A Social Network Based Patching Scheme for Worm Containment in Cellular Networks

Zhichao Zhu*, Guohong Cao*, Sencun Zhu*, Supranamaya Ranjan† and Antonio Nucci†
*Department of Computer Science and Engineering, The Pennsylvania State University, University Park, PA
†Narus Inc, Mountain View, CA
* {zzhu,gcao,szhu}@cse.psu.edu † {soups, anucci}@narus.com

***Abstract*—Recently, cellular phone networks have begun allowing third-party applications to run over certain open-API phone operating systems such as Windows Mobile, Iphone and Google's Android platform. However, with this increased openness, the fear of rogue programs written to propagate from one phone to another becomes ever more real. This paper proposes a counter-mechanism to contain the propagation of a mobile worm at the earliest stage by patching an optimal set of selected phones. The counter-mechanism continually extracts a social relationship graph between mobile phones via an analysis of the network traffic. As people are more likely to open and download content that they receive from friends, this social relationship graph is representative of the most likely propagation path of a mobile worm. The counter mechanism partitions the social relationship graph via two different algorithms, balanced and clustered partitioning and selects an optimal set of phones to be patched first as those which have the capability to infect the most number of other phones. The performance of these partitioning algorithms is compared against a benchmark random partitioning scheme. Through extensive trace-driven experiments using real IP packet traces from one of the largest cellular networks in the US, we demonstrate the efficacy of our proposed counter-mechanism in containing a mobile worm.**

I. Introduction

Cellular phone networks are increasingly receptive to open-API operating systems such as Windows Mobile, Iphone and Google's Android running over mobile phones in the networks. While this openness would allow richer applications to run over mobile phones, it also makes it easier for hackers to write malicious software that can take control of a mobile device by exploiting its vulnerabilities or that of the applications running on top of it. In this regards, cellular networks may witness a similar evolution of worms as has been seen in the wired world. Further, mobile worms could impose unwarranted bandwidth charges to customers, deterioration in quality of service, and ultimately loss of revenue for service providers.

The usual ways for mobile worms to propagate include Bluetooth [13] interface and Multimedia Messaging Service (MMS) [19] interface. One Bluetooth based mobile worm is Cabir [9], which can spread through Bluetooth connection to other Bluetooth-enabled devices it can find. As its name suggests, MMS messages are intended to contain media content such as photos, audios or videos, but they can also contain infected malicious codes. One noteworthy example is Commwarrior [10], which is the first worm that can propagate via MMS. It searches through a user's local address book for phone numbers and sends MMS messages containing infected files to other users in the address book.

The increasing popularity and unique property of MMS worms draws our focus on dealing with MMS worms in this paper. MMS worms could be sent out in just one click and travel to any mobiles all over the world with a larger chance of success in propagation, thus are potentially more virulent in terms of speed and area of propagation than Bluetooth worms. Note that worms that exploit plain-text Short Message Service (SMS) can not carry malicious payload, and hence usually only carry a URL in the message, from where the victim is lured to download the payload, e.g., the worm Symbos/Feak [8]. We consider worms that exploit SMS as similar to MMS in the way they spread (via address books or call records) and hence our methodology developed here would be applicable to both techniques.

Due to characteristics of slow start and exponential propagation exhibited by mobile worms, it is challenging to detect a worm outbreak at the early stage while it is hard to mitigate it at a later stage. However, even if network operators are unable to detect a worm propagation during the earliest stage, they still have a window of opportunity to react before the worm spreads to a larger population. This is especially true in mobile worm in which user interactions are required to download and install the malicious files on mobile devices. Therefore, unlike automatic Internet worms [22] which only take hours to infect millions of users, it usually takes much longer for mobile worms to spread to a severe level. In this paper, we focus on the methodology by which a mobile network operator would distribute a patch to arrest a worm's propagation before it causes complete network infection.

Patch propagation techniques have been developed for delivering worm signatures in the wired Internet [25]. However, such solutions are not directly applicable to mobile networks which have a unique constraint of lower data rates. In such a bandwidth-constrained environment, patches can not be propagated by a network operator to *all* devices at the same time. Moreover, the patch would have to compete with the bandwidth already being consumed by a propagating worm. Existing work on modeling and containment of worms in a mobile network [21] [3] [1] [11], do not take into account the unique capability of mobile worms to spread by using the social network of users by exploiting their address book or recent call records. In lieu of above observations, we take a hierarchical approach towards patching mobile devices such that those devices which act as a 'bridge' between social clusters within the network are patched first. The intuition being that such devices once infected have the ability to infect entire social clusters and hence they must be patched first.

