SAPIENZA
UNIVERSITÀ DI ROMA

FROM CELLULAR NETWORKS TO MOBILE CLOUD COMPUTING: SECURITY AND EFFICIENCY OF SMARTPHONE SYSTEMS

by

Marco Valerio Barbera

Submitted to the Department of Computer Science

in partial fulfillment of the requirements for the Degree of

DOCTOR OF PHYLOSOPHY in **COMPUTER SCIENCE**

at the

SAPIENZA UNIVERSITY OF ROME

October 2012

Thesis Committee

Prof. Alessandro Mei (First Member)
Department of Computer Science
Sapienza University of Rome, Italy

Prof. Luigi V. Mancini (Second Member)
Department of Computer Science
Sapienza University of Rome, Italy

Prof. Adolfo Piperno (Third Member)
Department of Computer Science
Sapienza University of Rome, Italy

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Prof. Alessandro Mei Thesis Advisor

Approved for the University Committee on Graduate Studies.

External Reviewers

Prof. Edith Ngai
Department of Information Technology
Uppsala University
Uppsala, Sweden

Prof. Mirco Musolesi
School of Computer Science
University of Birmingham
Birmingham, U.K.

The sky above the port was the color of television, tuned to a dead channel.

Acknowledgements

First and foremost I would like to thank my advisor, Professor Alessandro Mei, for giving me the opportunity of working with him. The trust he always put in me has been a great source of motivation for my work and I treasure each and every of his advices. I owe a special thank to Julinda Stefa too. Her always positive attitude and her determination have given me the strength to face head-on the many difficulties I met during my first hesitating steps in the world of academic research.

I would like to thank the members of the internal commitee, Professor Luigi Mancini and Professor Adolfo Piperno, as well as those of the external committee, Professor Mirco Musolesi and Prosessor Edith Cheuk-Han Ngai for their very helpful insights and comments. I owe a lot to my collaborators Aline Carneiro Viana, Marcelo Dias de Amorim and Pan Hui too. Working with them has helped me improving in many different ways.

A special mention deserve Professor Angelos Keromytis, Vasileios Kemerlis, Vasilis Pappas, George Portokalidis and Michalis Polychronakis. They are an exceptional example of how dedication, passion and hard work can bring terrific achievements. I would also like to thank my colleagues and friends at the CS Department for all the special moments I shared with them.

It would have certainly not been able to get to this point without the continuous, unreserved support of my family and closest friends. I will always do my best to try to not disappoint them. Finally, a very special thank goes to Dora for helping, encouraging and sustaining me with love in any situation, and for always staying close to me during these intense years.

Contents

Introduction

In my first year of my Computer Science degree, if somebody had told me that the few years ahead of me could have been the last ones of the so-called *PC-era*, I would have hardly believed him. Sure, I could imagine computers becoming smaller, faster and cheaper, but I could have never imagined that in such a short time the focus of the market would have so dramatically shifted from PCs to personal devices. Today, smartphones and tablets have become our inseparable companions, changing for the better numerous aspects of our daily life. The way we plan our days, we communicate with people, we listen to music, we search for information, we take pictures, we spend our free time and the way we note our ideas has been totally revolutionized thanks to them. At the same time, thanks also to the rapid growth of the Cloud Computing based services, most of our data and of the Internet services that we use every day are just a login-distance away from any device connected to the Internet that we can find around us. We can edit our documents, look our and our friends' pictures and videos, share our thoughts, access our bank account, pay our taxes using a familiar interface independently from where we are. What is the most fascinating thing is that all these new possibilities are not anymore at the hand of technically-savvy *geeks* only, but they are available to newer and older generations alike thanks to the efforts that recently have been put into building user interfaces that feel more natural and intuitive even to totally unexperienced users.

Despite of that, we are still far from an ideal world. Service providers, software engineers, hardware manufacturers and security experts are having a hard time in trying to satisfy the always growing expectations of a number of users that is steadily increasing every day. People are always longing for faster mobile connectivity at lower prices, for longer lasting batteries and for more powerful devices. On top of that, users are more and more exposed to new security threats, either because they tend to ignore even the most basic security-practices, or because virus writers have found new ways to exploit the now world-sized market of mobile devices.

For instance, more people accessing the Internet from their mobile devices forces the existing network infrastructure to be continuously updated in order to cope with the constantly increase in data consumption. As a consequence, AT&T's subscribers in the United States were getting extremely slow or no service at all because of the mobile network straining to meet iPhone users' demand [5]. The company switched from unlimited traffic plans to tiered pricing for mobile data users in summer 2010. Similarly, Dutch T-Mobile's infrastructure has not been able to cope with intense data traffic, thus forcing the company to issue refunds for affected users [6].

Another important aspect is that of mobile security. Around a billion of people today have their personal information on Facebook and half of them access Facebook from their mobile phone [7]; the size of the online-banking in America has almost doubled since 2004, with 16% of the American mobile users conducting financial-related activities from their mobile device [8]; on 2010, customers spent one billion of dollars buying products on Amazon via mobile devices [9]. These numbers give an idea of the amount of people that today could find themselves in trouble by not giving enough care into protecting their mobile device from unauthorized access. A distracted user who loses his phone, or just forgets it in a public place, even if for a short time only, could allow someone else to get unrestrained access to his online identity. By copying the contents of the phone, including passwords and access keys, an attacker could steal money from the user's bank account, read the user's emails, steal the user's personal files stored on the cloud, use the user's personal information to conduct scams, frauds, and other crimes using his name and so on.

But identity theft is not the only security problem affecting mobile users. Between 2011 and 2012, the number of unique viruses and malwares targeting mobile devices has increased more than six times, according to a recent report [10]. Typically, these try to get installed in the target device by convincing the user to download an infected app, or by making them follow a link to a malicious web site.

The problems just exposed are major issues affecting user's experience nowadays. We believe that finding effective, yet simple and widely adoptable solutions may require a new point of view, a shift in the way these problems are tackled. For these reasons, we evaluated the possibility of using a *hybrid* approach, that is, one where different technologies are brought together to create new, previously unexplored solutions.

We started by considering the issues affecting the mobile network infrastructure. While it is true that the usage of mobile connectivity has significantly increased over the past few years, it is also true that socially close users tend to be interested in the same content, like, the same Youtube videos, the same application updates, the same news and so on. By knowing that, operators, instead of spending billions [11] to update their mobile network, could try an orthogonal approach and leverage an *ad-hoc* wireless network between the mobile devices, referred to in literature as *Pocket Switched Networks* [12]. Indeed, most of the smartphones on the market today are equipped with short-ranged radio interfaces (i.e., Bluetooth, WiFi) that allow them to exchange data whenever they are close enough to each other. Popular data could be then stored and transferred directly between devices in the same social context in an ad-hoc fashion instead of being downloaded multiple times from the mobile network. We therefore studied the possibility of channeling traffic to a few, socially important users in the network called ***VIP delegates***, that can help distributing contents to the rest of the network. We evaluated VIP selection strategies that are based on the properties of the social network between mobile devices users. In Chapter 2, through extensive evaluations with real and synthetic traces, we show the effectiveness of VIP delegation both in terms of coverage and required number of VIPs – down to 7% in average of VIPs are needed in campus-like scenarios to offload about 90% of the traffic. These results have also been presented in [1].

Next we moved to the security issues. On of the highest threats to the security of mobile users is that of an identity theft performed using the data stored on the device. The problem highlighted by this kind of attacks is that the most commonly used authentication mechanisms completely fail to distinguish the honest user from somebody who just happens to know the user's login credentials or private keys. To be resistant to identity theft attacks, an authentication mechanism should, instead, be built to leverage some intrinsic and difficult to replicate characteristic of each user. We proposed the ***Personal Marks*** and ***Community Certificates*** systems with this aim in mind. They constitute an authentication mechanism that uses the social context sensed by the smartphone by means of Bluetooth or WiFi radios as a *biometric* way to identify the owner of a device. Personal Marks is a simple cryptographic protocol that works well when the attacker tries to use the stolen credentials in the social community of the victim. Community Certificates works well when the adversary has the goal of using the stolen credentials when interacting with entities that are far from the social network of the victim. When combined, these mechanisms provide an excellent protection against identity theft attacks. In Chapter 3 we prove our ideas and solutions with extensive simulations in both simulated and real world scenarios—with mobility traces collected in a real life experiment. This study appeared in [2].

Another way of accessing the private data of a user, other than getting physical access to his device, could be by means of a malware. An emerging trend in the way people are fooled into installing malware-infected apps is that of exploiting existing trust relationships between socially close users, like those between Facebook friends. In this way, the malware can rapidly expand through social links from a small set of infected devices towards the rest of the network. In our quest for hybrid solutions to the problem of malware spreading in social networks of mobile users we developed a novel approach based on the *Mobile Cloud Computing* paradigm. In this new paradigm, a mobile device can alleviate the burden of computationally intensive tasks by offloading them to a software *clone* running on the cloud. Also, the clones associated to devices of users in the same community are connected in a social peer-to-peer network, thus allowing lightweight content sharing between friends. ***CloudShield*** is a suite of protocols that provides an efficient way stop the malware spread by sending a small set of patches from the clones to the infected devices. Our experiments on different datasets show that CloudShield is able to better and more efficiently contain malware spreading in mobile wireless networks than the state-of-the-art solutions presented in literature. These findings (which are not included in this dissertation) appeared in [3] and are the result of a joint work with P.h.D student S. Kosta from Sapienza University. My main contribution to this work was in the simulation of both the malware spreading and of the patching protocol schemes on the different social networks datasets.

The Mobile Cloud Computing paradigm seems to be an excellent resource for mobile systems. It alleviates battery consumption on smartphones, it helps backing up user's data on-the-fly and, as CloudShield proves, it can also be used to find new, effective, solutions to existing problems. However, the communication between the mobile devices and their clones needed by such paradigm certainly does not come for free. It costs both in terms of bandwidth (the traffic overhead to communicate with the cloud) and in terms of energy (computation and use of network interfaces on the device). Being aware of the issues that heavy computation or communication can cause to both the battery life of the devices [13], and to the mobile infrastructure, we decided to study the actual feasibility of both mobile computation offloading and mobile software/data backups in real-life scenarios. In our study we considered two types of clones: The off-clone, whose purpose is to support computation offloading, and the back-clone, which comes to use when a restore of user's data and apps is needed. In Chapter 5 we give a precise evaluation of the feasibility and costs of both off-clones and back-clones in terms of bandwidth and energy consumption on the real device. We achieved this by means measurements done on a real testbed of 11 Android smartphones and on their relative clones running on the Amazon EC2 public cloud. The smartphones have been used as the primary mobile by the participants for the whole experiment duration. This study has been presented in [4] and is the result of a collaboration with P.h.D. Student S. Kosta from Sapienza University. S. Kosta mainly contributed to the experimental setup, deployment of the testbed and data collection.

Chapter 1

Introduction to Pocket Switched Networks And Properties of Human Mobility

The rapid growth of the market of mobile devices that has characterized the latest years has given rise to a very prolific research field of the so-called Pocket Switched Networks (PSNs), or, more generally, mobile social networks. These are networks composed of mobile devices communicating in an opportunistic manner by means of short-range radios like Bluetooth or WiFi. In the following chapters, we will see how, by leveraging the well known properties of the mobile social networks, we will be able to solve important problems affecting the mobile network infrastructure and the security of mobile users. Here, we will briefly overview what we believe are some of the most interesting results of this field, hoping that this will help the reader understand better our work.

1.1 Pocket Switched Networks

A Pocket Switched Network (PSNn) is a distributed wireless network that emerges from the meeting patterns of people carrying their wireless mobile device (i.e., their smartphone). PSNs belong to the family of Delay Tolerant Networks (DTN) [14]. In Delay Tolerant Networks, messages usually are sent in a store-and-forward fashion from one node to the other, until the message reaches the intended destination(s). This approach allows to exploit communication opportunities that last even for few seconds, thus enabling messages to travel between disconnected, moving portions of the network. In PSNs, the store-and-forward strategy is adapted to human mobility: wireless communication links are created and dropped in time, depending on the physical distance of the device holders. Thus, human mobility creates communicating opportunities that define the very network. Examples of such network include people in working places, students in university campuses, and citizens in metropolitan areas.

The body of the research in PSNs can be roughly divided into three parts. First, it has been necessary to collect data on the meeting patterns of people so as to understand the main properties of human social behaviour. Then, based on the collected data, a number of so-called *forwarding algorithms* have been defined to allow reliable and fast communication between nodes. Finally, new mobility models have been designed with the aim of overcoming the limitations affecting the collected data on human mobility.

1.2 Properties of Human Mobility

With the aim of fully understanding the actual feasibility of the model of communication used in PSNs, researches made numerous efforts in collecting actual data that could shed a light on the main properties of human social behaviour. These datasets were mostly collected by means of experiments in which a set of participants were given small Bluetooth devices (iMotes) able to sense and record their respective presence [12, 15]. Many of these experiments were performed in conference and campus environments with a number of participants varying between 10 and 90 people, and for a period spanning between few days and few weeks. Other data was also collected that, however, used information collected by WiFi access points [16]. This allowed to increase the size of the dataset at the expense of granularity. Thanks these datasets, researches could give precise characterizations of the human mobility patterns leveraged by PSNs. Those that attracted the most of the attention were the *inter-contact time*, the *contact duration* and the *number of contacts*. Together, these properties characterize the latency before a message reaches the destination and the amount of data that can be transferred.

The inter-contact time is defined as the amount of time that passes in-between communication opportunities. The statistical properties of the inter-contact time of the collected dataset has been mainly studied by Chaintreau et al. in [17], by Karagiannis et al. in [18] and by Conan et al. in [19]. Their studies show how the cumulative distribution of the inter-contact times between all the pair of nodes initially follows a power-law but then decays exponentially. Chaintreau et al. also study the relationship between the exponent of the power-law distribution and the expected communication delay.

Barabasi et al. in [20], study the trajectory of a very large number of anonymized mobile phone users whose position is tracked for a six-months period. They observe that human trajectories show a high degree of temporal and spatial regularity, each individual being characterized by a time independent characteristic travel distance and a significant probability to return to a few highly frequented locations. They also show that the probability density function of individual travel distances are heavy tailed and also are different for different groups of users and similar inside each group. Furthermore, they plot also the frequency of visiting different locations and show that it is well approximated by a power law with an exponential decay.

The duration and the number of contacts, instead, are studied by Hui at al. in [21]. The authors show how the social relationship between pairs of users can be classified in four ways according to the number and duration of their contacts. These are *friends*, *strangers*, *community* and *familiar strangers*. The first set contains pairs of users having long but not necessarily frequent contacts; The second group contains people that meet sporadically; The community of a person, instead, is the set of people that person meets in a regular way and for long periods of times (e.g., colleagues); Finally, familiar strangers are pair of people meeting often but for short periods of time.

To further understand the social structure of the nodes, Hui et al. in [22] study how nodes divide into communities. They first build a *contact graph* where there edges are weighted according to either the number of contacts or to the cumulative contact time between the nodes. Then, they run a number of weighted and unweighted community detection algorithms [23–25] on them. Finally, they propose distributed algorithms that do not need to know the whole network topology to accurately detect communities among nodes.

According to their study, complex community structures emerging from the meetings patterns of the nodes can arise even in small networks.

One last important characteristic of the human social behaviour is the wide heterogeneity in the people *centrality*. Broadly speaking, as observed in all social networks, in mobile social network people can have different degrees of sociality: There is very popular people, that meet a great number of people every day, and there is less popular people, that always meet the same small subset of people. This variety of social behaviour has both a positive and a negative impact on PSNs. On one side, popular people happen to be very valuable when it is necessary to send a message to a large set of users. On the other side, not-so-popular people can sometimes be hard to reach, since they may be available for short periods of time only during a whole day. In [26] the authors give experimental data on this matter and show how the number nodes with high centrality is usually very low with respect to the size of the network. They also show how centrality, as opposite to what one might think, is not directly correlated to the number of unique nodes met, but, rather, to the frequency of such contacts.

1.3 Forwarding in PSNs

Achieving reliable and fast communication in PSNs is a hard task. While it is true that, as discussed in the previous section, the underlying social structure of these network produces strong regularities in the meeting patterns of the nodes, human mobility has an unpredictable nature that makes end-to-end communication difficult.

In general, in a forwarding algorithm a message follows a multi-hop path from the source to the intended destination(s). When a node carrying a message meets another node, it has to decide whether give it a *replica* of the message or not. To date, Epidemic Forwarding [27] remains the protocol with the highest performance in terms of success delivery and delay. The success delivery is defined as the percentage of the sent messages that are actually delivered to destination before the message timeout. The delay is measured as the average time of delivery of these messages. Epidemic forwarding is based on flooding: Upon contact between A and B, node A forwards message m to node B unless B already has a replica of m. As it can be imagined, this protocol is inefficient in terms of message replicas created in the network, and network bandwidth. Further work has been done in designing efficient and highly performing forwarding protocols for both DTNs and PSNs. Most of them utilize flooding techniques aiming at approximating the performance of Epidemic by trying to keep a low cost. Spray and Wait [28] limits the number of copies to a number specified a-priori and dependent on the environment.

Other forwarding techniques exploit the "probability of delivering" of nodes. Schemes based on this technique forward messages in a greedy fashion to nodes that are more likely to meet the destination. One of the most representative protocols in this group is SimBet Routing [29], where ego-centric centrality of nodes and their social similarity is used to compute the forwarding quality of a possible relay node. SimBet is very efficient in terms of message replicas created in the system, but also very well performing: It succeeds in delivering messages even when there is a unique time-path between nodes. Unfortunately this routing protocol needs for the nodes to maintain and update routing tables. This make it unfeasible in PSNs where the movement of the nodes is highly dynamic and very unpredictable.

Delegation Forwarding [30] is another protocol that falls under the same group of protocols of SimBet. In Delegation Forwarding, each node is given a forwarding quality that can vary from the overall number of contacts to the time of last contact, and from the contact frequency to the time of last contact with the destination. Similarly, each message is given a rating that initially matches the forwarding quality of the source node. Upon contact between A and B, every message in A with a rating lower than the forwarding quality value of B updates its rating to this value. A forwards to B those messages whose rating is updated, unless B already has a replica. Intuitively, the rating of a replica indicates the value of the highest quality node that that replica has seen so far. Delegation is shown to have a very high performance, similar to Epidemic, yet keeping the cost much lower.

Other forwarding protocols for PSNs are based on techniques that exploit social aspects of node movement such as being part of a community. One of them is Bubble [21]. This protocol bases its forwarding strategy on the popularity of the individuals within their communities and within the whole network. Messages are first forwarded to popular nodes within the network, till they reach the target community. Then, the forwarding continues within the community through the most popular nodes, until delivery to target nodes.

Another interesting approach to forwarding is that of SANE, presented in [31]. SANE is a social aware, stateless forwarding algorithm that exploits people with similar interests, represented as Interests Profiles (IPs). In the paper, the author show how considering people with similar interests of the message recipient is a valuable way to quickly and efficiently route data on PSNs. Moreover, they present a *multicast* forwarding algorithm that allow a user to quickly send a message to the set of all the people that may be interested in its contents.

1.4 Mobility Models

As we already discussed, the whole body of work on the PSNs is based on the data collected during real-life experiments. Unfortunately, these datasets are often limited in size and inaccurate. Indeed, collecting real data is an expensive process both in terms of monetary cost and in terms of the human resources involved (i.e., volunteers). Moreover, hardware failures and distracted users are often the cause of “holes” in the data that represent a tough obstacle to researchers. For instance, it is not uncommon for nodes to disappear from the dataset for long period of times or to report incorrect information. The use of WiFi based traces partially solves these problems, but introduces many inaccuracies that are not easy to smooth out [32].

For these reasons, in the past years, the quest for *mobility models* tailored for the representation of the human social behaviour has attracted the attention of many researchers. The father of these models in undoubtedly the Random Waypoint [33]. In the Random Waypoint model, nodes move in a squared area by first selecting a uniformly random point inside the area and then reaching that point at a fixed speed. Needless to say, this model cannot be used to accurately reproduce the same characteristics of the human contact patterns found in the real datasets. For this reason, Musolesi and Mascolo in [34] proposed a variation of the Random Waypoint model that leverages a pre-existing social network to influence the movement of the nodes. More in detail, the probability that a node choses a given destination is proportional to the amount of friends that are moving towards the same destination. The main drawback of this model, however, is that it is not clear whether it is able to reproduce the critical property, observed in real dataset, that the distribution of the inter-contact times of all the pair of nodes follow, first, a power law, and then an exponential distribution. The SWIM mobility model proposed in [35], on the other hand, has been designed with this aim in mind. Briefly, in the SWIM model, nodes select their destination according to both the its popularity among the other nodes, and according to the distance to the "home" of the node. The speed at which the node approaches the destination, then, is proportional to the distance from the node to the destination. Finally, the amount of time the nodes wait before selecting the next destination follows a powerlaw distribution. SWIM has been formally proven to be able to reproduce the inter-contact time characteristics of the real datasets. Moreover, when tested using on traces generated following the SWIM model, the most important forwarding algorithms show the same performances they have on the real datasets.

We consider SWIM the state-of-the-art in the modelling human social behaviour for PSNs and we will be our reference model for our experiments. Other interesting mobility models for mobile social networks are those presented in [36–38].

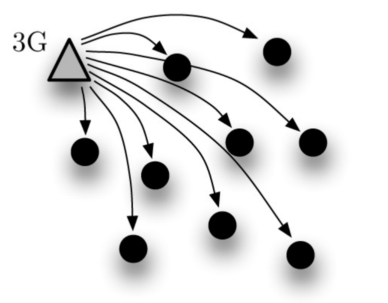
Chapter 2

Data Offloading in Social Mobile Networks Through VIP Delegation

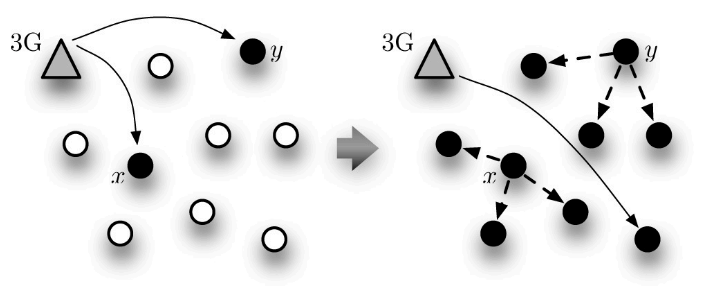
Since the modern smartphones have been introduced worldwide, more and more users have become eager to engage with mobile applications and connected services. This eagerness has boosted up sales in the market and, today, smartphones sales are higher than PCs sales [39]. Simultaneously, smartphone owners are using an increasing number of applications requiring the transfer of large amounts of data to/from mobile devices. Delay-tolerant applications related to social networks [40, 41], global sensing [42, 43], and content distribution [44] are just a few of the examples. As a consequence, the traffic generated by such devices has caused many problems to mobile network providers, forcing them to stop selling unlimited data plans [5, 6]. Despite of that, it can be observed that, in the case of popular data (e.g., news), the redundant traffic on the 3G infrastructure can be drastically reduced through opportunistic offloading (see Figs. 2.1(a) and 2.1(b)).[1]

In this chapter, we propose *VIP delegation*, a solution to this problem based solely on the inherent social aspects of user mobility. Our idea is to exploit a *few, important* subscribers (users) that with their movement and interactions are able to potentially contact on a regular basis all the rest of the network users. These VIP devices would act as a bridge between the network infrastructure and the remaining of the network, each time

[1]There are several other offloading alternatives (through WiFi access points for example, as discussed in Section 2.1). Here we focus only on the case of opportunistic offloading.



(a) Pure infrastructure mode. In this case, the 3G infrastructure sends the same data to all the nodes, generating a lot of redundant traffic.



(b) By using opportunistic offloading, it becomes possible to reduce redundant traffic sent over the 3G infrastructure. Note that in this example the 3G infrastructure sent only three copies of the data (two copies in the beginning and one copy later to cover all the nodes).

Figure 2.1: Leveraging opportunistic offloading to reduce traffic in 3G networks.

large amount of data has to be transferred. Because nodes move and meet each other, it suffices to contact a subset of them to reach a large number of nodes (possibly all of them). In the example of Figure 2.1(b), nodes $\{x, y\}$ can serve as VIPs on behalf of the others, as they meet by themselves the rest of the network.

VIP delegation can help alleviate the network traffic in different delay-tolerant scenarios. Collection of urban-sensing related data [42, 43], distribution of content to users by service providers, and free updates of (lite versions) mobile software's ad pools are examples of applications that would directly benefit from the use of VIPs. As these examples are typical applications that induce large amount of traffic in the network, delegating the collection/distribution of such traffic to a few important nodes would result in immediate data offload.

