# Mirroring Smartphones For Good: A Feasibility Study [⋆]

Bo Zhao[1], Zhi Xu[1], Caixia Chi[2], Sencun Zhu[1], and Guohong Cao[1]
[1] Department of Computer Science and Engineering
The Pennsylvania State University
[2] Bell Laboratories
Alcatel-Lucent Technologies
bzhao@cse.psu.edu, zux103@cse.psu.edu, chic@alcatel-lucent.com,
szhu@cse.psu.edu, gcao@cse.psu.edu

**Abstract.** More and more applications and functionalities have been introduced to smartphones, but smartphones have limited resources on computation and battery. To enhance the capacity of smartphones, an interesting idea is to use Cloud Computing and virtualization techniques to shift the workload from smartphones to a computational infrastructure. In this paper, we propose a new framework which keeps a mirror for each smartphone on a computing infrastructure in the telecom network. With mirror, we can greatly reduce the workload and virtually expand the resources of smartphones. We show the feasibility of deploying such a framework in telecom networks by protocol design, synchronization study and scalability test. To show the benefit, we introduce two applications where both the computational workload on smartphones and network traffic in telecom networks can be significantly reduced by our techniques.

## 1 Introduction

Smartphones have become more and more intelligent and powerful. Many complex applications, which used to be only on PCs, have been developed and running on smartphones. These applications expand the functionalities of smartphones and make it more convenient for users to get connected. However, they also greatly increase the workload on smartphones and introduce a lot of data transmissions between smartphones and telecom networks. The heavy workload and traffic affect both smartphone users and Telecommunication Service Provider (TSP). For users, heavy workload and traffic drain smartphone battery quickly. As we tested on Android Dev Phone 1, scanning a folder with 200MB files would take about 40 minutes and cause 10% battery; and uploading a 20MB file would cost more than 10% battery.

Recently some research [1] [2] [3] [4] has been done leveraging cloud techniques to help smartphones. An augmented execution model is provided to off-load application execution from smartphone to its clones in cloud [2]. In the in-cloud antivirus system [3], smartphones can send suspicious files to the antivirus service in cloud for scanning so as to avoid performing resource consuming scanning applications locally at phone.In addition, Zhang *et al.* [4] suggested to build elastic applications which can be partitioned into function independent components. Those components can be executed in cloud instead of running on smartphones. Cloudlets [1] are designed for applications that require real-time interactive response, such as augmented applications. In their framework, the smartphone delivers a virtual machine (VM) overlay to the cloudlet infrastructure. Via this overlay, smartphones deliver part of execution to launch VMs,

[⋆] This work was supported in part by NSF CAREER 0643906

which provide various services and return results to smartphones. Smartphones are connected with cloudlets via high speed wireless network connections.
*Contributions:* We propose a new framework which keeps a mirror for each smartphone on a computing infrastructure in the telecom network. With the mirror we can shift some computational workload from a smartphone to its mirror. Since a mirror is synchronized with its corresponding smartphone, some operations, such as file sharing and virus scanning, can be performed on the mirror directly. In this way, we essentially reduce the workload and virtually expand the resource of a smartphone.

The design of our framework leverages the emerging Cloud Computing and virtualization techniques. On the smartphone side, a client side synchronization module is deployed to collect smartphone user input data and transmit them to the mirror server, a powerful application server. With Cloud Computing techniques, the mirror server is capable of hosting hundreds of mirrors and each mirror is implemented as a VM. To keep loose synchronization between mirror and smartphone, the mirror server replays inputs to smartphone on its mirror with exactly the same order (Section 3).

To illustrate the feasibility and compatibility of deploying the proposed framework in 3G networks, we present a network architecture based on UMTS, a 3G network architecture currently used by T-mobile (Section 3.4). We show that only minor modifications are needed on the current UMTS 3G network. Moreover, we use multimedia service (MMS) and web browsing service as examples to illustrate our protocol design. To show the benefits of our framework, we present two applications (Section 4). One is the data caching application, which saves both the power consumption of the smartphones and network traffic of 3G networks by caching smartphones' data in a mirror server. Another application is virus scanning in a smartphone.

