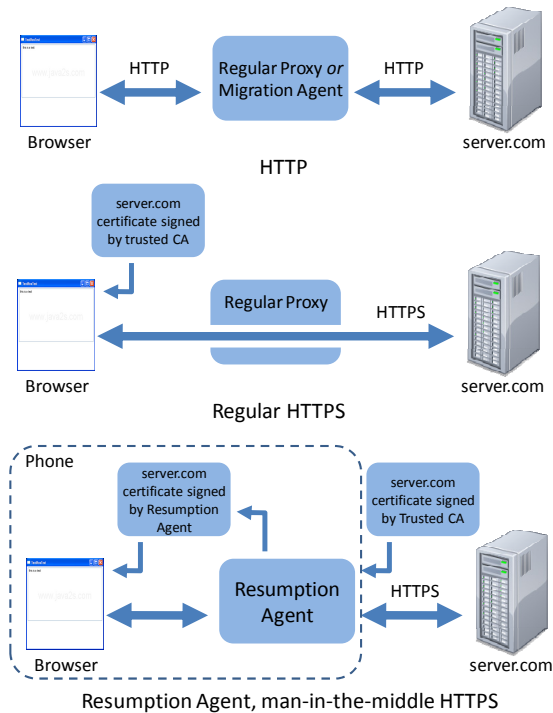
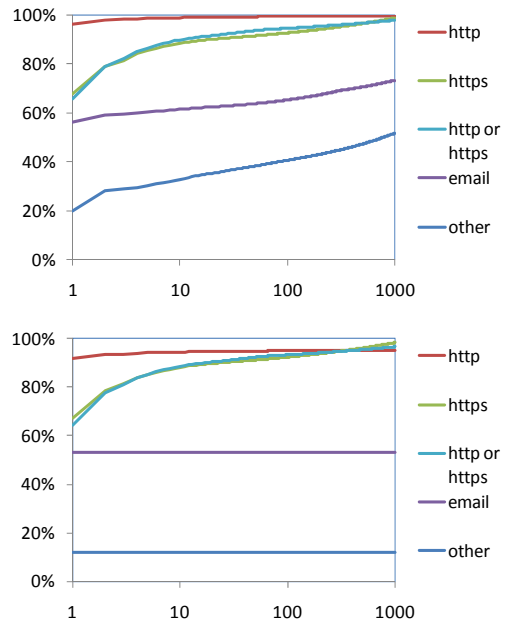Fig. 8: Regular proxy operation and Resumption Agent man-in-the-middle operation for a browser application

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

applying an offset, will Resumption Agent abort the resume and the transfer will fail.

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, upon validating the eligibility request, will create 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.

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, in order 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. 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. Furthermore, to completely avoid this latency, the device can use the standard CONNECT command without man-in-the-middle operation the first time the user accesses a new site, but generate the certificate for subsequent accesses.

In order 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 than application specific warnings. Therefore, Resumption Agent can in fact reduce bad decisions by users and increase overall security.

Finally, in order to improve protection against rouge applications that may be running directly on the phone, it is important for Resumption Agent to generate per-device random CAs in order to prevent fake certificates copied across devices running Migration Agent.
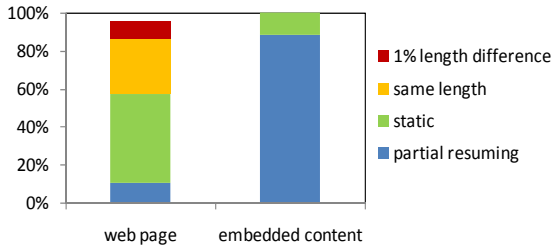
Fig. 10: Most web pages and all of their embedded content (among our top 100 pages) are supported by Resumption Agent

Fig. 11: Non-static web pages 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)

C. 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 probability of the system attempting a switch. 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.

1) Wait-n-Migrate

As mentioned in Section IV.A, 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. We have used our traces to calculate the percentage of flows that Wait-n-Migrate can successfully migrate to the new network without disruption, shown in Fig. 9. For example, Wait-n-Migrate successfully migrates all web flows for 90% and 95% of cases for wait-time values of 10 and 100 seconds, respectively.

