# Clone2Clone (C2C): Peer-to-Peer Networking of Smartphones on the Cloud[*]

Sokol Kosta[*], Vasile Claudiu Perta[*], Julinda Stefa[*], Pan Hui[†], and Alessandro Mei[*]
[*]*Sapienza Univ, Rome, Italy,* [†]*HKUST, Hong Kong and Deutsche Telekom Labs, Berlin, Germany.*
[*]*{lastname}@di.uniroma1.it,* [†]*pan.hui@telekom.de.*

## Abstract

In this work we introduce Clone2Clone (C2C), a distributed peer-to-peer platform for cloud clones of smartphones. C2C shows dramatic performance improvement that is made possible by offloading communication between smartphones on the cloud. Along the way toward C2C, we study the performance of device-clones hosted in various virtualization environments in both private (local servers) and public (Amazon EC2) clouds. We build the first Amazon Customized Image (AMI) for Android-OS—a key tool to get reliable performance measures of mobile cloud systems—and show how it boosts up performance of Android images on the Amazon cloud service. We then design, build, and implement C2C. Upon it we build CloneDoc, a secure real-time collaboration system for smartphone users. We measure the performance of CloneDoc by means of experiments on a testbed of 16 Android smartphones and clones hosted on both private and public cloud services. We show that C2C makes it possible to implement distributed execution of advanced peer-to-peer services in a network of mobile smartphones *reducing* 3 times the cellular data traffic and *saving* 99%, 80%, and 30% of the battery for respectively security checks, user status update and document editing.

## 1 Introduction and Motivation

Smartphones have changed the way we interact with our mobiles. We use them to call/text as well as to send emails, tweet, play video-games, shop, watch videos and so on. All this comes at a price: Battery life. With the apps becoming always more complex, we are destined to suffer battery limits even more in the future. Considerable research work have proposed solutions to address the issues of computational power and battery lifetime by offloading computing tasks to cloud. Frameworks like MAUI [7] and ThinkAir [10] are just a few of the many offloading methodologies that achieve mobile computation efficiency and prolong battery life.

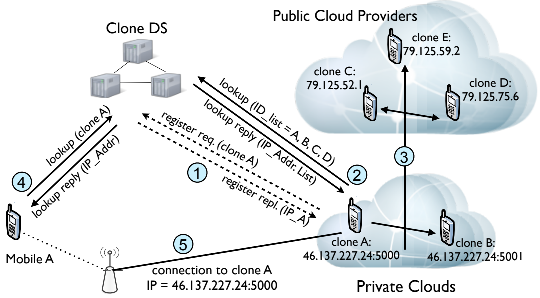

Figure 1: The C2C architecture and networking.

In this work we push the smartphone-cloud paradigm to a further level: We aim at smartphone-to-smartphone communication offloading, together with computation offloading. With this in mind, we develop Clone2Clone (C2C), a distributed platform for cloud clones of smartphones. C2C associates a software clone on the cloud to every smartphone and interconnects the clones in a peer-to-peer fashion exploiting the networking service within the cloud. A wireless P2P network between smartphones is notably hard to realize due to many factors: Severe battery limitations, frequent loss of cellular coverage in e.g. subway or rural areas, the way carriers NAT mobile Internet, and so on. These problems are mitigated with the C2C platform, where the clones are virtually always on and peer-to-peer connectivity lies upon the high-bandwidth network of the cloud. Most importantly, C2C opens up a wealth of novel ways to explore the integration of mobile computing, networking, and the cloud. It makes it possible to build P2P-based protocols for smartphones, as well as services like content sharing, search, distributed execution among the C2C network users, and so on. All this, without the need of relying on a continuous, and, impossible to achieve, P2P connection between real devices. In addition, coupled with any of the existing offloading techniques [7, 10], it also can help offload

[*]This work has been performed in the framework of the FP7 project TROPIC IST-318784 STP, which is funded by the European Community. The Authors would like to acknowledge the contributions of their colleagues from TROPIC Consortium (http://www.ict-tropic.eu).

heavy mobile computational tasks.