We have built a prototype of the mirror server using Dell PowerEdge 1900 (Section 5). By measuring its workload with different number of mirrors, we show that the scalability of our framework is acceptable. We also tested the power consumption of running synchronization client side module on Android Dev Phone 1. According to the result, the performance overhead in the smartphone is very small.
*Scope of This Work:* Given the complexity of our whole system, it is infeasible to describe every detail in this paper. From the system point of view, issues like how to implement and deploy the client-side software and how to design mirror servers efficiently are of great importance, but they require independent study. We believe with the advances of technology in Cloud Computing and VM techniques, such issues could be solved in the near future. Instead, in this stage of our work, we focus on describing the design from the networking point of view and demonstrating the feasibility of this framework by analyzing and evaluating its scalability and its benefit/cost. Note that due to the hardware diversities of existing commodity smartphones, building mirrors supporting hardware-specific features, such as accelerometer sensor, could be very difficult and costly. In this paper, we focus on common features, such as touch screen, key pad, and 3G connection. In other words, currently we do not provide mirroring services to applications whose functionalities rely on sensor inputs.

## 2 Related Works

The idea of leveraging cloud computing techniques to enhance smartphones has been discussed in recent years. Our framework distinguishes itself from two aspects. First of all, in our architecture, the cloud is located in TSP. A smartphone is connected with its mirror via 3G telecom network. Secondly, our mirror is for more general purposes, although it can certainly be used for specific applications. Smartphones and their mirrors are kept synchronized via a loose synchronization mechanism.

Satyanarayanan *et al.* proposed a cloudlet infrastructure [1], where a smartphone is connected with a cloudlet via high speed wireless LAN. In the proposed infrastructure, the mobile device delivers a small VM overlay to the cloudlet infrastructure. Smartphones provide input to launch VMs in the cloudlet. Launch VMs provide services based on the input and return the result. The focus of this work is to provide real-time interactive responses between smartphones and the launch VM, which are essential to such applications as augmented reality applications. It tries to connect service providers in the cloudlet with smartphones while making this process invisible and seamless to user. Different from this work, first in our proposed architecture, smartphones are connected with cloud via Telecom networks. A server in our case has much wider coverage than a cloudlet. On the other hand, the data transmission rate of existing 3G networks cannot compete with that of wireless LANs. Secondly, different from launch VMs, mirrors are always synchronized. Files exist on a smartphone also exist on its mirror.

Oberheide *et al.* extended the CloudAV platform [5] to a mobile environment [3]. In the proposed model, smartphones send suspicious files to a remote in-cloud antivirus network service, which provides a scanning service. The purpose of [3] is to move the mobile antivirus functionality to an in-cloud antivirus network service, so as to save the resources on the smartphones. The author claims that it is worthwhile to get the antivirus scanning service by paying for the file transmission cost. Different from [3], first we propose a generic framework which is not limited to the antivirus service. Secondly, because a smartphone and its mirror are always synchronized, scanning can be performed on-the-fly and directly on the mirror, reducing the file transmission cost in [3].

Zhang *et al.* suggested an elastic application model [4]. In this model, one application is partitioned into components called weblets. By partitioning, some workload can be outsourced from smartphones to cloud elasticity services. In our framework, no application partitioning is required.

Chun and Maniatis proposed a new architecture [2] to augment the capabilities of smartphones by moving, in whole or in part, the execution of resource expensive applications to smartphone clones at an external computational infrastructure such as a nearby PC. Their work focus on how to move parts of execution of applications to the augmented clones that are more powerful than smartphones. To get the clone of a smartphone, they proposed to perform synchronization via whole-system replication, which is expensive because of the power consumption in synchronization. Different from [2], we discuss and evaluate the feasibility of implementing a mirror for a smartphone in a real telecom network. We focus on saving power on smartphones and minimizing the network traffic between smartphones and telecom network.