As discussed in Section IV.A, there is a special case of Wait-n-Migrate that is employed when the system can only remain connected to one network, which waits for the moment where there are no ongoing flows to switch the network. For our evaluation, we assume that this special case 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 our policy does not wait for presumably non-interactive flows (i.e. non-web) to end, we can see a significantly larger number of disconnections for those flows. Yet, 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 III. However, we believe that increased complexity and multitasking in future applications will increase the performance difference of Wait-n-Migrate and its special case.

2) Resumption Agent

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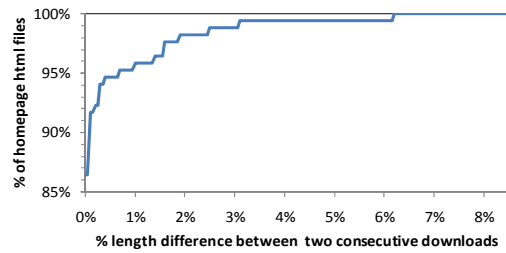
We have tested YouTube and it is fully supported by Resumption Agent; the stream is based on standard HTTP protocols and our experiments show that YouTube servers indeed support resuming videos at an arbitrary location.

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) since we found that many deeper, pages may depend on previous state information, e.g. 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 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% of embedded media is static and therefore supported by Resumption Agent. 91% of those support resuming from the middle of a transfer; i.e. without the need to re-transfer the already transferred part. Among the HTML homepages, 57% were static, and 9% 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 Cumulative Distribution Function (CDF) for the length differences between two consecutive downloads for our top 100 pages. We can see that another 30% had content lengths within 1% of each other. Therefore, we expect them to be supported by Resumption Agent, as described in Section IV.B.1. We note that only 16% of the HTML pages can be resumed from the middle of the transfer, vs. 89% of the embedded content. The remaining pages that do not support HTTP resume functionality incur an extra overhead of re-downloading 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% of the HTML pages were marked as such. However, of the HTML pages that disallowed caching, 36% in fact had static content, and 44% had content with the same length. Therefore, we conjecture that the no-cache response header may possibly be ignored by Resumption Agent.

```
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

V. 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.

A. 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:

(*i*) *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. Towards 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.

(*ii*) *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 outgoing sockets. 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 quite small, as it simply changes a system setting, and takes less than 300ms 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 in order to receive calls.

(*iii*) *Terminating Flows:* As previously mentioned, it may be necessary to force the migration of specific flows, such as those that are known to be of long 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 which kills the connection. When the application reconnects it is automatically routed through the new network.

(*iv*) *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 behaviour.

B. Resumption Agent

We implemented Resumption Agent in 1400 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 non-blocking 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
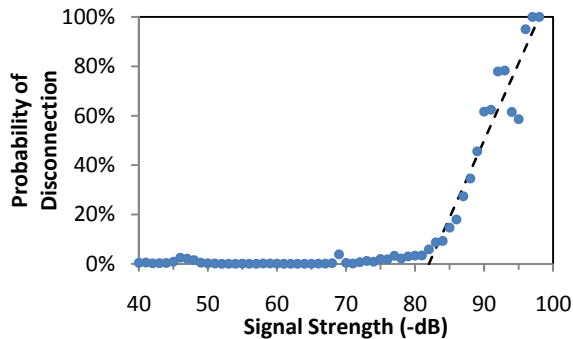
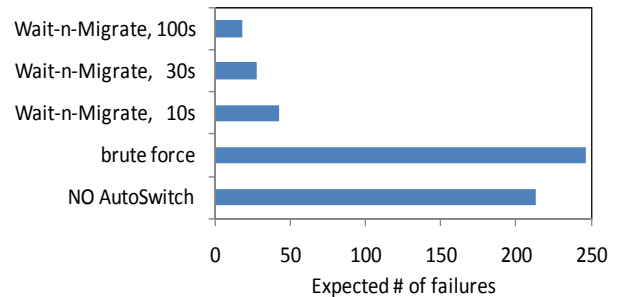Fig. 13: Probability of disconnection vs. Wi-Fi signal strengths

Fig. 14: AutoSwitch significantly reduces the expected number of disruptions in 1120 hours of interactive usage traces with Wi-Fi enabled, for different wait-time values