The question here is *how to compute an appropriate VIP set given some requirements*. For this, we present, formalize, and evaluate two methods of VIPs selection: *global* and *neighborhood* VIP delegation (see Section 2.2). While the former focuses on users that are *globally* important in the network (namely, *global VIPs*), the latter selects users that are important within their *social communities*. The importance of a user within the network is given in terms of well-known attributes such as centrality (betweenness, degree, and closeness) and page rank. In both cases, we observe that a short observation period (one week) is enough to detect users that keep their importance during long periods (several months). Selected nodes are then used to cover the network during a certain time window through solely direct wireless contacts with the remaining nodes (see Section 2.3). In this chapter, we provide significant extension over a companion work by deeply investigating coverage aspects [45].

Through extensive evaluations on real-life and synthetic traces, we evaluate the performance of the global and neighborhood VIP delegation methods in terms of network coverage, by varying the number of VIPs chosen (see Section 2.5). We compare our solution with an optimal benchmark computed from the full knowledge of the system. The results reveal that our strategies get very close to the performance of the benchmark VIPs: Only 5.93% page-rank VIPs against almost 4% of the benchmark VIPs are required to offload about 90% of the network in campus-like scenarios. Additionally we discuss on possible VIP incentives, the way VIPs offload the traffic accumulated to the network, and leveraged applications in Section 2.7. Finally, we conclude with Section 2.8.

The results presented in this chapter appear in [1].

2.1 Network Offloading

So far, the most reasonable solution to alleviate the load on the mobile network infrastructure is that of offloading to alternate networks, such as femtocells and WiFi. Femtocells exploit broadband connection to the service provider's network and leverage the licensed spectrum of cellular macro-cells to offer better indoor coverage to subscribers [46]. As a side effect, automatic switching of devices from cellular network to femtocells reduces the load of the network. Besides from being localized (indoors only), such solution suffers from the non-proliferation of femtocells to subscribers' homes. Moreover, charging users for the necessary equipment as the network providers are currently doing (150 USD for AT&T's microcell) will not help in this direction.

On the other hand, the proliferation of modern WiFi enabled smartphones, together with the network providers' tendency towards already existing technologies has turn WiFi offloading into a reality. More and more carriers worldwide are investing in this direction [47], by installing access points and hot-spots close to overloaded cellular towers, and by providing to clients WiFi access within tiered monthly subscription. In this direction, Balasubramanian et al. propose a system to augment access to 3G network through WiFi offloading [48]. This system, called Wiffler focuses only on Internet access from *moving vehicles*. It leverages delay tolerance and fast switching of devices to overcome the poor availability and performance of WiFi.

Even though offloading to WiFi seems to be the best solution so far to cellular network overloading [49], the continuous increasing of mobile data-traffic demand suggests for integration of WiFi with other offloading methods. Indeed, according to a report released from CISCO on February 2010 mobile data traffic is growing today 2.4 times faster than global fixed broadband one [50]. The mobile data consumed annually is expected to reach 40 exabytes by 2014, more than 90% of which will be driven by laptops' and smartphones' *air cards*. As we will show, our solution is essentially different from WiFi and femtocell-based offloading; nevertheless, it can be integrated to these methods to further help alleviate mobile data overloads.

To the best of our knowledge, Han et al. were the first to exploit opportunistic communication between mobile devices to alleviate data traffic in cellular networks [51]. However, conversely from ours, their solutions only apply to information dissemination problems such as broadcasting. They focus on selecting k target users to which the information is first sent through cellular network. These target users will then, through *multi-hop* opportunistic forwarding, disseminate the information to all users in the network. Multi-hop forwarding is not feasible in our scenario where large amount of data is to be transferred: applications that require collection of sensing data would be costly in terms of energy [42, 43]. Moreover, multi-hop forwarding requires collaboration of all users in the network. Even though such collaboration can be stimulated by incentive mechanisms [52], there is no guarantee on packet delivery. Rather, our solution relies on upgrading a crucial small set of users' devices (down to 5.93% according to experiments with real campus-like data traces), that through direct contact with network members help alleviate the data traffic in both upload and download, assuring that no packet is being lost.

2.2 VIP Delegation in a Nutshell

We propose an offload method based solely on the inherent social aspects of user mobility. Our idea is to detect subscribers (users) that, with their inherent mobility, are able to *encounter* a large number of users (possibly all them) in a regular fashion. These VIP devices would act as a bridge between the network infrastructure and the remaining of the network, each time large amount of data has to be transferred.

As the research in the field of Pocket Switch Networks has shown, our movement ois not random; rather, it is a manifestation of our social behavior. The analysis of such mobility patterns and the understanding of how mobile users meet play a critical role at the design of solutions/services for such kind of networks. Therefore, in general lines, this chapter investigates the following questions:

- How to gain insights into social mobile networking?
- How to utilize such insights to design solutions allowing alleviating the network traffic in the current overloaded 3G networks?

In particular, though the number of network users can be very high, just a few of them have an “important” role within the social graph induced by the encounters. The natural behavior of these VIP nodes, which are considerably fewer than the rest of the network, can be a valuable resource in both information dissemination and collection to/from the rest of the network. Motivated by the fact that opportunities for users to exchange data depend on their habits and mobility patterns, our idea is the following: turn those *few* VIP nodes into bridges between regular users and the Internet, each time large amount of data is to be uploaded/downloaded by these latter ones. In a word, VIPs would act as delegates of the network infrastructure builder. As a side effect, this would immediately drop down the 3G network usage.

In our scenario, we assume that users download/upload large amount of data. This would demand a lot of networking resources and thus, makes the use of multi-hop protocols unfeasible. Indeed, it is quite hard to convince the average user to act as a relay for others, even though to the closest access point, of such an overloading traffic. Rather, our solution relies on the upgrade of a small, crucial set of VIP nodes that regularly visit network users and collect (disseminate) data to them on behalf of the network infrastructure. Such upgrade would serve as incentive to users to play the role of VIPs (see Section 2.7 for discussions on possible VIP incentives). When this happens we say that the *network is covered*. Similarly, when a VIP visits a user, we say that the *user is covered*.

Now the problem becomes the following: *how to choose the smallest VIP set that with their natural movement in the network cover potentially all users in a certain time window?*. More formally, the problem is defined as follows: Let $N = \{n_1, \ldots, n_n\}$ be the network nodes and let $G_i = (V_i, E_i)$ be the graph whose set of vertexes $V_i \subset N$ are the network nodes that have at least a meeting during the time window i, whereas the set of edges E_i represents those meetings, i.e. $\{u_1, u_2\} \in E_i$ iff u_1 and u_2 meet in the time window i. We are looking for $S \subset N$ such that $S = argmin_{S \subset N} |\cup_i [(V_i \setminus (S \cup \{u \mid \{u, s\} \in E_i, s \in S\})]|$. That is, we are looking for the smallest set of nodes S that dominate most of the network during the time window i. Note that set S can be seen as the smallest set of vertexes from N that dominates most of the nodes in each G_i according to the respective E_i. Though it has a dominating set flavor, this problem is different from it: Indeed, here we deal with a series of graphs instead of a singular one.

As previously mentioned, we solve this problem by presenting two VIP selection methods that rely on either a global or a local view of the network (the methods are detailed in Section 2.3). We also present a benchmark solution for VIP delegation. The benchmark provides an optimal selection method that (i) requires the complete pre-knowledge of users' behavior and (ii) is based on an adaptation of the well known NP-hard problem of the Minimum Dominating Set [53]. Such a method is clearly not feasible in real-life applications, but useful to evaluate the goodness of our social-based VIP selection methods.

2.3 VIP selection methods

The selection of VIPs in a social mobile network is based on the ranking of nodes according to their social structural attributes and requires knowledge on their mobility. For this, a social graph describing the tightness of links in the network has to be designed. As the authors of [54] show, the performance of network protocols is strictly related to the goodness of the mapping between the mobility process and the network social graph. They propose an online algorithm that uses concepts from unsupervised learning and spectral graph theory to infer the “correct” graph structure. However, this approach is not applicable in our case, where VIPs are to be predicted. We thus decide to follow a simpler method in detecting the network’s social graph, based on the following intuition: The movement of users guided by their interests generates repeatability in their behaviors (e.g., go to work/school every day, hang out with the same group of friends). Thus, *observing users’ movement and meeting patterns for a short monitoring period reveals enough information to characterize the tightness of the social links in the network graph.*

In a real-life application, we could imagine the network infrastructure builder asking participating users to log their meetings for a certain time, called here as *monitoring period*. These logs serve then to build the networks’ social graph on which the VIPs selection is made. More specifically, from mobility patterns and wireless interactions of users in a network, we establish a *social* undirected graph $G(V;E)$, where V is the set of users and E is the set of social ties (encounters) among them. Note that such social graph is different from each of the $G_i = (V_i, E_i)$ we mentioned in the previous section. Indeed, G_i represents *exactly* who meets who in a certain time window i; whereas G is only a representation of the tightest *friendship* relations among network nodes that appears during the monitoring period, which is composed of a set of time windows i. The details of the construction of the social graph will be given in Section 2.5.3. However, we anticipate that social ties (edges) in the graph $G(V;E)$ are strictly related to users’ contact frequency: A link exists between two users if the number of times they meet is larger than a certain threshold, which depends on the considered networking scenario.

At the following, we present our global and local VIPs selection methods as well as the social structural attributes used at nodes ranking.

2.3.1 VIPs selection methods

Global VIP selection: In the global selection, all network nodes are first ordered according to their importance in the network, determined by their social structural attributes (see Section 2.3.2). Afterwards, the smallest VIP set over the *global social graph* that covers the network during a certain time window through direct contacts is chosen, by applying one of the following VIP promotion methods:

- *Blind global promotion.* It selects the top-ranked nodes not yet promoted, till the network is covered.
- *Greedy global promotion.* This is a set-cover flavored solution. In particular, it starts with promoting to VIP the top-ranked node. After this promotion, the nodes covered by this VIP are dumped and ranking on the remaining nodes are re-computed. Again, the procedure is repeated till the network is covered.

Hood VIP selection: The second strategy, neighborhood VIP delegation, is based on the intuition that repetitive meetings among people happen usually in the same places. The mobile social network generated by this behavior encompasses, besides contact locality, well tight social-community sub-structures. With this in mind, the hood strategy aims to cover each community at a time, independently from other communities. It selects *hood VIPs* by their importance within the communities they belong to. Before doing so, we first detect social-communities using the *k-clique* community algorithm [23]. The reason behind this choice is that the *k*-clique algorithm can detect *overlapping* communities, i.e., nodes may appear to belong to more than one community. This characteristic of the *k*-clique algorithm makes it well suited for a scenario like ours in which people belong to more than one social community (e.g. gym members that are also computer science students, etc.). This is also the reason why the *k*-clique algorithm is widely used in the area of social mobile networking [21,26,55]. Finally, to study the overlapping between communities, we use the Jaccard similarity index [56], which, for two sets A and B is computed as: $J_{A,B} = \frac{|A \cap B|}{|A \cup B|}$.

Afterwards, we rank members of each community according to their importance in the network (see Section 2.3.2). Then, we start covering each community by promoting its members to VIPs similarly to the global VIPs methods:

- *Blind hood promotion.* It continuously selects the top-ranked nodes not yet promoted in the community, till the network is covered.

- *Greedy hood promotion.* The highest-ranked member in the community is promoted, nodes it covers within the community are dropped, and rankings are computed again in the remaining graph.

In both promoting ways, when the whole community is covered, the procedure continues with another one, until all the communities are covered.

2.3.2 Social structural attributes of nodes

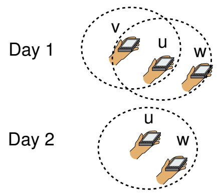
We define the importance of a node in the network by applying to the network social graph several structural *attributes*: betweenness centrality, closeness centrality, degree centrality, and Page rank. All these are well-known attributes in social network theory [57,58]:

Betweenness centrality measures the number of occurrences of a node in the shortest-path between pairs of others nodes. It thus determines "bridge nodes" that, with their movement, act as connectors between node groups (communities). For a given node k it is calculated as: $C_B(k) = \sum_{\substack{j=1 \\ j \neq k}}^{N} \sum_{\substack{i=1 \\ i \neq k}}^{N} \frac{g_{i,j}(k)}{g_{i,j}}$, where N is the number of nodes in the network, $g_{i,j}$ is the total number of shortest paths linking i and j, and $g_{i,j}(k)$ is the number of those shortest paths that include k.

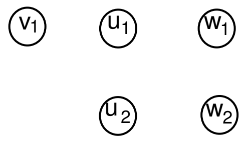
Degree centrality ranks nodes based on the number of their direct ties (i.e., neighbors) in the graph. It identifies the most popular nodes, also called *hubs* in social network theory, possible conduits for information exchange. Degree centrality is calculated as: $C_D(k) = \sum_{i=1}^{N} a(k,i)$, where $a(k,i) = 1$ if k and i are linked, and $a(k,i) = 0$ otherwise.

Closeness centrality ranks higher nodes with lower multi-hop distance to other nodes of the graph. It describes "independent nodes" that do not dependent upon others as intermediaries or relayers of messages due to their closeness to other nodes. The closeness centrality for a node k is calculated as $C_C(k) = \frac{N-1}{\sum_{i=1}^{N} d(k,i)}$, where $d(k,i)$ is the length of the shortest path between nodes k and i. To deal with disconnections it is computed within the subgraph induced by the elements of the connected component to which k belongs.

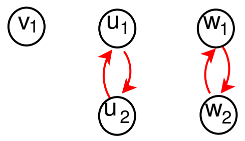
Page rank, the well known Google's ranking algorithm, measures the likelihood of nodes in having important friends in a social graph [58]. In particular, page rank of a node i in the social graph is given by the equation $PR(k) = \frac{1-d}{N} + d \sum_{i \in F(k)} \frac{PR(i)}{|F(i)|}$, where d $(0 \leq d \leq 1)$



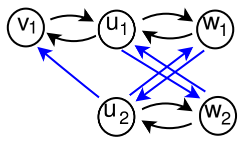
(a) Encounter example.



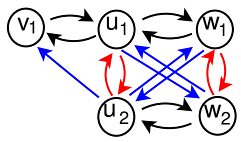
(b) Rule 1 in graph G.



(c) Rule 2 in graph G.



(d) Rule 3 in graph G.



(e) Final graph G.

Figure 2.2: (a) Meeting between u, v, and w during days 1 and 2. (b)-(d) Rules for the construction of graph G. (e) Final representation of graph G.

is the damping factor and $F(k)$ is the set of neighbors of k in the social graph (the graph is undirected). The damping factor d controls the amount of randomness in page ranking: Values close to 1 will give high page rank to socially best-connected nodes.

2.4 Benchmark approach

To evaluate the efficiency of our strategies, we propose a benchmark approach that gives the optimal solution: 100% of user coverage in a time window of one day, with minimum number of VIPs. It is important to underline that the benchmark serves only for comparison purposes, as it requires knowing the future to compute the exact set of VIPs.

2.4.1 Application scenario's abstraction

Suppose the network has to be covered daily (i.e., the time window i is of one day) by VIP delegates, for a period P during which the activity of all network users is known. Let also P be n days long. We construct a directed graph $G = (V, E)$ through the following rules (a step-by-step generation of graph G is illustrated in Fig. 2.2):

Rule 1 *Graph G has a vertex u_i for each day i in which user u is active (i.e., u has at least one contact during the day). This vertex impersonates u during day i in G and is referred to as the clone of u during that day (see Fig. 2.2(a) and 2.2(b)). The set of clones representing user u in G is denoted as C_u.*

Rule 2 *For every user u, C_u forms a clique in G, i.e., each pair of clones u_i, u_j of user u in G is connected by two directed edges, namely, (u_i, u_j) and (u_j, u_i) (see Fig. 2.2(c)).*

Rule 3 *If users u and v meet on day i, then every member u_t of C_u is connected to v_i through an edge (u_t, v_i). Similarly, every member v_t of C_v is connected to u_i through an edge (v_t, u_i). In particular, G also contains edges (v_i, u_i) and (u_i, v_i) representing that u and v met on day i (see Fig. 2.2(d)).*

The graph G constructed with the above rules represents users' behavior in the network during the whole period P. Take for example a certain user u. According to Rules 1 and 2, user u is "expanded" in G into a clique C_u, containing clones of u only for the days u is active (see Fig. 2.2(b) and 2.2(c)). Moreover, if u meets v in day i, Rule 3 guarantees that all members of C_u point to v_i (see Fig. 2.2(d)). The intuition behind this rule is that outgoing edges from $u's$ clique indicate that "u can be a delegate for v on day i".

Rule 3 is applied to every day on which user u is active. As a consequence, all members of the $u's$ clique in G point to the same members of *other users' cliques*. Thus, any clone of user u in G (any member of C_u) is enough to determine the set of users v for which u can be a delegate, and on which days.

2.4.2 Benchmark delegates selection

Intuitively, in order to cover all the network day by day, it is enough to select as delegates the members of a minimum out-dominating set of graph G. Moreover, such a set of delegates is the smallest set that can achieve full coverage. The following theorems prove such intuition.

Theorem 1 *Let MDS be a minimum out-dominating set of G. The set MDS can cover 100% of the active users for each day $i \in P$.*

Proof 1 *First recall that according to Rule 2, the set C_u of clones of a same user u form a clique in G. Since MDS is minimum, it contains at most one clone for every user u. When members of such a set are promoted to delegates, we get at most one delegate-instance per user.*

Suppose, without loss of generality, that user v is active during day i, i.e., $v_i \in G$. As MDS is a dominating set, either of the following cases might happen: (i) some clone v_t of v is in MDS or (ii) there is at least one other node u's clone $u_l \in MDS$ such that the edge (u_l, v_i) is in G. In case (i), since $v_t \in MDS$, v is promoted to a delegate and is covered by itself. Case (ii) can only happen if edge (u_l, v_i) was added by Rule 3, i.e., u and v met during day i. Given that $u_l \in MDS$, u is promoted to a delegate. Thus, v is necessarily covered on day i.

Theorem 2 *Let MDS be a minimum out-dominating set of G. Let also S be the smallest set of VIP delegates able to cover, for every day $i \in P$, 100% of the active network users on day i. Then, $|MDS| \leq |S|$.*

Proof 2 *Suppose, on the contrary, that $|S| < |MDS|$. By construction, and with a similar reasoning used in the proof of Theorem 1 it is easy to see that S is an out-dominating set of G. Then, by the minimum cardinality of MDS, we are done.*

The above theorems indicate how to proceed to find the best possible solution to our problem: after constructing graph G according to Rules 1, 2, and 3, find a minimum out-dominating set of G and use the members of such set as benchmark VIP delegates.

The minimum dominating set is notably a NP-hard problem. Thus, to individuate our benchmark VIP delegates, we reduce our problem to Set Cover (equivalent to *MDS* under L-reductions [53]) for which a simple greedy algorithm is known to be only a logarithmic approximation factor away from the optimum [53]. Moreover, the innaproximability result for this problem shows that no polynomial algorithm can approximate better than $(1 - o(1))\log n$ unless NP has quasi-polynomial algorithms. Thus, there is no polynomial-time

Table 2.1: Details on the datasets and respective monitoring period.

Data set	Taxi	Dartmouth	SWIM-500	D-SWIM-1500	A-SWIM-1500
Total nodes	536	1142	500	1500	1500
AVG active nodes/day (trace)	491	1060	499.98	1500	1500
AVG active nodes/day (monitoring)	429	1061.5	500	1500	1500
AVG contacts/node/day (trace)	7804	292	128	130	380
AVG contacts/node/day (monitoring)	7656	284	131	129	378

algorithm with a smaller approximation factor. The delegates obtained by this heuristic are then used as benchmark VIPs in our experiments.

2.5 Experimental setting: From data-sets to social graphs

We now give detailed information on the data-sets (real and synthetic) we use in the evaluation of our strategies.

2.5.1 Real data-sets

Two real data-sets are used: Dartmouth [59] (movement of students and staff in a college campus) and Taxis [60] (movement of taxi cabs in San Francisco). The vehicular mobility of the cabs is different from human mobility (Dartmouth). However, the purpose of using the taxis trace is to test our solution's extendibility to different contexts.

Dartmouth. Dartmouth includes SNMP logs from the access points across the Dartmouth College campus from April 2001 to June 2004. To generate user-to-user contacts from the data-set, we follow the popular consideration in the literature that devices associated to the same AP at the same time are assumed to be in contact [61]. We consider activities from the 5th of January to the 6th of March 2004, corresponding to a 2-month period during which the academic campus life is reasonably consistent. We chose to work with the set of nodes that have a fairly stable activity in time: at least 500 contacts per week with *any* other device. This results in a set of 1,146 nodes with an average of 1,060 daily active devices and 292 daily contacts in average per device.

Taxis. The Taxi data-set contains GPS coordinates of 536 cabs collected over 24 days in San Francisco. Here, we assume that two cabs are in contact when their geographical distance is smaller than 250 m (following suggestions of Piorkowski et al. [60]). This yields an average of 491 active nodes per day and 7,804 daily contacts per node.

2.5.2 Synthetic data-sets

Synthetic traces are generated using the SWIM model [62], shown to simulate well human mobility from statistical and social points of view. We use SWIM to simulate a 500-node version of the Cambridge Campus data-set (of only 36 Bluetooth enabled iMotes, 11 days long) [62]. We call this trace SWIM-500. It simulates user activity during 2 months, yielding 128 daily contacts per node in average. Then, we scale up to 1,500 nodes in two ways: (i) by keeping density constant (D-SWIM-1500) and (2) by keeping the area constant (A-SWIM-1500). The purpose of the two different scalings is to study the behavior of our strategies in two scenarios: D-SWIM-1500 simulates an urban growing in both area and population and A-SWIM-1500 refers to a sudden over-population of a given city.

Table 2.1 summarizes the details of the data-sets. Note that, although both data-sets represent campus scenarios, they yield different activity per node per day as they used distinct technologies (WiFi capable APs in Dartmouth and Bluetooth-like characteristics in SWIM) in the two data-sets.

2.5.3 Monitoring period and social graph

As we have already discussed in Section 2.3, we observe nodes encounters during an observation/monitoring period and exploit repeatability of users' movement patterns and recurrence of contacts among them. The used length of the monitoring period is not casual: *It is as short as 1 week* and divided in time windows of *1 day*. Usually, our life and the activities we conduct are organized on a week-base, mostly having a common routine repeated day by day (e.g., go to work/school or have lunch in the same place). Such repetition also infers the common meetings generated by those activities.

In the case of the Taxi data-set, the repeatability of contacts is due to several factors including the popularity of geographical zones in the city (e.g., center, stations, and airports), the fixed tracks leading to such zones, and the common city areas covered by groups of taxis. As shown by Piorkowski et al., popularity of areas generates clusters of connectivity among cabs [60]. Taxis' movements are guided by clients' (humans) necessity to reach a specific geographic location. Thus, one week observation is again enough to predict future meetings.

Our intuition on the length of the monitoring period is also confirmed by the results shown in Table 2.1. Indeed, the properties of the monitoring period are very close to the whole trace, for each considered scenario. Hence, this makes prediction of future meetings easy: the monitoring period we have chosen allows us characterizing social relationships. We are then able to generate a *social graph*, where two users are connected only if they have met with a certain frequency – that we call *social connectivity threshold* – during the monitoring period. The social connectivity threshold depends on the scenario considered:

- In the Dartmouth data-set, social connectivity in mostly due to the frequentation of the same classes, or studying in the same library, or living in the same dormitory. All these activities generate lots of meetings among people. We thus set the social connectivity threshold in this case to be at least 1 contact per day, for at least 5 days during a week.

- The social connectivity threshold in the Taxi data-set is higher due to higher speeds: at least 8 contacts per day during the monitoring period were considered.

- As the SWIM-500 trace also represents a University campus, we use the same social connectivity threshold of the Dartmouth trace: at least 1 contact per day for at least 5 days of the monitored week. This leads to a set of 498 nodes. When scaling up with constant density (D-SWIM-1500) the social connectivity threshold remains constant. It increases to at least 8 contacts per day for at least 5 days of the monitored week when scaling up with constant area (A-SWIM-1500).

The social graphs generated by the social connectivity thresholds are then used to individuate the VIP delegates, according to each of the strategies of Section 2.3.

2.5.4 Community detection

Our *hood VIPs* selection strategies operate on a community basis and aim at covering single communities by selecting members that are important within each community.