## 3 System Design

Our system provides an architecture for shifting computing from smartphones to their mirrors in the telecom network. Fig. 1 illustrates a high-level overview of our system. On the smartphone side, a client-side synchronization module, called *Syn-Client*, is deployed within the smartphone operating system (OS) to collect smartphone input data,

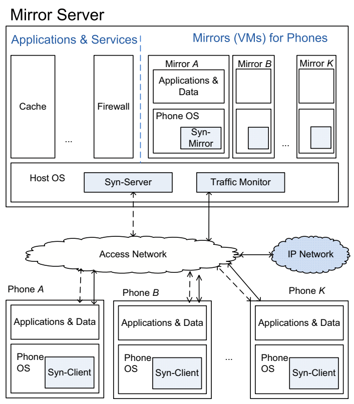


**Fig. 1.** The System Design

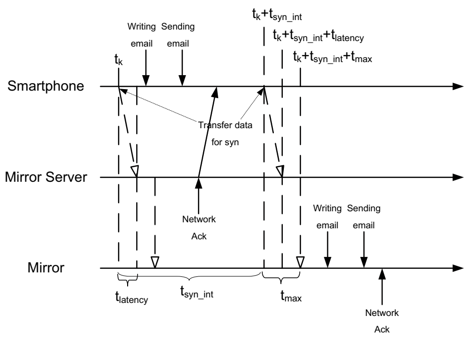


**Fig. 2.** Timelines for mirror synchronization

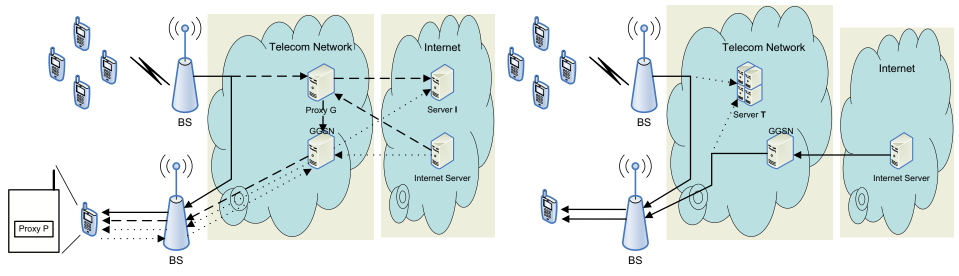
including user keyboard inputs, and transmit them to the mirror server for synchronization. The Syn-Client module is designed according to the specification provided by TSP and manufacturer. On the mirror server side, the mirror server is a powerful application server maintaining a set of VMs. Each VM is a mirror to one smartphone. To keep mirrors and smartphones synchronized, the server side synchronization module, called *Syn-Server*, updates mirrors based on the data provided by *Syn-Clients* and network traffic between smartphones and IP network, which are collected by the *Traffic Monitor* module. Next we will focus on location of mirror server, mirror design and synchronization. The detailed designs and implementations of the other modules are out of the scope of this paper.

### 3.1 Location of Mirror Server

In the proposed architecture, the mirror server monitors all network traffic of smartphones. In our case, we consider traffic from both ISPs (e.g., Internet websites) and the TSP (e.g., AT&T). Bluetooth or WiFi connections are not considered in our discussion. We only consider the case when a smartphone is connected through a 3G network.

To deploy the mirror server, there are two options. One is to deploy it in ISPs, i.e. the server I shown in Fig. 3(a). The dotted line and dashed line represent the message flow with proxy P and proxy G. And the other option is to deploy it in the TSP, i.e. the server T shown in Fig. 3(b). In this case, a copy of traffic from both TSP and ISP is forward to server T within telecom network. This forwarding requires modification to existing telecom network and we present details of modification in the next section.

In the systems proposed in [4], the Cloud assisting smartphones is deployed in ISPs and connected with smartphones via IP networks. However, in our architecture, if we want to deploy the mirror server in the ISPs, an additional proxy will be needed to collect incoming traffic of smartphones and forward it to the mirror server. As shown in Fig. 3(a), such a proxy can be deployed either besides GGSN in TSP or the smartphone, represented as proxy G and proxy P respectively. However, placing proxy in GGSN will also need to change the telecom network in order to forward messages, and then similar to placing mirror server in the TSP. Considering these concerns of placing mirror server in ISPs, we deploy the mirror server in the TSP.



(a) In the ISPs (b) In the TSP

**Fig. 3.** Placing the mirror server

## 3.2 Mirror Design

Compared with PC, creating mirrors for smartphones has many advantages. First of all, all smartphones of one model share the same hardware specifications, default factory settings, OS, and a lot of application software. Moreover, under the restriction of usage agreement with TSP, users are not allowed to modify the hardware or OS of their smartphones. Therefore, for one model, the mirror server only needs to keep one template of its factory default state.

