CloudShield: Efficient Anti-Malware Smartphone Patching with a P2P Network on the Cloud

Marco V. Barbera, Sokol Kosta, Julinda Stefa, Pan Hui, and Alessandro Mei

***Abstract*—The battery limits of today smartphones require a solution. In the scientific community it is believed that a promising way of prolonging battery life is to offload mobile computation to the cloud. State of the art offloading architectures consists of virtual copies of real smartphones (the *clones*) that run on the cloud, are synchronized with the corresponding devices, and help alleviate the computational burden on the real smartphones. Recently, it has been proposed to organize the clones in a peer-to-peer network in order to facilitate content sharing among the mobile smartphones. We believe that P2P network of clones, aside from content sharing, can be a useful tool to solve critical security problems on the mobile network of smartphones. In particular, we consider the problem of computing an efficient patching strategy to stop worm spreading between smartphones. The peer-to-peer network of clones is used to compute the best strategy to patch the smartphones in such a way that the number of devices to patch is low (to reduce the load on the cellular infrastructure) and that the worm is stopped quickly. We consider two well defined worms, one spreading between the devices and one attacking the cloud before moving to the real smartphones; we describe CloudShield, a suite of protocols running on the peer-to-peer network of clones; and we show by experiments that CloudShield outperforms state-of-the-art worm-containment mechanisms for mobile wireless networks.**

***Index Terms*—Cloud computing, smartphones, P2P.**

I. Introduction

The number of mobile apps available for smartphones has grown exponentially in the last years. They are distributed by online stores such as the App Store for the iPhone and the Android Market for Android systems. The App Store makes a number of checks before making the applications available for download. Of course, the checks give some reasonable confidence that the applications run correctly but does not guarantee that they are immune to viruses and malware. The Android Market is using a different strategy that helps Android spread faster—the online store is open without particular limits or quality checks to application developers that want to distribute and advertise their applications. Clearly, this makes Android an even easier target to viruses and malware.

Another strong and very recent trend in mobile computing is to offload computation of heavy mobile applications on the cloud. This is useful in many ways. Offloading computation from mobiles on the cloud helps mitigate the well known energy problems of modern smartphones. Recent work such as MAUI [4], CloneCloud [3], ThinkAir [5], and SociableSense [10] show how method-based or application partitioning-based offloading techniques to the cloud drastically improve computation efficiency and prolong battery life of mobile devices. More recently, in the C2C platform [6] every smartphone is associated with a software clone on the cloud and the clones are interconnected in a peer-to-peer fashion exploiting the networking service within the cloud. C2C seems like a perfect candidate to prevent malware spread. Indeed, the peer-to-peer network of clones on the cloud can be used both to check newly installed applications and to monitor virus spreading on the mobile devices.

In this paper we advocate the idea that C2C, a peer-to-peer network of virtual smartphone clones running on the cloud, can help stop worm attacks spreading from smartphone to smartphone on the mobile network. The worm propagates by using bluetooth connection, mms messages, phone calls, or any other means of infection available among smartphones. We work under the assumption that the links of the peer-to-peer network of the clones reflect the sociality among the real smartphone users (this is easily done since every clone on the cloud synchronizes with its corresponding real device). The first problem that we consider in this paper is to devise a mechanism to patch the smartphones in such a way that the number of devices patched is low, to make the scheme cheap, and that the probability of stopping the worm is high. As a solution, we propose CloudShield—a suite of schemes that cope with worm spread on cellular networks by using a peer-to-peer network of clones on the cloud to compute the patching strategy. The idea behind CloudShield is as follows: It selects few "important" clones in the cloud to patch; in turn, the patched clones transmit the patch to the respective smartphones by thus effectively containing the worm spread in the cellular network. We test our strategies on two different datasets—Facebook [11] and Live Journal [1], [7])—and compare them to the state-of-the-art worm-containment schemes on dynamic social networks [12], [9]. The experimental results show that our approaches outperform both [12], [9] in yielding lower infection rates after patching a smaller number of nodes.