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, in order to reduce CPU usage and bandwidth without increasing latency, the worker threads employ a large read size of 2KB, 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 timeout option to be specified, to report a disconnection if no data has been sent after the specified timeout. Unfortunately, while the iPhone appears to implement this option, it failed to function. Thus, we implemented our own timeout detection, with the same behaviour. If a timeout is detected, or the socket throws any error, the worker thread re-establishes 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 potential flooding the network, if the server or network becomes unavailable for an extended period 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% 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.

VI. EXAMPLE POLICY: AUTOSWITCH

In order 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 [44] – 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, e.g. [5], but they typically rely on a mobility gateway / proxy to handle network switches,

with inherent latency and deployability drawbacks as discussed in Section II.B.

A. AutoSwitch Design

AutoSwitch attempts to migrate TCP flows from Wi-Fi to cellular before Wi-Fi coverage is dropped or Wi-Fi becomes unreliable, and migrate back to Wi-Fi when a reliable Wi-Fi connection becomes available again. For simplicity, and without losing generality, we assume that cellular coverage is always available.

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 [45] and our previous work [46] 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 [8, 47], to initiate a network switch before losing Wi-Fi coverage completely.

In order 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.

B. Trace-based Evaluation

We have used the traces from LiveLab to evaluate the efficacy of AutoSwitch using Wait-n-Migrate, during rou-
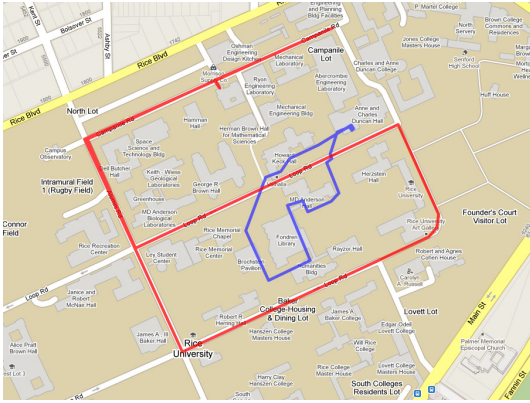
Fig. 15: Map of paths travelled in Rice University, for walking (1 km, Blue loop) and driving (3 km, Red loop) scenarios
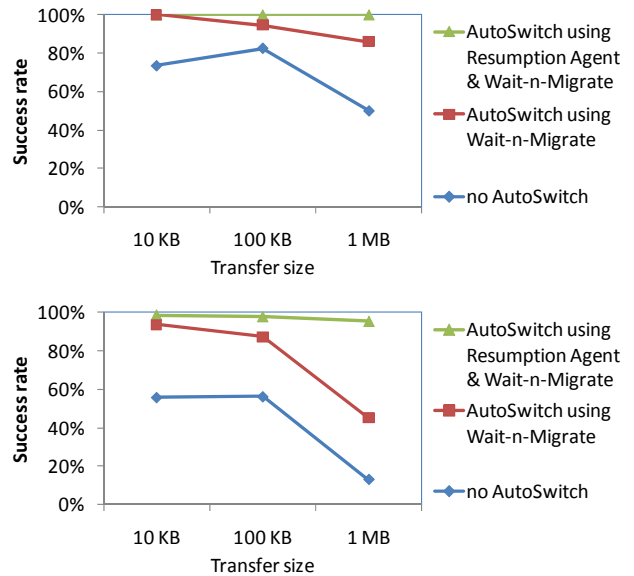

Fig. 16: AutoSwitch significantly increases the success rate of 10KB, 100KB, and 1MB transfers when walking (top) and driving (bottom) on Rice Campus

tine interactive usage. As mentioned in Section VI.A, 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 disruption 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.

We present the expected number of disconnections for AutoSwitch using Wait-n-Migrate, with wait-times of 10, 30, and 100 seconds, respectively, in Fig. 14. We compare it with two cases, one where Wi-Fi is left on (no AutoSwitch), and another case where AutoSwitch switches between networks in a brute force manner, without utilizing Wait-n-Migrate.

Using 1120 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% 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 II.A and prior work [44]. Therefore, they may turn off Wi-Fi altogether in conditions they know it is prone to failing. Hence, we expect that the results in this section, obtained from the traces when Wi-Fi was enabled, underestimate the potential benefit from AutoSwitch using Wait-n-Migrate.

C. 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 1MB, as well as Wait-n-Migrate alone. We then measured the 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 travelling 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 signal strength or more), and was covered approximately 95% of the time by a Wi-Fi signal. The driving path was approximately 3 km long and only had one area of good Wi-Fi signal strength, but still had about 80% Wi-Fi coverage. Each test run lasted approximately one hour, and included over 1000 transfer attempts.

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% for different file sizes.