On the other hand, we also notice that advanced smartphones have additional ways of system input, such as accelerometer sensors, which do not exist on PCs. These input methods introduce much difficulty to synchronization between a smartphone and its mirror. Currently, our mirror only supports general user interfaces, which are provided by most existing commodity smartphones, such as touch screen, keypad, and 3G network connection.

To build a new mirror for a smartphone, the mirror server creates a VM with exactly the same hardware/software specifications using the factory default template. If the smartphone is in its initial state, i.e. the first-time use, the mirror server can quickly update the mirror by supplying user information to the mirror. If the smartphone is not in its initial state, the mirror server copies software specifications and user personal data from the storage of the smartphone to the mirror. During the copying process, the state of the smartphone will be frozen and no operation from user is allowed.

## 3.3 Synchronization Mechanisms

To keep smartphone and its mirror synchronized through the telecom network, the mirror needs to be updated when the state of smartphone changes. The choice of synchronization mechanisms depends on applications which the mirror server supports.

**Loose Synchronization** In this paper, we present one option for loose synchronization, which requires identical hardware specification, storage (e.g. SD card), OS, and installed application software. The loose synchronization is easier to achieve and is sufficient for applications like firewall, data caching applications, and virus scan applications. Indeed, because a smartphone rarely changes its hardware and OS, and user applications are not frequently changed either, in the first stage of our work, we mainly consider the issue of keeping storage consistency between a smartphone and its mirror.

An intuitive way for storage synchronization is data replication. That is, copying every new or updated data item to the mirror. This however is too expensive when the data activity is frequent in the phone. Instead, we propose a periodical synchronization mechanism by replaying smartphone inputs at its mirror periodically.

The first question for such a design is: why data storage is uniquely determined by inputs? We noticed that, the change of smartphone state is always triggered by user inputs (e.g. keyboard typing) or network inputs (e.g. arrival of data packets), and most applications are deterministic. That is, given a list of inputs, the application will generate deterministic outputs. In this case, if we apply the same list of inputs to a smartphone and its synchronized mirror, both will generate the same output and will be synchronized again after applying the inputs. Note that in our design, to save communication overhead, only the user inputs are forwarded to the mirror server. The mirror server caches the network inputs to a smartphone in the past synchronization period, so later it can directly feed in the cached data to the mirror. Also, the outputs from the mirror execution do not actually go out of the mirror server. We use the global clock of the telecom network to time stamping all kinds of inputs/outputs. For example, on the smartphone side, the user pushes a button to open an email client application, types an email, and sends the email by clicking an icon on the screen. On the mirror side, if we exactly replay the "pushing", "typing", and "clicking" actions in order, the mirror will send the same email and both will have the same archived email in the storage.

Based on this observation, we make an assumption that, by applying exactly the same inputs to a smartphone and its mirror in the same order, the storage of a phone and its mirror will be synchronized. This assumption may be untrue for certain applications, such as the application that generates a random number. However, it is suitable for most applications on current smartphones, such as email, IM, web browser, VoIP, MMS/SMS, and the present service. Therefore, we monitor all the inputs to the smartphone and later apply to the mirror in order.

The second question is: what are the pros and cons of periodical synchronization and the impact of the synchronization period on the system? Qualitatively speaking, the periodical synchronization improves the transfer efficiency and reduces much power consumption by sending the inputs in a batch. In normal cases, it does not cause problems to user applications, as long as the interaction between a mobile phone and the network need not involve mirror's cooperation. One counter example we can think of is related to our cache based file uploading application. In this application, when a user wants to share a file with a friend, to save power no file is actually sent out from the mobile; instead, the mirror will make the data transfer after it receives the send command through synchronization. Thus, no file transfer happens immediately.

