MultiNets: Policy Oriented Real-Time Switching of Wireless Interfaces on Mobile Devices

Shahriar Nirjon[1], Angela Nicoara[2], Cheng-Hsin Hsu[3], Jatinder Singh[4], and John Stankovic[1]

[1]Department of Computer Science, University of Virginia, Charlottesville, VA
[2]Deutsche Telekom R&D Laboratories USA, Los Altos, CA
[3] Department of Computer Science, National Tsing Hua University, Hsin-Chu, Taiwan
[4]Department of Electrical Engineering, Stanford University, CA
smn8z@virginia.edu, angela.nicoara@telekom.com, chsu@cs.nthu.edu.tw, jatinder@stanford.edu,
stankovic@virginia.edu

Abstract—In this paper we present MultiNets, a system which is capable of switching between wireless network interfaces on mobile devices in real-time. MultiNets is motivated by the need of smartphone platforms to save energy, offload data traffic, and achieve higher throughput. We describe the architecture of MultiNets and demonstrate the methodology to perform switching in Linux based mobile OSes such as Android. Our analysis on mobile data traces collected from real users shows that with real-time switching we can save 27.4% of the energy, offload 79.82% of the data traffic, or achieve 7 times more throughput on average. We deploy MultiNets in a real world scenario and our experimental results show that depending on the user requirements, it outperforms the state-of-the-art Android system either by saving up to 33.75% energy, or achieving near-optimal offloading, or achieving near-optimal throughput while substantially reducing TCP interruptions due to switching.

Keywords-3G; WiFi; TCP; Switching; Offload; Energy;

I. INTRODUCTION

Cellular networks today provide nationwide coverage in several countries all over the globe. The proliferation in mobile applications like mobile TV, video on demand, video conferencing, tele-medicine, and numerous location based services is attracting an increasing number of consumers. However, the challenges to effective use of mobile networks remain manifold. Firstly, the mobile data traffic is soaring at a high rate. A recent study forecasts that global mobile data traffic will increase by 39 times with a compound annual growth rate of 108% over the next five years [2]. Secondly, the usable battery lifetime of mobile devices has become alarmingly low with feature packed mechanisms like touch screens and accurate positioning system. Thirdly, modern mobile applications typically require high throughput and/or fast response time which are difficult to deliver with scarce cellular bandwidth and expensive spectrum.

Contemporary mobile devices come equipped with network interfaces such as WiFi. This offers an attractive proposition to alleviate the staggering increase in data traffic over cellular networks owing to high bandwidth and low cost (or, often no cost) offered by WiFi networks. Although, the spatial coverage of WiFi is not comparable to the cellular networks due to short coverage range of the WLAN technology, the availability of WiFi is becoming more pervasive in houses, offices, campuses, stores, coffee shops, and even many public transport systems - locations where most mobile users tend to spend most of their time.

The state-of-the-art devices tend to leave the choice of selecting the network to the end-user, which we argue is not only inefficient but also undesirable in terms of smartphone usability. A user should instead be able to decide the high level goal and the device should switch to suitable interface to achieve that intent. By *real-time switching*, we mean activating a new network interface and deactivating the current one - dynamically and without interrupting existing connections. Real-time, in this context, stands for switching interfaces in real-time as opposed to meeting any deadline. Multifold benefits can be realized by switching in real-time. For example, battery life is prolonged if the device stays over the cellular network during its idle time and switches to WiFi during its data activity. An end-user who is concerned about the battery life can set the device to energy-saving mode, and the device can monitor the user's activity and perform the switching when appropriate. Similarly, by switching to the network that has the highest bandwidth, the mobile device can provide a better user experience with faster data rates. Switching is also a solution to the skyrocketing data problem faced by the mobile operators who want to offload data traffic to WiFi network to conserve cellular bandwidth.

Switching from one network interface to the other is challenging due to the connection-oriented nature of the ongoing data sessions. Unless properly dealt with, switching between interfaces results in interruptions, loss of data, and undesirable user experience. Existing works that attempt to solve this problem either require additional infrastructure supports such as gateways [16], [3], [12], [17], [15], [14] and masters [11], or require changes in the network protocols [7], [18], [22], [8], [13], and thus are not practical

since modification to infrastructure tends to be extremely expensive and modification to a standard network protocols is not a compatible pathway with regard to existing and deployed systems and applications.

In this paper we present MultiNets, a pragmatic client-based solution, which does not require any changes to the network protocols, and enables existing applications to run transparently without any modification. MultiNets is able to switch between cellular and WiFi interfaces in real-time and makes switching decisions based on one of its three interface selection policies: *energy saving*, *data offloading*, and *performance*. These policies address three crucial needs of a mobile device in being able to conserve battery power, offload data to WiFi, and increase throughput, respectively. A user of mobile device equipped with MultiNets can select one of these high-level policies and MutiNets performs switching accordingly. We note that, the authors in [5] characterize TCP flows on iPhones to analyze the feasiblity of flow migration between interfaces. In contrast to our work, they do not consider issues such as the policies determining when to switch, or rigorously quantify different benefits that are achieved by switching.

We have implemented MultiNets on Android-based mobile devices. However, the design and principles of MultiNets are general enough to be adopted in any other mobile OSes. Like all state-of-the-art mobile OSes, Android does not perform dynamic switching. Access to network interfaces in Android is exclusive, i.e., either the cellular or the WiFi is active at a time. The cellular network is the default network and is assumed to always be present. On the other hand, WiFi has to be manually turned on, and typically the user is prompted to select WiFi when it is available. A limitation of Android is that switching is not seamless i.e., all the TCP connections are bound to be interrupted. Furthermore, when the device is connected to WiFi, switching back to the cellular network can only be done by manually turning off the WiFi connectivity. In MultiNets, we obviate this exclusive network access and make it possible to keep both the interfaces on simultaneously for seamless and non-disruptive switching. In addition, this feature can be used to simultaneously access multiple network interfaces by the applications that are developed on top of MultiNets.