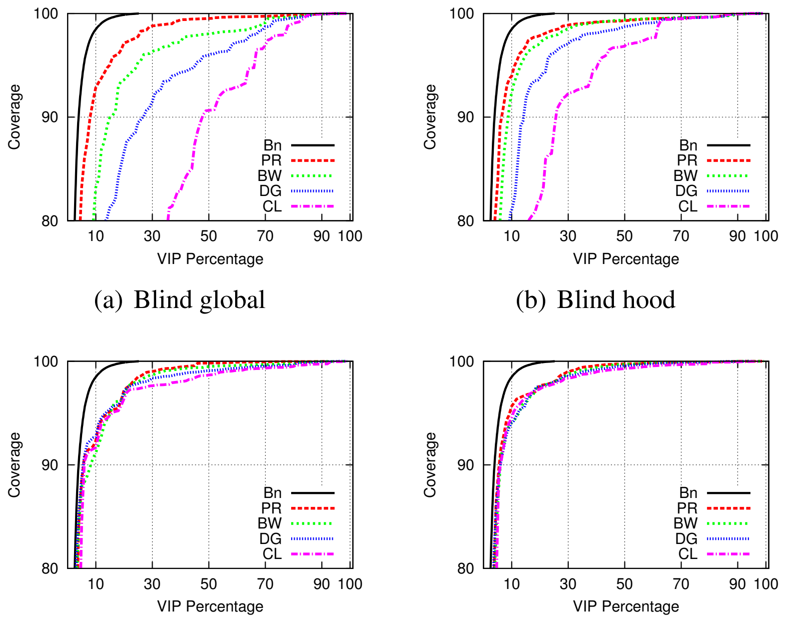
After applying the *k*-clique algorithm [23] to determine the communities and with respect to the campus-like scenarios, we have the following parameters: the Dartmouth data set has 24 communities of 41 members in average, the SWIM-500 trace has 16 communities with 32 members in average, the D-SWIM-1500 trace has 39 communities with 39.6 members in average, and the A-SWIM-1500 has 35 communities with 44 members in average. Note that constant-area scaling yields less, bigger communities.

The communities are well-knit and do not show much intersection between them. Indeed, the average Jaccard similarity index [56] between intersecting communities is 0.038 in the Dartmouth case and about 0.025 in SWIM-500 and D-SWIM-1500 case. This result supports recent findings on universities' communities detected with the *k*-clique algorithm [26]. Conversely, in the constant-area scaling of A-SWIM-1500, the communities have a higher overlapping: the Jaccard similarity index in this case is 0.045.

The Taxi data-set, due to the large number of contacts and the high mobility of nodes, does not present any community sub-structuring. When applying the *k*-clique algorithm, we observe one huge community containing almost 80% of the nodes, whereas the remaining 20% do not belong to any community. Thus, we decided to apply only the global VIP selection strategies to this trace.

2.6 Experimental Results

We analyze the performance of all our strategies in terms of coverage when applied to real and synthetic traces. For better understanding the quality of the VIPs selected by each strategy, we investigate the coverage trend with regard to an increasing number of the VIPs. The set used for coverage is updated from time to time following the order in which each strategy selects VIPs. For the sake of comparison, the results for the benchmark ("Bn") are included in the plots. We use the same technique as above to build the benchmark's trend: updating the VIPs set and the corresponding network coverage, following the order



(a) Blind global (b) Blind hood

(c) Greedy global (d) Greedy hood

Figure 2.3: Performance of the selection strategies on the Dartmouth data-set. “Bn” refers to the benchmark, “PR” to the page-rank, “BW” to betweenness centrality, “DG” to degree centrality, and “CL” to closeness centrality.

in which the benchmark promotes nodes to VIPs. For the page rank attribute, we noted that, varying the damping factor in the interval $[0.51;0.99]$ does not change the performance of page rank VIPs with respect to the VIPs selected according to other centralities. However, we decided to use $d = 0.85$, since, for page rank, it results in the best performance in terms of network coverage.

2.6.1 Results with real data-sets: Dartmouth case

Blind promotion. We show in Fig. 2.3 the coverage obtained by each of the promotion strategies. The *blind* promotion in the global and hood VIP selection strategies yields the results presented in Fig. 2.3(a) and 2.3(b). Notice that there is a coverage efficiency gap between page rank VIPs (referred as “PR” in the figure) and those of other centralities (referred as “BW”, “DG”, and “CL”). In addition, page rank is very close to the benchmark,

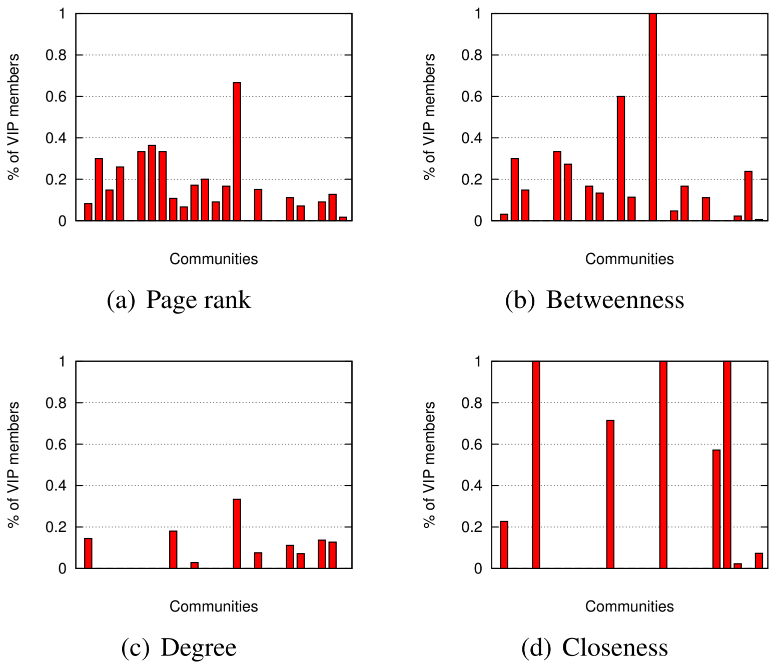
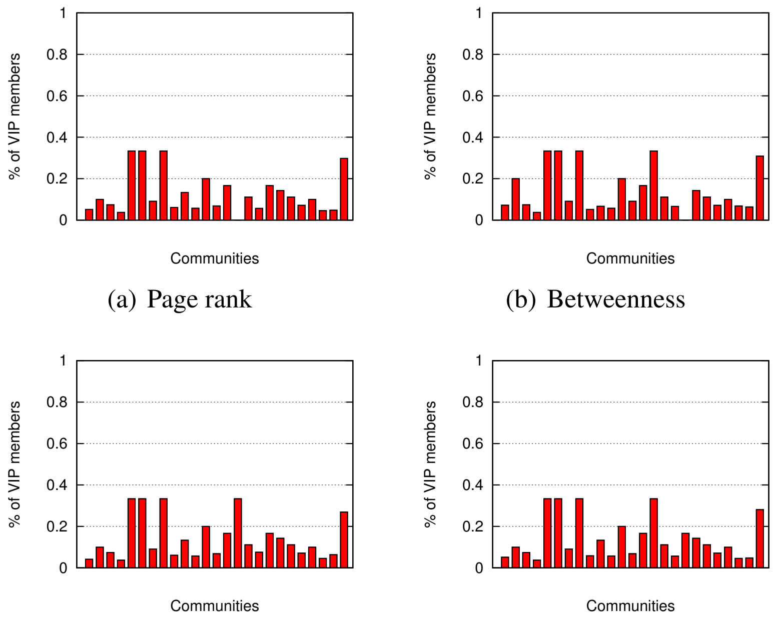


Figure 2.4: Distribution of VIPs per social attribute on the Dartmouth data-set with the *blind global* promotion strategy. The x-axis represents different communities detected.

even for small percentages of delegates considered. For instance, in the *global blind strategy*, to get to 90% of coverage, page rank only requires the promotion of 5.93% of nodes as delegates against 3.92% with the benchmark approach (see Table 2.2). Another consideration to be made is that hood selection is more effective than global selection. Hence, aiming to cover the network by forcing VIP selection within different communities seems to be a very good strategy. Nevertheless, there exist social attributes such as page rank that do not gain much from the hood selection. Indeed, global and hood page rank VIPs perform very similarly in both data-sets. This is because, on the one hand, page rank VIPs already target different communities, even in the global case. On the other hand, betweenness, degree, and closeness centrality tend to over-select VIPs from a few network communities, and consequently, leave uncovered many marginal ones. The tendency of these social attributes to target only a few communities is attenuated with the hood selection that boosts their efficiency in covering the network. In Fig. 2.4, we show how the global strategy distributes

Table 2.2: VIP sets cardinality to get 90% coverage on Dartmouth. The benchmark needs 3.92% of nodes.

	G-Blind (%)	H-Blind (%)	G-Greedy (%)	H-Greedy (%)
PR	8.98	6.89	5.93	6.19
BW	15.96	9.16	8.98	6.19
DG	26.96	15.09	5.93	6.19
CL	47.993	26.0035	5.93	6.19



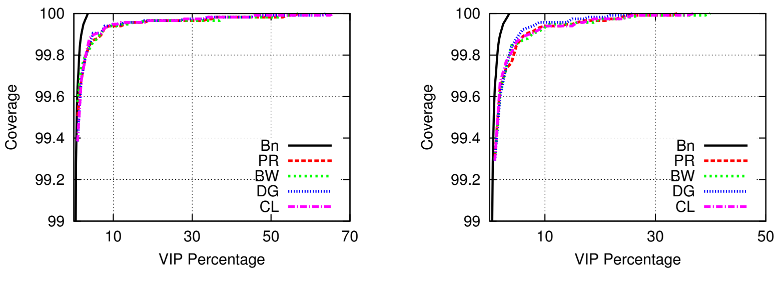
(a) Page rank (b) Betweenness

(c) Degree (d) Closeness

Figure 2.5: Distribution of VIPs per social attribute on the Dartmouth data-set with the *greedy global* promotion strategy. The x-axis represents different communities detected.

VIPs among communities of different centralities.

Greedy promotion. When applying the *greedy promotion*, the performance of all strategies improves considerably (see Figs. 2.3(c) and 2.3(d)). In addition, VIPs obtained with each social attribute perform very similarly to each other, in both hood and global selections. This is due to the capacity of the greedy approach to *not* promote as VIPs nodes that are too close to each other in the social graph. Indeed, after every node's promotion to VIP,



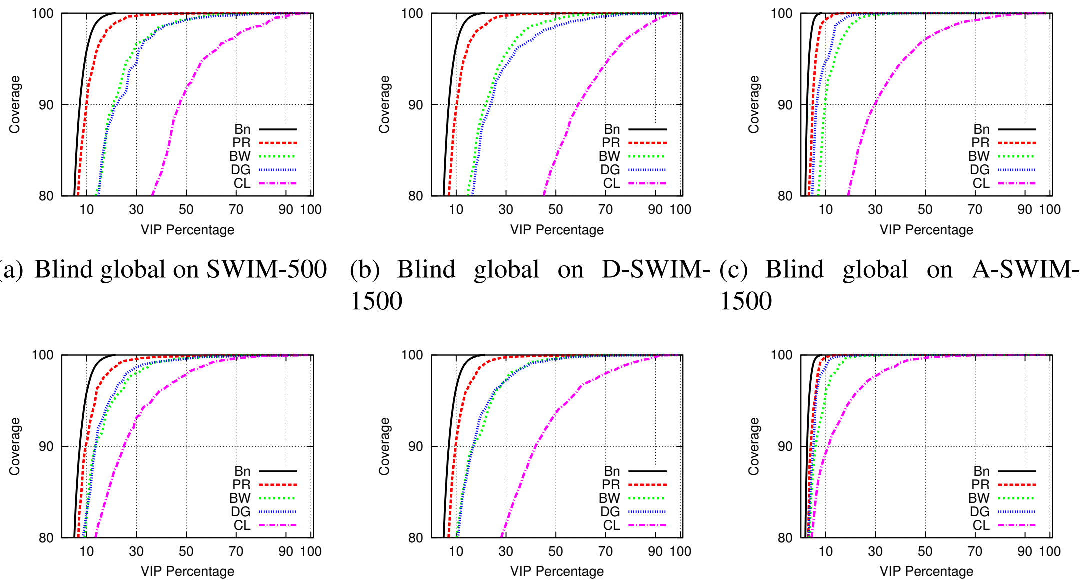
(a) Blind global (b) Greedy global

Figure 2.6: Performance of blind global and greedy global selection strategies on the **Taxi** data-set. "Bn" refers to the benchmark, "PR" to the page-rank, "BW" to betweenness centrality,"DG" to degree centrality, and "CL" to closeness centrality.

all its neighbors in the social graph and their links are removed. Since communities are very well tight, the promotion of one member can remove a large part of the community (if not all of it). Thus, attributes such as betweenness and closeness *do not* concentrate their selection on a few communities as in the global selection. This is also confirmed by Fig. 2.5, where we show how the greedy strategy distributes VIPs among communities for different social attributes.

2.6.2 Results with real data-sets: Taxi case

As discussed in Section 2.5.4, the community sub-structuring of the Taxi data-set is flat. This means that, due to the high mobility of nodes, a huge unique community containing 80% of nodes is detected and the 20% remaining nodes do not belong to any community. Thus, only the global selection strategy is applicable to this data-set. Fig. 2.6 shows the performance of blind and greedy global selection strategies in terms of coverage for the Taxi data-set. As we can see, due to the high mobility of nodes, all strategies perform very well in this scenario. Moreover, the sets selected by each strategy to guarantee up to 90% of coverage are exactly of the same (small) size: Only 0.93% of network nodes. The benchmark guarantees the same coverage with 0.2% of network nodes selected.



(a) Blind global on SWIM-500 (b) Blind global on D-SWIM-1500 (c) Blind global on A-SWIM-1500

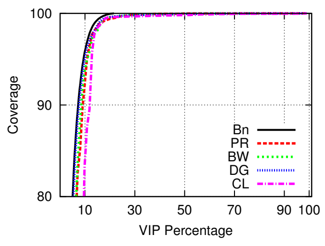
(d) Blind hood on SWIM-500 (e) Blind hood on D-SWIM-1500 (f) Blind hood on A-SWIM-1500

Figure 2.7: Performance of (a)-(c) blind global and (d)-(f) blind hood selection on SWIM. “Bn” refers to the benchmark, “PR” to the page-rank, “BW” to betweenness centrality, “DG” to degree centrality, and “CL” to closeness centrality.

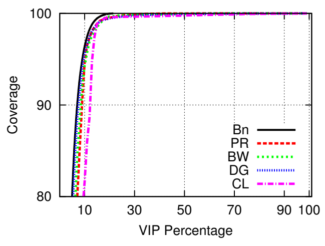
2.6.3 Results with synthetic data-sets: SWIM

As discussed in Section 2.5.2, starting from SWIM-500 (a 500-node simulation of a University scenario [63]), we generate two scaled versions with 1,500 nodes: D-SWIM-1500 (constant density scaling) and A-SWIM-1500 (constant area scaling). Our purpose is to study the reaction of the different strategies in two cases: an urban growing in both area and population (constant density) and a sudden over-population of a city (constant area).

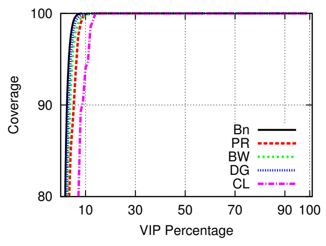
We start from *blind promotion* (see Fig. 2.7). Again, like in the Dartmouth scenario, page rank VIPs are more efficient than VIPs of other centralities. The reason is the same as discussed in the previous section, i.e., page rank global VIPs are better distributed within communities with respect to VIPs obtained with other centralities. For instance, Fig. 2.9 shows such distribution for the trace D-SWIM-1500 (the traces SWIM-500 and A-SWIM-1500 show a similar trend). Once again, aiming to cover the network by forcing VIPs to fall in different communities (hood selection) is a winning strategy.



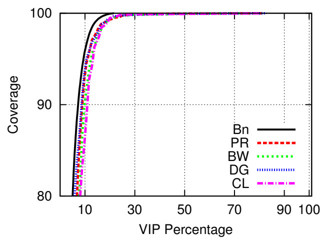
(a) Greedy global on SWIM-500



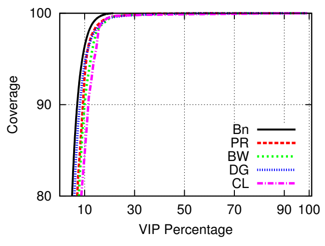
(b) Greedy global on D-SWIM-1500



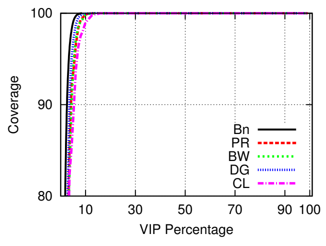
(c) Greedy global on A-SWIM-1500



(d) Greedy hood on SWIM-500



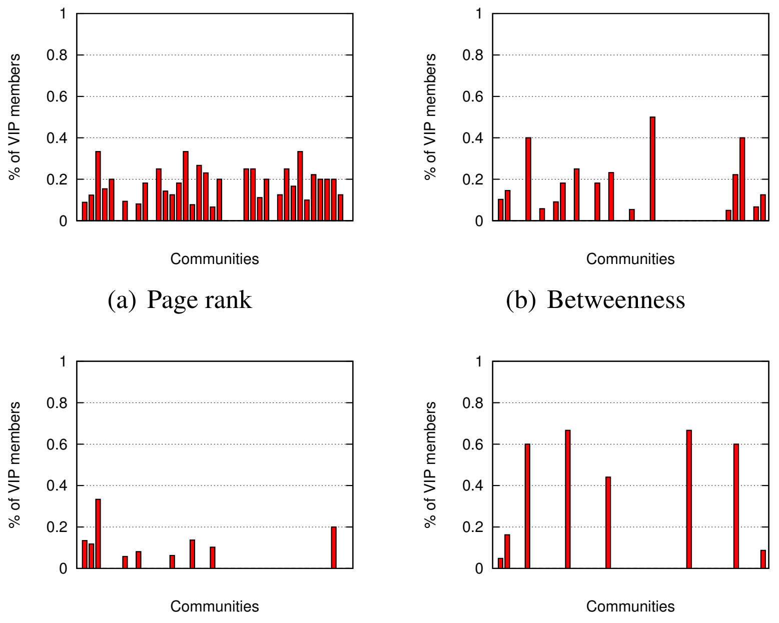
(e) Greedy hood on D-SWIM-1500



(f) Greedy hood on A-SWIM-1500

Figure 2.8: Performance of (a)-(c) greedy global and (d)-(f) greedy hood selection on SWIM. “Bn” refers to the benchmark, “PR” to the page-rank, “BW” to betweenness centrality, “DG” to degree centrality, and “CL” to closeness centrality.

Results related to *greedy promotion* are presented in Fig. 2.8. As in the Dartmouth case, the performance of all strategies is boosted up by the greedy selection of VIPs, yielding a better distribution of delegates within communities (see Fig. 2.10) and thus, much better coverage results with respect to the *blind promotion*. What is interesting to notice here is the impact of the way of scaling in our strategies. When passing from SWIM-500 to D-SWIM-1500 (constant density), all strategies perform very similarly in both blind and greedy promotions. Conversely, in an emergency situation where the network is suddenly much more overloaded as a result of the over-population of the network area (A-SWIM-1500), our strategies perform even better (see Fig. 2.7(c), 2.7(f), 2.8(c), and 2.8(f)). This is also confirmed by the results shown in Tables 2.3, 2.4, and 2.5 that contain, for each data-set, the percentage of delegates needed by the different strategies to cover 90% of the network.



(a) Page rank (b) Betweenness

(c) Degree (d) Closeness

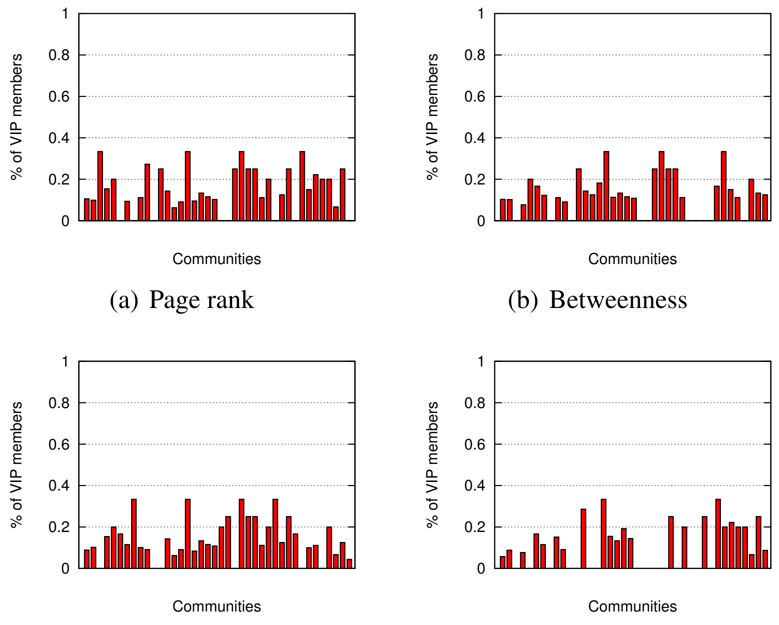
Figure 2.9: Distribution of VIPs per social attribute on the **D-SWIM-1500** data-set with the *blind global* promotion strategy. The x-axis represents different communities detected.

Table 2.3: Delegates given by each strategy to get 90% coverage on SWIM-500. The benchmark approach needs 7.4%.

	G-Blind (%)	H-Blind (%)	G-Greedy (%)	H-Greedy (%)
BW	21	14	9	10.6
CL	48	25.4	12.8	11.6
DG	23	13.6	8	8.8
PR	10.8	10.2	9.8	9.6

2.6.4 Coverage potential

To complete our study, we investigate the *coverage potential* of the first 10% of nodes promoted to delegates according to each strategy. To this end, we measure, for each delegate, the ratio of non-delegates neighbors on the social graph (i.e., the number of non-delegates neighbors of delegate *i* over the total number of neighbors in the social graph). We average then the result over the set of all delegates chosen by the corresponding strategy. Intuitively,



(a) Page rank (b) Betweenness (c) Degree (d) Closeness

Figure 2.10: Distribution of VIPs per social attribute on the **D-SWIM-1500** data-set with the *greedy global* promotion strategy. The x-axis represents different communities detected.

Table 2.4: VIP sets cardinality to get 90% coverage on D-SWIM-1500. The benchmark approach needs 7.06%.

	G-Blind (%)	H-Blind (%)	G-Greedy (%)	H-Greedy (%)
BW	22	17.26	9	9.93
CL	59	42.06	12.93	11.6
DG	24	17.2	8	9.06
PR	10.9333	10.06	9	9.93

the bigger this value, the larger the coverage potential of the strategy, and vice-versa. In Tables 2.6 and 2.7 we present the results for every strategy/social attribute for respectively the Dartmouth and Taxi traces. As a representative of the SWIM traces, on Table 2.8 we show the results obtained on D-SWIM-1500, since, being the least dense of the synthetic traces, it is the most interesting for the study of the coverage potential.

Table 2.5: VIP sets cardinality to get 90% coverage on A-SWIM-1500. The benchmark approach needs 2.53%.

	G-Blind (%)	H-Blind (%)	G-Greedy (%)	H-Greedy (%)
BW	10	6.73	4	4.4
CL	30	11.6	9	6.06
DG	7	5.13	4	3.53
PR	5	4.33	6	4.4

Table 2.6: Coverage potential for each strategy on Dartmouth. The benchmark's potential is 0.91.

	G-Blind	H-Blind	G-Greedy	H-Greedy
PR	0.830	0.069	1.0	0.831
BW	0.657	0.067	1.0	0.969
DG	0.530	0.061	1.0	0.890
CL	0.144	0.059	1.0	0.886

Table 2.7: Coverage potential for each strategy on TAXI. The benchmark's potential is 0.96.

	G-Blind	G-Greedy
PR	0.760	1.0
BW	0.758	1.0
DG	0.742	1.0
CL	0.745	1.0

Note that in the *global blind* selection, page-rank is the one with the highest value, followed by betweenness, degree, and finally closeness. This again supports the results of Fig. 2.3(a). The potential falls drastically when considering the *hood blind* selection (second column of Table 2.6): delegates are forced to be in the same community, in a blind way. Thus, with high probability, they are socially connected with each other. However, page rank remains the attribute with the highest value, supporting the results of Fig. 2.3(b).

The *global greedy* selection naturally yields the highest coverage potential for every attribute: after each node promotion, its neighbors are eliminated from the graph; thus,

Table 2.8: Coverage potential for each strategy on D-SWIM-1500. The benchmark's potential is 0.99.