**Synchronization Timelines** Next we explain the detailed operations and timelines in our periodical synchronization mechanism. Let $t_{latency}$ be the network transfer latency between the mirror server and the smartphone, and $t_{max}$ be the maximal network transfer latency. In 3G networks, $t_{max}$ is a constant value, 100 ms [6]. Let $t_{syn\_int}$ be the synchronization interval and the last synchronization time point be $t_k$. Thus, the data input to the smartphone between $t_k$ and $t_k + t_{syn\_int}$ are batched and transferred to its mirror at $t_k + t_{syn\_int}$. Due to the network transfer latency, it will arrive at the mirror server at $t_k + t_{syn\_int} + t_{latency}$. To make the replay interval even, the mirror server replays the network input data and keyboard input of the smartphone to the mirror at time $t_k + t_{syn\_int} + t_{max}$. As a result, the state of the mirror is $t_{syn\_int} + t_{max}$ later than that of the smartphone. Fig. 2 illustrates how this approach works with the email service by showing the time lines at the three parties: smartphone, mirror server and mirror. Note that in this figure the network acknowledgement is monitored and replayed by the mirror server to the mirror. Solid arrow lines stand for input data, and dash arrow lines stand for synchronization messages.

When a smartphone boots up, the initial time for the synchronization is set. At this time, on the smartphone side, the user can configure the value of $t_{syn\_int}$ for balancing the efficiency and reaction delay, while on the network side, the mirror server starts up this smartphone's mirror. When a smartphone is turned off, the mirror server stops this smartphone's mirror and saves its states.

### 3.4 Network Design

The mirror server stays in 3G networks as an application server, which supports such applications as security, storage and computation delegation services. It is easy to *configure* the nodes of 3G networks to route all incoming/outgoing messages (such as session control messages, data packets, and other 3G signaling protocols) of a smartphone to traverse the mirror server without any interaction with the smartphone. We do not need to modify any protocols or software in network nodes.

Taking UMTS as example, for a user who has subscribed to our mirror service, we will update his/her user profile in the Home Subscriber Server (HSS) node, as well as registering user policy and mirror service profile to the CSCF. HSS periodically updates the SGSN with the changes of user information; thus, SGSN becomes aware of the mirror service for this user. When SGSN receives a message from/to this user, it will either forward the message to CSCF, or forward it to the mirror server, subject to whether the message belongs to a service with session control.

Due to the limit of space, please refer to the technical report [7] for the details of deploying the mirror server in UMTS network. In this technical report, we present two sample message flows of typical telecom services to show that the mirror server works compatibly with UMTS network with only minor changes.

## 4 Applications

The proposed framework can support many applications. In this section, we identify two of them.

### 4.1 Data Caching Application

Recently, more and more people use their smartphones to share multimedia data with friends. Although data downloading/uploading speed between smartphone and BS has been significantly increased in recent years, receiving and sending bulk data still take time and consume a lot of power.

The current file sharing on smartphones has many weaknesses. For example, if the receiver runs out of battery or has very slow downloading speed in the current location (e.g. weak signal), the user may have to give up the file downloading. Also, if a user sends the same file to different people in a short time period, for each send, the smartphone has to do a separate file uploading, wasting lots of bandwidth and power.

Basing on the proposed framework, we present a data caching application. It is deployed on the mirror server which brings more flexibility to file downloading and significantly saves battery and time in file uploading.

**Bulk Data Downloading** For bulk data downloading, the data caching service provides a temporary storage service. When the user wants to download a file from the Internet, the caching service will first download the file to its temporary storage. Once the downloading is complete, the caching service sends a “file-cached” notification to the user via the synchronization message. This file will be kept for a period of time depending on the TSP policy. When the user decides to download the cached file, he can download it from the mirror server directly. Moreover, according to the synchronization mechanism, once the smartphone starts to download a file from the mirror server, its corresponding mirror will also download the file from the mirror server. Therefore, no extra storage synchronization is needed.

For the file sender, the caching service provides an illusion that the file transfer has been completed. For the file receiver, he is able to choose a time which is suitable for the actual downloading. For example, when he moves to a place with better downloading speed or has the access to battery charging.

**Bulk Data Uploading** For bulk file uploading, the data caching service utilizes the mirror of the smartphone on the server. The mirror has the same storage as its corresponding smartphone and can replay all actions performed on the smartphone. Thus, if a user wants to send a bulk file to multiple people, he does not need to upload the file multiple times through the wireless link. Instead, the data caching service can send the file directly from the mirror to multiple people and save the wireless bandwidth.