This paper makes the following contributions:

- We conduct a three months long empirical study and summarize the TCP characteristics in Android smart-phones, complementing a similar study with iPhone users in [4], [5]. We devise a switching technique which is client-based, transparent to applications, and does not require any protocol changes.
- We design and implement three switching policies. Our analysis on usage data collected from real mobile device users shows that with switching, we can save 27.4% of the energy, offload 79.82% data traffic, or achieve 7 times more throughput on average.
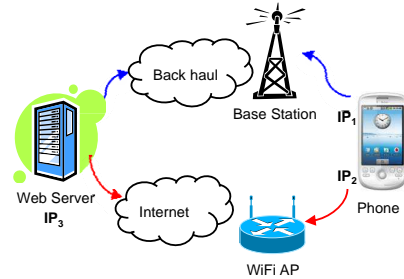

Figure 1.   A phone is trying to switch TCP sessions from 3G(IP_1) to WiFi(IP_2).

- We present MultiNets, which is to the best of our knowledge, the first complete system of this kind and demonstrate its performance in a real-world scenario. MultiNets outperforms the state-of-the-art Android system either by saving up to 33.75% energy, or achieving near-optimal offloading, or achieving near-optimal throughput while substantially reducing TCP interruptions due to switching.

II. SWITCHING NETWORK INTERFACES

Switching from one wireless interface to another is not as trivial as it may appear to be at first. Simply turning on one interface and turning off the other does not work as it results in interruptions, partially loaded web pages, loss of data, annoying error messages and user dissatisfaction in general. It is rather a challenging problem to *transfer connection oriented data traffic from one interface to another under the constraints of no user interventions, no interruptions, no changes of network protocols and requiring no extra support from the existing network infrastructure*.

Figure 1 illustrates this problem briefly. An end host, having two interfaces (IP_1 and IP_2) creates a TCP connection at it's port A with the remote server's (IP_3) port B. The connection is uniquely identified by the pair ($IP_1/A, IP_3/B$). We now analyze what happens if the host decides to turn off its interface, IP_1 and wants to continue the communication over IP_2. By changing the routes of all outgoing packets, the host may be able to send the next data packets using IP_2, but these packets will not be recognized as belonging to the same session at the server, as to the server, IP_1 and IP_2 appear as two different hosts. If we change the packet headers at the host to carry IP_1 as their sources even if they are sent using IP_2, the packets will be either dropped at IP_2's network or even if they get to the server, the ACKs will not reach the host as the server will send ACK to IP_1 which is closed.

A. Potential for Switching Interfaces

A client based solution that deals with this problem has to wait for all ongoing sessions over the first interface to finish, before it turns off the interface and activates the other one in order to prevent any interruptions. This waiting time can be theoretically infinite, but in practice, it depends on the usage
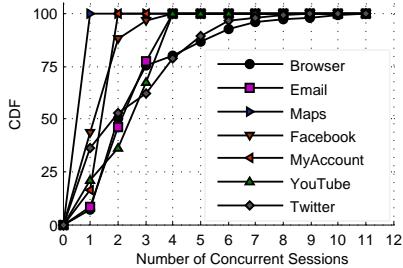
Figure 2. A steep rise in CDF in between $1-3$ indicates that the mean concurrency lies in that interval.

Figure 3. Applications with high concurrency tend to have most of their sessions with a lifetime of < 15 sec.

Figure 4. Data activity in long sessions are not continuous, rather they have an average burst length of ~ 3 sec.

of the phone and the characteristics of the applications that are running. To understand the type of data flows in mobile phones, we conduct a 3-months long experiment to collect the usage data from 13 Android phone users. These users are of different ages, demography, sex, and used a variety of applications. The results are therefore not homogeneous, rather highly diverse as evident later in Figures 11, 12, and 13. The results are also consistent with a 3-months long study involving 27 iPhone users in [4], [5].

In our study, the users used a total of 221 applications, and 35 of these applications require Internet connectivity. We analyze the collected data traces of these 35 applications and study the characteristics of the TCP sessions. We study TCP since our earlier work shows that almost all (99.7%) mobile traffic is TCP [4]. From this log, we try to answer three questions: *(1) how many concurrent TCP sessions are there at any instant of time within a mobile phone? (2) what are the durations of these sessions? and (3) how much activities are there over these sessions?* Answers to these questions are crucial since if we see that there are a large number of TCP sessions having long durations and high data activities, it is not worthy to wait for them and not wise to close them.

Figure 2 shows the cumulative distribution functions (CDF) of concurrent TCP sessions of the most popular 7 applications in the order of their usages. This is averaged over 10-minutes time windows of all users. This figure has to be studied in conjunction with Figure 3 which classifies these sessions into five classes based on their durations. In Figure 2, we observe that the concurrency of the TCP sessions has a steep rise in between $1-3$. This means that the average value (~ 2) lies in this region. Therefore, turning off the interface at any time may interrupt about 2 sessions on average, assuming that only one application is active at a time. In case of multi-tasking phones, the number of interruptions is not much higher. Applications like Browser and Twitter seem to show high concurrency (of maximum 10-11), but Figure 3 shows that about 80% of their sessions have lifetime < 15 seconds. For these applications, we may on average have to wait for 15 seconds before switching to the other interface. Applications like Email, Maps, and
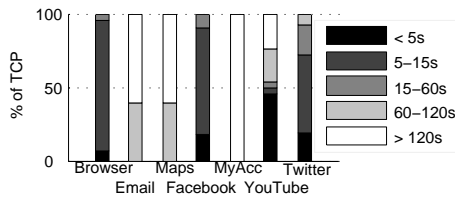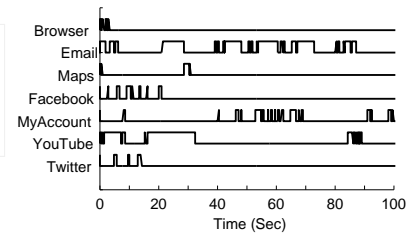
T-Mobile's MyAccount have very high percentage of long lasting sessions (> 120 seconds) which may seem a barrier to waiting for them to finish. However, the number of such long TCP sessions are very small (about 1), and with these TCP sessions, the applications keep themselves connected to a specific server (e.g., in case of Google Maps, it is `74.125.45.100`) for the entire lifetime, and that explains why they are so long.

