

# Mirroring Smartphones For Good: A Feasibility Study

Bo Zhao  
Department of Computer  
Science and Engineering  
The Pennsylvania State  
University  
bzhao@cse.psu.edu

Zhi Xu  
Department of Computer  
Science and Engineering  
The Pennsylvania State  
University  
zux103@cse.psu.edu

Caixia Chi  
Bell Laboratories  
Alcatel-Lucent Technologies  
chic@alcatel-lucent.com

Sencun Zhu  
Department of Computer  
Science and Engineering  
The Pennsylvania State  
University  
szhu@cse.psu.edu

Guohong Cao  
Department of Computer  
Science and Engineering  
The Pennsylvania State  
University  
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 an innovative framework which keeps a mirror for each smartphone on a computing infrastructure in the telecom network. With the 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. For TSP, traffic increase demands more investments for bandwidth expansion. The AT&T company also expressed its dissatisfaction on huge data traffic brought by iPhone users [5].

Recently some research has been done leveraging cloud techniques to help smartphones, including [13] [8] [12] [14]. [8] suggests an augmented execution model, in which application execution is off-loaded from smartphone to its clones in cloud. [12] focus on antivirus application and proposes an in-cloud antivirus system. 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. [14] suggests to build elastic applications which can be partitioned into function independent components. Those components can be executed in cloud instead of running on smartphones. [13] emphasizes on applications requiring real-time interactive response, such as augmented applications. In their framework, the smartphones delivers a VM overlay to the cloudlet infrastructure. Via this overlay, smartphones delivers part of execution to launch VMs, which provides various service and return results to smartphones. Smartphones are connected with cloudlets via high speed network connections.

*Contributions:* We propose an innovative 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 virtual machine (VM). To keep synchronization between mirror and smartphone, the mirror server replays all inputs to smartphone on its mirror with exactly the same order (Section 3). We show that the hardware cost will be in a reasonable range with our framework to support millions of telecom users (Section 6).

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Copyright 20XX ACM X-XXXXX-XX-X/XX/XX ...\$10.00.

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 AT&T (Section 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 5). One is the uploading/downloading 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, which is instead performed in its mirror.

*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.

## 2. RELATED WORKS

The idea of leveraging cloud computing techniques to enhance smartphones have been intensively discussed in recent years. Our framework distinguishes itself from two aspects. First of all, in our architecture, the cloud locates in TSP. A smartphone is connected with its mirror via 3G telecom network. Secondly, the mirror is for general purpose, not launched and configured only for specific applications. Smartphones and their mirrors are kept synchronized via a loose synchronization mechanism.

In [13], the authors propose a cloudlet infrastructure, where smartphones are 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 of all, in our proposed architecture, smartphones are connected with cloud via Telecom networks. A server in our case has much wider coverage than 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 will also exist on its mirror.

In [12], the author extends the CloudAV platform [11] to a mobile environment. In the proposed model, smartphones send suspicious files to a remote in-cloud antivirus network service, which provides a scanning service. The purpose of [12] 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 [12], first of all, 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

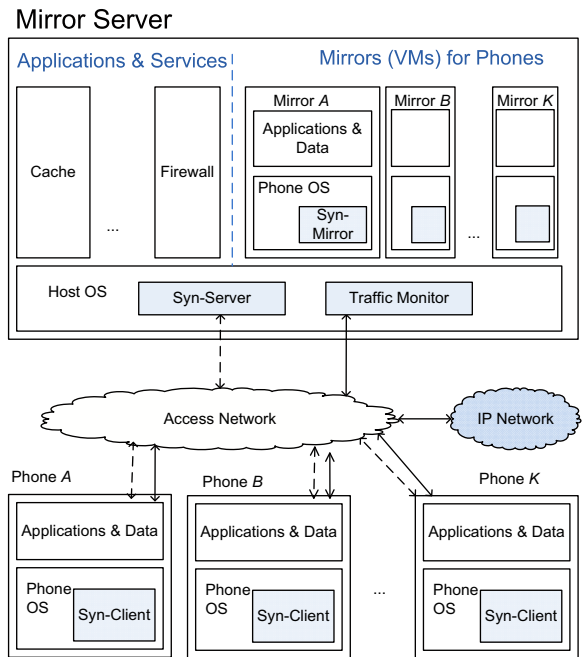


Figure 1: The System Design

be performed on-the-fly and directly on the mirror, reducing the file transmission cost in [12].