On top of C2C we implement CloneDoc, a secure, real-time collaboration system for smartphone users that work simultaneously on the same document, dealing with an insecure server. We use CloneDoc as a paradigmatic P2P-like application, and show how C2C achieves communication offloading, by dramatically decreasing the data traffic handled by the smartphone users (3 times less). In addition, we demonstrate that, thanks to C2C, all the heavy crypto and security tools required by CloneDoc are handled on the cloud. This translates in 99%, 80% and 30% of energy saving for respectively security checks, user status update, and document editing.

## 2 Related work

In the scientific community it is a common belief that a promising way around the problem of short battery life of smartphones is offered by offloading mobile computation on the cloud. Early works in this direction, like [13, 5] show how to benefit from cloud offloading in terms of security and energy-efficiency. In [7] the authors describe MAUI, a method level code offloading system based on the Microsoft .NET framework. MAUI exploits estimation models and application profilers to offload particularly heavy methods on the cloud. CloneCloud [6] uses a process-based offloading methodology: The binary of the application is partitioned and an off-line analysis decides which binary pieces are to be migrated to the cloud. Experiments with clones hosted on private cloud (local servers) show up to 20x energy saving with applications such as virus scanning, image search, and behavioral profiling. The first study of the energetic overhead of the device-clone synchronization is presented in [2]. The works [1, 11, 3] show how offloading benefits from associating a clone on the cloud to each smartphone.

Note that none of these offloading mechanisms provide P2P inter-connection of smartphones through the cloud, nor do they provide communication offloading, as we will see from our experiments that C2C does. Note also that our C2C platform offers the possibility to users to adopt an offloading methodology totally independent from that of other users in the platform.

## 3 How to clone on the cloud

For our C2C platform we exploit Android x86—an Android port to the x86 architecture. We consider two hosting strategies: (1) Private cloud, or, (2) public (commercial) cloud computing service. On the private cloud (Linux local servers, CPU Core(TM)2@1.83GHz), we consider three different virtualizing methods: Xen, VirtualBox (VB), and through the QEMU emulator [4]. As for the public cloud service, we opt for the Amazon EC2 platform (High-CPU Med., 2 virt.cores 2.5 ECUs). Its virtualization environment is Xen, and it expects a compatible Amazon Machine Image (AMI). Unfortunately, an AMI for Android-x86 that runs on top of Xen does not exist. In addition, aside from the processor emulator QEMU [4], no other virtualization environments are compatible with Amazon EC2 (including VB). So, public cloud cloning so far is only possible by using the Android emulator which runs on top of QEMU. Note that QEMU runs on top of a virtualized Linux AMI, which in turn runs on top of Amazon's Xen. This is far from being efficient—there are two virtualization layers between Android and Xen. So, we designed and built a *bundle*—a custom Amazon Machine Image for Android-x86 (*Ax86AMI*). This was not easy. It involved re-designing Android drivers from scratch, adding virutalization support for Xen, disabling kernel level modules and sensors, recompiling the kernel of Android x86 etc. We omit the details of this procedure due to lack of space[1].

| Test | QEMU L. | VB L. | Xen L. | QEMU A. | Ax86AMI |
|---|---|---|---|---|---|
| CPU(MFlops) | 2.5 | 34 | 45 | 2.6 | 57 |
| I/O Rd | 0.5 | 5.5 | 6.2 | 0.7 | 7.2 |
| I/O Wr | 0.3 | 5.1 | 5.8 | 0.45 | 6.8 |

Table 1: Clone performance. L is for local; A is for Amazon.

We compare the performance of the various clones in terms of CPU (through the Java version of Linpack[2]), and bufferized file access (through the standard Java API). During the test the clone makes use of one processor core. Each experiment is repeated 100 times and the average performance with each cloning method is shown in Table 1. One can notice that Xen clones win over all the other virtualization methods, on both Private and Public cloud (recall that the Ax86MI runs on top of Xen). Most importantly, our Ax86MI bundle boosts enormously the performance of the clones on Amazon: Indeed, QEMU clones perform 23 times less in terms of CPU than the Ax86AMI clones. In addition, our clones are more than 13 (13.5) times faster in performing I/O Read (Write) operations. The benchmark results also show that public cloud clones outperform private cloud ones. This is expected: Amazon's High-CPU Medium instances are more powerful than our local servers.