We conduct further investigation to examine the data activities in these long sessions. Figure 4 shows the presence of data activity of the longest lasting TCP sessions of each of these applications for a randomly selected user for the first 100 seconds as an example. This shows that, the data activities over these longest TCP sessions are not continuous, but rather sporadic. Averaged over all usages we see an average of 3 seconds data activity between any two gaps over these sessions. This indicates that we may have to wait on average about 3 seconds for these applications before switching to a new interface to prevent any data loss. Although this technique will cause that TCP session to terminate, luckily, mobile phone applications are written keeping in mind of the sudden loss of network connectivity. Therefore, in such cases, when we switch to a new interface, the application considers it as a loss of connectivity and re-establishes the connection with the server. We empirically observed this in all 35 applications.

The characteristics of TCP sessions in mobile phones can be summarized as follows:

- Average lifetime of TCP sessions is ~ 2 seconds.
- Average concurrency of these sessions is < 2.
- TCP activities are in bursts of average ~ 3 seconds.
- There exists some sessions that are alive during the entire lifetime of the application, keeping the application connected to its server. Disconnections of such sessions are automatically reestablished by the application.

B. Switching in MultiNets

MultiNets handles connectionless and connection-oriented sessions separately during the switching. UDP and TCP are the dominant transport protocols that we have observed

in Android, and therefore we use them in this section for illustration.

1) Connectionless Sessions: Connectionless sessions (e.g., UDP) are rare: less than 0.3% traffic amount. They are easier to switch. UDP applications communicate using `DatagramSocket` and each connection is bound to a port and assigned an IP address of an available interface by the OS. To switch network interface, MultiNets first turns on the new interface and removes the default route over the old interface. We have found that doing so does not affect the functionality of `DatagramSocket`: the out-bound traffic is sent with the IP of the new interface, while the in-bound traffic is received at the old interface. MultiNets then turns off the old interface, which initially incurs some packet loss of the in-bound traffic, but we have observed that, in most cases, this is handled by the application layer.

2) Connection-Oriented Sessions: Connection-oriented sessions are mostly TCP, comprising of about 99.7%. These are trickier to switch as explained earlier. MultiNets performs the following steps for switching these sessions:

Step 1: MultiNets counts the number of ongoing TCP connections on the *old* interface. We should not harm these connections. We exclude the sessions that have gone past the ESTABLISHED state during the counting.

Step 2: If the count is non zero, MultiNets adds new routing table entries for all these connections explicitly specifying the destination address, gateway and mask fields for the *old* interface. This is to ensure that the ongoing TCP sessions still remain in the *old* interface.

Step 3: MultiNets now brings up the *new* interface and adds routing table entries for it including the default route and removes the default route of the *old* interface from the routing table. Any new connections start using the *new* interface from now on.

Step 4: MultiNets waits for the ongoing TCP sessions over the *old* interface to finish or until a timeout (determined experimentally) – whichever happens first. Finally, it tears down the *old* interface completely and the system moves on to the new interface.

The users of MultiNets have to configure the WiFi network by providing the authentication information only once. After that, MultiNets switches the interfaces dynamically without requiring any manual intervention. The proposed switching solution in MultiNets is fully client based – it does not require additional support from the access points or gateways. Furthermore, MultiNets does not require any modification to the network protocols. It only reads the transport information and adds or removes routing table entries to perform a switch. This is why, existing applications run transparently on MultiNets without any change. There is a possibility that during the Step 4 of the switching, a very long TCP session may get interrupted due to timeout. We empirically derive that, a timeout value of 30 seconds is optimum for making this interruption a rare phenomena.

Due to page limitations, we present the details of this experiment in our technical report in [19]. We notice that the proposed switching technique may also be applicable to other applications, e.g., Alperovich and Noble [20] propose a similar technique to switch among homogeneous WiFi access points, which is quite different from MultiNets.

3) Switching API: MultiNets uses the API shown in Figure 5 to switch to a new interface. The method `switchInterface()` takes the name of the interface as an argument and returns either success or failure. Upon failure, it throws an exception explaining the reason of failure.

```
1  SwitchingManager mgr = new SwitchingManager(getSystemService("SM"));
2  try {
3      if (mgr.switchInterface(mgr.MOBILE) == true) {
4          // Success. New sessions start over 3G now
5      }
6  } catch(SwitchingException ex) { ex.printStackTrace(); }
```

Figure 5.   Using switchInterface() method.

III. DESIGN AND IMPLEMENTATION

The design of MultiNets is modular, consisting of three principal components – *Switching Engine*, *Monitoring Engine*, and *Selection Policy* as shown in Figure 6. These components isolate the mechanism, policy and monitoring tasks of the system, and allow extending their capabilities without requiring any changes to the architecture.

A. The Switching Engine

The Switching Engine performs the switching between cellular and WiFi. It maintains an internal state machine to keep track of the connectivity status. It also has a Switching Utility module that performs some low level tasks related to switching. The Switching Core coordinates these two.

1) The State Machine: Figure 7 shows the state diagram together with all the states and transitions. The system remains at *NoConnectivity* state (S_0) when neither cellular nor WiFi is available, and keeps seeking for a network to connect to. The states *ConnectedToCellular* (S_1) and *ConnectedToWiFi* (S_3) are similar. At these states, the device uses only one wireless interface and periodically checks with the Selection Policy (see Section III-C) to see if a switch is needed. The states *SwitchingToWiFi* (S_2) and *SwitchingToCellular* (S_4) are the transition states. Both of the interfaces are active during these states, but only the new connections start over the new interface while existing sessions still remain in the previous interface. The engine stays at these states as long as the old interface has active TCP sessions or until a timeout. Under normal circumstances, the system moves around within the states $\{S_1, S_2, S_3, S_4\}$ circularly. To cope with any loss of connectivity, the system makes some transitions shown in dotted arrows. A loss of WiFi connectivity at S_2 takes the system to S_3, but it immediately starts switching back to cellular upon detecting such a disconnection.
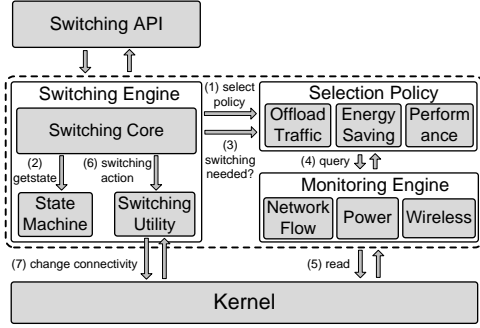
Figure 6. MultiNets Architecture.