[14] suggests an elastic application model. In this model, one application is partitioned into components called weblets. By partitioning, some workload can be outsourced from smartphones to cloud elasticity services. However, this requires application developers carefully develop sophisticated applications to fit this model. Also, partitioning, scheduling, and result assembling are all complex. In our framework, no application partitioning is required.

In [8], Chun and Maniatis proposed a new architecture 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 propose to perform synchronization via whole-system replication, which is expensive because of the power consumption in synchronization.

Different from [8], we discuss and evaluate the feasibility of implementing a mirror for a smartphone in the real telecom network. We focus on saving power consumption on smartphones and minimizing the network traffic between smartphones and the 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, 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 appli-

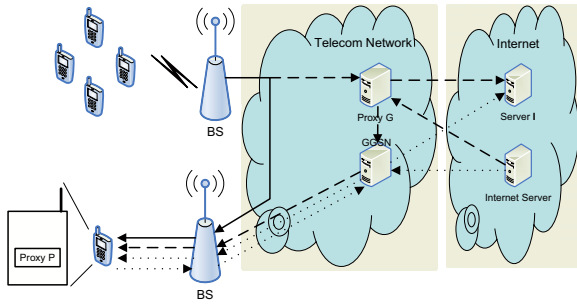


Figure 2: Placing the mirror server in the Internet

cation 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 needs to monitor all the incoming traffic of smartphones. Currently, we only consider the case when a smartphone is connected through a telecom network. So, the incoming traffic of a smartphone consists of data originating from both the Internet (e.g., Internet websites) and the telecom network (e.g. AT&T). Bluetooth or WiFi connections are not considered.

To deploy the mirror server, we consider two candidate locations. One location is on the Internet, e.g., the *Server I* shown in Figure 2. In this case, the mirror server itself can not collect the whole incoming traffic data directly. Instead, it will need a proxy deployed either in the telecom network or on the smartphone to collect incoming traffic information for it. We present how these proxies can be deployed in Figure 2. The *Proxy G* and dashed lines represent the proxy in telecom network and its message flows. The *Proxy P* and dotted lines represent the proxy on smartphone itself and its message flows. Either *Proxy G* or *Proxy P* is needed to collect incoming traffic of a smartphone and forward it to the mirror server *Server I*.

Unfortunately, both proxies have obvious disadvantages. To implement *Proxy G*, it has to sit close to the gateway GGSN. And we have to make modifications to the existing telecom network in order to forward a copy of incoming traffic from both the telecom network and Internet to *Proxy G*. On the other side, putting a *Proxy P* on a smartphone will bring communication and battery consumption to the smartphone, because *Proxy P* has to collect and send incoming traffic information to *Server I*.

Considering these disadvantages, we choose to deploy the mirror server within the telecom network, as shown in Figure 3. And we make modification to existing 3G telecom network to forward a copy of all the incoming traffic of smartphones to the mirror server, as shown by the dashed line in Figure 3. The details of our network design will be presented in Section 4.

### 3.2 Mirror Design

Compared with PC, it is much easier to create mirrors for smartphones because all smartphones of one model share the same hardware specifications, default factory settings, OS, and a lot of ap-

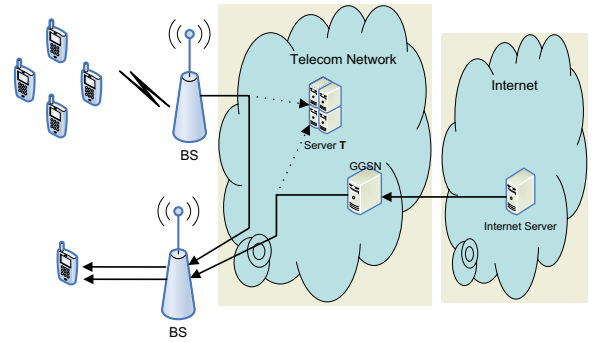


Figure 3: Placing the mirror server in the telecom network

plication 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.