Note that is not necessary for the clones to be images of the Android OS running on the cloud—they could be Linux x86 OS running JVM applications. However, from a software engineering point of view running Android OS images makes it straightforward to install/uninstall user apps on the clone and use them for offloading.

## 4 C2C: Architecture Design

To enable peer-to-peer networking among smartphone clones, the C2C platform needs a mechanism that "notifies" clones about the presence of others and gives information on how to connect to them. Here we stick to

[1] Our AMI for Android-x86 is open source. If you need it to run experiments, just drop an email to one of the authors of this paper.

[2] http://www.netlib.org/benchmark/linpackjava/

a simple baseline architecture for our platform (see Figure 1). It includes a *directory service* (CloneDS in the figure) which takes care of mapping users to clones and clones to IPs. The CloneDS is always up and its IP is known (made public by e.g. the C2C platform builder). We assume also that all the entities in the system—users, cloud providers, and the CloneDS—have a private/public key pair and can securely verify their authenticity. A user willing to join the C2C platform requests a clone, equipped with a private/public key par and a public IP, to her cloud provider of choice. The public key is signed by the user, so everybody can verify the clone's owner. The newly created clone performs the following steps, shown also in Fig. 1: (1) *DS register:* The clone sends to CloneDS its respective device ID (mobile A), its ID (clone A), its IP address, and its public key. (2) *DS lookup:* Clone A receives a list signed by CloneDS of all other clones entries along with their IPs and public keys. (3) *C2C connect:* Clone A starts P2P connections with the other clones. (4) *User lookup:* User A can always get her clone's IP through a CloneDS lookup. (5) *User-clone connection:* The user is now ready to connect to its clone through its public IP, and installs whatever she likes in her cloned devices (the newly created clone is a default Android-x86 image running on the cloud, and does not contain any user data). This includes apps that she already has in her real smartphone. In addition, the user negotiates a symmetric key with her clone (this is done in a standard way thanks to the authenticated private/public keys). This key will be used to encrypt and sign user-clone communication.

## 4.1 C2C and security

In C2C, we assume that the users trust their own cloud provider. However, they do not trust the cloud providers of the other users. The interaction between the mobile device of a user and its clone is secured by using a shared symmetric key. In this way encryption of packets and signatures performed by the real device can be implemented efficiently, which allows devices to save energy.

In the architecture we described, the CloneDS is an external entity with respect to the cloud providers and is trusted by all the users in the system. Therefore, correct users can trust that the information provided by the CloneDS is consistent. However, since this information is provided by the cloud providers of the other users, malicious cloud providers can make it impossible to connect to their own users. Or make other users connect to fake malicious clones. Since the user trusts her cloud provider, we cannot do much about this.

An alternative architecture for the CloneDS is to implement it as a distributed service among the cloud providers. In this distributed version, the CloneDS is replicated on the cloud providers, users connect to the replica on the cloud they trust, and the replicas are kept consistent by broadcasting signed updates among the cloud providers that are part of the system. Of course, malicious providers can again forge information about their own users. But correct users and cloud providers can connect correctly. This distributed alternative might be more available, since the cloud typically guarantee high availability.

# 5 Secure real time collaboration

Many recent works like [12, 14], and the most recent SPORC [9], argue that secure, real-time group collaboration on sensitive information, e.g., Banking, or shared documents between cooperating companies, can be efficiently deployed by making use of untrusted external servers. Their goal is to force a global order on the concurrent users' operations. Due to the sensitiveness of the information, the server is considered as potentially malicious: Its goal may be to partition the clients in disjoint groups with different views of the document. So, platforms like e.g. GoogleDocs are not to be used in these cases. Prevalent solutions to the problem [9, 12, 14] involve recurrent P2P communication among users, as well as heavy cryptography to guarantee crucial security and system properties. Clearly, all this is unfeasible for our resource-limited smartphones.