Figure 7. State Diagram.

Figure 8. Layered Implementation.

2) The Switching Utility: The Switching Utility provides the utility methods to perform the switching. It includes the following tasks – counting the ongoing TCP sessions over a specific network interface, updating the routing table to keep the existing TCP sessions over a specific interface, adding and deleting default routes of the network interfaces, and connecting, re-connecting or tearing down interfaces. These methods are called by the core switching module to perform a switch.

B. The Monitoring Engine

The Monitoring Engine is responsible for monitoring all the necessary phenomena pertaining to switching. It contains several different monitors, each of which observes one or more system variables, and holds the latest values of those variables. We have implemented 4 monitors – *(i) Data Monitor:* Monitors the amount of transmitted and received data over WiFi and cellular interfaces in bytes and packets since the interface is turned on, *(ii) Wireless Monitor:* Monitors the connectivity status, signal strengths, and information of access points, *(iii) Network Flow Monitor:* Monitors the number and state of all TCP and UDP sessions, routing information from the routing table, and *(iv) Power Monitor:* Monitors the state of the battery and its voltage, current, and capacity. All these monitors are singleton and are created in an on-demand basis. They have a common interface to answer to all the queries. The query and its response form a $< key, value >$ pair. The Selection Policy component (see Section III-C) issues these queries. The modular design of the monitors and a common interface to talk to them allow us to add new monitors into the system and to extend the capability of the existing monitors easily.

C. The Selection Policy

The Selection Policy defines the policy for interface switching. By separating the policies from the rest of the system, we are able to add new policies or modify the existing ones with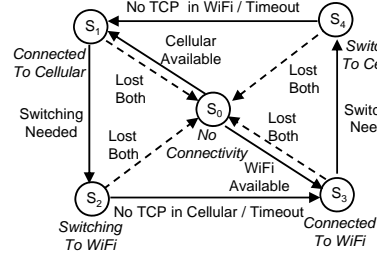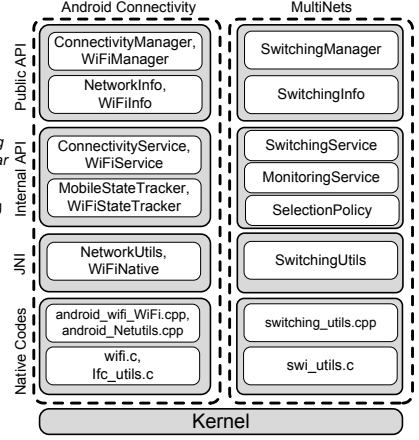out requiring any change to the other parts of the system. For example, one of our policies is based on the fact that WiFi is much faster than the cellular network. But in future, this situation may change and cellular data connectivity may outperform WiFi and thereby requiring a change in current policy or adding a new policy that leverages that. Currently, we have developed and implemented three policies which are described next. Only one of these policies is active at a time. The user of MultiNets determines which policy is to be used.

1) Energy Saving Policy: The aim of this policy is to minimize the power consumption. We describe an optimum energy saving algorithm in [19], which requires the knowledge of future data traffic. For a realistic setup, we propose a switching heuristic, which is inspired by our energy measurements. According to this policy, the mobile device connects to the cellular network when it is idle, and starts to count the number of bytes sent over the cellular network after the user launches an application. As soon as the total amount of data over the cellular network exceeds a threshold τ, the device decides to switch to WiFi. The mobile device switches back to the cellular network once the WiFi network is idle for ζ seconds. We empirically derive the best τ and ζ values in Section IV-C1. This policy levarages one fact that the idle power of WiFi is much higher than that of cellular. Techniques such as [9], [10] may save some part of the energy that is consumed for scanning WiFi APs– which accounts for about 40% of the idle energy. But even after applying such techniques, WiFi's idle power remains more than 50 times higher. Hence, switching interfaces dynamically is a better option to save energy.

2) Offload Policy: The aim of this policy is to offload cellular data traffic to any available WiFi network. According to this policy, whenever WiFi is available, we switch to WiFi. We only switch back to the cellular network when WiFi's signal strength is dropping below a threshold η dBm. The advantage of this policy is to reduce data traffic on cellular networks. But the downside is that, if the network is not being used, only to keep the WiFi interface idle is more

Method	Description
getInfo	Returns the status, state and current policy.
activateEngine	Activates or deactivates the engine.
setPolicy	Sets the current Selection Policy.
switchInterface	Request to switch to a particular interface.
useInterface	Request to use a specific interface.

Table I
DESCRIPTION OF SWITCHINGMANAGER API.

expensive in terms of energy.

3) Performance Policy: The aim of this policy is to maximize the network throughput. It achieves this by switching to the interface with the highest bandwidth. Let, $B_W(s)$ and $B_C(s)$ be the bandwidth functions for WiFi and cellular networks respectively where s denotes the signal strength, which is read via Android system API. We empirically derive the bandwidth functions $B_W(s)$ and $B_C(s)$ in Section IV-C3 through extensive experiments. The performance policy compares the values of these two functions every δ seconds, and switches to the network interface with the higher bandwidth.

D. Layered Implementation

We have closely studied the software architecture of the data connectivity in Android. Like Android, the implementation of MultiNets is layered. Classes and methods of our system that are similar to those of Android are implemented at the same layer. Yet, our system is vertically distinguishable from Android as shown in Figure 8. At the bottom of the architecture, we have the unmodified Linux Kernel. Right above the Kernel, we have a layer of native C/C++ modules that perform the lower level tasks of file I/O to get all the information used by the Monitors and some socket I/O to add, remove or update routing table entries. We improve the implementation of Android's `ifc_util.c`, `route.c`, and `netstat.c` by adding these non-existing modules and put them into our own module `swi_utils.c`. But no changes are made into the network protocols. These modules are wrapped by JNI and are called from the Internal Classes layer. The Switching Service, Monitoring Service and Selection Policy are implemented as system services at the Internal Classes layer. These services are created during the device start-up and they run as long as the device is running. We have modified the Android's System Server to start these services when the device starts. All the changes are done by adding 209 lines of C/C++ code and 650 lines of Java code to Android (Eclair 2.1).