Specifically, 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 the 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 are going to support in the mirror server.

#### 3.3.1 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, uploading/downloading caching application, and virus scan. 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 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 generating a random number. However, it is suitable for most applications on current smartphones, such as email, IM, web browsing, VoIP, MMS/SMS, and present service. Therefore, we monitor all the inputs to the smartphone and later applied 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, 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.

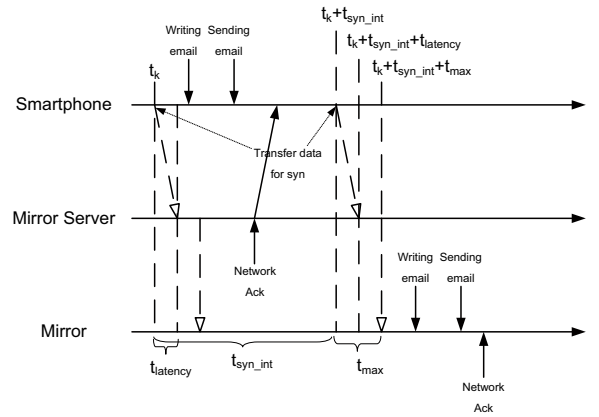
### 3.3.2 Synchronization Timelines

Next we explain the detailed operation 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 [3]. 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. 4 illustrates how this approach works with email services 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.

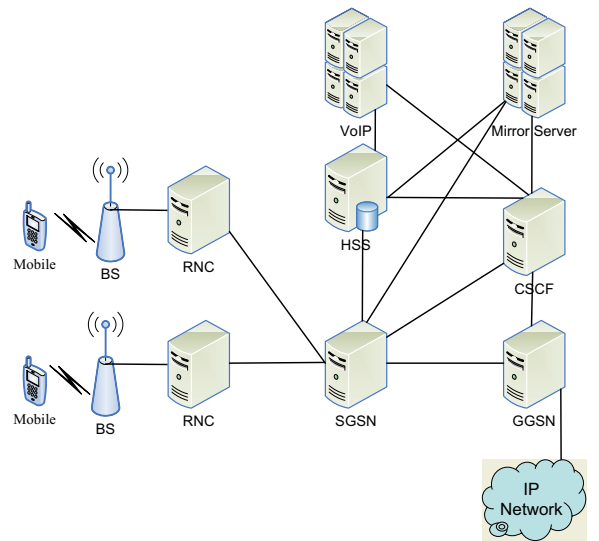
When a smartphone boots up, the initial time for the synchronization is set. At this time, on the smartphone side, the user can configure the value of  $t_{syn\_int}$  for balancing the efficiency and reaction delay, while on the network side, the mirror server starts up this smartphone’s mirror. When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

## 4. NETWORK DESIGN

In this section, we illustrate the feasibility of telecom network-based solution by describing how mirror server works compatibly



**Figure 4: Timelines for mirror synchronization (Solid arrow lines stand for input data, and dash arrow lines stand for synchronization messages)**



**Figure 5: The 3G Networks Architecture**

with other network nodes with minor change to 3G networks. In addition, two sample message flows of typical telecom services are described in detail. Note that in this paper, we only consider the situation that a smartphone is connected only through a 3G network, not through bluetooth or WiFi connections.

### 4.1 Network Architecture and Protocol 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. Fig. 5 gives the network architecture with the mirror server.