Then, we consider the possibility that the weak link is the cloud itself. We assume that a worm attacks the p2p network of clones on the cloud with the goal of spreading on the smartphones. In this case, the attack is probably faster and we assume that the patch is not yet available. We propose a two phase solution: At detection time, we make the clones weaken the incoming links from their peers in the cloud, so

Marco V. Barbera, Sokol Kosta, Julinda Stefa, and Alessandro Mei are with the Department of Computer Science, Sapienza University of Rome, Italy. Email {barbera,kosta,stefa,mei}@di.uniroma1.it. Pan Hui is with Telekom Innovation Laboratories T-Labs, Berlin, Germany. Email pan.hui@telekom.de.

Alessandro Mei is supported by a Marie Curie Outgoing International Fellowship funded by the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement n. 253461.

that the worm is contained as much as possible. Then, once the patch is released, we apply our CloudShield schemes to select effective patching sets, patch the respective clones, and finally, reset the capacity of the links. Our experimental results show that this combined strategy yields infection rates reduced with up to 20% of the nodes.

II. System model and motivation

In our system each cellphone is associated with a software clone in the cloud [3], [6]. The clone runs the same operating system and apps as the real device. The device sends updates to and gets updates from the clone whenever new data is generated and a new application is installed. In addition, the clones are connected to clones of other "friend" devices on the cloud. Friendship is determined by the rate of communication between the real devices: Smartphones that call/text/email each other frequently have the respective clones connected in the P2P networks on the cloud [6].

Previously presented infrastructure-based worm containment schemes consider worms that propagate smartphone-to-smartphone. They aim at individuating a set of crucial smartphones to patch so to contain the worm spread in the network. From the other side, proximity-based schemes distribute the patch along efficient but slow blue-tooth paths. In this case as well, flooding the network with the patch may not be an option since patches are usually large files and smartphones are battery limited devices. As we already discussed in the related work section, both schemes try to exploit the social relationships among the users to better select fewer and more effective nodes to patch. Here we take a similar approach, but on the P2P network of clones on the cloud: We send the patch to few, effective clones, that in turn transmit it individually to the respective devices. Recall that the clones are up-to-date with statistics on the communication of the real devices.

However, the P2P network of clones can potentially introduce a new attack to the real devices: A virus/worm that infects a clone, either during a synchronization with the (infected) device or during a communication with an (infected) friend clone, can propagate with enormous speed exploiting the P2P links in the cloud. The infection is then directly propagated to the real devices as soon as they communicate with their clones. One might think that this attack is mitigated if clone B does not accept digital content or does not install any new applications suggested by friend clones without consulting the user first. However note that users get bored of systems with features that require their intervention. Mr. Clippit for example, which was Microsoft's office assistant, was cut off the system because it annoyed the users with too many questions and suggestions[1]. Moreover, experience has taught us that people tend to blindly agree with settings that allow automatic download and installation of software suggested by already installed software or updates. Lastly, but probably most importantly, even if the user has to intervene and make a decision every time a friend clone suggests its clone to install an app, she will probably accept the suggestion. Past experience has shown that phenomena such as viral marketing in social networks are successful because people tend to follow suggestions from their friends almost blindly. This was the case of the Koobface worm that spread on Facebook in 2008: It sent infected links[2] to the list of friends of the victims.

III. The Methodology

We consider the P2P network of clones to be a graph $G = (V, E)$, where V is the set of clones and E is the set of edges representing the communication links among them. The links reflect the frequency of communication between the devices in the real world. Graph G is directed and weighted. The weights are derived from the frequency of calls/texts/emails/sharing of files through blue-tooth between the two devices in the real world.

Unfortunately we cannot replicate the experiments done in [12], since the dataset of cellphone calls used in that work is not publicly available. Thus, we use two social-network datasets that are available and that can be used to replicate the experiments: (1) *FB* [11], a large Facebook subgraph of 63'392 nodes used also in [9], and (2) *LJ* [1], [7], a directed graph representation of the LiveJournal social network of 4'847'571 nodes. We generate the Facebook's dataset oriented version by transforming each friendship link among nodes u and v the two directed edges (u, v) and (v, u) of the same weight.

The FB dataset is enriched with information on over 1.5M wall posts between users for the period September 2006–November 2009. Conversely, the LJ dataset does not include information of user posts. Note though that the direction of the edge (u, v) on the Live Journal social network means that v has subscribed to $u's$ journal, and thus, gets all the posts published by u. So, in this case, we suppose that the weight of all the out-links of a node is 1.