We provide API to configure and control the Switching Engine which is described in Table I. We use this API to extend Android's built in wireless control settings application so that the Switching Engine can be stopped, restarted and configured to run in different modes from the application layer. When the engine is stopped, this API can also be used by the application programmers to switch interfaces, send a specific flow using a specific interface, or to use multiple interfaces simultaneously. Two of the methods in the API are very useful from the application programmers' point of view. The first one is, the `switchInterface()` method, which allows the programmers to switch interface when needed. This is useful for those kinds of applications that need to send sensitive data (e.g. user credentials) over the cellular network, but for all other purposes prefer to be on WiFi. Another important method is, the `useInterface()` method. It is useful to send or receive data using a specific interface for a specific connection. Note that, it does not switch the interface, rather if the preferred network is available, it sends the data using that interface for the specified connection only. With this method, an application can use multiple interfaces simultaneously. Figure 9 shows an example usage of this method. Both of these methods are requests to the switching system. The requests may fail if the application does not have proper permissions or the Switching Engine is currently running.

```
1   SwitchingManager mgr = new SwitchingManager(getSystemService("SM"));
2   String ip = "12.71.54.184"; int port1 = 5050, port2 = 5051;
3   InetAddress a = InetAddress.getByName(ip);
4   try {
5       if (mgr.useInterface(a, mgr.MOBILE)){
6           Socket ms = new Socket(ip, port1);
7           // transfer sensitive data over secured cellular network.
8       }
9       if (mgr.useInterface(a, mgr.WIFI)){
10          Socket ws = new Socket(ip, port2);
11          // transfer less sensitive data over public WiFi.
12      }
13  } catch(SwitchingException ex){ ex.printStackTrace(); }
```

Figure 9.   Using useInterface() method.

IV. EVALUATION

We present three sets of experiments in this paper. First, we discuss the system overhead. Second, we describe a set of experiments, where we apply the three switching policies on the 3 month long collected data to demonstrate the benefits of switching. Third, we demonstrate the performance of MultiNets in a real world scenario.

A. Experimental Methodology

1) Hardware Setup: All experiments are performed on multiple Android Developer Phones 2 (ADP2) [1]. The mobile devices are running MultiNets which is developed on top of Android OS (Eclair 2.1-update 1). The devices are 3G-enabled T-Mobile phones that use 3G, EDGE, GPRS and WiFi 802.11 b/g connectivity and are equipped with an 528 MHz ARM processor, 512 MB flash memory, 192 MB RAM, and 1 GB microSD card.

2) Software Setup: We have developed a *data logger* that is capable of logging various important information of the running applications within the phone periodically and send it to a remote server. 13 volunteers from our research lab, including research scientists, graduate students, faculty, and staffs of age group 25 to 35, were equipped with these

| Benchmark	Description
Linpack	Solves dense NxN system of linear equations.
Fps2d	Measures 2D graphics frames per second.
CMark	Measurements performance of java programs involving prime generation, loop, logic, method, and floats.
Graphics	Draws opacity and transparent bitmaps.
Cpu	CPU performance of MWIPS, MFLOPS, and VAX MIPS (SP and DP).
Mem	Memory copy operation.
File	File create, write, read, and delete.

Table II

DESCRIPTION OF THE BENCHMARKS.
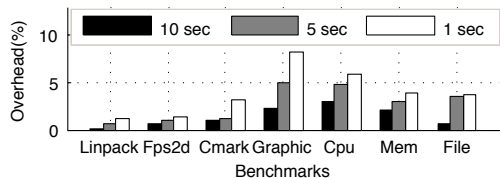
Figure 10. A monitring interval of 5 sec or more keeps the overhead below 5%.

phones with our data logger and they carried around the phones to wherever they wanted and used them for both voice and data connectivity for 90 days. The information that we collect from these logs include the names and types of the applications, the frequency and the duration of their usage, and the data usage information for each wireless interface for each user. For each of these applications, we have the total number of bytes and packets transmitted and received over cellular and WiFi. We modified the Linux's `netstat` tool for Android to get the information about all the TCP and UDP sessions, which include IP addresses, ports, start time, and durations.

We then implement a *traffic generator* to reproduce the data sessions. The traffic generator replays the exact same sessions as that are in the log except for that they are now using different server IPs which are situated in our lab instead of the original ones. We load the information about all the sessions into the traffic generator running on the phone and start a process that replays those sessions.

B. System Overhead

The Switching Engine starts several background system services at the device startup. Running such system services may add additional overhead to the system. The goal of this experiment is to derive a minimal sleeping interval for the monitoring services so that their overhead is reasonable. We run a set of benchmarks on the device, with and without the Switching Engine and compare the two scores. None of these benchmarks use any data connectivity and, hence, no switching happens during this experiment. The overhead is due to the engine's continuously monitoring and checking for an opportunity to switch only. We use seven sets of benchmarks that are available in Android Market that has been downloaded $10,000$ to $50,000$ times. Table II describes the benchmarks.
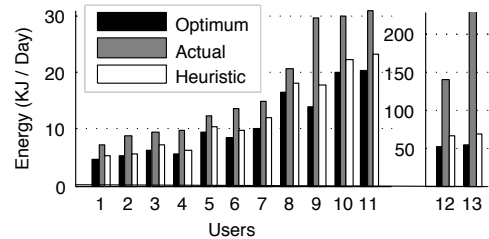

Figure 11. Energy saving heuristic cuts down the average daily energy usage by 27.4% and is close to optimum.

Figure 10 shows the benchmark scores of the device for running the Switching Engine at 10, 5, and 1 seconds sleeping intervals. The scores are normalized to the scores achieved by a phone running original Android. We see that the more the sleeping interval, the smaller the overhead and the closer the score is to that of without running the engine. But if this interval is large, the responsiveness of the engine becomes lazy. We therefore run the engine in 5 seconds interval which keeps the overhead below 5% and at the same time the responsiveness is also good. Note that, this overhead is due to the polling style implementation of the monitoring engine and is not an inherent problem of the switching technique itself. A more efficient implementation of the engine is left as our future work.