Before we describe how the mirror server works in 3G networks, we need introduce the overview of the Universal Mobile Telecommunications System (UMTS) network architecture [1] as a concrete example of a 3G network. The signals of mobile devices

first go through Base Station (BS), and then are handled by the Radio Network Controller (RNC), which is the point where wireless link layer protocols terminate. The Serving GPRS Support Node (SGSN) is responsible for sending data to and from servers and network nodes. The Gateway GPRS Support Node (GGSN) is the gateway for UMTS packet services to external packet data networks such as the Internet. It routes packet data from the core network to the external network on request by SGSN [2]. The Home Subscriber Server (HSS) is the central repository for user-related information, such as location information, security information and user profile information, which are required to handle multimedia sessions. The Call/Session Control Function (CSCF) is an essential SIP server, which processes SIP signaling. The rest of the nodes, such as VoIP and mirror server, are related to the application server.

Taking UMTS as the example of a 3G network, for a user 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 (which is a SIP server). 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.

For the services with session control, such as presence service, VoIP, or multimedia service (MMS), their session control messages, such as SUB, INVITE, are forwarded by SGSN to CSCF. With the mirror service registered, the CSCF first routes session messages to the mirror server. The mirror server processes the session messages following the actual service protocol on behalf of the target. Once the entire session is finished, the mirror server initiates a new session with the original target. In this way, the mirror server is able to record all the messages coming to or going out of a registered smartphone without ruining the function of the session protocol.

For the services without session control, such as browsing webpages or email service, when SGSN receives a message from/to a registered user, it will first obtain the user policy from the CSCF, and then forwards the message to the mirror server. The mirror server will record the message and then forward it back to SGSN so that the message will be delivered to its original target in a normal way.

Note that the above protocol design is not necessarily the only way or the most efficient way. Our design principle is the most compatibility, so that our system is feasible for real world deployment. As we stated earlier, in this design no modification is made to the existing software or protocols in the network nodes except changing configurations and registering user profiles in SGSN, CSCF and HSS. If we are allowed to redesign the protocols in SGSN and CSCF, we can let them repeat their protocols with both the original target of a message and the mirror server simultaneously without first going through the mirror server, thus saving message overhead.

Next, for better understanding, we show the message flows involved in two typical services, MMS service (with session control) and email service (without session control).

## 4.2 Message Flows For Example Services

Fig. 6 shows the original message flow in the MMS service, based on the 3GPP standard [4]. The sender “Mobile #1” alerts the receiver “Mobile #2” by sending an INVITE message and receives the MSRP:VISIT message; the receiver accepts and sends 200 OK message back. Next, the sender sends a MSRP:SEND message, which contains the multimedia data, to the receiver. Now suppose the receiver, “Mobile #2”, has the mirror service. Fig. 7 shows the