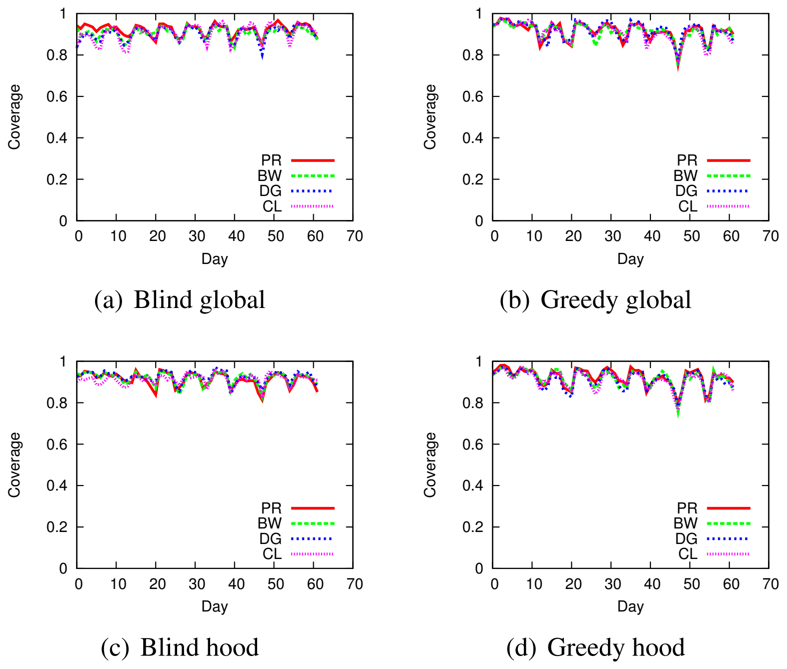
	G-Blind	H-Blind	G-Greedy	H-Greedy
PR	0.888	0.558	1.0	0.992
BW	0.688	0.451	1.0	0.993
DG	0.607	0.288	1.0	0.994
CL	0.180	0.094	1.0	0.993

the ratio of non-delegate neighbors of a node is 1. The *hood greedy* selection (fourth column of Tables 2.6 and 2.8) leads to smaller values. This is because selection is done on a community basis and *only* community neighbors are eliminated after promotion. Since communities are not totally distinct, it might happen that two VIP neighbors in the social graph belong to different communities and, consequently, are eliminated after the promotion, decreasing thus the coverage potential of the strategy. This effect is smaller for high betweenness nodes: they tend to belong to the same group of communities (the ones that they connect). Closeness/degree attributes suffer less from this effect, as they select nodes that are central to communities. Finally, page rank is the one that suffers most: high page rank nodes are well distributed within the community to which they belong (being communities well-knit). Thus, they are more likely to have high page rank neighbors belonging to other communities (that the hood greedy selection does not eliminate).

It is worth to note that the coverage potential just gives a hint on the real coverage power of a method: It does not affect the real ability of the selection method/attribute in covering the network. Indeed, for all traces (see Tables 2.6, 2.7, and 2.8) the coverage power of the benchmark in all traces is less than all the values related to the global greedy selection. Regardless of the coverage potential, the benchmark performs better with respect to every strategy. In addition, the results with 90% coverage presented in Tables 2.2–2.5, confirm page ranks's high performance ability when combined with every strategy.

2.6.5 Coverage stability

Finally, we investigate the stability of coverage of our strategies in time. We focus on the delegates set needed to reach, in average, 90% coverage for each strategy on all tracers. In



(a) Blind global (b) Greedy global

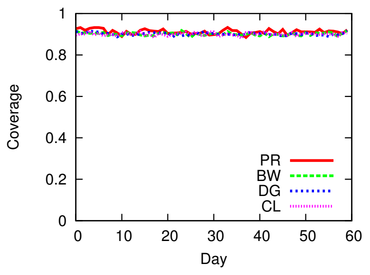
(c) Blind hood (d) Greedy hood

Figure 2.11: Coverage stability in time for all strategies (Dartmouth data-set).

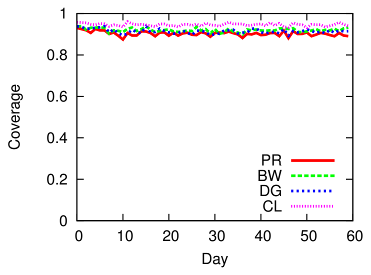
Fig. 2.11 and 2.12, we plot the coverage per day of the Dartmouth and D-SWIM-1500 data sets. We observe that coverage is quite constant in time for every strategy. This reinforces our intuition on both the monitoring period and the way the social graph is generated. With minimal information on the scenario and a very short observation of the network, our strategies are able to compute VIP sets that are small, efficient, and stable in time. The results we obtained with the Taxi, SWIM and A-SWIM-1500 confirm these results.

2.6.6 Coverage intervals vs. VIPs

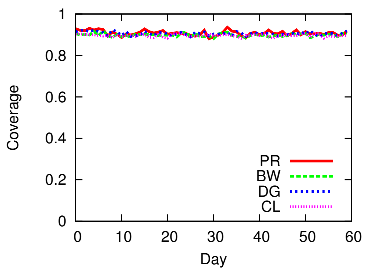
The VIPs selected by our strategies are expected to cover all nodes every day of the data-trace by carrying data traffic from/to the users. All our target applications (e.g., urban-sensing related data, software updates) are delay-tolerant and would not suffer from the 1-day latency of the daily coverage of VIPs. But what happens for applications that require coverage intervals different from 1-day? How does the length of the coverage interval



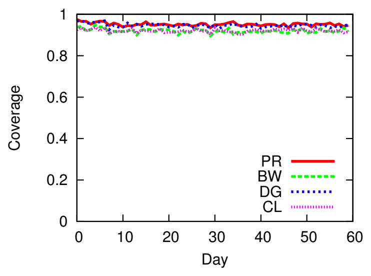
(a) Blind global



(b) Greedy global



(c) Blind hood



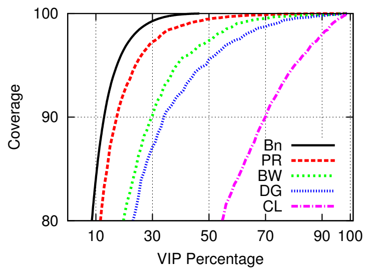
(d) Greedy hood

Figure 2.12: Coverage stability in time for all strategies (**D-SWIM-1500** data-set).

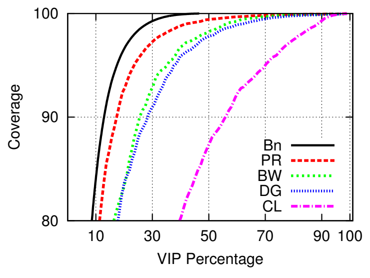
Table 2.9: VIP sets cardinality to get 90% coverage on D-SWIM-1500. The benchmark approach needs 13%. Half-day coverage interval.

	G-Blind (%)	H-Blind (%)	G-Greedy (%)	H-Greedy (%)
BW	30	25.53	15	15.63
CL	70	56.33	15	15.69
DG	34	28.66	15	15.65
PR	18	17.46	15	15.6

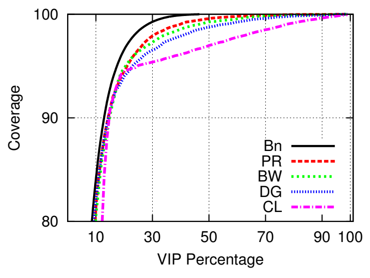
impact the selection of VIPs? Clearly, if the coverage interval is longer, the 1-day coverage VIPs are a superset of the required number of delegates. Indeed, for a coverage interval long e.g. 2 days, the 1-day coverage VIPs would perform as good as in the 2-day coverage case. However, if the coverage interval required is smaller, the VIP set required to cover the network is likely to change. To quantify such change we have also studied the half-day network coverage for all the traces. For both the Taxi and the A-SWIM-1500 traces we noted absolutely no difference from the 1-day coverage case. We believe this is due



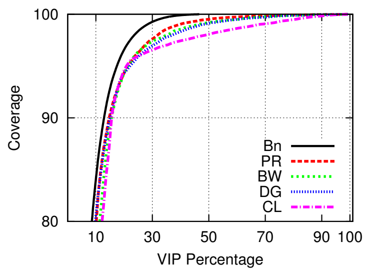
(a) Blind global



(b) Blind hood



(c) Greedy global

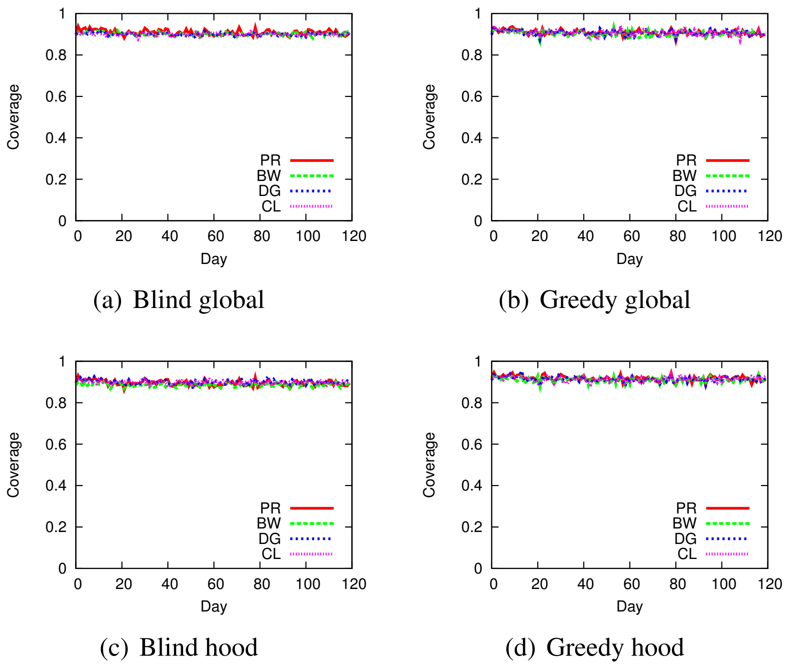


(d) Greedy hood

Figure 2.13: Performance of the selection strategies on the **D-SWIM-1500** data-set. “Bn” refers to the benchmark, “PR” to the page-rank, “BW” to betweenness centrality, “DG” to degree centrality, and “CL” to closeness centrality. Half-day coverage interval.

to the high mixing and speed of movement of cabs in the Taxi case, and due to the high node density in the A-SWIM-1500 trace. Recall that A-SWIM-1500 is obtained scaling SWIM-500 with constant area.

In Dartmouth, SWIM-500 and D-SWIM-1500 we observed a growth in the VIPs number required to assure 90% of network coverage. Intuitively, this is because the meeting patterns of the first half of the day are different from those of the second half. The growth on the number of delegates required to cover 90% of the network in the half-day coverage case is also reflected in the benchmark’s VIPs, which are almost doubled with respect to the 1-day coverage interval case (see Table 2.9). Hence, one would expect that the same should happen also with the VIP sets selected by our strategies. However, this is not the case. For instance, from the comparison of the 1-day coverage interval results with the D-SWIM-1500 dataset of Table 2.4 with the half-day coverage interval results with the same dataset



(a) Blind global
(b) Greedy global
(c) Blind hood
(d) Greedy hood

Figure 2.14: Coverage stability in time for all strategies (**D-SWIM-1500** data-set). Half-day coverage interval.

of Table 2.9 we note that the VIP sets have increased of about 60%. This again means that selecting VIPs according to their "importance" in the network is a good strategy: Indeed, most of the important people during the morning remain so also during evening. However, the coverage interval length indeed does impact the cardinality of VIPs. This suggests for application developers or network infrastructure builders to trade off between data transfer frequency and number of VIPs.

To conclude, Fig. 2.13 and 2.14 show respectively the coverage trend, and the performance of the different selection strategies for the case of half-day interval coverage. Again we note that page-rank wins over the other centralities, and the hood selection strategy wins over the global selection one. Finally, Fig. 2.14 confirms the stability in time of our VIPs, regardless of the length of the coverage interval.

2.7 Incentivizing VIPs and data transfer

Being the human nature inherently selfish, it is more likely that no user would accept the promotion to VIP. However, the number of VIPs selected by our strategies to guarantee 90% coverage is quite low (8% in SWIM, 5.93% in Dartmouth, less than 1% in Taxi). In view of this, VIPs could have their devices upgraded to more fancy, recent ones, and get paid for carrying them around and "working" for the network provider/application builder. Considering the amount of funding that Governments worldwide are putting into global-sensing research [64–66] this incentive is more than real. Another possibility involves considering users' traffic load at the delegates selection and use it to establish a maximum load threshold per delegate. Accordingly, combine it with the social attributes for delegate selection considering fairness and resource constraint among delegates.

It is clear that cellular networks cannot handle the data-traffic from/to VIPs in the classic way (whenever VIPs like), because it would not be of any benefit to offloading. However, the 3G network can still be used in different moments of the day to transfer the data. So, the network load would result distributed in time rather than concentrated in highly congested hours. Another possibility is to transfer the data through wired networks, whenever a VIPs device gets connected to a broadband network during the day. After all, if VIPs are being paid to perform such task, this assumption is more than reasonable.

Another consideration to be made on our strategies is the following: "How to handle the traffic of 10% of network users that remain uncovered by our delegates?" As discussed above, the coverage of 90% of nodes requires the promotion as delegates of very few and constant in time network members. This confirms the advantage of our opportunistic delegation approach for covering a high percentage of nodes in a daily basis. Additionally, we claim that the impact of the few 10% non-covered nodes on the 3G network will be small and typically generated by nodes that are marginal to the network (e.g., people frequenting peripheral areas of a city). Usually, nodes having a high activity or mostly visiting central areas in the network will be represented in the constructed social graph, stressing their frequent encounters. In this way, we believe that such more active nodes will be mostly responsible for the traffic overloading previously mentioned and will be covered by the selected delegates, with high probability. Therefore, to answer the previous consideration, the few remaining uncovered nodes could directly transfer their data using 3G cellular networks, at the end of the day, once no delegate visit was detected.

2.8 Conclusions

Dense metropolitan areas are suffering network overloading due to the data-traffic generated by the proliferation of smartphone devices. In this chapter, we describe VIP delegation, a mechanism to alleviate such traffic based on opportunistic contacts. Our solution relies on the upgrade of a small, crucial set of VIP nodes that regularly visit network users and collect (disseminate) data to them on behalf of the network infrastructure.

VIPs are defined according to well known social network attributes (betweenness, closeness, degree centrality and page-rank), and are selected according to two methods: global (network-based) and hood (community-based) selection. Our observations reveals that 1 week of monitoring period is enough to characterize the tightness of the social links in the network graph. Hence, all methods rely on this network monitoring period and select VIP sets that result small, efficient, and stable in time. Extensive experiments with several real and synthetic data-sets show the effectiveness of our methods in offloading: VIP sets of about 7% and 1% of network nodes in respectively campus-like and vehicular mobility scenarios are enough to guarantee about 90% of network offload. Additionally, the performance of the VIPs selected by our methods is very close to an optimal benchmark VIPs set computed from the full knowledge of the system (i.e., based on past, present, and future contacts among nodes).

Chapter 3

Personal Marks and Community Certificates: Detecting Clones in Wireless Mobile Social Networks

You have left your smartphone on the table at a cafeteria. You soon realize and go back to take it. Fortunately, it is still there! You feel safe—while you are not safe at all. Someone has connected your smartphone to a laptop and dumped all of its memory, including public and secret cryptographic keys. It is a matter of seconds or, at most, minutes. You do not revoke your certificates and passwords (you feel safe!) but, a month later, you discover that your credentials have been used by someone else. If you think this cannot happen—people take very good care of their personal devices—consider that according to a report 478 laptops have been lost or stolen from the IRS (the Internal Revenue Service is the United States federal government agency that collects taxes and enforces the internal revenue laws) between 2002–2006; 112 held sensitive taxpayer data, including SSNs. Furthermore, recent statistics show that in the United States around a thousand phones are lost every hour [67].

Our idea on how to solve this identity *cloning* (or identity *theft*) attack is to leverage social physical contacts, securely collected by wireless radio equipped personal devices (e.g., smartphones), as a biometric way to authenticate the owner of the device: Users prove who they are not with something they know (passwords), or something they have (certificate), but with an unforgeable proof that they actually meet physically and regularly the people they are supposed to meet. Indeed, as the research on Pocket Switched Networks has proved, our physical contacts are not random. Rather, they are highly influenced by the fact that almost everything in our everyday life is guided by social relationships (e.g. friends, family) and interests (e.g. work, school, gym, or playing chess) which yields a regularity in the communities we live in and in circle of friends that we usually meet in person.

We will see how to use this regularity, complemented with other essential mechanisms, to detect the clone attack by using wireless networks of mobile devices and to prevent the misuse of stolen certificates. We will introduce two protocols: Personal Marks and Community Certificates. Personal Marks is a simple cryptographic protocol that provides a way to detect the attack when the adversary is an *insider*, a person that tries to use the cloned identity in the *social community* of the victim. Community Certificates is a solution based on certificates that tells how the owner of a device is expected to behave in terms of social contacts. If the adversary is an *outsider*, someone that wants to use the cloned identity outside the community of the victim, the certificates soon expire, and the cloned identity cannot be used anymore.

The results presented in this chapter appear in [2].

3.1 The clone attack

The detection of the clone attack is one of the most investigated security issue in wireless ad-hoc networks. As far as static wireless networks are concerned, there are three main approaches to the problem: Centralized, local and distributed random based techniques. The centralized techniques like [68] require a base station to collect the location information of the nodes and to check for anomalies (same node ID with different locations). Local-based schemes like [69] make use of voting mechanisms within nodes' neighborhoods to detect clones. Finally, the distributed and random based techniques like [70, 71] require nodes to send signed location information to randomly selected destinations on the network in a hop-by-hop fashion. All these techniques, by relying on fixed geographical position of the nodes in the network are not apt to be used in mobile scenarios such the one we consider [12, 72].

With the spreading of the mobile pay systems (72.8 million of users at the end of 2009, expected 220 million by 2011 in China only [73]), and especially those based on credit card transactions, phone cloning has become a vicious threat to the users portfolio [74]. Thus a lot of research on detecting such attacks have been done, most of which is based on the use of neural networks by the carrier to detect possible anomalies. For a good survey see [75].

The use of biometrics to authenticate has become a well-known area of investigation in computer security. Several biological measurements have been used to identify people to computer systems: voice, face, iris, keystroke dynamics (the way we type is often sufficiently unique to identify ourselves), and others. As an excellent reference and starting point on computer biometrics, see [76].

The authors in [77] propose two intrusion detection systems that have similar biometrics ideas: The first one is built upon Radio Frequency Fingerprinting (RFF), whereas the second one leverages User Mobility Profiles (UMP). However both detection systems are not distributed, and rely on the fact that the intruder (the clone) behaves substantially differently from the real user in terms of geographical movements. Thus, compared with the solutions proposed by us, both systems are not able to detect anomalies when the clone behaves similarly to the original node (for example, when the clone attack happens in a building).

In [78], Chaw et al. propose a framework for authentication based on behavioural information collected by a portable device like a smartphone. There are some key differences with our solution. First, they require an always online Data Aggregator that must be invoked during the authentication phase, while our solution is distributed. Second, they need a Trusted Platform Module to be installed in the device that checks if the device sensing components have been tampered. Third, they are not able to detect an anomaly when the adversary is able to reproduce the behaviour of the victim. Finally, they do not leverage the social characteristics of the user.

In the system presented by Miettinen and Asokan in [79] a device adjusts its access control policies based on the context in which it resides. If the context, defined as the set of devices in range, is not familiar to the device, then the device locks itself and asks the user to type a password or pin to unlock it. On the other hand, if the set of devices in range is mainly composed of known devices, then the device assumes the environment is safe and applies a shallow locking mechanism.

To the best of our knowledge, we are the first presenting biometric authentication techniques based on the social contacts of the owner of the device to detect the clone attack in mobile wireless networks of smartphones. The techniques, namely *Community Certificates* and *Personal Marks*, are *only* built upon the socially-guided meeting patterns of users in the mobile social network. They are thoroughly orthogonal to trust or voting mechanisms, or geography based techniques to detect cellphone theft or credit card fraud, that are based on different ideas and can be used at the same time.

The rest of this chapter is structured as follows: In Section 3.2 we give a detailed description of our system and of the model of the adversary; In Section 3.3 we describe the details of the Personal Marks and Community Certificates protocols; In Section 3.5 we show some experimental results; Finally, in Section 3.6 we discuss some interesting extensions to our system.

3.2 The System

Our network setting is made of last generation smartphones. Smartphones are not-so-small devices that can easily handle video/audio streaming, 3D games, web surfing and SSL sessions, and other applications. Therefore, we can safely assume that nodes are able to perform public key cryptography. The nodes are equipped with public/private key pairs, and the former is signed by a trusted authority CA.

Nodes are loosely time synchronized. Loose time synchronization is very easy to get, if a precision in the order of the second is enough, like in our protocols. We also assume that the trusted authority is able to send a message to any node in the system, for example using the cellular network. When a clone is present, the message is received by the original node and by the clone as well (of course it is perfectly possible that the clone has turned off its interface to the cellular network).

We also assume that the users have access to an alternative way to authenticate to the authority. There are several examples of such mechanisms. One example is GMail: If you forget your password, you can still authenticate by responding to a list of personal questions that, most probably, only you can respond. In other systems, you might be able to authenticate by using a smart-card at your desktop at home. Another example is the PUK code used in GSM mobile phones. In any case, we assume that the alternative mechanism to authenticate is secure but long, burdensome, and we definitely want to use it only in rare and exceptional circumstances like when we need to recover from a clone attack.

From now on, we use $\langle m \rangle_E$ to denote a message m signed by the entity E (e.g. a node or the CA).

3.2.1 The Adversary and the Problem

We consider a scenario in which an adversary has cloned a device by copying the relevant contents of its memory, including passwords, cryptographic keys and certificates. By storing the stolen data on his device, the adversary is now able to interact in the network by using the victim's credentials.

We assume that the victim is unaware of the attack and keeps on using his device in a normal way. This makes the problem more difficult to solve, since, if the user knew that his identity has been cloned, he could renew his keys and passwords to stop the attack.

Our adversary can be an outsider or an insider. An outsider tries to use the cloned identity outside of the victim's community. On the other hand, an insider is interested in using the cloned identity inside the victim's community. We fight outsiders with the Community Certificates subsystem, and insiders with the Personal Marks subsystem.

Our model of the adversary is rather general: He can turn the cloned device on and off at will, refuse to follow any security protocol and try to eavesdrop any data transmitted or received by any other device (if he is in its transmission range). Also, we do not make any assumption on the way he moves. For instance, he could start following the victim or any other node at all times, though this would require some effort.

A powerful adversary may also infect the victim's device with a malware that is able to continuously steal the certificates and the keys from it. We believe that this kind of attack should be tackled with malware detection techniques [80] and we do not address it directly.

3.3 The Protocol

In this Section we are going to give a detailed description of the Personal Marks protocol and the Community Certificates and show how, used together, they make it possible to detect the clone attack in mobile networks of smartphones.

3.3.1 Personal Marks

Personal Marks is a simple cryptographic protocol that can be used by a node to check if it has been interacting with two different nodes claiming to be the same node. Personal Marks is executed each time a node, say node i, meets a node j in its community. The aim of the protocol is to make it possible for node j to realize if there is another node claiming to be node i. Personal Marks is made of three parts: channel-creation, mark-check and mark-exchange.

The channel-creation is done first and it's meant to make it impossible for an adversary that has cloned node i (i.e. has node's i private key) to eavesdrop the data exchanged between the two nodes. The protocol is inspired by the SSL handshake [81]: Node i randomly generates a symmetric key and sends it to node j encrypted with the public key of node j. Starting from this moment until the end of the Personal Marks protocol, the two nodes will encrypt their messages using this key. Since the key is encrypted using node's j public key, an adversary who has cloned node i cannot read it. The adversary, however, could also try to perform a man-in-the-middle attack to make node i believe it is node j and eavesdrop the channel. To make sure that he is using the public key of node j, node i can, for example, use the certificate signed by the CA to node j. Another possibility could be that of making node i and node j exchange keys the first time they meet through either the wireless channel (assuming that an attack doesn't happen in that short moment), or, for example, by means a Near Field Communication [82].

The mark-exchange consists on the two nodes exchanging a *mark*, that is an object in the form $\langle \text{MARK}, t \rangle_{ji}$, signed by both nodes, where t is a timestamp. To exchange the mark, node j first sends $\langle \text{MARK}, t \rangle_{j}$ to node i, and node i replies with $\langle \text{MARK}, t \rangle_{ji}$. Note that if the protocol is not completed for any reason (may be for lack of continuous connectivity, or because one of the two nodes is malicious), the peers can safely assume that it has not happened.

The mark-check is done before the mark-exchange protocol when it is not the first time that node j meets node i. Node j sends a request to which node i has to reply with $\langle \text{MARK\_RPL}, t, m \rangle_i$, where t is a timestamp and m is the latest version of the mark node i received from node j. Note than m was signed by both nodes in the previous contact, so node i cannot forge it. The mark-check continues with node j checking if the mark m received by node i is the same as the mark exchanged during the previous contact with node i. If this is not the case, then node j can be sure that the node i met during the previous contact was not the same as the current one. Therefore, one of them must be a clone of the other.