C. Trace Driven Experiments

1) Energy Efficiency: Using the energy model in [19], we estimate the average daily energy consumption for each user in our data traces. We have considered data transfer over WiFi and 3G, and also considered the idle power. We then compute the optimum energy consumption of each user assuming they switched optimally. We use dynamic programming to get this optimum value, which is described in [19]. While the algorithm achieves the optimum energy consumption, it assumes that the future data usage is known, which is not realistic. Therefore in MultiNets, we use a simple heuristic to switch interfaces. As data communication in WiFi is cheaper, for switching from 3G to WiFi, we use a data threshold of τ KB. If the phone crosses this limit, we switch to WiFi. On the other hand, since idle power of WiFi is much higher than 3G, we switch the phone back to 3G when data activity is absent over WiFi for the last ζ seconds. We systematically tried various τ and ζ values using the data traces, and found that $\tau = 1$ KB and $\zeta = 60$ seconds minimizes the deviation from the optimum energy saving. Therefore, we use these two values throughout the paper if not otherwise specified.

Figure 11 shows the average daily energy consumptions of all the users for three strategies: optimum, actual and the heuristic that we use in MultiNets. This figure shows that switching optimally saves on average 24.17 KJ energy per user per day, which is as high as $89 - 179$ KJ for some users (e.g., 12, 13). We also see that our simple heuristic achieves
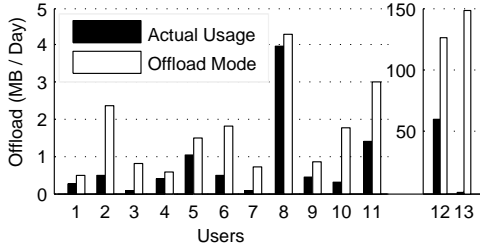
Figure 12. An average of 22.45 MB more data per day per user is offloadable using dynamic switching.

Figure 13. The achievable throughput is 7 times higher with switching.

near-optimal energy consumption with an average deviation of only 13.8%, and we are able to cut down the daily energy usage by 27.4% (21.14 KJ) on average.

2) Offloading Traffic: In order to estimate how much data traffic we are able to offload from 3G to WiFi network with MultiNets, we analyze the data traces that we have collected. For each user, we compute the average daily WiFi usage and compare it to the amount of data that is possible to offload if MultiNets was used. Our offloading strategy is to switch all 3G traffic to WiFi whenever we find a connectible access point. We consider an access point connectible if and only if its signal strength s is above $\eta = -90$ dBm and has been used by the user in the past. The threshold -90 dBm is derived from real experiments: when the signal strength is below it, the WiFi is not usable. Figure 12 shows this comparison for each of the users. We see that, for some users (e.g., 3, 7) we are able to offload $11 - 14$ times more data, and for some users who does not tend to use available WiFi at all (e.g., 13) this difference is about 150 MB per day. Considering all users, with switching, we are able to offload on average 22.45 MB of data per day per user which is 79.82% of the average daily usage (28.13 MB).

3) Performance: Performance of web applications get a significant boost by switching to the interface with the higher bandwidth. In our data trace, we have recorded the signal strengths of both the cellular and all available WiFi networks at 30 seconds intervals. We conduct extensive measurements using `iperf` tool to find the correlation between signal strength s and bandwidth B. We run `iperf` server on our server, and `iperf` client on Android phones, and `iperf` packets traverse through the Internet. We have taken measurements in both indoor and outdoor environments and report the average of 10 measurements at varying signal strengths in Table III. We define the bandwidth function $B_W(s)$ (for WiFi) and $B_C(s)$ (for cellular) using linear interpolations on measurement samples in this table.

Using the bandwidth functions, we calculate the average daily throughput of each user for his actual usage, and we also do the same if MultiNets was used. Figure 13 shows that, with MultiNets, it is possible to achieve an average throughout of 2.58 MBits/sec, which is 7 times more than the actual usages. For some users (e.g., 1, 2, 4, 13) this gain
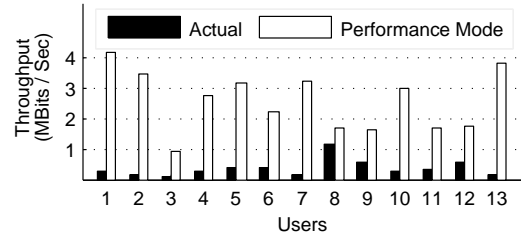
is about $14 - 24$ times. These are the users who tend to remain in the 3G network even if WiFi is available for them to connect.

Note that, based on our measurement results, even if we take decisions to switch based on the signal strengths, state-of-the-art 3G network being always slower than WiFi, the policy selects WiFi almost as if it were in Offload mode. This is, however, not always true, e.g., with the rapid advancement of cellular data network technology, this gap is diminishing. Our measurements with recent High Speed Packet Access Network (HSPA+) in Table III shows that this network is about eight times faster than 3G. We believe, in near future, cellular networks will have a comparable bandwidth to WiFi, and the performance mode of MultiNets will have a higher impact at that moment. Finally, measurement studies report that WiFi throughput may be lower than 3G throughput under certain practical circumstances [3].

WiFi		HSPA+		3G	
Signal (dBm)	Bandwidth (Mbps)	Signal (dBm)	Bandwidth (Kbps)	Signal (dBm)	Bandwidth (Kbps)
≤ -50	8.58	-65	929	-63	138
$(-50, -60]$	7.06	-73	858	-89	115
$(-60, -70]$	6.16	-89	746	-101	104
$(-70, -80]$	3.99	-97	509		
> -80	1.25				

Table III
BANDWIDTH OF WiFi, HSPA+, AND 3G AT DIFFERENT SIGNAL LEVELS MEASURED BY `IPERF`

D. Real Deployment

To quantify the performance of our system in a real-world scenario, we conduct actual experiments at Stanford University campus. We have chosen this campus since it has WiFi connectivity both inside and outside of the buildings and also has several areas where WiFi is either completely unavailable or has a very poor signal strength. High availability of WiFi is important for us since we want to demonstrate that our system is switching back to 3G to conserve energy even in presence of WiFi. On the other hand, loss and reconnection of WiFi connectivity is important to demonstrate that our system is capable of switching smoothly. We take 4 ADP2 phones with us. Two of these have our system installed and the other two run Android (Eclair 2.1). All 4 phones are fully charged and their screen brightnesses are set to the lowest level. For a fair comparison, we use our traffic generator to replay the
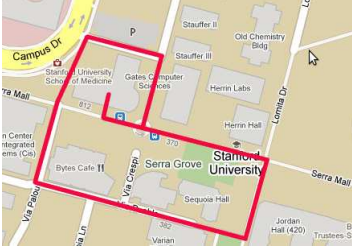
Figure 14. We encounter 37 WiFi APs, average signal strengths of -68.46 dBm (inside) and -82.34 dBm (outside), and 28 WiFi disconnections during the tour.