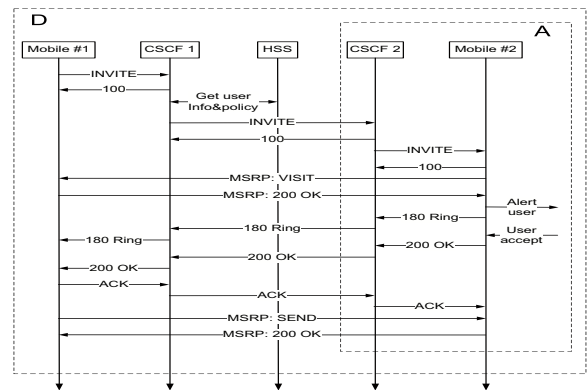


Figure 6: Original Network Message Flow of MMS

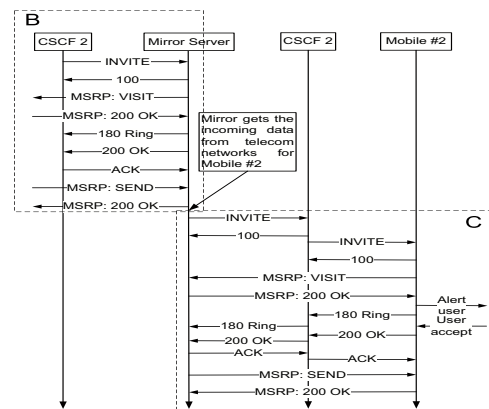


Figure 7: Mirror Based Message Flow of MMS

new message flow in the receiver side network. Here the message flows in zone A of Fig. 6 and zone B of Fig. 7 are similar except that no user action is taken. The message flows in zone C of Fig. 7 and zone D of Fig. 6 are also similar, except that no HSS and the sender side nodes are involved. Basically, the CSCF 2 of the receiver side network redirects the received session control messages to the mirror server, which acts as the receiver. This process is transparent to the sender “Mobile #1”. Once the session is over, the mirror records the multimedia data in the MSRP:SEND message. Next, to deliver the data to the real receiver, the mirror server initiates a new session to the receiver through the same CSCF 2.

The original message flow of browsing a webpage is illustrated in Fig. 8. With the mirror service, Fig. 9 shows that once the SGSN knows that the receiver “Mobile #1” has the mirror service from the CSCF, it forwards the packet to the mirror server first. After the mirror server handles this packet, it forwards this packet back to the receiver through the same SGSN.

## 5. APPLICATIONS

The proposed framework can support many applications. In this

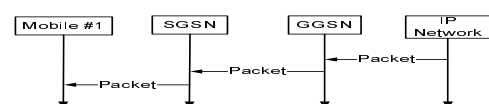


Figure 8: Original Web Browsing Message Flow

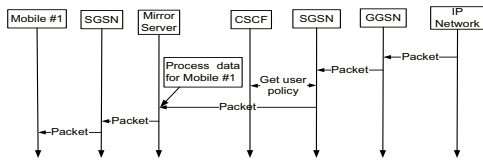


Figure 9: Mirror Based Web Browsing Message Flow

Table 1: The download and uploads speeds for EDGE and 3G

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

section, we identify two of them.

## 5.1 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 have been significantly increased in recent years, receiving and sending bulk data still take time and consume power, as shown in Fig. 11. The figure is drawn based on the downlink and uplink speed measured in State College as shown in Table 1.

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.

### 5.1.1 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 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 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 access to battery charging.

### 5.1.2 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 per-

formed 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, 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. In Fig. 15 and Fig. 14, we compare the battery cost for synchronization and file downloading/uploading using ftp with 20MB file. We also show the power consumption of 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. When synchronizing periodically, we choose a case in which user generates 0.5 event per second in average and the smartphone synchronization period is 60 seconds. In this case, the average power consumption is around 0.116835% per minute.

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 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.

## 5.2 Antivirus Scanning service

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 signature, 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 scan adds inconvenience to users and drains the battery quickly.

To see the impact of antivirus scan, we install SMobile Virus-Guard [6] 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. 13. In Fig. 12, we also measured the CPU usage during the scanning, using the Android NetMeter application. The result of our measurement is presented in Fig. 12. In normal usage, the CPU usage is less than 10%. When scanning starts, the CPU usage increases dramatically to almost 100%. Most of CPU usage is taken by the scanner. And the high CPU usage continues until the scan is complete. During the scan, the user can explicitly feel the slow down of the smartphone.

In the proposed framework, antivirus scanner can be deployed as a service on the mirror server, and the scanner can access the file system on mirrors. Further, a hook application is installed on the phone. When a user launches the hook application on the smart-

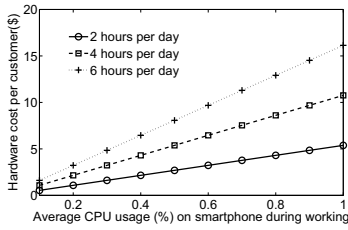


Figure 10: The hardware cost per customer to support mirrors on the server

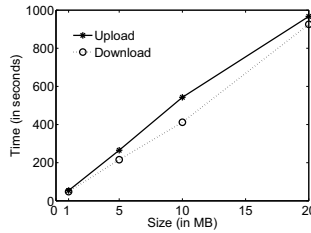


Figure 11: The average time for uploading/downloading files in EDGE networks

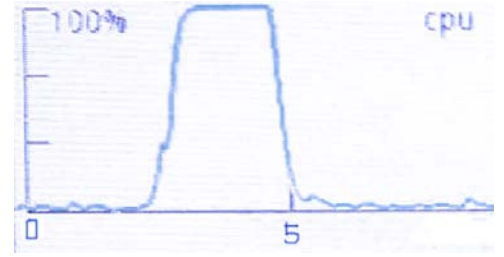


Figure 12: CPU usage during the scanning

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

phone, 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 antivirus service outside of protected VM has been proved in the paper [10]. The VMwatcher [10] can be deployed directly on our framework.