To achieve this, the caching service registers the “upload” event in the synchronization modules. When the Syn-Client module intercepts the uploading action on the smartphone, instead of uploading file immediately, it replaces the default uploading action by a “fake” action without actual uploading. When the next synchronization is performed, the mirror sends the same data. In normal cases, the data sent from a mirror will be dropped by the mirror server. In the file uploading service, the data sent from the mirror will be delivered to the receiver pretending that it is sent by the smartphone. For both the file sender and the receiver, the caching application provides an illusion that the file is sent from the smartphone. Compared with uploading from smartphone, performing synchronization costs much less battery power, especially when the file size is large.

Note that the caching service can be used for all file uploading and downloading activities on the smartphone, such as IM client and web browser, etc. To reduce the delay caused by the synchronization, the user should have the flexibility to activate/deactivate the data caching service. For example, a user may want to send small files from the smartphone immediately, but use the caching service for files larger than 500KB.

### 4.2 Antivirus Scanning Service

Recent research [8] [9] shows there is an incoming number of mobile phone malware such as worm and virus in 3G networks. Virus scan is one of the most common functionalities provided by almost all antivirus software. In virus scan, the scanner reads files, compares them with some virus signatures, and returns a result of “virus found” or not to the user. Obviously, the scan process is CPU and I/O intensive. In case of smartphone, performing such scans adds inconvenience to users and drains the battery quickly.

In the proposed framework, an antivirus scanner can be deployed as a service on the mirror server, and the scanner can access the file system on the mirrors. Further, a hook application is installed on the phone. When a user launches the hook application on the smartphone, it sends a request with parameters to the mirror server. Instead of scanning the real files on the smartphone, the scanner service performs scanning on the synchronized mirror. The result will be sent back to the smartphone. As the smartphone and its mirror are synchronized, scanning on the mirror generates the same result as scanning on the smartphone. The feasibility of building an antivirus service outside of protected VM has been proved in the VMwatcher [10] which can be deployed directly on our framework.

Shifting antivirus scans are threefold. First, it saves battery power on smartphones. Second, as the CPU and I/O intensive workloads are shifted to the mirror server, the scanning will not affect the common usages of smartphones. Finally, the scan speed on mirror will be much faster than that on the phone due to its limited hardware capabilities.

## 5 Evaluation

In this section, we first describe our prototype. Then, we evaluate the synchronization cost and the scalability of our framework with this prototype.

### 5.1 The Prototype

Because the UMTS 3G network is very complicated with multiple protocols and many different nodes, currently only big telecom equipment vendors, such as Nokia, Alcatel-Lucent and Ericsson have real prototypes. In order to evaluate the performance, scalability and feasibility of our framework, we setup a simplified prototype. In this prototype, the Android Dev Phone 1 is used as the smartphone, while Dell PowerEdge 1900 is used as the mirror server which has two Quad Core Intel Xeon 1.60GHz CPUs, 8GB Memory, 320GB Disk and installed Ubuntu 8.04 Desktop OS. The mirror server is connected to the IP network and can be accessed by the smartphone through T-mobile 3G network. In this prototype, for the synchronization feature, because we can not route or collect the incoming and outgoing messages of the smartphone in T-mobile 3G networks, we only replay the user input data on the mirror server so that the mirror takes the same tasks as the smartphone.

**The Smartphone** We use Android Dev Phone 1 as our smartphone. It is a SIM-unlocked and hardware-unlocked device that is designed for advanced developers. We implement a syn-client module on it to monitor the user keyboard input. The user keyboard input can be got from system program "/dev/input/event2" in real time. Whenever the user enters a key, this program generates 32 bytes data which includes the key pressing time, key releasing time and key id. We develop a syn-client module to collect the input data from "/dev/input/event2" in real time and transfer it through 3G network to our mirror server periodically.

**The Mirror Server** We use Android emulator as the mirror of Android Dev Phone 1. The size of the whole running Android emulator is 165MB. The Android OS is based on the simplified Linux kernel OS. Many virtualization techniques can be applied to our mirror server, such as OpenVZ [11] and QEMU. However, the current OpenVZ does not support Android OS. Android emulator basing on QEMU is simple and totally virtualization of the Android system.