To understand why this is true, assume that node i has been cloned at time t_1 and that node j gets in physical contact with node i for the first time after the attack at time $t_2 > t_1$. In this case both node i and its clone have the same mark $m_0 = \langle \text{MARK}, t_0 \rangle_{ji}$ in memory, where $t_0 < t_1$. Therefore, independently on whether node i is the clone or the original node, the mark-check protocol doesn't fail, and, at the end of the mark-exchange protocol, node i gets a new mark $m_1 = \langle \text{MARK}, t_2 \rangle_{ji}$. The detection is triggered when eventually node j meets at time $t_3 > t_2$ the *other* node, that, during the mark-check protocol, sends the message $r = \langle \text{MARK\_RPL}, t_3, m_0 \rangle_i$ which doesn't pass the test made by node j. Now, node j cannot tell *which* node is a clone, but it can *prove* that node i has been cloned. The proof consists on the pair of messages r and m_1.The reason this is a proof of the attack is that, in r, node i declares that at time t_3 the latest mark was m_0 (generated at time t_0), but at the same time node j has a mark m_1 (generated at time $t_1 > t_0$) signed by node i that contradicts this declaration.

Note that the messages r and m_1 are signed with the key of node i, thus the proof cannot be forged by node j. This is an important detail because it makes it impossible for an adversary to build fake reports and threaten the honest nodes. This proof can be either sent to the authority by using GSM or broadcasted in the network, and the credentials of node i are thus revoked. Then, the legitimate node i is invited to re-authenticate and get new credentials.

3.3.2 Community Certificates

A community certificate is a cryptographic object used by a node i to prove that he hasn't abruptly changed his social behavior. When node i joins the system, it automatically enters a training period during which it securely collects signed and timestamped logs of the physical contacts with the other nodes. At the end of the period the logs are reported to the authority. The authority uses the logs to build a signed certificate $ComC_i$ that is sent back to node i. All these messages are encrypted and authenticated. The certificate is of the form $ComC_i = \langle\langle FS_i, FI_i, k_i\rangle_{CA}, SU_i\rangle$.

In the certificate, FS_i is the community of node j; FI_i is a mapping that tells the "strength of the relationship" between node i and his best friends. More specifically, for every $j \in FS_i$, $FI_i(j)$ is a value computed as a function of the inter-contact times between nodes i and j observed during the training period. Moreover, for every $j \in FS_i$, $SU_i(j)$ is an object $\langle \text{TIMESTAMP}, t\rangle_j$ signed by node j where t is a timestamp. This object, called *signed timestamp*, certifies at what time there has been the last contact between node i and j. Signed timestamps are similar to the marks of Personal Marks. However, the two objects *cannot* be used interchangeably.

Before sending a signed timestamp to node i, a node j in the set FS_i must first check node i by using the Personal Marks protocol described in the previous Section. Upon completing the Personal Marks protocol with success, node j sends the signed timestamp to node i. This is an important detail we will talk about in a short time.

Node i receives signed timestamps through an encrypted channel created with the same channel-create procedure used in the Personal Marks protocol (it could even be the same channel created during the execution of the Personal Marks protocol). This prevents the adversary with node i's private key to get a copy of the timestamp through eavesdropping or a man-in-the-middle attack.

Given node $j \in FS_i$, we say that the timestamp $SU_i(j)$ is *fresh* at time t if $t < SU_i(j) + FI_i(j)$. Certificate $ComC_i = \langle\langle FS_i, FI_i, k\rangle_{CA}, SU_i\rangle$ is valid at time t if and only if at least k signed timestamps in set SU_i are fresh at time t. In other words, the certificate is valid only if the node has been able to collect enough fresh signed timestamps by physically meeting people in his circle of friends.

When the authority generates the certificate, SU_i can be prepared with timestamps signed by the authority, just to make the certificate immediately valid. Note that all this process is totally automatic. Note as well that we assume that no attack occurs during the training period. This is quite reasonable indeed, if such an attack takes place, the authority would receive training logs from both the victim and the clone at the same time, thus revealing the attack.

3.3.3 Personal Marks and Community Certificates

Personal Marks and Community Certificates, when used together, are able to defend any honest node i from a clone attack. Let us see how: In our system, having a valid community certificate is a requirement for the authentication of a node i to the other nodes. Indeed, when node i wants to set up a communication or to use one of the services provided by a node j, it will first be asked to show a valid certificate $ComC_i$. This applies also in the case when node i needs to get a signed timestamp from node j.

Note that node i should never send the certificate $ComC_i$ directly since this would make it possible for the node receiving it to use node i's identity for the remaining validity time of the certificate. Node i should instead sign and send an object like $\langle ComC_i, t, j\rangle_i$ where t is a timestamp and j is the public key of the node to which the certificate is being sent. This would bind the certificate to both the moment it is being used and to the node that is receiving it, thus making it impossible for any node to reuse it at a later time. Also, the certificate should always be sent through an encrypted channel created in such a way that an adversary who copied node i's private key cannot eavesdrop it (see the channel-create protocol described in Section 3.3.1).

Let us now consider the scenario where an adversary has just cloned node i, getting, along with all the keys stored in the node, the valid community certificate $ComC_i$. By using the certificate, the adversary can interact with any node j of the system pretending to be node i. Now, if the adversary is an insider, then the attack will eventually be detected by node j thanks to the Personal Marks protocol. On the other hand, if the adversary is an outsider, he will be able to use node i's identity only until the certificate $ComC_i$ expires.

The only way an outsider adversary can try to keep on using the cloned identity is to get enough fresh signed timestamps from the nodes in the community of node i and use them to maintain $ComC_i$ up to date. However, as explained earlier, before giving a new signed timestamp, a node j in the community of node i will require the adversary to go through the Personal Marks protocol and, therefore, will detect the attack.

This is the *key idea* behind Personal Marks and Community Certificates: While Personal Marks discourages the adversary from using the cloned identity to interact with the nodes in the community of the victim, at the same time Community Certificates requires him to interact with them. Therefore the adversary will either be detected with Personal Marks or will be able to use the cloned identity for a limited amount of time.

In the experiments with real traces, we show that it is possible to choose FS_i, FI_i, and k in such a way that the certificates are continuously and consistently valid when carried by the legitimate owner and expire quickly when carried by outsiders. Therefore, a combined use of Community Certificates and Personal Marks is a good way to provide efficient and secure authentication and to thwart the clone attack.

3.3.4 Dynamic Community Certificates

So far, we have described Community Certificates as a static system. However, in real life it may be possible, even if it is not common, that we change our own community. In general, it is reasonable to imagine the following scenarios: (i) our community changes completely since, for example, we move to another town; and (ii) one of our friends moves away, or a new node is our new best friend. Here, we see that it is easy to design protocols to dynamically change the community certificate in a secure way.

In case (i), it is enough to start off a new training phase and to get a new certificate. In case (ii), we can initiate a selective update of the certificate to remove one node, or to add a new one, or to update the parameter of a node that is already part of our ring. Of course, the addition and/or the removal can change all the parameters of the certificate, like mapping FI_i or k_i. The procedure can be easily secured. Indeed, when the procedure starts, the authority sends a GSM message to the node. If a clone requests the procedure to change the certificate according to his own communities, then the message is received by the original owner as well, that promptly detects the attack and sends to the authority a signed request of certificate revocation.

3.4 Multiple, Coordinated Clone Attacks

Here we consider the problem of detecting a single clone attack. Generally speaking, if the adversary is very powerful, it is however possible that it clones a whole set of mobile nodes. While we do not explicitly deal with this case in this work, it is still useful to see what is going to happen with Personal Marks and Community Certificates.

Let us start with Personal Marks. An insider adversary who clones more than one node has one main advantage: He can use the stolen private keys either to eavesdrop on the encrypted channels that two victims establish during the Personal Marks protocol, or to make the cloned nodes forge marks to each other. In most of the cases, however, this doesn't stop Personal Marks to detect the attack anyway. The only thing the adversary can do is to clone a node i and a node j in node i's community and make sure that, whenever he uses a clone of node i to communicate with node j, at the exact same time he uses the clone of node j to exchange a mark with node i. This would make it impossible for the two honest nodes to detect the attack since they will store the same mark. Moreover, whenever the adversary wants to repeat this procedure, he will first have to eavesdrop a mark-exchange between the honest node i and node j in order to be sure he has the latest version of the mark.

Though in principle this attack is possible, we think that the amount of effort it requires is big enough to drastically reduce the probability that the adversary performs it.

A similar argument can be made for Community Certificates: By using the stolen private keys, an outsider adversary can either eavesdrop on the encrypted channels that two victims establish to exchange the signed timestamps, or make the cloned nodes forge signed timestamps to each other. This means that if the adversary clones a node i and some of the nodes in its community, he could now get signed timestamps for node i without having to exchange marks with the honest nodes. If the number of cloned nodes in the community of node i is high enough, then the adversary can build a valid certificate $ComC_i$ (Section 3.3.2).

There are two possible ways we can try to fight back this kind of attack. The first one

Table 3.1: Details on the datasets (DS) and training periods (TR).

Data set	Dartmouth	UCSD	Reality	SWIM
Total nodes	1099	32	45	1500
DS AVG active nodes/day	1034	27	37	1500
TR AVG active nodes/day	980	28	38	1500
DS AVG contacts/node/day	283	49	15	132
TR AVG contacts/node/day	263	51	16	131

is to use a fairly big value of k in the $ComC_i$. This would tradeoff between the risk for false positives for the node i and the number of nodes the adversary has to clone in order to generate a valid certificate. A second possibility would be requiring that, to be considered valid, $ComC_i$ has to contain a valid $ComC_j$ for each fresh signed timestamp $SU_i(j)$. This extra requirement, however, shouldn't be applied recursively to the internal $ComC_j$'s so as to avoid the certificate to become too big.

In this way we would make it harder for the adversary to forge a valid $ComC_i$ because, in order to do that, he would also have to forge valid $ComC_j$ for at least k_i of the nodes in FS_i. The adversary would therefore be forced to search for a set of nodes which form a clique and can mutually certificate each other. Considering that, in general, communities are not closed sets of nodes [23], it should be hard find such a clique. While studying this attack is out of the scope of this study, it is certainly a open and interesting research direction.

3.5 Experimental Results

To validate our mechanisms we use four different data-sets. We first start with describing them and their properties. Then, we show the effectiveness of Personal Marks and Community Certificates in detecting insiders and outsiders in all datasets.

3.5.1 Datasets

For the experiments we use an event-driven simulator, fed with 4 traces of three types: WLAN (real)—*Dartmouth* [59] and *UCSD* [83]; bluetooth (real)—*Reality* [84], and *SWIM* [35] simulated traces. In the simulated scenario we are able to select the number of nodes and the length of the experiment. Rather, in the real scenarios (WLAN and bluetooth), the datasets suffer from data loss due to periods of low node activity (e.g. people not always have their devices on), thus we are forced to select a trace period in which nodes are reasonably active.

Dartmouth

It contains SNMP logs of the access points across the Dartmouth College campus from April 2001 to June 2004. To infer the contacts between the nodes we follow the assumption widely used in the literature: two nodes can communicate whenever associated to the same access point [85] [72]. From this trace we have selected a time span of 8 weeks, from January 5, 2004 to March 1, 2004, during which 1099 nodes have at least 50 contacts a day for at least the 80% of the days. This ensures us that these nodes remain fairly active during the whole period.

UCSD

This trace is part of the Wireless Topology Discover project (WTD) [83]. It contains logs extracted from PDAs carried around campus by a set of about 275 freshmen students of the University of California San Diego. The trace spans a period of 11 weeks between September 22, 2002 and December 8, 2002, during which each PDA periodically recorded the signal strength of all the APs in its range. To infer contacts between nodes we again use the assumption that two nodes communicate when they are associated to the same access point. As reported in [85] the trace is characterized by a steady decline of the user population that especially affects the last two weeks. Thus, we restrict our tests to the first 8 weeks and use the nodes that are fairly active during this period by discarding nodes that record no contact at all for more than 20% of the time. This selection yields a set of 32 nodes.

Reality

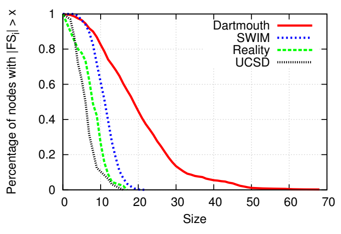
Differently from UCSD and Dartmouth which are WiFi–based, the *Reality* [84] trace contains the *bluetooth* records collected by 94 cellphones distributed to student and faculty on MIT campus during 9 months (from Sept. 2004 to June 2005). This trace is one of the few bluetooth–based existing traces this long encompassing a relatively large node number. Nonetheless, it includes many nodes which have recorded very few to no sightseeings for long periods of time. In order to keep the amount of active nodes high we thus restrict ourselves to a period of 8 weeks (Oct. 18th–Dec. 13th 2004) and discard the nodes that record no contacts (do not appear) for more than 30% of the time. This selection yields a final set of 45 nodes.

SWIM

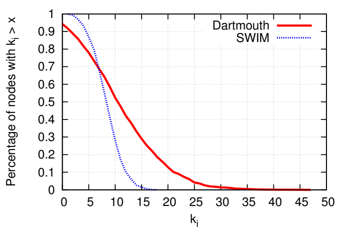
This trace is generated by the SWIM model [35, 86], shown to simulate well human mobility in conference and university campus environments and to properly scale a reference scenario by keeping the nodes density constant. With SWIM we replicate the statistical and social properties of the Cambridge Campus bluetooth data set [87] (only 11 days long) increasing both the number of nodes and the time span while preserving the dynamics of the original real trace in terms of average number of contacts per user: The generated trace contains 1500 nodes and is 8 weeks long. The simulated nodes do not suffer from battery insufficiency or other problems that generate periods of low activity in the network. Thus, with SWIM we use the whole trace with all the 1500 nodes in it.

3.5.2 Training Period

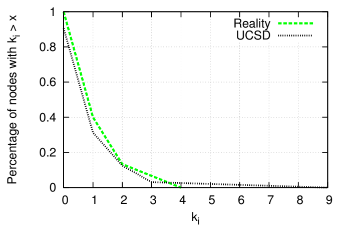
In order to work, Personal Marks and Community Certificates need that nodes “know” their Friends Set (FS_i) and the values FI_i to be put in $ComC_i$ so that the Authority can incorporate this information in the certificates. For this reason nodes go through a *training period* during which they collect logs of contacts with other nodes. In our experiments we use the first part of each trace as the training period, whereas the remaining of the trace is used to test the performance of our security mechanisms. We have experimentally observed that the training period doesn’t need to be long: Only the first 2 weeks of each



(a) Distribution of the sizes of the sets FS_i.



(b) Distribution of the optimal k_i values on Dartmouth and SWIM.



(c) Distribution of the optimal k_i values on UCSD and Reality.

Figure 3.1: Distribution of FS_i sizes and parameter k for all traces.

trace are enough for our system to quickly detect both insiders and outsiders. Table 3.1 includes statistical details of traces and training periods. It is interesting to note that the characteristics of each trace are similar to those of the respective training period, even though the training period is just a subset of 2 weeks of the whole trace.

3.5.3 Selecting the right friends is good for your phone

Our mechanisms are built upon a given node's i friend set (FS_i)—the set of nodes whose signed timestamps are collected by node i—and the respective FI_i telling the "strength of the relationship" between the node and his friends in FS_i. As we anticipated in Section 3.3.2, $FI_i(j)$ defines the period of time in which a signed timestamp given to node i by a node $j \in FS_i$ remains fresh (valid). The choice of FS_i and FI_i is crucial: Large FS_i (too many "best" friends) and big values of FI_i (too long time before expiration) clearly reduce the chances of a honest node to fail proving its identity (the false positive rate). Nonetheless, they also increase the chances that the adversary can use cloned certificates for a long time. Also, a too big FS_i could include nodes that do not have regular contacts with node i. Thus, this would reduce the effectiveness of Personal Marks in detecting an attempt of the adversary to refresh the community certificate of the victim.

That said, it is meaningful to restrict FS_i to the set of nodes that i is likely to encounter more often. More in details, we define the set FS_i as follows: We compute the average number of encounters between i and other network nodes. Then we put j in FS_i if the number of encounters with i is larger than the average value. Simultaneously, we set $FI_i(j)$ to be the average time that usually passes from two consecutive encounters between i and j (their average inter-contact time). In this way, we implicitly include in the certificate i's habitual behavior in the network, that is, his "best friends" and how often he meets them. Note that both FS_i and FI_i are computed taking into account the training period only.

The distribution of the size of FS_i's for all traces is shown in Figure 3.1(a). In Reality, UCSD and SWIM traces, most of the sets' sizes are between 7 and 15. Instead, in Dartmouth, FS_i sets' sizes are a bit larger—between 10 and 30 nodes. This happens for two reasons: (1) Datmouth is more than 20 times bigger than Reality and UCSD, in terms of number of nodes; (2) even though SWIM is bigger than Dartmouth, the longer range of the Wi-Fi communication in Dartmouth (vs the bluetooth in SWIM) induces more contacts among nodes. Even so, we want to stress that the size of the FS_i's in Dartmouth is still much smaller than the total number of nodes in the network (at most 2%), meaning that our selection strategy scales well.

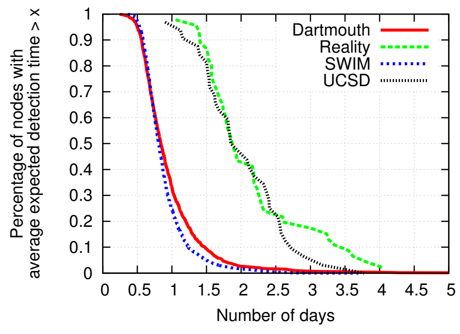
3.5.4 Inter-contacts and detection

Suppose node i has been cloned, and that the adversary is an insider that can easily meet node i's friends. Take, for example, node $j \in FS_i$. Either node i meets j before the clone does, or vice versa. If the former happens, node i exchanges a mark with friend node j thus making obsolete the clone's mark. As a consequence, the next time the clone meets node j it will be detected. Rather, if it is the clone who first meets node j after the cloning and exchanges marks with it, victim node's i mark will result outdated and the anomaly will eventually be detected during the next meet between nodes i and j. Thus, the amount of time the clone can get away inside i's community is bounded by the time-period between two consecutive meets of i with his friends—the inter-contact time.

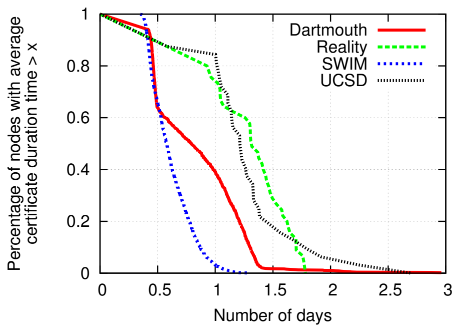
In a similar way, the validity of the certificate $ComC_i$ depends again on the inter-contacts of i and its friends. Indeed, FI_i is the average inter-contact time between i and its friends in FS_i. Overall, the better the friends in FS_i of a node i are selected, i.e., the more frequently i meets them, the faster is the detection of insiders (through Personal Marks) and outsiders (through Community Certificates). In the real-life we would expect that each of us has in his FS work colleagues, family members etc.. People that we meet daily or even more often (e.g. office co-workers). And this is what our mechanisms are based upon. However, the real-traces we use for our experiments are not exactly representing the real life: The people taking part in the experiment might not be friends, thus, might not frequent each other with the same rate as it happens to us with our âĂIJbest friendsâĂİ or family. This is confirmed when looking at the inter-contacts of nodes with their friends (nodes in FS) of the real-life traces. The average of such inter-contact time distribution is little more than 2 days in UCSD and Reality, whereas it is little more than 1 day in Dartmouth and little less than 1 day in SWIM. We would expect then that this is also reflected in the clone detection time in each trace.

3.5.5 Personal Marks vs Insiders

An insider is one that clones a victim i's device and hangs out in victim's community to keep refreshing his certificate. It can attempt to do so at any moment after the cloning. Suppose, without loss of generality, that the cloning happens after the meeting of i with friend node j. As we already discussed, through the Personal Marks mechanism the anomaly will be detected at most after both the clone and the victim meet with j again (no matter in what order this happens). We do not know when the attacker will meet j, nor we do know when exactly the cloning happened (if right after i meets j or later on). However, the clone is interested in seeing j again, to keep renewing the certificate $ComC_i$ (remind that j is one of the nodes in FS_i, and that $ComC_i$ expires as dictated by FI_i–the averages of inter-contacts between i and $j \in FS_i$). So, to test the effectiveness of Personal Marks we measure, for each node i and each $j \in FS_i$, the expected amount of time it will take for them to meet again starting from any point in time in order to count for the possible instants the attacker



(a) Distribution of the average expected detection time of insiders.



(b) Distribution of the average community certificates durations.

Figure 3.2

could meet j, and possible cloning times. Then we average the results over all network nodes and communities. Figure 3.2(a) shows, for all traces, the distribution of the detection time of an insider with Personal Marks. As expected, the detection time is shorter with SWIM, where nodes have, on average, shorter inter-contacts. A little bit longer detection times are yielded by the Dartmouth trace, where again inter-contacts are longer than in SWIM. However, in both traces the detection for more than 90% lasts less than 1 day and a half. Accordingly, UCSD and Reality–the two traces with longer inter-contacts among nodes–yield longer detection times than Dartmouth: less than 2.7 days in UCSD and less than 3.5 days in Reality for more than 90% of nodes.

However, we note that in real life typically inter-contacts with the people we meet more often are even shorter than in SWIM: we meet our office co-workers or family members at least twice a day, vs 1 per day as in SWIM. This makes us think that in a real deployment the clone would be detected much faster.

3.5.6 False Positives

As we already discussed in Section 3.3.2 a certificate $ComC_i = \langle\langle FS_i, FI_i, k\rangle_{CA}, SU_i\rangle$ is valid at time t if and only if at least k signed timestamps in set SU_i are fresh at time t. That is, if node i manages to refresh his certificate by meeting with at least k of its friends. So, the value k is crucial: It determines the trade-off between high detection performance (high

k) and low false positives (low k). We first study the lower bound of the false positives in each trace by setting $k = 1$, the minimum possible value. In our tests we assume that, when the community certificate of an honest node expires, then the user asks the Certificate Authority for a community certificate with a validity of one day. The results of this first test are encouraging. In Reality, only for 8 over 45 nodes we get false positives: The certificate expires once for 4 of them, while for the remaining 4 it expires twice. In UCSD only 3 nodes over 32 show false positives: the certificate expires only once for one node, whereas it expires respectively 2 and 3 times for the remaining 2 nodes. In the case of the Dartmouth trace (1099 nodes) only 63 (less than 6%) show false positives. For all of them, the certificate expires at most 3 times during the whole two months span of the trace. Finally, in SWIM no node at all suffers from false positives.

In Figures 3.1(b) and 3.1(c) we show the distributions of the values k_i that we have found to be the largest we can use on each node without generating false positives. As we can see, the optimal values found for the Dartmouth and SWIM traces are bigger than those found in the Reality and UCSD traces. The reason is that, as already discussed, in Dartmouth we have bigger sets FS_i (more nodes) and the range of the nodes is higher, while, in SWIM, nodes have more contacts and, consequently, more chances to exchange signed timestamps with respect to the Reality and UCSD traces. By comparing the values k_i with the sizes of the corresponding sets FS_i, we have noted that in the UCSD and Reality traces, choosing a value of k_i around the 10% of the size of FS_i is good enough for the 90% of the nodes. In Dartmouth, for 90% of the nodes, k_i can be chosen to be around the 25% of the size of FS_i, while in SWIM it can be set to the 50%. Taking these values into consideration, we believe that a good initial choice of the value k_i could be the 10% of the size of FS_i and, in case this doesn't produce false positives, k_i could be safely raised up to values around the 20% or 25% using a tuning procedure like the one described in Section 3.3.4.

3.5.7 Community Certificates vs Outsiders

After we studied the lower-bound of the false positive rate with the Community Certificates we are ready to study the effectiveness of the mechanism in detecting outsiders, that is, attackers that try to get away with cloning a victim's device by using the device far from the victim's community, where the Personal Marks mechanism would promptly detect the attack. To do so we compute the average expiring time of a cloned certificate (e.g. from a legitimate node i) in absence of contacts with nodes in the set FS_i. This gives us a measure of the expected amount of time the adversary has available to use a cloned identity outside the victim's community. In the experiment, we set k_i to be, for each node, the highest value for which the node is not affected by false positives. The results of our measurements are plotted in Figure 3.2(b). The first observation we make is that in all traces the duration of the certificate is never more than 2 or 3 days at worst (UCSD and Reality, respectively), and roughly between 0.5 and 1.5 days at best (SWIM and Dartmouth, respectively). Again, the detection is faster in traces with shorter average inter-contacts among nodes (SWIM), which makes us think that in real-life the performance of the mechanism would be even better.