The benefits of shifting antivirus scan are significant. First, it saves battery power on smartphones. Second, as the CPU and I/O intensive workload are shifted to the mirror server, scanning will not affect the common usages of smartphone. Finally, the scan speed on mirror will be much faster than that on the phone due to its limited hardware capabilities. Table 2 also shows much less time cost for scanning the same folders on PC using Symantec.

## 6. SCALABILITY AND COST ESTIMATION

In this section, we explain the scalability of the present Cloud Computing and demonstrate that our framework is scalable. We also estimate the cost of deploying our service to millions of users.

### 6.1 Scalability Study

Due to rapid advances in technology, scalability is not that critical for Cloud Computing. Based on the data in [9], for a large data center which supports 50,000 servers with Cloud Computing, its network cost is \$1.6 per MB/sec/month, and its storage cost is \$0.40 per GB/month. The service to end users is also very cheap. For example, the Elastic Compute Cloud (EC2) from Amazon Web Services (AWS) sells 1.0-GHz x86 ISA slices for \$0.1 per hour. Amazon's Scalable Storage Service (S3) charges \$0.12 to \$0.15 per GB/month [7].

Our mirror service relies on smartphone mirrors on Cloud Computing, and hence its scalability depends on that of the Cloud Computing. The scalability of mirror service is better than that of the PC virtualization service on Cloud Computing because the work-

Table 3: The Hardware Configuration Data

Phone Device	CPU	Memory	Battery Life
Android G1	528 MHz	192 MB	6 hours talk
iPhone 3G	620 MHz	128 MB	6 hours talk
Server Device	Price	CPU HZ	Memory
Dell PowerEdge 1900	\$1566.00	12.8 GHz	8 GB

load of smartphone mirror is much less than PC virtualization. For example, the OSes of the smartphones are much simpler, and it uses much less amount of system resources compared to the normal OSes of PCs. In addition, the network traffic of a smartphone is much less than those of PCs. Compared to the workload of a real smartphone, the workload of its mirror can be further reduced, since it does not need to handle wireless transmission, display, and audio functions. Therefore, the scalability of our framework should not be a problem. To implement our mirrors on the mirror server, there are many virtualization techniques, such as Xen, OpenVZ and QEMU for us to choose.

### 6.2 Cost Estimation

We emphasize that it is only a rough estimation. The accurate data about the cost and the scalability depends on the Cloud Computing environment and the business model.

Based on the hardware configuration data listed in Table 3, we calculate the cost of each mirror. We use CPU as the basic parameter to roughly estimate how many Android emulators the server (PowerEdge 1900) can support. Each PowerEdge 1900 has two CPUs which have 8 cores running at 1.6 GHz.

Suppose  $\alpha$  is the working hours per day and  $\beta$  is average CPU usage (%) during working.  $\alpha$  is at most 6 hours per day according to iPhone battery usage description. Since on the mirror side we do not need to process wireless signals or handle display functions, some resource can be saved. Let  $M$  denote the CPU usage percentage saved on the server side per VM. and  $H$  denote the CPU usage overhead percentage caused by the virtualization technique per VM. Let  $N$  denote the number of customers a server can support. Then,

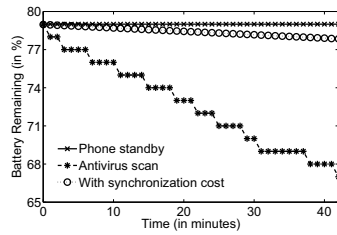
$$N = 12800/528 \times 24/(\alpha \times \beta \times (1 - M) \times (1 + H))$$