VII. 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. The system mechanisms we have presented here can also be utilized to enable the immediate deployment of many performance and efficiency-enhancing policies studied in the 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 support striping larger transfers, i.e. download different parts of the transfer simultaneously through different networks, as long as the content supports resuming. Resumption Agent can be extended to download separate chunks over each interface and then amalgamate these chunks before sending them to the client. For striping content that contains dynamic parts, as described in Section IV.B.1, it is necessary to ensure the dynamic portions are downloaded in single chunks.

Mirroring: For pages that do not support striping, or that are very small compared to the latency, Resumption Agent can be extended to simultaneously request the same page on multiple networks, and return whichever finishes first. While this method reduces efficiancy, it can provide substantial reduction in 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.

VIII. 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] Rahmati, A. and Zhong, L. Context-for-Wireless: Context-Sensitive Energy-Efficient Wireless Data Transfer. *Proc. Int. Conf. Mobile Systems, Applications and Services (MobiSys)*. 2007.

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

[3] Thompson, N., He, G. and Luo, H., Flow scheduling for end-host multihoming. in *IEEE INFOCOM*, Citeseer, 2006.

[4] Kandula, S., Lin, K.C.-J., Badirkhanli, T. and Katabi, D. FatVAP: aggregating AP backhaul capacity to maximize throughput. *Proc. USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2008.

[5] Balasubramanian, A., Mahajan, R. and Venkataramani, A. Augmenting Mobile 3G Using WiFi. *Proc. Int. Conf. Mobile Systems, Applications and Services (MobiSys)*, 2010.

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

[7] Pering, T., Agarwal, Y., Gupta, R. and Want, R. CoolSpots: Reducing the Power Consumption of Wireless Mobile Devices with Multiple Radio Interfaces. *Proc. Int. Conf. Mobile Systems, Applications and Services (MobiSys)*. 220-232, 2006.

[8] Giannoulis, A., Fiore, M. and Knightly, E.W. Supporting vehicular mobility in urban multi-hop wireless networks *Proceeding of the 6th international conference on Mobile systems, applications, and services*, ACM, Breckenridge, CO, USA, 2008.

[9] Lai, A. and Nieh, J. Limits of wide-area thin-client computing. *SIGMETRICS Perform. Eval. Rev., 30* (1). 228-239, 2002.

[10] Agresti, A. and Coull, B.A. Approximate Is Better than "Exact" for Interval Estimation of Binomial Proportions. *The American Statistician, 52* (2). 119-126, 1998.

[11] Gustafsson, E. and Jonsson, A. Always Best Connected. *IEEE Wireless Communications, 10* (1). 49-55, 2003.

[12] Pahlavan, K., Krishnamurthy, P., Hatami, A., Ylianttila, M., Makela, J., Pichna, R. and Vallstron, J. Handoff in hybrid mobile data networks. *IEEE Personal Communications, 7* (2). 34-47, 2000.

[13] Maltz, D. and Bhagwat, P., MSOCKS: An architecture for transport layer mobility. in *IEEE INFOCOM*, 1998.

[14] Chalmers, R. and Almeroth, K., A mobility gateway for small device networks. in *IEEE PerCom*, Citeseer, 2004.

[15] Sharma, P., Lee, S., Brassil, J. and Shin, K., Handheld routers: Intelligent bandwidth aggregation for mobile collaborative communities. in *IEEE BroadNets*, Citeseer, 2004.

[16] Pucha, H. and Hu, Y. Overlay TCP: Ending end-to-end transport for higher throughput. *Poster in ACM SIGCOMM*, 2005.

[17] Hsieh, H. and Sivakumar, R. pTCP: An end-to-end transport layer protocol for striped connections. *Proc. IEEE ICNP*, 2002.

[18] Han, H., Shakkottai, S., Hollot, C., Srikant, R. and Towsley, D. Overlay TCP for multi-path routing and congestion control. *IEEE/ACM Trans. Networking*, 2006.

[19] Kandula, S., Katabi, D., Sinha, S. and Berger, A. Dynamic load balancing without packet reordering. *SIGCOMM Comput. Commun. Rev., 37* (2). 51-62, 2007.

[20] Traw, C. and Smith, J. Striping within the network subsystem. *IEEE Network, 9* (4). 22-32, 1995.