A. Worm propagation model

We use the same worm propagation model used in [9], [12]: The worm is able to explore the social strength of the communication links among nodes. Once it infects a node, it tries to expand to its friends by exploiting communication opportunities from between the two to send infected files or suggest installation of malware apps. Thus, the probability of the actual infection happening depends on both the communication frequency (link weight) and on how likely is it that the possible victim accept suggestions sent by the infected neighbor. This factor is related to the level of trust that the possible victim has towards the infected node: We are more likely to follow links included in emails or messages received by our close friends (which we trust more) than from strangers.

We measure the trust level of a node v towards its neighbor u as follows:

$$\tau_{v,u} = \frac{|OF_u \cap OF_v|}{|OF_v|}. \quad (1)$$

The intuition behind the above equation is that we deem as more valuable (and thus trust more) people that have many

[1] http://www.guardian.co.uk/media/2001/apr/11/advertising2

[2] http://gawker.com/5103848/why-the-koobface-virus-spread-so-fast

friends in common with us. Then, the probability that a worm spreads from node u to node v through link (u, v) becomes:

$$p_{(u,v)} = w_{(u,v)} \cdot \tau_{v,u}, \quad (2)$$

where $w_{(u,v)}$ is the weight of the directed edge (u, v). Again as in [9], [12] we assume that the time that the worm takes to spread from an infected peer to another is inversely proportional to the communication frequency between the peer and this specific out-friend.

B. The CloudShield Scheme

We base our solution on the idea that socially-important nodes are typically very influential in terms of information spreading in general, and in worm spreading in particular. This is due to a combination of two factors: Their position in the network and the large number of links. Moreover, we want the solution that implements this idea to be simple and computationally efficient in finding the subset of nodes to be patched. The simplest it is, the less it costs to deal with the dynamics of the network (links that appear and disappear). With this in mind we build the following three versions of our CloudShield solution based on different methodologies on selection of nodes to patch:

1) *Page-rank CloudShield (PR-CS)*: It deems as important nodes with high page-rank in the network. The intuition behind this is as follows: A node is at risk of infection if it interacts frequently with other nodes with high risk of infection. Of course, this is very similar to PageRank [2].
2) *Degree CloudShield (DG-CS)*: It deems as important nodes with high out-degree weight in the network. Intuitively, once a node gets infected, the highest its out-degree, the highest its contribution in further spreading the worm to other nodes.
3) *Greedy Degree CloudShield (G-DG-CS)*: In social networks, the nodes with highest degree tend to cluster. Following this intuition, we present G-DG-CS, where, similarly to DG-CS, nodes with high out-degree are candidates to be patched first. However, the selection is different: After the highest out-degree weight node is chosen, all its in-links are dumped and the out-degree weight of its incoming friends is updated.

As we will see from our experimental results, our simple schemes outperform previous ones that require complicate computation of network clustering or community substructures (e.g. [12], [9]). This is due to the fact that the worm exploits the social links/paths in infecting the network. The more a scheme manages to "destroy" such links by patching crucial nodes, the more the structure of the network changes, as far as the worm is concerned. Most importantly, our schemes require little computation and can be even computed in a distributed way—without the need of an authority—at the cost of inducing some traffic overhead on the C2C network. This makes it easier to handle network dynamics such as insertion or deletion of edges (new social relationships that start or old ones that end).

IV. Experimental Results

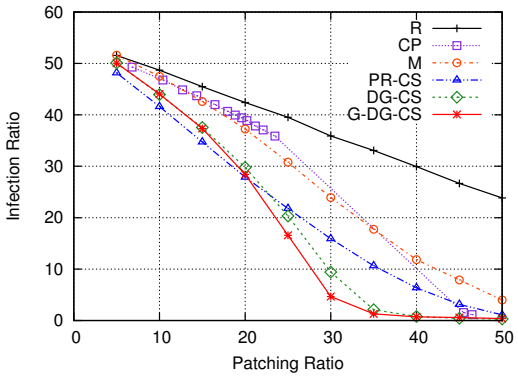
To validate our CloudShield patching methods we compare them with the states of the art in terms of infrastructure- based patching schemes for cellular networks: Clustered Partitioning (CP) [12] and Community-based partitioning (M) [9]. To the comparison we also add Random Partitioning, used as a benchmark in [12].