3.6 Add-ons and Caveats

Community Certificates can be complemented and extended in such a way to design more sophisticated and complete versions. Here we discuss two of these addons.

Traveling If we travel, especially alone, our community certificate is going to expire soon. While this is generally true, we can easily design a solution. The user can prepare two different *profiles*, in the same certificate, that can be selected when changing community. Of course, it is fundamental that these solutions can be used only by the legitimate owner by performing the burdensome alternative authentication that is used in case of exceptional events.

Complex patterns It is fairly easy to extend Community Certificates in such a way that it is not enough to get k fresh signed timestamps from your circle of friends. Rather, the certificate needs k_1 fresh signed timestamps from one sub-community (for example your family) and k_2 fresh signed timestamps from another sub-community (for example your colleagues) to be valid. A community certificate like this can easily be more efficient and more useful in particular situations. For example when we need at least one signature from one of our bosses to be able to perform critical operations. Or at least one signature from a fixed infrastructure in the building of our company, for example.

Electronic handshakes In addition to standard physical contacts, Community Certificates can be extended to use *electronic handshakes*. Assume that, when meeting a friend, you use a particular secure "handshake" made with your devices (like bumping the two together, for example). Handshakes can be used at the place of physical proximity. In this way, contacts are more trustworthy, but, on the other hand, they need human intervention and are much fewer in number. Alternatively, handshakes could be used to exchange a stronger signed timestamp. By using *complex patterns*, one or more of these handshakes can be required to enable a higher level of authentication to perform critical operations.

3.7 Conclusions

We introduced Personal Marks and Community Certificates. The fundamental idea of Community Certificates is that, in networks of mobile people, authentication can be based on the notion of community, and nodes can authenticate by showing that they indeed meet the people that are part of their community. While the social structure of these networks has been extensively used in networking, to the best of our knowledge this is the first time that this has been used as a biometric to authenticate device. We also present Personal Marks, a way a community can use to protect itself against insiders performing a clone attack. The combined used of these mechanisms deliver an excellent protection of the social mobile network against the clone attack. Indeed our experiments show that the detection is fast enough considered the slow dynamics of the trace we have used. In any real system, we believe that the much faster dynamics of meetings can help deliver much faster detection times.

Chapter 4

Introduction to Mobile Cloud Computing

A topic that has recently attracted the attention of the research community is the integration of cloud computing in the mobile environment, which has given birth to the concept of *Mobile Cloud Computing*. With Mobile Cloud Computing, resource constrained mobile devices can extend their battery lifetime, improve their security, increase their data storage and their processing power by leveraging the cloud computing infrastructure. To better frame the topic of the next chapter here we give a short introduction to the aspects of mobile cloud computing that have been source of inspiration for our work. These include the recently proposed [88] Clone to Clone system. Note that Mobile Cloud Computing is a broad, constantly evolving area of research. We refer the reader to [89, 90] for a more complete overview of the subject.

4.1 Mobile Cloud Computing

The recent explosion of the market of mobile devices has convinced many of us [91,92] that we are shifting from (or have already shifted to) an era dominated by Personal Computers (PCs) to one dominated by portable, wireless devices. Today, that the vast majority of the activities that in the past required us to seat in front of our big, noisy, clumsy computer can be performed in a much more comfortable, intuitive way using our smartphone or tablet. It is therefore not hard to foresee a future where PCs will start disappearing from the houses and the life of the most of the people, replaced with a wide variety of mobile devices.

From the point of view of the programmers and software engineers, however, things have become much worse than before: If a few years ago their software was running on powerful machines with lot of processing power, RAM, hard drive space and with a reliable, high-speed, uncapped Internet connectivity, today they are forced to deal with an environment that has a very limited set of resources available. The resource that is lacking the most in our devices is for sure the battery power, which, in turn, affects all the other components (i.e., clock speed). Moreover, building smaller and faster hardware, that doesn't need an active cooling system, is another serious challenge too. On top of that, people expect their smartphones and tablets to perform always more complex tasks, like gaming and video processing, following the same trend that characterized the PC market in the past.

This increasing gap between the demand for complex applications and the availability of resources on mobile devices is one of the major concerns of the modern tech industry and is pushing the research community into exploring solutions to it. This has encouraged the idea that *computation offloading* should be taken into consideration as a way to increase the efficiency of mobile devices by, at the same time, increasing their battery life. Computation offloading is the practice of migrating parts of a software running on a system (e.g., a mobile device) to a more resourceful, remote system. Computation offloading, however, should not be mistakenly confused with the concept of thin-clients [93] that always offload *all* their computation to some server.

As Kumar explains in [89], the research on computation offloading dates back to the middle or late 90's. Back then, the efforts were focused on trying to make the offloading process feasible, since the wireless technology was still young (less than 2 Mbit/s) and represented a severe bottleneck. As the wireless technology (both WiFi and mobile) improved, the attention shifted to the problem of understanding if, by selecting with care the portions of code to be offloaded, this technique could really solve the problems of today's mobile devices.

But the improvements in the wireless technology are not the only reason why today we can realistically consider solutions based on computation offloading. A major role is also the one played by the cloud computing infrastructure that, with the ability to quickly allocate virtually infinite resources on-demand, and with the heavy usage of virtualization techniques, is perfectly able to play the role of the always reachable, reliable and powerful remote server where the offloading can happen.

In the following sections, we will briefly expose some examples of the most recent advances in the field of the *Mobile Cloud Computing*, and we will conclude with the description of the recently proposed Clone to Clone system [88].

4.2 Computation Offloading

While, in principle, offloading computation to a remote, powerful, server seems like an easy way to overcome many of the limitations of today's mobile devices, in practice it presents several difficulties. The most important one is that sending code and data to be processed to the cloud can be both a slow and an energy consuming process. It is therefore important to understand *when* the offload has to happen and *what* to offload. Offloading a piece of computationally intensive code to the cloud may not be a good idea if, for example, the amount of data that that code depends on is big. In this case, offloading the code may cost the device more energy and bandwidth than running it locally. Also, there are pieces of code that heavily depend on the sensors installed on the mobile phone (e.g., the camera) and that, for this reason, may not be trivial to be offloaded. On the other hand, offloading a piece of code that consumes a lot of memory, or one that does not require much data (like a chess engine) may help saving a considerable amount of battery power to the mobile device.

The authors in [89] identify two main approaches in the literature for the decision of when the offloading should happen. One, is to make these decisions statically, that is, during the development phase. The other is to use a decision engine that, based on the observed conditions of the device (i.e., battery level, available bandwidth, connectivity type and so on), dynamically decides when it is the right time to perform the offloading. Obviously, unless the conditions in which the code will be executed can be predicted with reasonable accuracy, the dynamic method is to be preferred. This is especially true for mobile devices that, sometimes, have slow, intermittent mobile connectivity, while, some other times, can access to a fast and stable WiFi connection.

The process of detecting offloadable portions of the code is generally referred to as *code partitioning*. A typical approach is that of representing the program as a graph where the nodes are the functions to be computed and the edges are the dependencies between them. Partitioning the code, then, can be reduced to a graph partitioning problem.

Recently, Cuervo et al. proposed MAUI [94], a fine grained, energy aware offloading technique based on the .NET Common Language Runtime. MAUI performs dynamic decisions on whether to offload computation by combining network and CPU cost with measurements of the wireless connectivity. The decisions are based on a linear programming formulation of the code offloading problem. MAUI uses programming reflection provided by the .NET environment in order to identify the context needed for the methods to be offloaded with minimal intervention from the programmer's side. Experimental results show that MAUI can both save up to 47% of energy to the mobile device, and improve the performances of mobile applications.

In [95] Chun et al. propose an offloading mechanism named CloneCloud. CloneCloud, as opposed to MAUI, is usable even for applications that were not originally intended to be offloaded because it does not need any support from the programmer. CloneCloud determines the pieces of code to be offloaded by means of an offline static analysis of the binary. Plus, it uses a dynamic profiling to build a cost model for the execution and migration of the computation. In CloneCloud, the code is offloaded to one or more *device clones* running on the cloud, which allows native execution of the code to be offloaded. The authors show how with CloneCloud virus scanning, image search and behavior profiling sample applications can get a $20x$ speed up and, at the same time, a $20x$ reduction in the energy consumed on the device.

In [96], Kosta et al. propose ThinkAir, an offloading mechanism that allows method-level computation offloading with the help of simple annotations on the code. Like MAUI and CloneCloud, to decide whether a method should be offloaded for execution to the cloud, ThinkAir uses a runtime mechanism which considers execution time and energy consumption together with environmental parameters like, for example, the network state. ThinkAir creates virtual machines running the same device's operating system (i.e., Android) and allows parallel execution of the code on the cloud.

These are just three examples of the most recent studies about computation offloading. Other interesting systems are Cloudlets [97] and the Virtual Smartphone over IP proposed in cite [98]. For a more complete review of this field, see [89].

4.3 Data Storage

Storage capacity is another limitation of today's mobile devices that can be alleviated thanks to the Mobile Cloud Computing paradigm. Indeed, today's fast wireless connectivity enables mobile devices to store and access relatively large data on the cloud without using their local memory. In [99], the authors propose ImageExchange, a cloud based service that allows users to upload and share the photos immediately after captured them with their mobile device(s). Flickr [100], Facebook [101], Google Music [102] and iCloud [103] are vary popular services that, based on the mobile cloud paradigm paradigm, allow users to store on the cloud the personal data they have on their phone, thus saving precious space on their device.

Also, systems that regularly send user's data to remote servers for backup are very valuable. The system proposed in [104], besides from backup/restore, also allows for sharing information in smartphones among groups of people. The authors test their system in terms of time needed (on the phone) to backup three different data types: SMS, calendar events, and contacts. In [105] the authors argue that not only contacts and emails—synced by e.g. Google sync on Android OS—but also application settings, game scores etc., are important to users. With this in mind they build ASIMS, a tool that has the goal of providing a better application settings integration and management scheme for Android mobiles. ASIMS is based on SQLite, it stores other applications' settings and syncs them to the Internet. Its interface makes it possible for other applications to store settings in one common place and for users to select which applications they want to sync.

An interesting example of how mobile cloud computing can simplify the managament of data stored on mobile devices can be found in [106]. The authors present Dessy, a mobile search and synchronization framework inspired by desktop search applications. Dessy manages synchronization of data and index files between all the mobile devices of a user and the cloud, and between the devices themselves. By offloading the indexing and ranking of the data on the cloud, Dessy allow mobile devices to save both energy for creating the index, and energy that could be required to download supplementary data from the Internet.

4.4 Security

The Mobile Cloud Computing paradigm can also help improving the security of mobile devices. Indeed, the constrained resources of mobile devices would not make it feasible for an antivirus software to continuously scans for viruses on downloaded files or to monitor the behaviour of the applications installed. This makes the cloud a perfect place where these checks could be performed with a minimal energy expense from the side of the device. Two interesting examples are the system proposed by Oberheide et al. in [107], and ParanoidAndroid, presented in [108]. Oberheide et al. present an approach to move the threat detection capabilities to the cloud. Their system is an extension of the cloud AV platform that provides an in-cloud service for malware detection. The main advantage of their solution is that moving the detection to the cloud enables the use of multiple antivirus engines in parallel. On the other hand, in ParanoidAndroid, execution traces of the apps on the device are recorded and transmitted for analysis on the cloud. The authors show how this paradigm not only improves detection accuracy, but also saves the battery lifetime up to 30%.

4.5 Clone to Clone

In [88] the authors propose the Clone to Clone (C2C) platform, which extends the Mobile Cloud Computing paradigm by developing a distributed platform for cloud clones of smartphones. Clone to Clone associates each smartphone to a software clone on the cloud (e.g., Amazon EC2) and interconnects the clones in a peer-to-peer fashion using the networking service of the cloud. The peers (clones) of the C2C network clearly do not suffer of battery limitations, unlike their mobile alter-egos since they are running on the cloud. Indeed, a wireless P2P network between smartphones would be very expensive to realize, as well as would it suffer from continuous on/off of devices, due to battery insufficiency or temporary loss of cellular coverage or Internet connection. Such problems are mitigated by the C2C platform, where the clones are always on and peer-to-peer connectivity lies upon the high-bandwidth network of the cloud. Not only can the C2C platform help offload heavy mobile computational tasks by adopting techniques similar to MAUI, CloneCloud and ThinkAir, but it also enables innovative services such as content sharing, search, and distributed execution among the C2C network users.

As a proof of concept of the functionalities of Clone to Clone, the authors implement CloneDoc and CloneBox. CloneDoc is a real-time collaboration system for smartphone users that work simultaneously on the same document. CloneDoc is structured as p2p-like application that makes use of heavy crypto primitives and communication among peers. On the other hand, CloneBox, is a system that provides smartphone users with a service similar to DropBox. With CloneBox users can decide which files/directory to share with whom. The files are actually pushed to the clone of the mobile and shared with the other authorized users connected on the C2C platform. In their work, the authors show how CloneDoc and CloneBox can execute on smart-phones by using a very limited amount of energy thus demonstrating that C2C enables a whole new class of distributed applications on mobile devices.

Chapter 5

To Offload or Not to Offload? The Bandwidth and Energy Costs of Mobile Cloud Computing

The more eager we get when using our smartphones by installing new apps, the less happy we are with the lifetime of the battery. The problem is that we rely upon a number of crucial pieces of information that are only stored in the device (phone numbers, addresses, notes, appointments, etc.), or, in some cases, that can be got only by using the Internet on the fly as many of us are used to do. It is so important to keep our smartphone operational that everyday we pay attention to our battery and try to save it by reducing the number of phone calls, or by avoiding to watch too many videos, just enough to be able to reach home and recharge it. But that means that we cannot use our device to the fullest.

As discussed in the previous chapters, Mobile Cloud Computing seems an excellent candidate solution to help reduce battery consumption of smartphones, as well as to backup user's data. Indeed, recent works have focused on building frameworks that enable mobile computation offloading to software clones of smartphones on the cloud as well as to backup systems for data and applications stored in our devices. Both mobile computation offloading and data backup involve communication between the real device and the cloud. However, this communication does not come for free, in terms of both energy consumption (utilization of network interfaces to send the data) and bandwidth. None of the many previous works related to mobile cloud computing explicitly studies the actual overhead in terms of bandwidth and energy to achieve full backup of both data/applications of a smartphone, as well as to keep, on the cloud, up-to-date clones of smartphones for mobile computation offload purposes. In this chapter we address both issues, and, for the first time (to the best of our knowledge), we provide results with a real testbed of 11 Android powered smartphones and associated clones running on the Amazon EC2 platform. This work, appeared in [4], is the result of a collaboration with P.h.D. Student S. Kosta from Sapienza University.

5.1 Energy efficiency of Mobile Cloud Computing

The mobile cloud computing paradigm has, among its main objectives, that of saving mobile devices' battery life. For this reason, in literature, there are a number of studies that try to give at least an approximate answer about whether there can be an actual energy gain for mobile devices in performing computation offloading. Analytical models have been proposed, for instance, in [109–111]. In [109], the authors give a formula that expresses the amount of saved energy in terms of factors like the number of instructions to be offloaded, the computational power of the mobile device, the available bandwidth and the energy required to send the data to the cloud. In [110] the authors present an analytical study to find the optimal execution policy. This is identified by optimally configuring the clock frequency in the mobile device to minimize the energy used for computation and by optimally scheduling the data rate over a stochastic wireless channel to minimize the energy for data transmission. By formulating these issues as a constrained optimization problem they obtain close-formed solutions which give analytical insight in finding the optimal offloading decision. The authors of [111] discuss on the main factors that affect the energy consumption of mobile apps in cloud computing, and deem that such factors are workload, data communication patterns, and usage of WLAN and 3G. However, these works are limited to consider best case scenarios—ideal WiFi connectivity and WiFi interface always switched on.

In [112] the authors argue that previous frameworks that enable offloading are inaccurate in estimating the energy of a certain workload in the system. This is because they do not operate at kernel level. With this in mind they present AppScope, an Android-based energy estimator implemented as a kernel module that uses an event-driven monitoring method. By operating at kernel level their framework yields energy consumption estimation values that are very close to the ones obtained through the Mobile Device Power Monitor[1], a tool used also in other works in the area [94, 113].

To the best of our knowledge there is no study on the cost of keeping updated a clone or a backup on the cloud in a real setting with real mobile smartphone users, where Internet connectivity is often guaranteed by WiFi only at home and at work, by 2G/3G when moving outside these areas, and where coverage is often not present (think of subways, for example).

5.2 Off-clones and Back-clones

In our study we assume an architecture as Clone to Clone [88], CloneCloud [95] and ThinkAir [96], presented in Chapter 4, where each real device is associated to a software clone on the cloud. For the mobile computation offloading to work, the status of the applications running on the clone needs to be in sync (as accurately as possible) with the ones running on the real device. In order to allow users to offload mobile computation on the fly, the clone must run the same applications the real device is running. In addition, also the state of both the real application (the one running on the device) and the cloned application (the one running on the clone), has to be the same. We will refer to a clone that is used to offload computation on the fly as an *off-clone*, and to a clone that is used to backup user's data and to enable restore as a *back-clone* (backup clone). Since the goal of the back-clone is to restore user's data and the system (including installed apps) in case of system/data corruption or loss, it does not need to keep up with any single application's status in time, as in the case of the off-clone. Yet, it needs to be up to date with all sort of data generated/received by the user like notes, pictures, videos, contacts, calendar entries, messages, emails etc.

Our goal in this work is to study the feasibility of both mobile computation offloading and mobile software/data backups in real-life scenarios. We give a precise evaluation of the

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

feasibility and costs of both off-clones and back-clones in terms of bandwidth and energy consumption on the real device. Our contribution is as follows:

- We study the network availability (2G/3G, WiFi etc.) as well as the signal quality in a real testbed of mobile smartphone users, with reference to the requirements of offloading and backup on the cloud;
- we study the data communication overhead that is necessary to achieve different levels of synchronization (once every 5min, 30min, 1h, etc.) between devices and clones in both the off-clone and back-clone case;
- we report on the costs in terms of energy incurred by each of these synchronization frequencies as well as by the respective communication overhead.

To achieve all the above we design and build *Logger*, an Android app that runs in the foreground and collects data on the utilization of the device. Logger also handles the communication between the real device and the cloud. We run Logger on a testbed of 11 smartphones (associated to 5 different carriers) that make use of clones running on the Amazon's EC2 cloud platform.

5.3 The Logger

Studying the overheads incurred by the off-clones and the back-clones is not easy. It depends on how user actually uses her device, on the properties of the network technologies (WiFi/3G/ etc.) and on which of these technologies is actually available and used. So, we develop Logger, an Android app that continuously logs the events occurring in the device. These includes user and system generated events. Examples of user generated events are: Mail sending and receiving, phone calls (both incoming and outgoing), files exchanged over blue-tooth, device switching on and off, battery charging, installing and uninstalling applications. Examples of system generated events are: Access to network interfaces (e.g. regular heartbeat pings to Google's servers), access to GPS radio (e.g. Google Latitude or Facebook), generation and editing of internal files by apps, automatic updates, and so on.

The Logger service is structured into sub-components, each responsible for logging data coming from a given resource. While some information about the device state can be logged passively by our Logger using standard facilities provided by the Android OS, collection of other statistics require the Logger to actively and recurrently send requests to the OS. In the remaining of this section we give details on both the passive and active aspects of our Logger. Also, we describe the difficulties we faced in building this tool and how we overcame them.

5.3.1 Passive data collection

Some of the Android OS components are loosely coupled (e.g. network interfaces and the browser). To make them communicate the Android OS provides the Intents—instances of the *android.content.Intent* class. Intents are asynchronous messages and can be used to perform all sorts of operations: Requesting the system to launch applications, asking the user to select a WiFi network to connect to, sending notification messages to other applications, signalling the Android system that a certain event has occurred (e.g. message received, connection available, etc.). In this latter case, a component interested in a specific event does not have to actively send the system requests on the occurrence of the event. It suffices that the component registers to the specific event through the so called intent filters, and it will be notified by the system as soon as the selected event occurs.

Our Logger uses Intents to be notified about changes in the device connectivity (2G, 3G, WiFi), of the battery status (charging, discharging, current battery life), of the screen state (on/off) etc.

5.3.2 Active data collection

Data collected passively through Intents are not enough for our study. Statistics as network data usage, for example, cannot be obtained through Intents. In this case, the Android OS has to actively be sent specific requests by the interested component. In Android systems this is achieved through Alarms—actions scheduled to be executed recurrently, even when the device is in sleep mode.

So, we built a set of alarms and included them in our Logger architecture in order to log data exchanged through the network interfaces, the set of currently running applications, incoming and outgoing emails, SMSs, phone calls, and so on.

5.3.3 Filesystem activity

Monitoring the filesystem's activity includes logging information on when and how each file or directory was created, deleted, accessed, and modified. This information is clearly important in both the study of the off-clones (as far as files internal to single applications are concerned) and back-clones (user-generated files). The Alarms are not suited to achieve this: They would require very frequent scan of the whole filesystem, which would be inaccurate, aside from being very expensive in terms of battery consumption. Fortunately, the underlying Linux kernel on which Android OS is based includes *Inotify*—a filesystem monitoring service with a similar approach to that of Intents. Inotify allows applications to add so called *watches* to directories. Whenever one of the watched directories, or its content, is modified, the kernel promptly notifies the "watching application". The Android OS provides user-level applications with a Java interface to the Inotify subsystem.

The notifications are sent only when something actually happens on the filesystem. So, the interface to the Inotify subsystem allowed our Logger to get the most accurate information possible about the filesystem changes at a minimal cost in terms of resources. Inotify does not allow to watch for changes happening in the subdirectories of a directory. To overcome this problem the Logger, when started, adds watches to all the directories in the filesystem tree of the device. In addition, whenever a new directory is created/deleted the watch is added/removed automatically by the Logger. It is worth noting that adding each directory of the filesystem tree to the Inotify watch list is not expensive in terms of memory and computational resources, and it did not affect the user experience in a noticeable way.

5.3.4 File permissions

An Android device storage is usually composed of an external storage unit, in the form of an SD card, and one or more internal storage units. The external SD card is where most of the user files are stored (mp3s, pictures, etc.). Conversely, system files like config files and system binaries are stored in the internal storage unit. The SD card storage (usually a FAT32 filesystem) is the one with the highest capacity (for example 10 GB) and is accessible in read/write mode by all the apps without restrictions. While it is possible for an app to have its data stored on the SD card (this is very useful for large apps like games or other media-based ones), most of them are installed in the internal storage space.

As opposite to the external SD card, the access to the internal storage (an ext3 or ext4 filesystem) is restricted: A user-level app is only allowed to access its private data (usually stored in the /data/data/ directory). This prevents any user application (including our Logger), to add Inotify watches to the private directories of other apps. To overcome this limitation we root the devices of our experimental testbed. By rooting a device, an application can execute commands using the root permissions, thus going past the restrictions of user applications, including those on filesystem access. Our Logger, even though running on a rooted device, is not allowed to execute its own code with root permissions. It is however enabled to execute shell commands as root. This permits us to use the Unix shell included in Android OS to temporarily change the permissions of private application files. The permission changing step takes place during the starting phase of the Logger. The old permissions are saved into a database (internal to Logger), and are restored whenever it is stopped by the user. To make the permission changing phase as independent as possible from the Android platform we used BusyBox [114] a popular lightweight collection of Unix tools largely used and optimized for embedded systems. Finally, with these modifications, the Logger is enabled to add Inotify watches to files that are private to other apps.

5.3.5 Dealing with file modifications

Once Inotify sends a notification to our Logger about the modification/editing of a directory/file, we need to understand the amount of data that we should upload to the cloud so to keep the clone up-to-date with the user's device. Of course, uploading has to be performed in an efficient way—we certainly do not want to upload the whole file/directory each time it is modified. A standard technique is to send just incremental modifications of the file, instead of the whole file. Unfortunately, nor does Inotify provide details on the amount of data that was modified, neither does it keep track of the modified file portion. So we had to find a method to identify the file portions involved in a modification that was fast and lightweight. Fast, so that it could cope with frequent modifications occurring in the system. Lightweight, so that its overhead is low and it would not affect the user experience. For all these reasons we decided to apply the *rolling hash* [115] technique, widely used on popular backup/sync software *rsync* [116]. Rolling hash is one of the possible implementations of the so called *binary-diff*, used also by *Dropbox* to update users' modified files yet not wasting users' network bandwidth[2]. This technique allows to compute very quickly (in linear time with respect to file sizes) hashes of all possible file blocks (the block size is a parameter of the rolling hash technique). So, it makes possible to quickly compute the amount of data (in terms of blocks) that make two file versions differ. A very detailed and clear description of the rolling hashes technique (including how to compute it and how to deal with hash collisions) can be found in [115].