[21] Sivakumar, H., Bailey, S. and Grossman, R.L. PSockets: the case for application-level network striping for data intensive applications using high speed wide area networks *Proc. ACM/IEEE conf. on Supercomputing (CDROM)*, IEEE Computer Society, 2000.

[22] Hsieh, H., Kim, K., Zhu, Y. and Sivakumar, R. A receiver-centric transport protocol for mobile hosts with heterogeneous wireless interfaces. *MobiCom*, 2003.

[23] Rodriguez, P., Chakravorty, R., Chesterfield, J., Pratt, I. and Banerjee, S. MAR: a commuter router infrastructure for the mobile Internet *Proc. Int. Conf. Mobile Systems, Applications and Services (MobiSys)*, ACM, Boston, MA, USA, 2004.

[24] Kim, S. and Copeland, J., TCP for seamless vertical handoff in hybrid mobile data networks. in *IEEE GLOBECOM*, 661-665, 2003.

[25] Kim, K.-H., Zhu, Y., Sivakumar, R. and Hsieh, H.-Y. A receiver-centric transport protocol for mobile hosts with heterogeneous wireless interfaces. *Wirel. Netw., 11* (4). 363-382, 2005.

[26] Stemm, M. and Katz, R. Vertical handoffs in wireless overlay networks. *Mobile Networks and Applications, 3* (4). 335-350, 1998.

[27] Perkins, C., Alpert, S. and Woolf, B. *Mobile IP; Design Principles and Practices*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1997.

[28] Perkins, C. Mobile IP. *IEEE Communications Magazine*, *35* (5). 84-99, 1997.

[29] Huang, J., Xu, Q., Tiwana, B., Mao, Z., Zhang, M. and Bahl, P. Anatomizing Application Performance Differences on Smartphones. *Proc. Int. Conf. Mobile Systems, Applications and Services (MobiSys)*. 165-178, 2010.

[30] Nirjon, S.M.S., Nicoara, A., Hsu, C.-H., Singh, J.P. and Stankovic, J. MultiNets: Policy Oriented Dynamic Switching of WirelessInterfaces on Mobile Devices. *under review*

[31] Snoeren, A.C., Andersen, D.G. and Balakrishnan, H. Fine-grained failover using connection migration *Proc. USENIX Symp. on Internet Technologies and Systems*, USENIX Association, San Francisco, California, 2001.

[32] Alperovich, T. and Noble, B. The case for elastic access *Proc. ACM int. wrkshp. on Mobility in the evolving internet architecture (MobiArch)*, ACM, 2010.

[33] Chandra, R. and Bahl, B. MultiNet: Connecting to Multiple IEEE 802.1 1 Networks Using a Single Wireless Card. *Proc. IEEE InfoCom*, 2004.

[34] Falaki, H., Mahajan, R., Kandula, S., Lymberopoulos, D., Govindan, R. and Estrin, D. Diversity in Smartphone Usage. *Proc. Int. Conf. Mobile Systems, Applications and Services (MobiSys)*, 2010.

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

[36] Rahmati, A., Shepard, C., Nicoara, A., Zhong, L. and Singh, J.P. MobiCom 2010 Poster: Mobile TCP usage characteristics and the feasibility of network migration without infrastructure support. *SIGMOBILE Mob. Comput. Commun. Rev.*, *14* (4). 2010.

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

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

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

[40] Dogar, F. and Steenkiste, P. Catnap: Exploiting High Bandwidth Wireless Interfaces to Save Energy for Mobile Devices. *Proc. Int. Conf. Mobile Systems, Applications and Services (MobiSys)*, 2010.

[41] Buchanan, M. Gizmodo Blog: How Multitasking Works on a Phone. http://gizmodo.com/5527407/giz-explains-how-multitasking-works-on-a-phone, April 29, 2010.

[42] Squid Web Cache. Squid-in-the-middle SSL Bump http://wiki.squid-cache.org/Features/SslBump.

[43] Cryptopath blog. iPhone certificate flaws http://cryptopath.wordpress.com/2010/01/29/iphone-certificate-flaws/.

[44] Rahmati, A. and Zhong, L. A longitudinal study of non-voice mobile phone usage by teens from an underserved urban community, Tech. Rep. 0515-09, Rice University, 2009.

[45] Nicholson, A.J. and Noble, B.D. BreadCrumbs: Forecasting Mobile Connectivity. *Proc. Int. Conf. Mobile Computing and Networking (MobiCom)*. 46-57, 2008.

[46] Rahmati, A. and Zhong, L. Context-based network estimation for energy-efficient ubiquitous wireless connectivity. *IEEE Transactions on Mobile Computing*, 2010.