A. Worm attack model and patching threshold

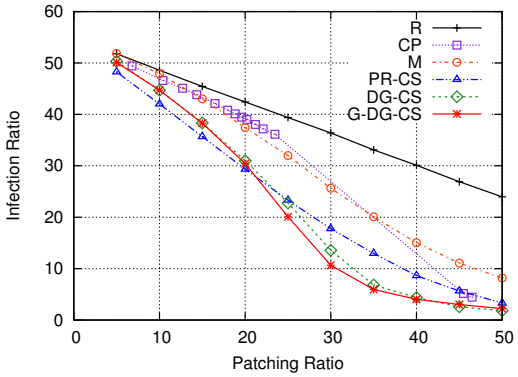
To model the worm attack at its initial phase we follow the method in [12], [9]: We first induce the infection to a small number of users (0.02% of nodes), randomly and uniformly chosen on the network. This is to simulate the initial worm sources during the early stages of the infection. The worm starts spreading in the system till it reaches a certain number of infection rate (percentage of nodes infected), given by a patching threshold parameter α. This parameter represents the timespan between the very first infection phase and the moment in which the worm is detected and the patch is generated. Then, we apply the patches to nodes selected by any of the above schemes. The simulation finishes when the worm does not expand any further in the network. The performance of each scheme is measured by the infection rate reached by the worm as a function of the number of nodes patched with each of the patching schemes. Each experiment is run 1000 times and the results are averaged. As far as CP [12] is concerned, since it is impossible to derive the number of nodes to patch that each k value yields, we compute the patching sets for every possible value k.

B. Stopping the worm on the cellular network

Here we assume the first attack: The worm is spreading in the cellular network, and does not attack the cloud. This scenario is justified by the large number of existing worms that spread between smartphones by using bluetooth, wi-fi, email, mms, and so on. For each scheme we study the rate of the infection as a function of the patching ratio. This is done for two values of patching threshold α: 2% and 10%. The results for both FB and LJ are shown respectively in Figures 1 and 2. Let us start from the FB dataset. First we note that our three schemes have the best performance. For patching ratios up to 20%, PR-CS outperforms both DG-CS and G-DG-CS. This is because the Facebook graph is very dense. As a result, DG-CS tends to send the patch to nodes that are "close" in the graph. When the values of patching ratio increase, we get a different scenario: For patching thresholds in the interval $[20\%, 35\%]$, G-DG-CS becomes the most performing. This is due to the fact that the higher is the number of the nodes patched, the larger is the number of edges dropped by the G-DG-CS scheme. This procedure makes so that very well connected clusters in the network get "destroyed" soon after a few cluster members are patched, and the scheme starts therefore selecting nodes in other parts of the network by better distributing the patches on the cloud. That said, this procedure eventually ends up "destroying" all the strongly connected clusters of the network: The graph results so sparse that the further selection choices

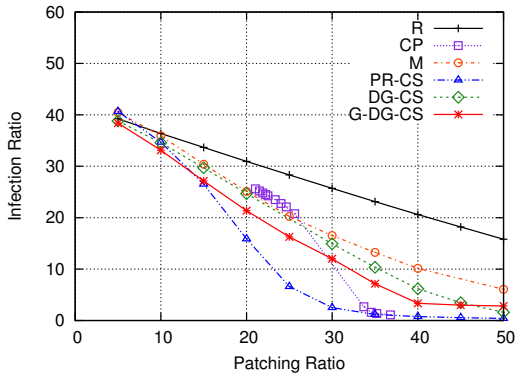


(a) $\alpha = 2\%$

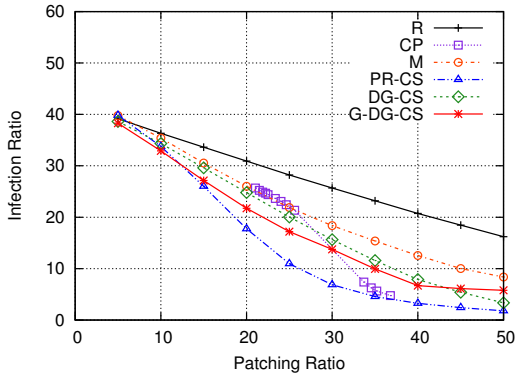


(b) $\alpha = 10\%$

Fig. 1. Infection rate vs the percentage of patched nodes for the various schemes on the FB dataset. **"R"** denotes the random scheme; **"CP"** denotes the Clustering Partition; **"M"** denotes the Community-Based scheme; **"PR-CS"**, **"DG-CS"**, and **"G-DG-CS"** denote respectively the Page-Rank, Degree and Greedy Degree CloudShield schemes.