We built from scratch a Java implementation of the rolling hash technique, and included it as one of the modules of our Logger app. Old rolling hashes of files are maintained into a SQLite database (stored in the phone's SD card) so that they are promptly available to be compared with the new rolling hashes each time files are modified. The block size we used in our implementation equals $8KB$, the same used by rsync.

Our implementation of the rolling hash technique turned out to be very efficient in computing and comparing hashes. Nonetheless, we believe that an implementation written in native code (C/C++) and running as a root process would dramatically improve the performances. We leave this as future work.

5.3.6 Logger output

The data collected by each component of the Logger is continuously written to a log file (a simple text file) that is periodically rotated, compressed and stored onto a special directory within the SD card reserved to the Logger. The user, on request, can also trigger sending emails with the content of this directory, from time to time, to a custom gmail account we created for this purpose. If the user does so, the log files that are successfully sent by email are deleted from the device.

[2]https://www.dropbox.com/help/8/en

Number, type & OS	CPU	RAM
7×Samsung Galaxy S Plus (Android 2.3)	1.4 GHz Scorpion	512 MB
2×Samsung Galaxy S (Android 2.3)	1 GHz Cortex-A8	512 MB
1×Samsung Galaxy Note (Android 2.3)	1.4 GHz dual-core Cortex-A9	1 GB
1×Samsung Galaxy Nexus (Android 4.1)	1.2 GHz dual-core Cortex-A9	1 GB

Table 5.1: Phone specifications.

5.4 Experimental results

Our experiments are based on a testbed of real users. In this section we describe the testbed, the results, and the observations of our study on the cost of keeping updated both off-clones and back-clones in the cloud.

5.4.1 Experimental setting

To gather data related to the device usage we set up a testbed of 11 smartphones running our Logger app. In addition, the testbed consists of an equal number of software clones—customized AMIs of the Android x86 OS [88]—running on the Amazon's EC2 platform. Logger makes the device communicate with the clones to collect data related to networking.

The details of the mobile devices involved in the experiment are shown in Table 5.1. The devices were used as primary smartphones for the whole duration of the experiment (the experiment lasted 3 weeks) and involved people living in the city of Rome, Italy, and Cambridge, UK. The profiles of the participants are heterogeneous in age and occupation—university students, faculties, and part-time and full-time workers that are completely external to the university. So is the technology they use to connect to the Internet with their mobile devices: One of the participants does not have a cellular data traffic plan, so he connects only through WiFi networks (home or work). Another participant does not have an available WiFi network neither at home nor at work, so he only makes use of 2G/3G technology. The remaining participants use habitually both cellular data traffic and WiFi

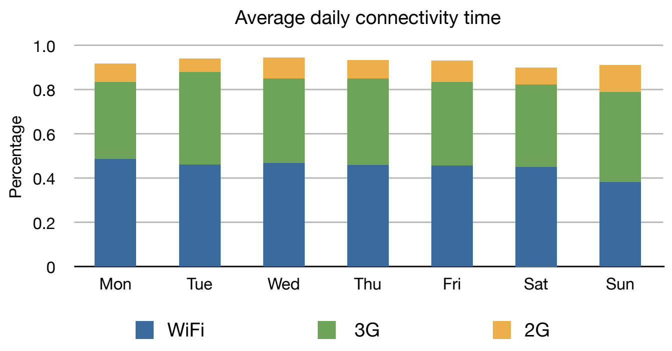


Figure 5.1: Average daily connectivity percentage for various technologies.

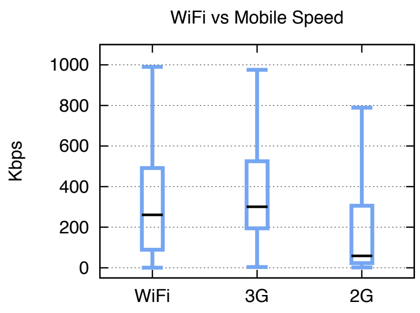
interchangeably.

During the experiment the participants used their own contract with the service provider of their choice. The data we gathered includes the four major cellular service providers in Italy, namely, TIM, Vodafone, Wind and 3, as well as O2, a major provider in UK. It is worth to note that, though each of these carriers relies on its own network infrastructure, they all provide cellular data traffic plans with the same upload and download speed in best case scenarios. The actual speed and availability depend, of course, on the quality of the signal and on the load of the cells that are used by the participants during the three weeks of the experiment. As this is an element that might impact the performance of mobile cloud paradigms, we start our study with network availability.

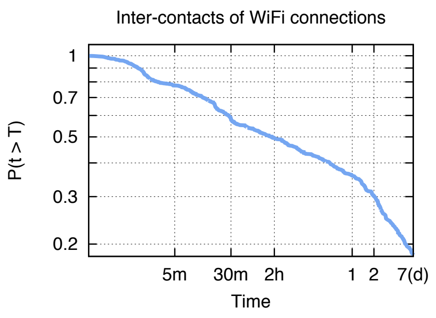
5.4.2 Mobile data traffic and network availability

Certainly one of the most important component to keep both off-clones and back-clones updated is the ability of the real devices to communicate often and efficiently with the cloud. Often, so that the user can send data to the cloud anytime she needs. Efficiently, so that the overhead of this communication is as low as possible and does not impact the usability of the system. Indeed, if the networking process becomes a bottleneck to usability then people will not be happy about it, and they will less likely accept and use mobile cloud systems.

The first network related result that we present is the average amount of time per day,



(a) Average (per user) daily upload speed. The graphics include the minimum and maximum speed value as well as the 25^{th}, 50^{th} and 75^{th} quartile.



(b) Cumulative distribution of WiFi connection inter-contact times.

Figure 5.2: Network properties.

in percentage, users spend connected either to 3G/2G or to WiFi networks (see Figure 5.1). The first observation that we make is that the percentage of the daily time period during which the devices are connected is quite high (more than 90%), and the trend is similar for all the days of the week. The periods of disconnection are either due to areas with no signal at all (e.g. subways), or due to periods in which the devices are switched off. The time of connectivity using the three different networking technologies (WiFi, 3G, 2G) is also quite stable from day to day: Around 45% of WiFi, around 40% of 3G, and the remaining of 2G. This partitioning is particularly positive being WiFi and 3G the two communication technologies with larger bandwidth.

Keeping off-clones and back-clones up-to-date requires the device to upload data to the cloud. For this reason we measure, separately for 3G/2G and WiFi, the average speed in upload during the real life conditions of our testbed. We achieve the measurement by making Logger send a file of 300kB from each device to its clone on the Amazon platform every 30 min, keeping track of the technology used to send the data, and averaging the results. We compute the average speed for every user (per day). Figure 5.2(a) shows the average as well as a representation of the distribution of the values: Minimum, 25th, 50th, and 75th percentiles, and maximum. By doing so we are able to depict in the graphic results related to scenarios where the user happens to be in zones that are not well covered by wireless technology.

Let us first focus on comparing 3G with 2G connectivity. When using 2G/3G technology, smartphones switch from 3G to 2G whenever the 3G signal is not present or lower than a certain threshold, which depends on the smartphone type. As expected, the 2G upload speeds are much lower than the 3G ones (see Figure 5.2(a)). Fortunately, this 3G to 2G switch typically happens only for short periods of time (see Figure 5.1). WiFi connections are faster than 3G ones, as far as download is concerned. However, the mobile cloud paradigm requires that the majority of traffic travels from the real devices to the cloud, to keep the clone up-to-date. The results in Figure 5.2(a) show that the average upload speeds are higher in the 3G case with respect to WiFi. This is not surprising: When accessing the WiFi users typically set up their devices to automatically connect to known and trusted WiFi routers that are at their home or work locations. These in Italy are generally ADSL WiFi routers, and thus they do not provide high speed in upload. This last consideration makes think that, when a device is being charged, it is more likely to be connected to a WiFi network—usually we charge our devices at home or at work where WiFi is available. Not to mention that ADSL connection providers make users pay fixed prices for unlimited bandwidth. So, exploiting WiFi networking to keep clones up-to-date is more likely to come for free in both terms of costs (no cellular data traffic is used) and energy (the device has more chances to be charging).

Figure 5.1 shows that the overall WiFi connection time per day is high, around 45% of the time. However, it is important to see how this period of time is distributed over the day. This is an important question to address. Indeed, as long as the device is under WiFi coverage, keeping off-clones and back-clones updated is cheap and therefore users might configure the system in such a way to use WiFi only to sync. So, we are interested in checking whether it is frequent that users are without WiFi connection for long period of times. A technical way to check this property is to compute the distribution of the intervals between the end of a period of WiFi coverage and the start of the next period of WiFi coverage. We refer to this time as the WiFi inter-contact time. The results of our experiments to this respect are shown in Figure 5.2(b).

From Figure 5.2(b) we can see that 20% of the WiFi inter-contact times are lower than 20 min, 40% are lower than 30 min, 50% are lower than 2h. Just 35% of them are higher than 1 day. Considering that among the participants there are also people that do not use WiFi at all, this result is excellent—50% of WiFi inter-contact times are as low as 2h and that means that 50% of the times users can rely on WiFi connection to sync their device's clones as frequently as every 2 hours or more. And, as we already discussed, this makes users save their cellular data traffic, and, possibly, save battery (if during the WiFi connectivity they are charging their device at home or at work.).

5.4.3 Overhead of Mobile Cloud Computing

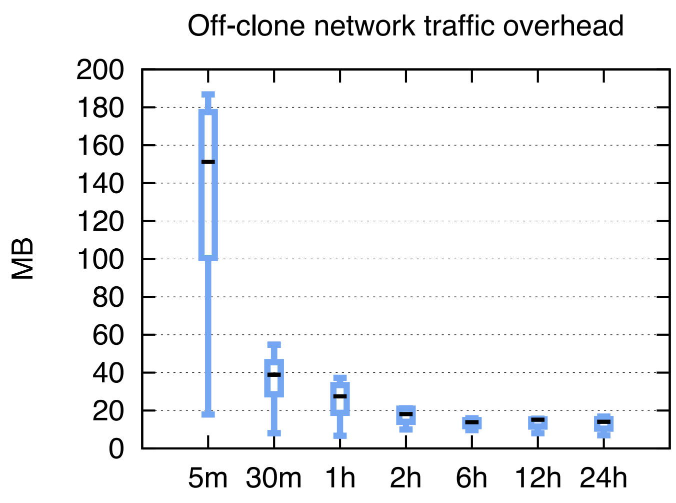
To keep off-clones and back-clones up-to-date devices need to recurrently access the network interfaces to send the latest user-generated data and app statuses to the respective clones. Clearly, the more often the device syncs with the clone, the better the clones represent the status of the device. Though, frequent syncs incur higher bandwidth and energy overhead to the real device. The overhead depends on the network technology involved in the device-cloud communication (WiFi vs 2G/3G), as well as on how the device is used—for example, the more pictures a user takes with her device's camera the more data have to be transferred to the user's back-clone. The user should be able to trade-off this overhead with the synchronization level of the clone of her device, and decide the rate of synchronization that better suits her needs.

Off-clone synchronization requires information on the user apps. For a given app this information includes the app's state, its internal private files, its settings specified by the user, and so on. Back-clone synchronization requires informations on the apps that are installed in the device and user's data (contacts, calendars, pictures, music, notes, emails, SMS etc.). As we already anticipated in Section 5.3, the Logger computes, at the end of specific time-intervals that correspond to sync frequencies, the amount of data that the device should send to the cloud in order to update each clone type. In particular we recall that, when a file is modified, we use the efficient rolling hash technique to compute the difference between the new and the old version of the file. The sync intervals that we have considered are 5 min, 30 min, 1h, 2h, 6h, 12h, and 24h. To keep it as real as possible, the Logger computes the amount of data to be sent at the end of each interval only if either 2G/3G or WiFi connectivity is available.

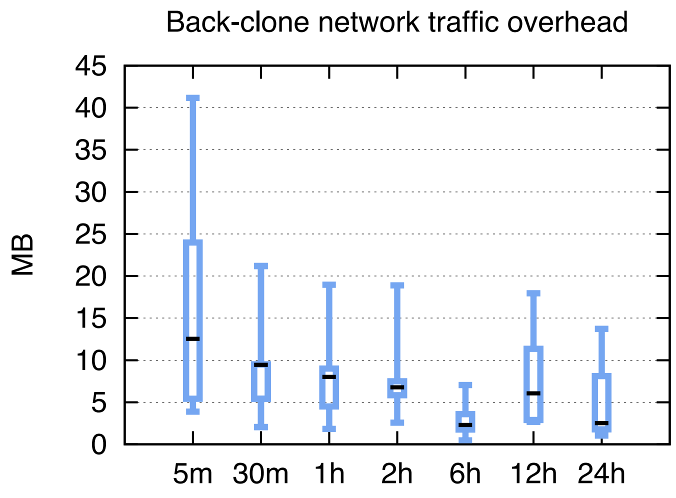
Network bandwidth overhead

In Figures 5.3(a) and 5.3(b) we plot, for both clone types, the quartiles of the distribution of the average (per user) daily traffic needed to be sent to the cloud in dependence of the sync frequency in order to keep off-clone and back-clone updated. The first simple observation that we make is that the traffic overhead incurred by the synchronization decreases as the sync interval increases. This phenomenon is straightforward—while the device is running there is a large number of files/directories generated and then destroyed in the system (temporary files). Typically, this happens more frequently with system or app private files rather than with user generated files. Indeed, the results in Figure 5.5, that show the complementary cumulative distribution of the file lifetime for both clone types in the device, confirm this intuition—more than 90% of off-clone files last less than 1s, while more than 90% of back-clone files last longer than 5 days. If the device-cloud synchronization interval is long, most of the temporary files generated during the interval do not last till its end—when the file diffs are computed. On the contrary, if the sync interval is short, it is more likely that temporary files are still "alive" at the end of the interval, so they are involved in the file diffs. This is what boosts up the overhead traffic incurred by small synchronization intervals even in the presence of a smart incremental backup system like that we implemented. As this is more likely to happen with system/app private files, the difference between the overhead traffic generated by the synchronization for different sync interval lengths is smaller for back-clones—these clones involve more user-generated data (like sent/received emails, texts, calls, and so on) which is rarely deleted by the user. Another important observation is that, for small sync intervals, the back-clones incur much less overhead (around 4 times less) than the off-clones. Again, this is due to the fact that the user generates data with much less frequency with respect to the system, and typically do not delete their data. The overhead difference between the two types of clones is attenuated when the interval of syncs increases (see Figures 5.3(a) and 5.3(b)).

Lastly, for both types of clones, the bandwidth overhead incurred is not excessive, if we compare it to the data traffic normally generated by the user's device to receive/send mail, access Facebook/Twitter accounts etc. For the sake of comparison we have plotted, in Figure 5.4, the quartiles of the average traffic generated normally by the users per day.



(a) Average (per user) off-clone traffic overhead.



(b) Average (per user) back-clone traffic overhead.

Figure 5.3: Bandwidth overhead for clone synchronization in dependence of the sync frequency. The graphics include the minimum and maximum speed value as well as the 25^{th}, 50^{th} and 75^{th} percentile.

Energy overhead

To measure the energy overhead incurred by the sync of back-clones and off-clones we make use the Mobile Device Power Monitor, an external power meter widely used to validate offloading frameworks esteems [94, 112, 113]. The device samples the smartphone's

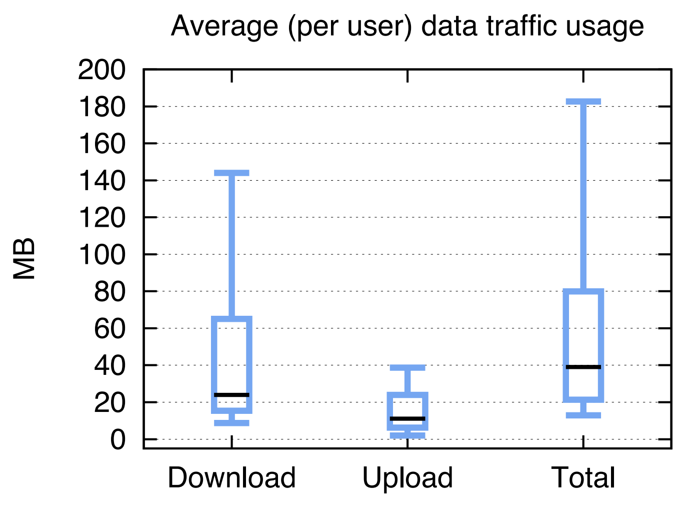


Figure 5.4: Average (per user) data traffic sent/received per day.

battery with high frequency (5000 Hz) so to yield accurate results on the battery's power, current, and voltage. Clearly, we could not do the measurement when the devices were with the users. So, at the end of the experiment, we gathered the devices and re-simulated the device-cloud communication for each smartphone with the device connected to the Power Monitor. The simulation included re-computation of the amount of data to be sent to the clones at the end of the time-interval considered, as well as the data sending process. The clones involved are customized Amazon AMIs (Amazon Machine Image) of the Android-x86 OS and run on the Amazon EC2 platform.

The simulation is run 20 times for every device: 10 times using WiFi connectivity and the other 10 times using 3G connectivity. Figures 5.6(a) and 5.6(b) show the trend of the energy overhead in the case of WiFi connectivity for both off-clones and back-clones in dependence of the sync frequency. Similarly, Figures 5.6(c) and 5.6(d) show the trend of the energy overhead in dependence of the sync frequency in case of 3G connectivity. There are several considerations to be made. First, let us consider WiFi sync compared to 3G sync of the same clone type. See, for example, Figure 5.6(a) and Figure 5.6(c) that depict the results for the off-clones. When the same clone type is considered, the energy overhead difference is determined by the communication technology used. Indeed, the energy dissipated to compute the file diffs is the same, being that we are comparing the

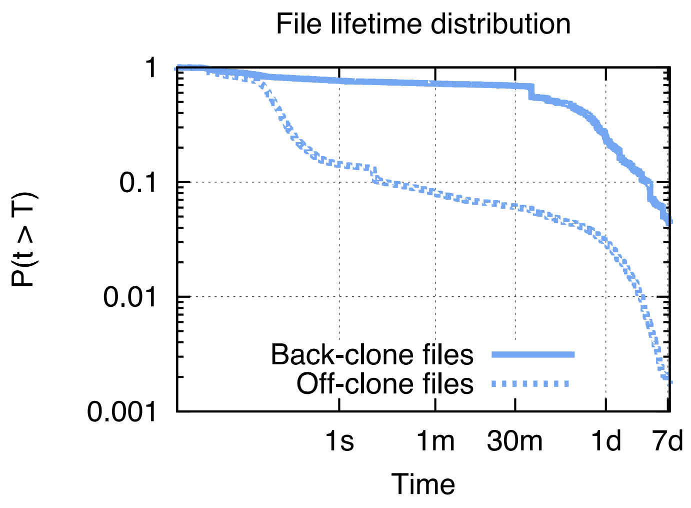
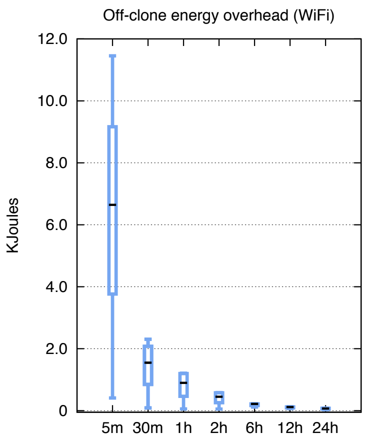


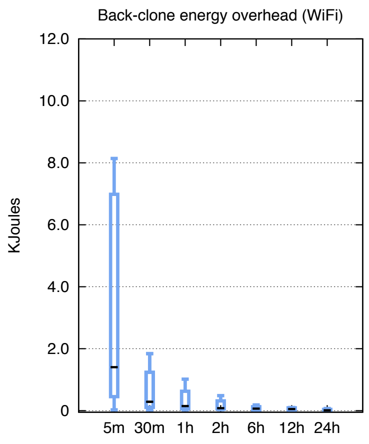
Figure 5.5: istribution of file lifetime relative to off-clones and back-clones.

same clone types among them. However, the energy consumed in computing the rolling hashes of the files dominates the energy required to actually send the diffs. As a result, the overall energy overhead of the sync through WiFi is slightly lower than the overhead of the sync done through 3G. The difference is of the order of tens of Joule.

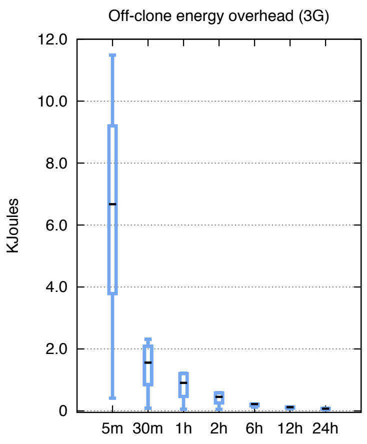
Now, let us focus on one communication technology, say, WiFi, and compare the energy overhead to sync off-clones (Figure 5.6(a)) with the energy overhead to sync back-clones (Figure 5.6(b)). It is worth noting that, although off-clones incur higher energy costs than back-clones, the difference in energy requirement is much more attenuated than the difference in traffic overhead. This is so for the following reason: As shown in Figure 5.5, off-clones generate many small temporary files (the system files). On the other hand, the files generated by the back-clones are user-generated files. Typically, these are bigger, are generated with a lower frequency, and last longer. So, the computation load of the file diffs is somehow balanced in the two cases—For the off-clones the rolling hashes are computed very frequently times on small files; for the back-clones, the rolling hashes are computed much less frequently on bigger files. The energy for this computation dominates the energy required to send the file diffs. So, even though back-clones generate 4 times less traffic overhead than off-clones, the difference of the overall energy spent to compute the diffs and to send them to the cloud for both clone types is attenuated. The same observation holds



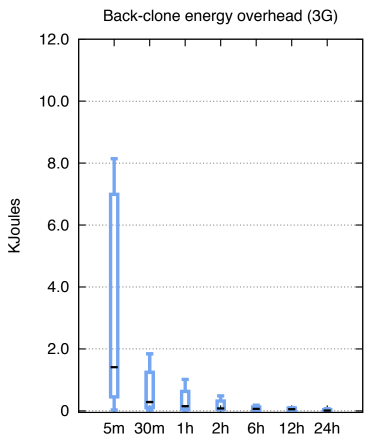
(a) Off-clone energy overhead (WiFi) per day.



(b) Back-clone energy overhead (WiFi) per day.



(c) Off-clone energy overhead (3G) per day.



(d) Back-clone energy overhead (3G) per day.

Figure 5.6: Energy overhead for clone synchronization through WiFi (a and b) and through 3G (c and d) in dependence of the sync frequency. The graphics include the minimum and maximum speed value as well as the 25th, 50th and 75th percentiles.

for the case when 3G technology is used in the device-clone communication.

That said, it is worth noting that mobile cloud computing does not impact much the life of the battery. The smartphones we used in our testbed are powered by Lithium–Ion batteries (1650 mAh, 3.7 V). These batteries, if fully charged contain 21.9 KJoule of energy. According to our experiments, synchronizing off-clones (back-clones) every 5 minutes, incurs, at max, around 11.8 KJoule (8 KJoule) of energy overhead per day (see Figure 5.6). This means that the sync cost is about 53% (36%) of the battery. These values certainly correspond to an extreme scenario—keeping the clone updated every 5 min certainly has its cost. As soon as lower synchronization frequencies are considered these values are drastically reduced. For 30 min (2h) sync intervals the synchronization cost drops to around 11% (2.7%) of the battery for the off-clone and around 8% (2.3%) of the battery for the back-clone.

Lastly, the differences in terms of energy overhead between off-clones and back-clones (being it WiFi or 3G) are attenuated with the increasing of the synchronization interval. In addition, large synchronization intervals yield much lower energy overhead than short ones.

5.5 Lesson learned and conclusions

In this work we have described our experience with a testbed of real users of smartphones and a mobile cloud system of smartphone software clones in the cloud. The goal of the experiment is to understand the feasibility of mobile cloud systems in a real setting—a setting consisting of people (participants in the experiment) using the smartphone as their primary device during their everyday life. In the experiment, the participants made use of their mobiles just as usual for three weeks. Here are some of the key observation we made during this experience:

- Most of the users are virtually always under the coverage of some wireless technology. In particular, almost 50% of the time smartphone users are connected to a WiFi access point. Indeed, this is often the case at home and at work (recall that these numbers are computed as the average of all the participants);
- In more than 50% of the cases, users lose WiFi coverage for just for 2 hours at most. This is probably due to commuting between places with WiFi, like work and home. For a systems point of view, it means that sync operations can optimistically wait

until the device is connected to a WiFi access point, and that most probably this is going to happen in a short period of time.