Figure 15. Energy saving mode saves about $28.4\% - 33.75\%$ energy as compared to Android.

same data traffic in all of them. The traffic generator runs in the phone and sends and receives data over the Internet to and from our server which is situated in our lab at 4 miles distant from the campus. The phones replay the traffic patterns of the most popular 6 applications from our data traces having sessions of varying numbers, durations, delays and concurrencies. Once started, the phones run each of these applications for 10 minutes followed by a 10 minutes break, repeatedly one after the other. We log the transmitted and received bytes, signal strengths, MAC addresses of WiFi APs, battery current, voltage, and capacity into the file system of the phone every 2 seconds for later analyses.

1) Energy Efficiency: In this experiment, we configure one of our phones into the energy saving mode. We take another two phones that run Android– one with WiFi enabled, and the other staying over 3G only. We start the traffic generator in all 3 phones and begin our 168 minutes long campus tour starting from the Computer Science building. We move around all 5 floors of the building for an hour, then take an hour long round trip tour within the campus, and finally get back to the building to spend the rest of the tour as shown in Figure 14. During this tour, we encounter 37 different WiFi APs, an average signal strength of -68.46 dBm inside the building and -82.34 dBm outside the building, 28 disconnections from WiFi to 3G, and a total of 49 switchings by our system. Using the instantaneous values of current and voltage obtained from the log, we compute the energy consumption of each of these phones and plot the cumulative energy consumption in Figure 15. Despite the fact that the battery voltages and currents read from the Android system are not in high precision, we still see a clear difference of the energy consumption among these 3 phones. We see that for the same data traffic, our system achieves about $28.4\% - 33.75\%$ energy savings as compared to state of the art Android systems.

2) Offloading and Throughput: In this experiment, we compare the offloading and throughput of our system with those of the state-of-the-art Android. In MultiNets, we set the switching timeout to be 30 seconds. Recall that, cellular network being much slower than WiFi, the outcomes of Offload and Performance modes are the same, although their decision mechanisms are completely different. Therefore, we
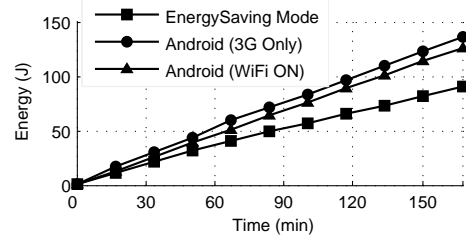
present them together in Table IV. This table also gives the achievable lower and upper bounds of Android on offloading and throughput. The lower bound is derived by disabling WiFi interface (3G Only), and the upper bound is achieved by always switching to WiFi whenever it is available (WiFi ON). This table shows that: MultiNets (i) leads to three times higher throughput than the Android lower bound, (ii) achieves near-optimal offloading and throughput, and (iii) experiences *zero* TCP disconnections throughout the experiments, while Android upper bound results in *eight* TCP disconnections.

System	Offload (MB)	Throughput (kbps)	Disconnections (Count)
MultiNets	45.41	116.20	0
Android (3G Only)	0	39.29	0
Android (WiFi ON)	44.54	116.26	8

Table IV
FOR NEAR-OPTIMAL OFFLOADING AND THROUGHPUT, MULTINETS EXPERIENCES NO TCP DISCONNECTIONS THROUGHOUT THE EXPERIMENTS.

3) Energy Efficiency vs. Offload Trade-off: It is interesting to see the trade-offs between the energy savings and offloading. Table V shows that, MultiNets in energy-saving mode consumes about 55.85% less energy than offload mode, but sacrifices about 14.25% of offloading capability. The reason behind is that, energy saving mode keeps the phone in 3G while it is idle. When data transmission starts, it keeps the phone in 3G mode for a while before completely switching to WiFi, and hence the overall WiFi offloading is slightly lower in this case. This experiment illustrates that, users of MultiNets achieve different objectives by putting the system in different modes.

Mode	Energy Consumption (J)	Offload (MB)
Energy Saving	90.36	38.94
Offload	204.65	45.41

Table V
ENERGY SAVING MODE SAVES 55.85% MORE ENERGY, BUT SACRIFICES 14.25% OFFLOADING.

V. RELATED WORKS

Switching among multiple network interfaces of a mobile device has been considered in the literature. New protocols

in various layers have been designed to support switching among access networks. Wang et al. [7] propose a rate control algorithm for multiple access networks, which needs to be integrated into application-layer protocols. Kim and Copeland [18], and Wu et al. [22] propose TCP variants that result in better performance by switching among access networks. Mobile IP [8] uses foreign/home agents to forward network traffic from/to a smartphone that move among access networks but incurs additional network latency. Mobile IP v6 [13] uses optimized routes for lower network latency, but it still relies on deploying foreign/home agents for mobility management. In contrast to MultiNets, widely deploying TCP variants and mobile IP agents in the Internet incurs tremendous costs and burden, and may take years to be done.

Gateways between smartphones and the Internet can be used for accessing multiple network interfaces. Balasubramanian et al. [3] design a system to reduce the network traffic over cellular networks by transmitting delay tolerant data over WiFi and real-time data over cellular networks. Sharma et al. [15] propose a system that uses gateways to aggregate network resources from multiple access networks among several collaborative smartphones. Armstrong et al. [21] use a proxy to notify the phone via sms about content updates and suggests interface to use. Unlike MultiNets, gateway solutions require deploying expensive gateways, incurs additional network latency, and may need users to configure the proxy settings in applications.