(a) $\alpha = 2\%$



(b) $\alpha = 10\%$

Fig. 2. Infection rate vs the percentage of patched nodes for the various schemes on the LJ dataset. **"R"** denotes the random scheme; **"CP"** denotes the Clustering Partition; **"M"** denotes the Community-Based scheme; **"PR-CS"**, **"DG-CS"**, and **"G-DG-CS"** denote respectively the Page-Rank, Degree and Greedy Degree CloudShield schemes.

do not impact much the infection rate. This is why for larger patching thresholds (from 40% and on) the simple degree scheme (DG-CS) yields better results.

Now, let us consider the LJ dataset. The graph is much larger and sparser than FB and the distribution of the out-degrees of the nodes in LJ decays faster than that of FB. So, high out-degree nodes are not only very well connected between them, but also are very central to the graph. Indeed, the average clustering coefficient for the LJ dataset is higher than that of the FB dataset (respectively 0.3123 for LJ and 0.22 for FB). So, the node sets chosen by our degree based schemes here tend to be even more clustered, thus yielding lower performance with respect to our PR-CS scheme, which is the best for the DJ dataset. Also note that the features of the LJ dataset make so that for low values of patching ratios the CP scheme performs very badly. This is due to the fact that CP yields very large partitions. So, even though CP "cuts" the bridge edges between clusters, it does not manage to stop the worm infecting the many nodes of a given partition.

Note that for higher values of patching threshold all the schemes perform worse. As the worm has already gotten to many nodes, it is very difficult to effectively contain it with the same number of patched nodes by all the schemes.

C. *Stopping the worm on the cloud*

We consider the second scheme in which the worm manages to break down the security of the cloud and overtake the clones. This scenario considers a worm that might exist in the future and that could try to exploit the p2p network of clones in the cloud. It firstly infects a subset of cloud clones, again considered to be as small as the 0.02% of the whole network. Then, it starts propagating towards other clones, exploiting the p2p cloud social links among them. As soon as it manages to infect the maximum number of clones possible, it makes the infected clones transmit the worm to the respective cellphones.

In this scenario our schemes contain the worm with the same efficiency as the previous one. However, here we are dealing with a stronger worm, for which the patch might be more difficult and complicated to get. What's most, even if the patch is released early, no one guarantees that it is installed in time by users—recall the example of the SQL Slammer malware [8], for which the patch was released 6 months earlier than the attack time, that still managed to infect more than 75'000 users that had not installed the patch in time. With this in mind, we introduce another parameter in our system, the detection threshold δ, representing the

Dataset	δ		
	2%	5%	10%
FB	$40\%\tau$	$30\%\tau$	$25\%\tau$
LJ	$50\%\tau$	$50\%\tau$	$45\%\tau$

TABLE I
VALUES OF QUARANTINE TRUST τ_q THAT ALLOW THE VIRUS TO EXPAND TO INFECTION RATE $\alpha = 20\%$. τ DENOTES THE INITIAL TRUST VALUE.