- Synchronizing back-clones (for backup purposes) requires less network traffic (down to 4 times less) and less energy overhead (around to 3 KJoule less) than synchronizing off-clones (that handle mobile computation offload).
- The difference in overhead incurred by the synchronization of the two clone types decreases drastically as the sync frequency decreases; reasonable sync frequencies like 30 min have a reasonable cost in term of energy spent on the device to keep off-clones and back-clones updated (11% of the battery for the off-clones and 8% for the back-clones sync). Recall that by paying this overhead, the user either has a very efficient backup system or can efficiently offload computation on the fly. This latter service has the potential, depending on the application, to reduce energy consumption by a factor that is much higher than the cost we computed in this experiment.
- Finally, WiFi technology incurs lower overhead with respect to 3G. However, the overall energy overhead, which depends also on the workload before the device-cloud communication, is almost the same with both communication technologies.

Our work supports the conclusion that mobile cloud computing can be sustained by continuous update of software clones in the cloud with a reasonable overhead in terms of bandwidth and energy costs, especially if the sync intervals are not too short.

Future Work and Concluding Remarks

The past few years have been characterized by an explosion of the market of mobile devices. It all started with the awkward and poorly designed first models of PDAs of the mid 90's, that had relative success only among a small market niche of early adopters with very specific needs. In the following years things changed slowly, until a "revolution" happened around 2007, when faster, more appealing devices suddenly became extremely popular to the general public too. We reached a point where everyone has, or longs for, a smartphone or a tablet (or both), and it is clear that it won't take long for the market to expand up to saturation like it happened in the past for the phone or the television [117]. During my P.h.D I studied what, in my opinion, represent the most important issues affecting this *PostPC*-era, which regard data connectivity, security and energy efficiency of the mobile devices. In trying to solve these issues I used an hybrid approach, taking the best of different technologies to create new solutions.

We started with the lessons learned from the field of Pocket Switched Networks (PSNs). The insights on the statistical properties of human meeting patterns allowed us to tackle two important problems, namely, the overloading of the network infrastructure and the identity-theft. For what concerns the first problem, we showed how it is possible to identify a set of important subscribers, the *VIPs*, that can act as a bridge between the other subscribers and the network provider whenever a large set of data has to be transferred to/from the network. Our experiments showed that this technique is indeed effective in reducing the load on the network infrastructure even when the set of VIPs is less than 10% of the nodes in the network. As a future work, we would like to consider a more realistic communication model, that takes into consideration, for example, the sizes of the data to be transferred.

Then, we showed how to exploit the regularity of the meeting patterns of people in order to define a new authentication mechanism that is resistant to the identity theft (or cloning). We introduced Personal Marks and Community Certificates. Personal Marks detect whenever a cloned identity is used inside the community of the victim. On the other hand, Community Certificates prevent the unauthorized use of the cloned identity outside of the community of the victim. Used together, these two systems allow to unmask an attack that other authentication mechanisms fail to detect. In the future, we will use a new dataset we collected with the purpose of testing the system in a more accurate way, using previously unavailable information about all the different aspects of the social life of the users.

From the PSNs we then moved to the clouds. We considered how the usage of software clones of mobile devices running on the cloud infrastructure can help fighting the spread of social mobile malware. In this context, CloudShield is a suite of strategies able to detect a small set of clones that, if used to patch the respective devices, can quickly stop the malware from infecting the whole network. We also considered the eventuality that the cloud itself becomes a channel for the spread of the malware and used a quarantine strategy to stop it. Our evaluations performed with real datasets have shown that our strategies are simpler, faster to compute and more effective than the state of the art of malware containment techniques. In the future, we plan to study the scenario of smarter malware that explicitly tries to fool our patching strategies.

Finally, we studied the feasibility, in terms of energy and bandwidth consumption, of mobile cloud systems in a real setting. We considered, again, a scenario where mobile devices have a respective software clone on the cloud that they can use to offload computationally intensive tasks or to backup their data. To do so, we collected data about the regular use of the smartphone of a set of 11 participants for a period of three weeks. Our data shows how much time users spend connected to WiFi or 3*g*, how much download and upload bandwidth they have available and how the frequency of updates sent from the smartphones to their clones in the cloud impacts the energy and bandwidth consumption. In the future we would like to extend the study to a larger set of volunteers.

Bibliography

[1] Marco Valerio Barbera, Julinda Stefa, Aline Carneiro Viana, Marcelo Dias de Amorim, and Mathias Boc. Vip delegation: Enabling vips to offload data in wireless social mobile networks. In *DCOSS*, 2011.

[2] Marco Valerio Barbera and Alessandro Mei. Personal marks and community certificates: Detecting clones in wireless mobile social networks. In *DCOSS*, 2012.

[3] Marco V. Barbera, Sokol Kosta, Julinda Stefa, Pan Hui, and Alessandro Mei. Cloud-shield: Efficient anti-malware smartphone patching with a p2p network on the cloud. In *P2P*, 2012.

[4] Marco Valerio Barbera, Sokol Kosta, Alessandro Mei, and Julinda Stefa. To offload or not to offload? the bandwidth and energy costs of mobile cloud computing. In *INFOCOM*, 2013.

[5] Customers Angered as iPhones Overload AT&T. New York Times, http://www.nytimes.com/2009/09/03/technology/companies/03att.html, Sep. 2009.

[6] iPhone overload: Dutch T-Mobile issues refund after 3G issues. Ars Technica, http://arstechnica.com/tech-policy/news/2010/06/dutch-t-mobile-gives-some-cash-back-because-of-3g-issues.ars, Jul. 2010.

[7] Facebook: Over 955 million users, 543 million mobile users. CNET: http://news.cnet.com/8301-1023_3-57480950-93/facebook-over-955-million-users-543-million-mobile-users/, July 2012.

[8] 2011 state of online and mobile banking. ComScore: http://www.comscore.com/Press_Events/Presentations_Whitepapers/2012/2011_State _of_Online_and_Mobile_Banking, Febr. 2012.

[9] Amazon customers now order 1$ billion of products per year via mobile. Mashable: http://mashable.com/2010/07/22/amazon-mobile-sales/, Jul. 2010.

[10] Mcafee threats report: Second quarter 2012. McAfee: http://www.mcafee.com/us/resources/reports/rp-quarterly-threat-q2-2012.pdf, Sept. 2012.

[11] At&t to pay $1.93 billion for qualcomm mobile spectrum. Bloomberg: http://www.bloomberg.com/news/2010-12-20/at-t-agrees-to-acquire-wireless-licenses-from-qualcomm-for-1-93-billion.html, Dec. 2010.

[12] Pan Hui, Augustin Chaintreau, James Scott, Richard Gass, Jon Crowcroft, and Christophe Diot. Pocket switched networks and human mobility in conference environments. In *SIGCOMM*, 2005.

[13] Battery life complaints causing operator headaches. Telecoms: http://www.telecoms.com/48822/battery-life-complaints-causing-operator-headaches/, Sept. 2012.

[14] Athanasios V. Vasilakos, Yan Zhang, and Thrasyvoulos Spyropoulos. *Delay Tolerant Networks: Protocols and Applications*. CRC Press, Inc., 2011.

[15] J. Scott, R. Gass, J. Crowcroft, P. Hui, C. Diot, and A. Chaintreau. CRAWDAD trace cambridge/haggle/imote/content (v. 2006–09–15). Downloaded from http://crawdad.cs.dartmouth.edu/cambridge/haggle/imote/content, September 2006.

[16] David Kotz, Tristan Henderson, and Ilya Abyzov. CRAWDAD data set dartmouth/-campus (v. 2007-02-08). http://crawdad.cs.dartmouth.edu/dartmouth/campus.

[17] A. Chaintreau, P. Hui, J. Crowcroft, C. Diot, R. Gass, and J. Scott. Pocket switched networks: Real-world mobility and its consequences for opportunistic forwarding. Technical report, University of Cambridge, 2006.

[18] Thomas Karagiannis, Jean-Yves Le Boudec, and Milan Vojnović. Power law and exponential decay of inter contact times between mobile devices. In *MobiCom*, 2007.

[19] V. Conan, J. Leguay, and Timur T. Friedman. Characterizing pairwise inter-contact patterns in delay tolerant networks. In *Proceedings of the 1st international conference on Autonomic computing and communication systems*, Autonomics, 2007.

[20] Marta C. Gonzalez, Cesar A. Hidalgo, and Albert-Laszlo Barabasi. Understanding individual human mobility patterns. *Nature*, 2008.

[21] Pan Hui, Jon Crowcroft, and Eiko Yoneki. Bubble rap: Social-based forwarding in delay tolerant networks. In *MobiHoc*, 2008.

[22] Pan Hui, Eiko Yoneki, Shu Yan Chan, and Jon Crowcroft. Distributed community detection in delay tolerant networks. In *MobiArch*, 2007.

[23] Gergely Palla, Imre Derényi, Illés Farkas, and Tamás Vicsek. Uncovering the overlapping community structure of complex networks in nature and society. *Nature*, 2005.

[24] M. E. Newman. Detecting community structure in networks. *EPJB*, March 2004.

[25] M. E. Newman. Analysis of weighted networks. *Phys Rev E Stat Nonlin Soft Matter Phys*, November 2004.

[26] Pan Hui. People are the network: experimental design and evaluation of social-based forwarding algorithms, 2008.

[27] A. Vahdat and D. Becker. Epidemic routing for partially-connected ad hoc networks. Technical report, Duke University, 2000.

[28] Thrasyvoulos Spyropoulos, Konstantinos Psounis, and Cauligi S. Raghavendra. Spray and wait: an efficient routing scheme for intermittently connected mobile networks. In *Proceedings of the 2005 ACM SIGCOMM workshop on Delay-tolerant networking*, WDTN '05, pages 252–259, New York, NY, USA, 2005. ACM.

[29] Elizabeth M. Daly and Mads Haahr. Social network analysis for routing in disconnected delay-tolerant manets. In *MobiHoc*, 2007.

[30] Vijay Erramilli, Mark Crovella, Augustin Chaintreau, and Christophe Diot. Delegation forwarding. In *MobiHoc*, 2008.

[31] Alessandro Mei, Giacomo Morabito, Paolo Santi, and Julinda Stefa. Social-aware stateless forwarding in pocket switched networks. In *INFOCOM*, 2011.

[32] Jungkeun Yoon, Brian D. Noble, and Mingyan Liu. Building realistic mobility models from coarse-grained traces. In *in Proc. MobiSys*, 2006.

[33] T. Camp, J. Boleng, and V. Davies. A Survey of Mobility Models for Ad Hoc Network Research. *Wireless Communications & Mobile Computing (WCMC): Special issue on Mobile Ad Hoc Networking: Research, Trends and Applications*, 2002.

[34] Mirco Musolesi and Cecilia Mascolo. Designing mobility models based on social network theory. *SIGMOBILE Mob. Comput. Commun. Rev.*, 2007.

[35] Alessandro Mei and Julinda Stefa. Swim: A simple model to generate small mobile worlds. In *INFOCOM'09*, 2009.

[36] C. Boldrini and A. Passarella. Hcmm: Modelling spatial and temporal properties of human mobility driven by usersâĂŹ social relationships. *Computer Communications*, 33(9):1056–1074, 2010.

[37] K. Lee, S. Hong, S. J. Kim, I. Rhee, and S. Chong. SLAW: A Mobility Model for Human Walks. In *INFOCOM '09: Proceedings of The 28th IEEE Conference on Computer Communications*, 2009.

[38] Injong Rhee, Minsu Shin, Seongik Hong, Kyunghan Lee, and Song Chong. On the levy-walk nature of human mobility. In *INFOCOM*, 2008.

[39] Smartphone sales overtake pcs for the first time. Mashable: http://mashable.com/2012/02/03/smartphone-sales-overtake-pcs/, Feb. 2012.

[40] N. Eagle and A. Pentland. Social serendipity: Mobilizing social software. *Pervasive Computing, IEEE*, 4(2):28–34, 2005.

[41] Shravan Gaonkar, Jack Li, Romit Roy Choudhury, Landon Cox, and Al Schmidt. Micro-blog: Sharing and querying content through mobile phones and social participation. In *ACM MobiSys*, 2008.

[42] E. Miluzzo, N. D. Lane, K. Fodor, R. Peterson, H. Lu, M. Musolesi, S. B. Eisenman, X. Zheng, and A. T. Campbell. Sensing meets mobile social networks: the design, implementation and evaluation of the cenceme application. In *ACM SenSys*, 2008.

[43] S. B. Eisenman, N. D. Lane, E. Miluzzo, R. A. Peterson, G. Ahn, and A. T. Campbell. Metrosense project: People-centric sensisng at scale. In *ACM SenSys*, 2006.

[44] S. Ioannidis, A. Chaintreau, and L. Massoulie. Optimal and scalable distribution of content updates over a mobile social network. In *IEEE Infocom*, 2009.

[45] M. V. Barbera, J. Stefa, A. Carneiro Viana, Marcelo D de Amorim, and M. Boc. Vip delegation: Enabling vips to offload data in wireless social mobile networks. In *IEEE DCOSS*, 2011.

[46] V. Chandrasekhar, J. Andrews, and A. Gatherer. Femtocell networks: a survey. *IEEE Communications Magazine*, 46(9):59 –67, September 2008.

[47] AT&T, Verizon Wireless join Wi-Fi interoperability group. http://news.cnet.com/8301-30686_3-20008476-266.html, June 2010.

[48] A. Balasubramanian, R. Mahajan, and A. Venkataramani. Augmenting mobile 3g using wifi. In *ACM MobiSys*, 2010.

[49] K. Lee, I. Rhee, J. Lee, Y. Yi, and S. Chong. Mobile data offloading: How much can wifi deliver? In *ACM SIGCOMM*, 2010.

[50] Cisco visual networking index forecast predicts continued mobile data traffic surge. http://newsroom.cisco.com/dlls/2010/prod_020910b.html, Feb. 2010.

[51] B. Han, P. Hui, V. S. A. Kumar, V. M. Marathe, G. Peig, and A. Srinivasan. Cellular traffic offloading through opportunistic communications: A case study. In *ACM CHANTS*, 2010.

[52] A. Garyfalos and K. C. Almeroth. Coupons: A multilevel incentive scheme for information dissemination in mobile networks. *IEEE Transactions on Mobile Computing*, 7(6):792 –804, June 2008.

[53] Vigo Kann. *On the Approximability of NP-complete Optimization Problems*. PhD thesis, Department of Numerical Analysis and Computing Science, Royal Institute of Technology, Stockholm., 1992.

[54] Th. Hossmann, Th. Spyropoulos, and F. Legendre. Know thy neighbor: Towards optimal mapping of contacts to social graphs for dtn routing. In *IEEE INFOCOM'10*, 2010.

[55] A. Mei and J. Stefa. Give2get: Forwarding in social mobile wireless networks of selfish individuals. In *ICDCS*, 2010.

[56] P. Jaccard. Étude comparative de la distribution florale dans une portion des Alpes et des Jura. *Bulletin del la Société Vaudoise des Sciences Naturelles*, 37:547–579, 1901.

[57] L. C. Freeman. Centrality in social networks conceptual clarification. *Social Networks*, 1(3):215–239, 1979.

[58] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Comput. Netw. ISDN Syst.*, 30:107–117, April 1998.

[59] T. Henderson, D. Kotz, I. Abyzov, and J. Yeo. CRAWDAD trace dartmouth/campus/movement/01_04 (v. 2005–03–08). http://crawdad.cs.dartmouth.edu/dartmouth/campus/movement/01_04.

[60] M. Piorkowski, N. S.-Djukic, and M. Grossglauser. A parsimonious model of mobile partitioned networks with clustering. In *COMSNETS*, 2009.

[61] A. Chaintreau, P. Hui, J. Crowcroft, C. Diot, R. Gass, and J. Scott. Impact of human mobility on the design of opportunistic forwarding algorithms. *IEEE Transactions on Mobile Computing*, 6(6):600–620, June 2007.

[62] A. Mei and J. Stefa. SWIM: A simple model to generate small mobile worlds. In *IEEE Infocom*, 2009.

[63] J. Leguay, A. Lindgren, J. Scott, T. Riedman, J. Crowcroft, and P. Hui. CRAWDAD trace upmc/content/imote/cambridge (v. 2006–11–17). Downloaded from http://crawdad.cs.dartmouth.edu/upmc/content/imote/cambridge, nov 2006.

[64] CENS Urban sensing. http://urban.cens.ucla.edu/projects.

[65] MIT Senseable City Lab. http://senseable.mit.edu.

[66] EARSeL. http://www.earsel.org/?target=SIGs.

[67] Lost and stolen smartphones will cost $30 billion in 2012. WebProNews: http://www.webpronews.com/lost-and-stolen-smartphones-will-cost-30-billion-in-2012-2012-03, Mar. 2012.

[68] R.R. Brooks, P.Y. Govindaraju, M. Pirretti, N. Vijaykrishnan, and M.T. Kandemir. On the detection of clones in sensor networks using random key predistribution. *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, 2007.

[69] Haowen Chan, Adrian Perrig, and Dawn Song. Random key predistribution schemes for sensor networks. In *IEEE Symposium on Security and Privacy*, 2003.

[70] M. Conti, R. Di Pietro, L. V. Mancini, and A. Mei. Distributed detection of clone attacks in wireless sensor networks. *IEEE Transactions on Dependable and Secure Computing*, 2010.

[71] Bo Zhu, Sanjeev Setia, Sushil Jajodia, Sankardas Roy, and Lingyu Wang. Localized multicast: Efficient and distributed replica detection in large-scale sensor networks. *IEEE Transactions on Mobile Computing*, 2010.

[72] Augustin Chaintreau, Pan Hui, Jon Crowcroft, Christophe Diot, Richard Gass, and James Scott. Impact of human mobility on opportunistic forwarding algorithms. *IEEE Transactions on Mobile Computing*, 2007.

[73] Mobile payment in china. http://www.mobilepaymentchina.com/mobilepayment/.

[74] Ramesh Singh, Preeti Bhargava, and Samta Kain. Cellphone cloning: a perspective on gsm security. *Ubiquity*, 2007.

[75] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 2009.

[76] Ruud Bolle, Sharath Pankanti, and Anil K. Jain. *Biometrics, Personal Identification in Networked Society: Personal Identification in Networked Society*. Kluwer Academic Publishers, 1998.

[77] Michel Barbeau, Jyanthi Hall, and Evangelos Kranakis. Detecting impersonation attacks in future wireless and mobile networks. In *MADNES*, 2005.

[78] Richard Chow, Markus Jakobsson, Ryusuke Masuoka, Jesus Molina, Yuan Niu, Elaine Shi, and Zhexuan Song. Authentication in the clouds: A framework and its application to mobile users. In *CCSW*, 2010.

[79] Markus Miettinen and Nadarajah Asokan. Towards security policy decisions based on context profiling. In *Proceedings of the 3rd ACM workshop on Artificial intelligence and security*, 2010.

[80] Markus Jakobsson and Karl-Anders Johansson. Retroactive detection of malware with applications to mobile platforms. In *HotSec*, 2010.

[81] D. Wagner, B. Schneier, et al. Analysis of the ssl 3.0 protocol. In *The Second USENIX Workshop on Electronic Commerce Proceedings*, pages 29–40, 1996.

[82] Near field communication. http://www.nfc-forum.org/home/.

[83] Marvin McNett. Wireless topology discovery. http://my.url.com/, 2008.

[84] Nathan Eagle and Alex (Sandy) Pentland. CRAWDAD data set mit/reality (v. 2005-07-01). Downloaded from http://crawdad.cs.dartmouth.edu/mit/reality, July 2005.

[85] M. McNett and G.M. Voelker. Access and mobility of wireless pda users. *ACM SIGMOBILE Mobile Computing and Communications Review*, 9(2):40–55, 2005.

[86] S. Kosta, A. Mei, and J. Stefa. Small world in motion (SWIM): Modeling communities in ad-hoc mobile networking. In *IEEE SECON*, 2010.

[87] J. Leguay, A. Lindgren, J. Scott, T. Riedman, J. Crowcroft, and P. Hui. CRAWDAD trace upmc/content/imote/cambridge (v. 2006–11–17), November 2006.

[88] S. Kosta, C.V. Perta, J. Stefa, P. Hui, and A. Mei. Clone2clone (c2c): Enable peer-to-peer networking of smartphones on the cloud. Technical Report TR-SK032012AM, T-Labs, Deutsche Telekom, 2012. url: http://www.deutsche-telekom-laboratories.de/∼panhui/publications/clonedoc.pdf.

[89] K. Kumar, J. Liu, Y.H. Lu, and B. Bhargava. A survey of computation offloading for mobile systems. *Mobile Networks and Applications*, pages 1–12, 2012.

[90] Hoang T. Dinh, Chonho Lee, Dusit Niyato, and Ping Wang. A survey of mobile cloud computing: architecture, applications, and approaches. *Wireless Communications and Mobile Computing*, 2012.

[91] Welcome to the post-pc era. Coding of Horror: http://www.codinghorror.com/blog/2012/03/welcome-to-the-post-pc-era.html.

[92] TechCrunch: http://techcrunch.com/2012/09/12/the-post-pc-era-apple-announces-steller-ipad-sales-numbers-84m-sold/.

[93] J. Nieh, S.J. Yang, and N. Novik. A comparison of thin-client computing architectures, 2000.

[94] E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl. Maui: Making smartphones last longer with code offload. In *Proc. of MobiSys '10*, 2010.

[95] B. G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti. Clonecloud: elastic execution between mobile device and cloud. In *Proc. of EuroSys '11*, 2011.

[96] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang. Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In *INFOCOM, 2012 Proceedings IEEE*, pages 945–953. IEEE, 2012.

[97] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. The case for vm-based cloudlets in mobile computing. *Pervasive Computing, IEEE*, 8(4):14 –23, oct.-dec. 2009.

[98] E.Y. Chen and M. Itoh. Virtual smartphone over ip. In *World of Wireless Mobile and Multimedia Networks (WoWMoM)*, pages 1–6. IEEE, 2010.

[99] Elina Vartiainen and Kaisa Väänänen-Vainio-Mattila. User experience of mobile photo sharing in the cloud. In *Proceedings of the 9th International Conference on Mobile and Ubiquitous Multimedia*, 2010.

[100] Flickr. http://www.flickr.com/.

[101] Facebook. http://wwww.facebook.com.

[102] Google music. http://music.google.com.

[103] icloud. https://www.icloud.com/.

[104] V. Ottaviani, A. Lentini, A. Grillo, S. Di Cesare, and G.F. Italiano. Shared backup & restore: Save, recover and share personal information into closed groups of smartphones. In *Proc. of IFIP NTMS 2011*, 2011.

[105] C. Ai, J. Liu, C. Fan, X. Zhang, and J. Zou. Enhancing personal information security on android with a new synchronization scheme. In *Proc. of WiCOM 2011*, 2011.

[106] Eemil Lagerspetz and Sasu Tarkoma. Mobile search and the cloud: The benefits of offloading. In *PerCom Workshops*, 2011.

[107] Jon Oberheide, Kaushik Veeraraghavan, Evan Cooke, Jason Flinn, and Farnam Jahanian. Virtualized in-cloud security services for mobile devices. In *Proceedings of the First Workshop on Virtualization in Mobile Computing*, MobiVirt '08, 2008.

[108] Georgios Portokalidis, Philip Homburg, Kostas Anagnostakis, and Herbert Bos. Paranoid android: Versatile protection for smartphones. In *Proceedings of the 26th Annual Computer Security Applications Conference (ACSAC)*, Austin, Texas, December 2010.

[109] K. Kumar and Y. H. Lu. Cloud computing for mobile users: Can offloading computation save energy? *IEEE Computer*, 43(4):51–56, April 2010.

[110] Y. Wen, W. Zhang, and H. Luo. Energy-optimal mobile application execution: Taming resource-poor mobile devices with cloud clones. In *Proc. of IEEE INFOCOM 2012*, 2012.

[111] A.P. Miettinen and J.K. Nurminen. Energy efficiency of mobile clients in cloud computing. In *Proc. of HotCloud 2010*, 2010.

[112] Chanmin Yoon, Dongwon Kim, Wonwoo Jung, and Chulkoo Kang. Appscope: Application energy metering framework for android smartphone using kernel activity monitoring. In *Proc. of USENIX ATC 12*, 2012.

[113] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R.P. Dick, Z.M. Mao, and L. Yang. Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In *Proc. of IEEE/ACM/IFIP CODES/ISSS '10*, 2010.

[114] Busybox. http://www.busybox.net/.

[115] A. Tridgell and P. Mackerras. The rsync algorithm. Technical Report TR-CS-96-05, The Australian National University, 1996.

[116] Rsync. http://rsync.net/.

[117] Are smart phones spreading faster than any technology in human history? Technology Review: http://www.technologyreview.com/news/427787/are-smart-phones-spreading-faster-than-any/2/, May 2012.