A master/slave solution [11] chooses an always-connected access network as the master network, and uses other access networks as slave networks for opportunistic routing. Different from MultiNets, deploying master/slave solutions requires complete control over multiple access networks, which is difficult due to business reasons. Higgins et al. [6] propose intentional networking where applications provide hints to the system and system chooses the best interface opportunistically. In contrast, MultiNets is completely transparent to the existing aplications. Rahmati et al. [5] demonstrate the feasibility of TCP flow migration on iPhones, but they do not address or rigorously quantify the policies and different benefits of switching interfaces.

VI. CONCLUSION

In this paper, we consider the problem of real-time switching between multiple network interfaces on mobile devices. We first conduct a three month long empirical study to understand the TCP characteristics on Android devices. Based on this study, we design a client based solution to the switching problem. We then present the MultiNets system that uses this technique to dynamically switch between WiFi and cellular networks based on three policies: energy efficiency, offloading data traffic, and higher throughput. Our evaluation results show that our system outperforms the state-of-the-art Android either by saving up to 33.75%

energy, or achieving near-optimal offloading, or achieving near-optimal throughput while substantially reducing TCP interruptions due to switching.

REFERENCES

[1] Android Developer Phone 2 (ADP2). http://en.wikipedia.org/wiki/Android_Dev_Phone.

[2] Cisco Visual Networking Index: Forecast and Methodology 2009–2014. http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-481360.pdf.

[3] A. Balasubramanian, R. Mahajan, A. Venkataramani. Augmenting Mobile 3G using WiFi. In *Proc. of MobiSys*, San Francisco, USA, March 2010.

[4] A. Rahmati, C. Shepard, A. Nicoara, L. Zhong, J. Singh. Mobile TCP Usage Characteristics and the Feasibility of Network Migration without Infrastructure Support. In *Proc. of MobiCom (Poster Session)*, Chicago, USA, September 2010.

[5] A. Rahmati, C. Shepard, A. Nicoara, L. Zhong, J. Singh. Seamless flow migration on smartphones without network support. *Technical Report 2010-1214, Rice University*, December 2010.

[6] B. Higgins, A. Reda, T. Alperovich, J. Flinn, T. Giuli, B. Noble, D. Watson. Intentional networking: opportunistic exploitation of mobile network diversity. In *Proc. of MobiCom*, Chicago, USA, September 2010.

[7] B. Wang, W. Wei, J Kurose, D. Towsley, K. Pattipati, Z. Guo, Z. Peng. Application-Layer Multipath Data Transfer via TCP: Schemes and Performance Tradeoffs. *Elsevier Performance Evaluation*, 64(9-12):965–977, October 2007.

[8] C. Perkins. Mobile IP. *IEEE Wireless Communications Magazine*, 35(5):84–99, May 1997.

[9] G. Ananthanarayanan, I. Stoica. Blue-Fi: enhancing Wi-Fi performance using bluetooth signals. In *Proc. of MobiSys*, Wroclaw, Poland, June 2009.

[10] K. Kim, A. Min, D. Gupta, P. Mohapatra, J. Singh. Improving Energy Efficiency of Wi-Fi Sensing on Smartphones. In *Proc. of INFOCOM*, Shanghai, China, April 2011.

[11] K. Pahlavan, P. Krishnamurthy, A. Hatami, M. Ylianttila, J. Makela, R. Pichna, J. Vallstron. Handoff in Hybrid Mobile Data Networks. *IEEE Personal Communications*, 7(2):34–47, April 2000.

[12] N. Thompson, G. He, H. Luo. Flow Scheduling for End-Host Multihoming. In *Proc. of INFOCOM*, Barcelona, Spain, April 2006.

[13] P. Nikander, J. Arkko, T. Aura, G. Montenegro. Mobile IP Version 6 (MIPv6) Route Optimization Security Design. In *Proc. of IEEE Vehicular Technology Conference (VTC'03-Fall)*, Orlando, USA, October 2003.

[14] P. Rodriguez, R. Chakravorty, J. Chesterfield, I. Pratt, S. Banerjee. MAR: a Commuter Router Infrastructure for the Mobile Internet. In *Proc. of MobiSys*, Boston, USA, June 2004.

[15] P. Sharma, S. Lee, J. Brassil, K. Shin. Handheld Routers: Intelligent Bandwidth Aggregation for Mobile Collaborative Communities. In *Proc. of BroadNets*, San Jose, USA, October 2004.

[16] R. Chalmers, K. Almeroth. A Mobility Gateway for Small Device Networks. In *Proc. of IEEE International Conference on Pervasive Computing and Communications (PerCom'04)*, Orlando, USA, March 2004.

[17] S. Kandula, K. Lin, T. Badirkhanli, D. Katabi. FatVAP: Aggregating AP Backhaul Capacity to Maximize Throughput. In *Proc. of NSDI*, San Francisco, USA, April 2008.

[18] S. Kim, J. Copeland. TCP for Seamless Vertical Handoff in Hybrid Mobile Data Networks. In *Proc. of IEEE Global Telecommunications Conference (GLOBECOM'03)*, San Francisco, USA, December 2003.

[19] S. Nirjon, A. Nicoara, C. Hsu, J. Singh, J. Stankovic. MultiNets: Policy Oriented Real-Time Switching of Wireless Interfaces on Mobile Devices. *Technical Report CS-2011-08, University of Virginia*, October 2011. http://goo.gl/DDAuj.

[20] T. Alperovich, B. Noble. The Case for Elastic Access. In *Proc. of the MobiArch*, Chicago, USA, September 2010.

[21] T. Armstrong, O. Trescases, C. Amza, E. Lara. Efficient and transparent dynamic content updates for mobile clients. In *Proc. of MobiSys*, Uppsala, Sweden, June 2006.

[22] X. Wu, M. Chan, A. Ananda. TCP HandOff: a Practical TCP Enhancement for Heterogeneous Mobile Environments. In *Proc. of IEEE International Conference on Communications (ICC'07)*, Glasgow, UK, June 2007.