However, with the C2C platform the tables are turned. C2C delivers efficient peer-to-peer networking for smartphones by moving computation and, most importantly, communication on the cloud. We exploit these features of C2C and modify SPORC's architecture to build CloneDoc—The first energy-efficient and real-time collaboration system for battery constrained smartphones. CloneDoc demonstrates that our idea of moving communication and enabling P2P networking on the cloud makes it possible to design energy-efficient non-trivial P2P applications for smart-phones.

## 5.1 CloneDoc: System Architecture

Even though similar to SPORC [9], CloneDoc makes use of the C2C platform which introduces more complexity to the system overall, yet reducing battery consumption by liberating the mobile devices from many tasks. CloneDoc is a typical P2P-like application with communication among peers and not-so-light computation due to cryptography. Thus, we use it as a stress-test for C2C.

The main idea in CloneDoc is to make the clone on the cloud receive operations from the mobile device, handle as many tasks as possible on the device's behalf and keep the device up-to-date. Recall that the clone is trusted by the user. In scenarios where CloneDoc is deployed, e.g. for Banking or intra-enterprise collaboration, the clones might reside inside the enterprises' private cloud platforms. The clone (similarly to the client in SPORC) maintains two states: the *pending queue* and the *committed queue*. The clone behaves as follows: (1) submits to the server operations that he gets from the user's real-device and (2) appropriately transforms (using Operational Transformation [8]) the operations of the other users received from the server. It is thus responsible for correctly handling the queues so that its view of the document is coherent to that of other clones. Last, but not least, the clone sends back to the real device the operations such that the user's view is coherent to that of other users in the system. However, the real and the cloned device are not physically the same. This translates into an unavoidable delay in their communications that, if not appropriately managed, may introduce inconsistency. So, CloneDoc includes a clone–user consistency protocol, that solves this problem, yet allowing the user to apply her operations optimistically on her device: We want a system that looks real-time to the user.

The clone in CloneDoc is also responsible for detecting server misbehavior. This involves encryption and decryption of each operation labeled and received by the server, checks on sequence numbers and hash chains of histories sent by other users' clones that help with the detection and so on. In addition, when the respective user is disconnected (e.g. due to loss of cellular coverage), the clone continues being on, receiving other users' operations and applying them locally. All the user has to do when she gets back is to pull from the respective clone the set of operations that she missed during her absence. This has many advantages. First, the user's device need not transform these operations past possible operations contained in its local pending queue (that the clone has). Indeed, the clone has already taken care of such transformations. Second, the clone optimizes the list of operations by canceling possible *add character* followed by a *del character* in the same position. This reduces the number of operations the real device receives and applies to the document. Third, the clone sends the whole sequence encrypted as a unique bundle. So, the device does only one decryption, instead of one per operation as in SPORC. Finally, the clone also takes care of all the crypto operations necessary to handle adding or deleting users from the system. Nonetheless, the communication between clone and real-device is always done using secure SSL channels to avoid eavesdropping. According to the experiments on the testbed, relieving the smartphones by handling all these tasks on the clone results in enormous energy and cellular bandwidth savings.

## 5.2 Experimental Setup and Results

To test CloneDoc we compare it with SPORC. Unlike [9], that exploits Google Wave's source code[3]we implemented both systems from scratch—Google Wave's source code is not compatible with Android.

| Number, type & OS | CPU | RAM |
|---|---|---|
| 6×Samsung Galaxy S+ (Android 2.3) | 1.4 GHz | 512 MB |
| 2×Nexus S (Android 4.0.1) | 1 GHz | 512 MB |
| 2×HTC Desire (Android 2.3) | 1 GHz | 576 MB |
| 6×HTC Hero (Android 2.1) | 528 MHz | 288 MB |

Table 2: Specifics of the mobile devices used in the testbed.

| Protocol& Smartphone | Phase 1 | Phase 2 | Phase 3 | Phase 4 |
|---|---|---|---|---|
| SPORC-Hero | 2300 | 140 | 180 | 5.5 |
| SPORC-Samsung | 1700 | 60 | 130 | 2.4 |
| CloneDoc-Hero | 1500 | 22 | 7 | 0.9 |
| CloneDoc-Samsung | 1250 | 18 | 5 | 0.8 |

Table 3: Energy consumption in mJ during the 4 test phases.