**3G Networks** To measure the uplink and downlink speed of T-mobile EDGE and 3G networks, we develop a test program, which sets up a socket connection between the smartphone and the server. Then it keeps downloading UDP data from the server to the smartphone or uploading data to the server. The result is measured in State College as shown in Table 1. The 3G bandwidth is much faster than the EDGE, but currently the EDGE has more coverage than 3G, especially in a suburban area.

**Table 1.** The download and uploads speeds for EDGE and 3G

| | Uploading (kbps) | | Downloading (kbps) | |
|---|---|---|---|---|
| | Advertised | Tested | Advertised | Tested |
| T-mobile EDGE | 75-135 | 140-160 | 75-135 | 170-190 |
| T-mobile 3G | 500-1200 | 708 | 700-1700 | 1259 |

To measure the average time for uploading or downloading files in EDGE networks, we transfer several different size of files. As shown in Fig. 4, to transfer same amount of data, the downloading takes less time than uploading. In addition, it takes several minutes to transfer 5 MB data which is around the normal video file size.

## 5.2 Synchronization Cost

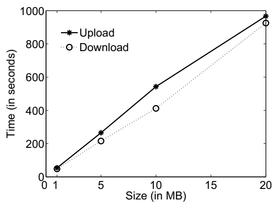


**Fig. 4.** The average time for uploading/downloading files in EDGE networks

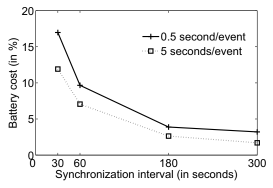


**Fig. 5.** The Synchronization cost for 6 hours with different interval

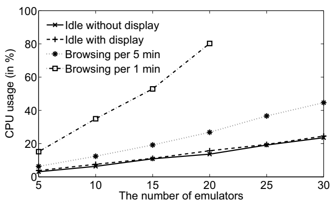


**Fig. 6.** The scalability for running multiple emulators on a server

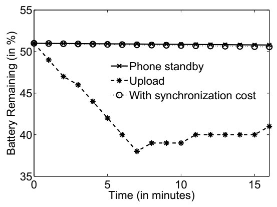


**Fig. 7.** The comparison of power consumption for uploading

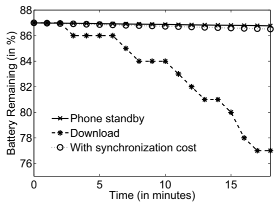


**Fig. 8.** The comparison of power consumption for downloading

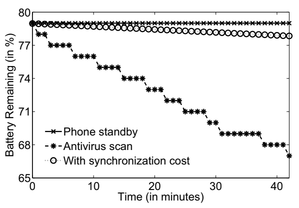


**Fig. 9.** The comparison of power consumption

In the proposed framework, the Syn-Client module inside the smartphone will send the input information to the mirror server periodically. The battery cost of this periodical sending is determined by two factors. One is the frequency of inputs and the other is the time interval between two sending operations. More frequent inputs will generate more data to send. A shorter time interval means a more timely synchronization but will lead to more frequent sending operations.

To measure the power consumption with different input frequencies and sending time intervals, we have measured the power consumption of periodical sending in two cases. In the first case, the smartphone is very active and generates two events every second. In the other case, the smartphone only generates one event every five seconds. Each event is encoded as a 32-byte synchronization message, and the sending is through a UDP connection. In both cases, we have measured the total battery cost with different sending time intervals in 6 hours.

The battery consumption is shown in Fig. 5. As can be seen from the figure, with the same frequency of event generation, the shorter sending interval will bring more battery cost. Also, with the same frequency of sending operations, more input events cause more data to be sent thus cost more battery power. For example, in case the user generates two input events per second, if the synchronization interval is greater than 180 seconds, the total battery cost for sending synchronization information on the smartphone will be less than 4%.

To show the benefit in downloading/uploading service, we also compare the battery cost for synchronization and file downloading/uploading using ftp with 20MB file. When synchronizing periodically, we choose a case in which the user generates 0.5 event per second in average and the smartphone synchronization period is 60 seconds. Fig. 7 and Fig. 8 show the power consumption of a smartphone in standby or when it synchronizes periodically with the mirror server. When in standby, the average power consumption is around 0.09% per minute in average. In this case, the average power consumption is around 0.117% per minute.