[47] Kim, S.K., Kang, C.G. and Kim, K.S., An adaptive handover decision algorithm based on the estimating mobility from signal strength measurements. in *Vehicular Technology Conference (VCT)*, 1004-1008 Vol. 1002, 2004.

Ahmad Rahmati is a graduate student of Electrical and Computer Engineering at Rice University, Houston, TX. He received his M.S. degree in Electrical and Computer Engineering from Rice University in 2008, and his B.S. degree in Computer Engineering from Sharif University of Technology in 2004. He has been a research intern at AT&T Labs Research, Motorola Labs, and Deutsche Telekom R&D Lab. USA, in 2006, 2008, and 2010, respectively. His publications have received the ACM SIGCHI/SIGMOBILE MobileHCI Best Paper Award in 2007, and have been featured as the spotlight paper of the IEEE Transactions on Mobile Computing twice, in 2010 and 2011.

Clayton Shepard is a graduate student of Electrical and Computer Engineering at Rice University, Houston, TX. He received his B.S. degree in Electrical and Computer Engineering from Rice University in 2008.

Chad C. Tossell is a graduate student in psychology at Rice University in Houston, Texas and a behavioral scientist in the United States Air Force. He received his M.S. in Applied Psychology from Arizona State University in 2006 and his B.S. in Psychology from the University of California, Berkeley in 2003.

Lin Zhong received his B.S. and M.S. from Tsinghua University in 1998 and 2000, respectively. He received his Ph.D. from Princeton University in September, 2005. He was with NEC Labs, America, for the summer of 2003 and with Microsoft Research for the summers of 2004 and 2005. He joined the Department of Electrical & Computer Engineering, Rice University as an assistant professor in September, 2005. He received the AT&T Asian-Pacific Leadership Award in 2001 and the Harold W. Dodds Princeton University Honorific Fellowship for 2004-2005. He coauthored a paper that is selected as one of the 30 most influential papers in the first 10 years of Design, Automation & Test in Europe conferences. He and his students received the best paper awards from ACM MobileHCI 2007 and IEEE PerCom 2009. His research interests include mobile & embedded system design, human-computer interaction, and nanoelectronics. His research has been funded by National Science Foundation, Motorola Labs, Nokia, Texas Instruments, and Microsoft Research.

Philip Kortum is currently a faculty member in the Department of Psychology at Rice University in Houston, Texas. Prior to joining Rice, he worked for almost a decade at SBC Laboratories (now AT&T Laboratories) doing human factors research and development in all areas of telecommunications. Dr. Kortum continues to do work in 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). He received his Ph.D. from the University of Texas at Austin.

Angela Nicoara is a Senior Research Scientist at Deutsche Telekom R&D Lab USA since 2008. She completed her graduate studies with a PhD from ETH Zurich, Switzerland in 2007 and her undergraduate studies at Politehnica University Timisoara, Romania with a Computer Science degree in 2002. Her research interests are in the areas of services and mobile platforms, adaptive software architectures, virtual machines, middleware for mobile and distributed computing, and distributed systems. Her current research activities include the development of open and programmable mobile platforms (e.g., Android) and novel information technology services to shape the emerging trends in fixed and mobile infrastructure and services sectors. Angela has served as a general chair, industry co-chair, program committee or scientific reviewer member for EuroSys, VMIL, LCN–WNM, MobiCASE, Communications of the ACM. She had contracts and internships with Google Inc. and Webquote in California, as well as a German software company Caatoosee.

Jatinder Singh is