Our C2C testbed consists of 16 real devices (see Table 2 for more details) and an equal number of clones deployed in a hybrid cloud platform: 14 hosted on Amazon, and 2 hosted on our private server. The untrusted server resides on our private cloud to make the scenario as heterogeneous as possible. Devices communicate with the untrusted server and the clones (in C2C) through WiFi.

The test, repeated 10 times, has four phases: (1) Editing phase–users edit the document concurrently; (2) Update of temporarily disconnected user status; (3) Security check to detect misbehavior; (4) User membership remove. The test is repeated 10 times and the results on single and meaningful operations are averaged. Here we present the experimental results relative to the two extreme cases: Samsung Galaxy S+ (the highest performing smartphone) and HTC Hero (the lowest performing one). We measure energy consumed by the smartphones on both systems with the Mobile Device Power Monitor[4], used by many other works in the area [7, 10]. This device samples the smartphone's battery with high frequency (5000 Hz) so to yield accurate results on the battery's power, current, and voltage. The results, from the energy consumption perspective, are presented in Table 3. The energy savings achieved with CloneDoc is high in all the phases—around 30% for document editing (phase 1), more than 80% for status update (phase 2), more than 99% for security check (phase 3), and more than 80% for the user membership remove (phase 4). This is not surprising: the clones deal with most of the heavy tasks in the system, by thus dramatically decreasing the burden of the real devices. Most importantly, in all cases the gap between the energy consumed from HTC Hero with that consumed by the Samsung Galaxy S+ is reduced on CloneDoc. This confirms that the C2C platform boosts up the performance of old-fashioned low-performing smartphones making them competitive with newer, more expensive, and more performing ones.

### 5.2.1 Network Traffic

We have measured the number of bytes sent/received during the test by the active smartphones when running

[3] http://www.waveprotocol.org/whitepapers/operational–transform

[4] http://www.msoon.com/LabEquipment/PowerMonitor/

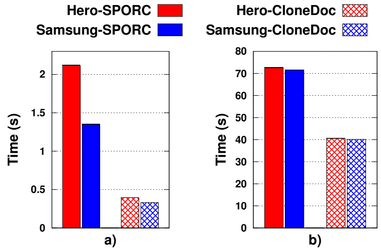

Figure 2: Time to (a) update the state of a temporarily disconnected user; (b) apply all edit users' operations.

SPORC and CloneDoc. The result is that the overall network bandwidth used by the smartphones with CloneDoc is 3 times less than with SPORC (see Figure 3). This is because the C2C platform enables offloading of both computation and communication. CloneDoc benefits of this feature by offloading most of the networking tasks on the peer-to-peer network of clones (e.g. communication with the server and with other users in the network). So, the smartphone uses less its networking interface. This, not only spares cellular traffic to the device, but it also reduces the energy consumption of its battery.

## 6 Lessons Learned and Conclusions

In this work we built the first Amazon Customized Image (AMI) for Android OS: A key tool to get reliable measurements of performance of mobile cloud computing systems. It involved implementing a virtual device driver for Android-x86, disabling OS services unnecessary for the cloud platform, and rebuilding its kernel. By using our Ax86AMI, we were able to get reliable performance of public and private cloud clones of Android phones in terms of CPU and I/O. Then we described Clone2Clone (C2C), a platform that offloads networking on the cloud by realizing peer-to-peer networking of clones of smartphones on the cloud. To the best of our knowledge, this is the first time that distributed computing among smartphones is made possible by exploiting cloud services. We designed the system, implemented it, and tested its functionalities with CloneDoc: A non-trivial application for secure group collaboration. For the testbed we used 16 android smartphones and the most performing clones on both public and private cloud, according to our performance study. Our experiments show that building protocols upon C2C (like CloneDoc) has many benefits:

1. Makes the battery last longer thanks to computation offloading on the cloud;
2. makes the device use the network interfaces less, by offloading communication on the cloud; thus
3. helps offloading the cellular network as the smartphones use less network bandwidth;
4. makes execution of complex operations faster; thus, the usability of the protocol improves;
5. boosts up the performance of old low-performing smartphones making them competitive with newer, more expensive, and more performing ones.

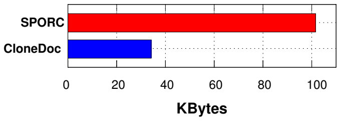

Figure 3: Cellular network traffic with SPORC and CloneDoc.

We believe that C2C opens up a wealth of novel ways to explore the integration of mobile computing, networking, and the cloud. Among others, C2C enables innovative applications such as content sharing and search in the huge amount of data stored by our devices and makes it possible to implement distributed execution of advanced services in a network of mobile smartphones, such as for example the Diaspora[5] social network.

## References

[1] Barbera, M. V., Kosta, S., Mei, A., Perta, V. C., and Stefa, J. CDroid: Towards a Cloud-Integrated Mobile Operating System. In *Proc. IEEE INFOCOM 2013*.

[2] Barbera, M. V., Kosta, S., Mei, A., and Stefa, J. To Offload or Not to Offload? The Bandwidth and Energy Costs of Mobile Cloud Computing. In *Proc. IEEE INFOCOM 2013*.

[3] Barbera, M. V., Kosta, S., Stefa, J., Hui, P., and Mei, A. CloudShield: Efficient anti-malware smartphone patching with a P2P network on the cloud. In *Proc. IEEE P2P 2012*.

[4] Bellard, F. Qemu, a fast and portable dynamic translator. In *Proc. USENIX ATEC '05*.

[5] Chen, E., and Itoh, M. Virtual smartphone over ip. In *Proc. IEEE WoWMoM '10*.

[6] Chun, B. G., Ihm, S., Maniatis, P., Naik, M., and Patti, A. Clonecloud: elastic execution between mobile device and cloud. In *Proc. EuroSys '11*.

[7] Cuervo, E., Balasubramanian, A., Cho, D., Wolman, A., Saroiu, S., Chandra, R., and Bahl, P. Maui: making smartphones last longer with code offload. In *Proc. MobiSys '10*.

[8] Ellis, C. A., and Gibbs, S. J. Concurrency control in groupware systems. *SIGMOD Rec. 18* (June 1989), 399–407.

[9] Feldman, A. J., Zeller, W. P., Freedman, M. J., and Felten, E. W. Sporc: Group collaboration using untrusted cloud resources. In *Proc. OSDI '10*.

[10] Kosta, S., Aucinas, A., Hui, P., Mortier, R., and Zhang, X. Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In *Proc. IEEE INFOCOM 2012*.

[11] Kosta, S., Perta, V. C., Stefa, J., Hui, P., and Mei, A. CloneDoc: Exploiting the Cloud to Leverage Secure Group Collaboration Mechanisms for Smartphones. In *Proc. IEEE INFOCOM 2013*.

[12] Mahajan, P., Setty, S., Lee, S., Clement, A., Alvisi, L., Dahlin, M., and Walfish, M. Depot: Cloud storage with minimal trust. In *Proc. OSDI '10*.

[13] Portokalidis, G., Homburg, P., Anagnostakis, K., and H.Bos. ParanoidAndroid: versatile protection for smartphones. In *Proc. ACSAC '10*.

[14] Shraer, A., Cachin, C., Cidon, A., Keidar, I., Michalevsky, Y., and Shaket, D. Venus: verification for untrusted cloud storage. In *Proc. of ACM CCSW '10* (2010).

[5] http://diasporaproject.org/