Because we do not know the value of  $M$  and  $H$ , we ignore them to simplify the model. Also let  $C$  denote the hardware cost per mirror. Then,

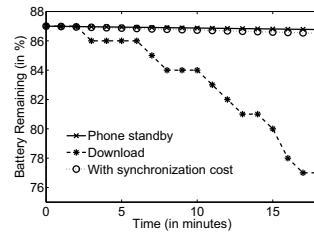
$$N = 12800/528 \times 24/(\alpha \times \beta)$$

$$C = 1566.00/N = 2.69 \times \alpha \times \beta$$

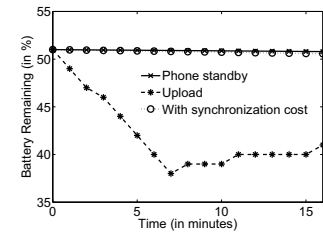
Fig. 10 shows that the total hardware cost per mirror is at most \$15 even when users use 100% CPU for the whole 6 hours. Normally the cost ranges between \$5 and \$10. In addition, if the cost



**Figure 13: The comparison of power consumption for scanning**



**Figure 14: The comparison of power consumption for downloading**



**Figure 15: The comparison of power consumption for uploading**

is averaged to the monthly payment from users, this service price should be less than the SMS (\$5 per month). The service price also depends on the TSP business model.

## 7. CONCLUSION AND FUTURE WORK

In this work, we propose an innovative framework, which use 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. 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.

## 8. REFERENCES

- [1] 3GPP release 1999 UMTS specifications.
- [2] 3GPP specification 25.410: UTRAN iu interface: General aspects and principles.
- [3] 3GPP specification ts 23.107: Quality of service (QoS) concept and architecture.
- [4] 3GPP specification ts 24.247: Messaging service using the ip multimedia (im) core network (cn) subsystem.
- [5] No shocker: iphone drives at&t metrics in 2q.
- [6] Smobile virusguard for google android.
- [7] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, and R. Katz. Above the clouds: A berkeley view of cloud computing. Technical report, UC Berkeley Reliable Adaptive Distributed Systems Laboratory, 2009.
- [8] B.-G. Chun and P. Maniatis. Augmented smart phone applications through clone cloud execution. In *HotOS XII: Proceedings of the 12th Workshop on Hot Topics in Operating Systems*, May 2009.
- [9] J. HAMILTON. Internet-scale service efficiency. In *Large-Scale Distributed Systems and Middleware (LADIS) Workshop*, September 2008.
- [10] X. Jiang, X. Wang, and D. Xu. Stealthy malware detection through vmm-based "out-of-the-box" semantic view reconstruction. In *CCS '07: Proceedings of the 14th ACM conference on Computer and communications security*, pages 128–138, New York, NY, USA, 2007. ACM.
- [11] J. Oberheide, E. Cooke, and F. Jahanian. Cloudav: N-version antivirus in the network cloud. In *SS'08: Proceedings of the 17th conference on Security symposium*, pages 91–106, Berkeley, CA, USA, 2008. USENIX Association.
- [12] J. Oberheide, K. Veeraraghavan, E. Cooke, J. Flinn, and F. Jahanian. Virtualized in-cloud security services for mobile devices. In *MobiVirt '08: Proceedings of the First Workshop on Virtualization in Mobile Computing*, pages 31–35, New York, NY, USA, 2008. ACM.
- [13] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. The case for vm-based cloudlets in mobile computing. *IEEE Pervasive Computing*, 8(4):14–23, 2009.
- [14] X. Zhang, J. Schiffman, S. Gibbs, A. Kunjithapatham, and S. Jeong. Securing elastic applications on mobile devices for cloud computing. In *CCSW '09: Proceedings of the 2009 ACM workshop on Cloud computing security*, pages 127–134, New York, NY, USA, 2009. ACM.