**Table 2.** The time and battery cost for performing antivirus scanning

| Folder Size (MB) | SMobile on smartphone | | Symantec on PC |
|---|---|---|---|
| | Time (sec) | Battery (%) | Time (sec) |
| 10 | 120 | 2 | 12 |
| 50 | 600 | 3 | 31 |
| 100 | 1140 | 6 | 57 |
| 200 | 2340 | 10 | 110 |

To show the benefit in antivirus scan applications, we install the SMobile VirusGuard [12] on an Android Dev Phone 1 and use it to scan folders with different sizes. All folders are filled with Android Package (APK) files which are used for application installation. As shown in Table 2, the time and battery needed for scan increases as the folder size increases. When the folder size is 200 MB, it takes more than 39 minutes and 10% battery to complete the scan, as shown in Fig. 9. During the scan, the user can explicitly feel the slow down of smartphone. Table 2 also shows much less time cost for scanning the same folders on PC using Symantec Endpoint Protection software.

### 5.3 Scalability

To evaluate the scalability of our framework, we study how many mirrors the server can support with different workload. In this experiment, we use web browsing service as an example to evaluate the scalability.

Fig. 6 illustrates the CPU usage versus different number of emulators, together with different workloads. In general, the mirror server can support at least 20 mirrors with heavy workload (browsing web every 1 minute) or more than 30 mirrors with light workload (open a webpage every 5 minutes). In this case, the CPU usage increases almost linearly with the number of mirrors.

## 6 Conclusion and Future Work

In this work, we propose a new framework, which uses Cloud Computing and VM techniques to shift the workload from smartphones to a computational infrastructure in telecom networks. It can greatly reduce the workload and virtually expand the resources of smartphones. We show the feasibility of deploying such a framework in telecom networks by protocol design, synchronization study and scalability test. A great number of issues are not addressed yet at this stage, especially those related to synchronization and actual system implementation. In our future work, we will study different synchronization mechanisms and find out what mobile applications can benefit from our framework. We will continue building the whole system and seek to evaluate its performance in a real 3G network. Finally, we will investigate the situation with additional incoming traffic through bluetooth and WiFi connections.

## References

1. M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, “The case for vm-based cloudlets in mobile computing,” *IEEE Pervasive Computing*, 2009.
2. B.-G. Chun and P. Maniatis, “Augmented smart phone applications through clone cloud execution,” in *Proc. HotOS XII*, 2009.
3. J. Oberheide, K. Veeraraghavan, E. Cooke, J. Flinn, and F. Jahanian, “Virtualized in-cloud security services for mobile devices,” in *Proc. MobiVirt*, 2008.
4. X. Zhang, J. Schiffman, S. Gibbs, A. Kunjithapatham, and S. Jeong, “Securing elastic applications on mobile devices for cloud computing,” in *Proc. CCSW*, 2009.
5. J. Oberheide, E. Cooke, and F. Jahanian, “Cloudav: N-version antivirus in the network cloud,” in *Proc. Security Symposium Conference (SS)* , 2008.
6. *3GPP Specification TS 23.107: Quality of Service (QoS) concept and architecture*, Std., Rev. Rel. 8. [Online]. Available: http://www.3gpp.org
7. B. Zhao, Z. Xu, C. Chi, S. Zhu, and G. Cao, “Mirroring smartphones for good: A feasibility study,” Tech. Rep., 2010. [Online]. Available: www.cse.psu.edu/~bzhao/mirror_report.pdf
8. Z. Zhu, G. Cao, S. Zhu, S. Ranjan, and A. Nucci, “A social network based patching scheme for worm containment in cellular networks,” in *Proc. IEEE INFOCOM*, 2009.
9. B. Zhao, C. Chi, W. Gao, S. Zhu, and G. Cao, “A chain reaction dos attack on 3G networks: Analysis and defenses,” in *Proc. IEEE INFOCOM*, 2009.
10. X. Jiang, X. Wang, and D. Xu, “Stealthy malware detection through vmm-based ”out-of-the-box” semantic view reconstruction,” in *Proc. CCS*, 2007.
11. “Openvz.” [Online]. Available: en.wikipedia.org/wiki/OpenVZ
12. “Smobile virusguard for google android.” [Online]. Available: www.smobilesystems.com