In this paper, we propose a new approach to contain MMS worms within a limited range at the earliest stage. We divide the mobiles in cellular networks into multiple partitions based on the social relationships between mobiles retrieved from a real cellular network trace. Mobiles in each partition closely interact with each other while mobiles across different partitions are less related. Security patches are distributed to key nodes that separate individual partitions to block the worm propagation from one partition to another. More specifically, the contributions of this paper are three-fold:

- We construct a social relationship graph of mobile devices by extracting their communication patterns based on a network trace. This graph describes the social relationships between mobile phones which are usually exploited by mobile worms for spreading.
- We propose a new containment strategy for MMS worms by partitioning the mobiles appropriately based on the social relationship graph. Two partitioning algorithms: **balanced partitioning** and **clustered partitioning** are proposed and their performance is evaluated.
- We experimentally compare our targeted patching algorithms (balanced and clustered) against a benchmark uniformly random patching strategy. Our experiments show the efficiency of targeted patching: both balanced and clustered patching algorithms achieve a lower infection rate than the random strategy while patching a significantly smaller number of nodes.

The rest of this paper is organized as follows. Section II presents motivations behind the trace-driven partitioning approach. Section III describes how this social relationship graph can be built by using a network traffic trace. Section IV introduces the graph partitioning theory and two corresponding patching schemes. Section V evaluates the performance of our worm containment strategy. Finally, Section VI concludes the paper and provides future research directions.

II. Motivation

Mobile worms that spread using MMS [10] or SMS [8] typically exploit the social network of users to propagate from one mobile device to another. These worms search through a user's local address book and recent call records for phone numbers and send messages to other users. Note that randomly scanning does not work on mobile worm environment, as any malicious message from an untrusted stranger would not be opened and activated. In the case of MMS, the message itself could be the malicious payload, while in the case of SMS, the user would be lured to download the payload from a URL. A victim mobile receiving this message will most likely open and download the message since he believes it comes from someone he knows and trusts. Thus, an effective worm containment approach must take in to account the social relationship graph between mobile devices in a cellular network. By figuring out the social interactions between mobile devices, i.e. which devices are more likely to exchange messages with each other, we can predict the propagation path of such mobile worms. In this way the vulnerable mobiles or connections could be marked and be protected.

Given that there has not been any instance of mobile worm that has propagated far and wide across a cellular network 'in the wild' as yet, there is limited knowledge of propagation paths of mobile worms. In this regards, we make the following assumption, that the propagation path of a mobile worm can be approximated by the social network of mobile devices. Given that a user *Joe* has a higher probability to open and download a message from *Jane* with whom he periodically exchanges messages, this pair of users, *Joe-Jane* would be considered more vulnerable. In contrast, if *Joe* doesn't exchange messages with *Mary*, he is unlikely to be infected by a worm sent by *Mary* and hence the pair of users, *Joe-Mary* is considered less likely to be included in the worm's propagation path. In summary, we use the amount of traffic exchanged between two mobile devices as an indicator of whether this pair of devices would be present in a worm's propagation path. This propagation model would be reflective of worms that spread by exploiting the call records of infected hosts. Such a social relationship graph can be accurately built by a mobile network operator by looking at the call and messaging records at which the operator stores for billing purposes. Even for mobile worms which spread by using the address book of an infected host, the social relationship graph built by using the call and messaging records would be reflective of the propagation path of the worm, which is similar for worms which spread by randomly generating a hit list of potential devices. This is on account of the fact that humans are much more likely to open and download a message from someone with whom they have communicated in the past.

Our worm containment strategy would be implemented at a mobile service provider's messaging gateways or base-station controllers. Service providers typically store records of all traffic generated by a user per session for billing and accounting purposes. We use an anonymized trace from one of the largest cellular network providers over a two week period in April 2008. The trace summarizes the total amount of traffic generated by every user for a variety of applications such as SMS, MMS, SIP based VoIP, Push-To-Talk and so on. We use all traffic exchanged between a pair of devices regardless of application types, as an indicator of their likelihood to infect each other.

We use the social relationship graph to decide on an effective patch distribution strategy. A mobile that receives a patch becomes immune to the worm and could then be used to propagate the patch further. However, as we will discuss in Section IV, disseminating patches to all mobiles may not be a practical method due to the time and bandwidth limits. Thus, a faster way of patch dissemination, or an appropriate order of patch distribution is needed. Intuitively, the one with the highest risk to be infected or the one with the highest probability to infect others should have the highest priority for security upgrades. Under our partitioning based approach, security patches need not reach all the mobiles if the worm could be contained in each small partition. Therefore, only those key nodes that separate the graph into individual partitions should be patched in the first place. We next discuss how to determine this set of key nodes.