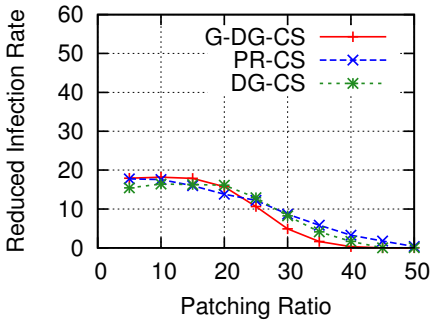


Fig. 3. Reduction of the infection rate on the FB dataset for the various schemes when introducing the quarantine phase ($\delta = 10\%$). **"PR-CS"**, **"DG-CS"**, and **"G-DG-CS"** denote respectively the Page-Rank, Degree and Greedy Degree CloudShield schemes.

fraction of nodes infected at the moment in which the attack is detected. We improve the efficiency of our approach in worm containment in the following way: When the attack is detected, the clones enter in a "quarantine" state, during which they became more cautious and trust less their incoming friends. This assumption is indeed realistic: When users know that a worm is spreading, we pay more attention on what links we follow and on what software we install. To simulate this behavior in the p2p network of clones we make each healthy clone diminish the trust τ towards its incoming friends. The goal of the quarantine phase is to contain the worm as much as possible from spreading, till the patch is released.

We study the impact of the quarantine phase on our schemes by fixing the patching threshold to 20% and the detection threshold δ to 10%. We run the following experiment: When an infection rate of δ is reached, clones diminish their incoming links trust from τ to τ_q until the worm infects a rate of $\alpha = 20\%$ of the network nodes. Then, we patch the nodes according to each of our schemes (PR-CS, DG-CS, and G-DG-CS), restore the trust to its initial value, and wait till the worm propagation is stopped. Each experiment is run 1000 times and the results are averaged. Figures 3 and 4 present the reduction of the infection rate for each scheme when coupled with the quarantine phase, for three different values of detection rates δ. As can be noticed from the figures, the gain is very high for low patching rates (up to 20% on the FB dataset), and it lowers with the increase of the patching ratio.

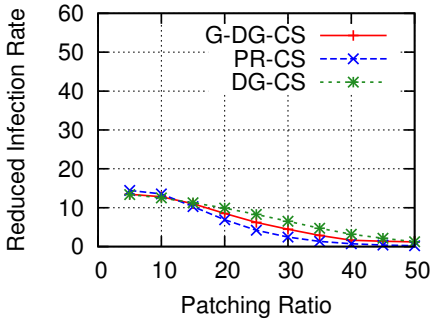


Fig. 4. Reduction of the infection rate on the LJ dataset for the various schemes when introducing the quarantine phase ($\delta = 10\%$). **"PR-CS"**, **"DG-CS"**, and **"G-DG-CS"** denote respectively the Page-Rank, Degree and Greedy Degree CloudShield schemes.

V. CONCLUSION

In this paper we advocate the use of a p2p network of smartphones clones on the cloud as a mechanism to worm containment. We consider the problem of stopping worms on the mobile network of smartphones, and then we consider the problem of stopping a worm spreading on the p2p network. We introduce simple mechanisms that can be computed quickly by the P2P network of clones and that outperform the state of the art on worm containment for mobile cellular networks.

REFERENCES

[1] L. Backstrom, D. Huttenlocher, J. Kleinberg, and X. Lan. Group formation in large social networks: membership, growth, and evolution. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '06, 2006.

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

[3] 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.

[4] 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.

[5] 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 *Prod. of IEEE INFOCOM 2012*, 2012.

[6] S. Kosta, C. 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.

[7] J. Leskovec, K. Lang, A. Dasgupta, and M. Mahoney. Community Structure in Large Networks: Natural Cluster Sizes and the Absence of Large Well-Defined Clusters. *Internet Mathematics*, 6(1):29–123, 2009.

[8] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver. Inside the slammer worm. *IEEE Security & Privacy*, 1(4):33–39, August 2003.

[9] N. Nguyen, Y. Xuan, and M. Thai. A novel method for worm containment on dynamic social networks. In *Military Communications Conference (MILCOM 2010)*, 2010.

[10] K. K. Rachuri, C. Mascolo, M. Musolesi, and P. J. Rentfrow. Sociablesense: Exploring the trade-offs of adaptive sampling and computation offloading for social sensing. In *Proc. of Mobicom '11*, 2011.

[11] B. Viswanath, A. Mislove, M. Cha, and K. P. Gummadi. On the evolution of user interaction in facebook. In *ACM SIGCOMM Workshop on Social Networks*, 2009.

[12] Z. Zhu, G. Cao, S. Zhu, S. Ranjan, and A. Nucci. A social network based patching scheme for worm containment in cellular networks. In *INFOCOM 2009, IEEE*, 2009.