III. Trace-driven Social Relationship Graph

In this section, we describe how a service provider can construct a social relationship graph by using an example traffic trace collected at the network layer at one of the largest mobile phone networks in the US. The endpoints present in the trace were anonymized while preserving the uniqueness of the identifiers of ip-addresses and phone numbers involved. The trace provides a session-level information for traffic (bytes and packets) exchanged between two endpoints per application over a two week period in April 2008. The trace contains information about 2 million users across 65000 base station cells all over the US. According to this trace, about 35% of users in this network exchange about 0.4 million MMS messages every day. Besides MMS, the trace also contains traffic volume information for SIP based VoIP sessions exchanged between users, SIP based Push-To-Talk and SMS.

Definition 1 (Cellular-Social Relationship Graph): An undirected weighted graph $G = (V, E)$ consists of a set of vertices V and a set of edges E, such that each vertex $u \in V$ denotes a mobile in the cellular network, while each edge $e(u, v)$ denotes that at least one traffic flow was exchanged between mobile u and v. Let d_u denote the degree of vertex u, $u \in V$ (the number of mobiles or vertices having a link with u). Let $m(u, v)$ denote the amount of traffic initiated from u to v. If there are functions f and g that map each vertex $u \in V$ and each edge $(u, v) \in E$ to a real number, then the graph is considered to be weighted with f and g determining the vertex-wights and edge-weights, respectively. The weight-mapping functions are as following:

$$f(u) = d_u \tag{1}$$

$$g(u, v) = m(u, v) + m(v, u) \tag{2}$$

An example social relationship graph is shown in Table I. We use the number of sessions exchanged between two mobiles u and v over one week as our weights $m(u, v)$. Alternatively, the total number of bytes or packets exchanged between two mobiles could also be used as the weights. For the sake of generalization, we count all sessions exchanged between two mobiles regardless of the application type, as all types contribute to the worm propagation patterns. Each entry in the table shows how many times any two mobiles communicated with each other every week on an average. We note this metric as WAT (weekly averaged traffic). If we abstract each mobile as a vertex and normalize WAT between any two mobiles by dividing the maximum WAT over the week, we get a relationship graph as Figure 1.

TABLE I
Communication Traffic Records

Between Phones	WAT	Between Phones	WAT
A and B	1	A and G	3
A and H	3	B and C	2
B and H	1	B and I	1
C and D	10	C and I	1
D and E	1.5	D and I	1.5
E and F	5	F and G	1
F and I	5	G and H	2
G and I	1		

The weights of vertices and edges together contribute to a *significance level* which represents the chance of being infected by worms. As can be seen from Equation (1), the weights of vertices depend on the node degrees. Intuitively, the mobile with the highest risk to be infected or infect others is the key node that a worm can use to spread and thus has the highest priority for containment. For MMS worms, a mobile with a higher in-degree means that it is more likely to be infected while a mobile with a higher out-degree is more likely to infect other mobiles. Therefore, those high-degree mobiles, either in-degree or out-degree, should be assigned a higher vertex weight and get higher priority for patching consideration. The in-degree and out-degree of a mobile are not necessarily dependent, but may be correlated. The phone number of the mobile that has large address book tends to appear in the address books of many others.

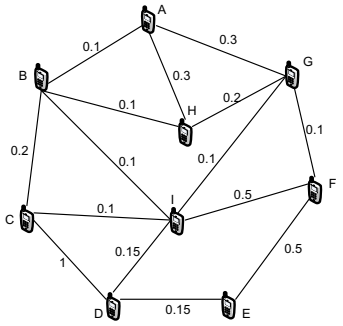


Fig. 1. Cellular Social Relationship Graph

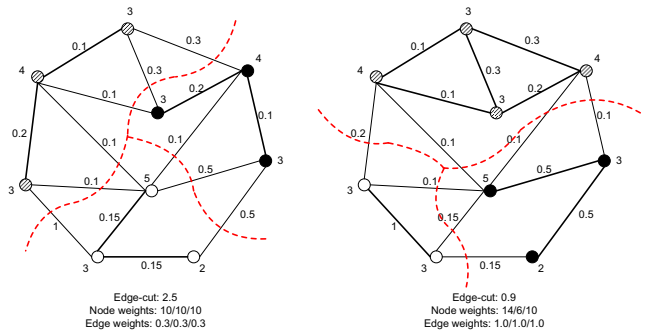
The social interactions [5] between mobiles can be used to explain Equation (2). Whenever there is a traffic record between two mobiles, they have a chance to be friends and therefore a larger probability to open and activate a worm message received from each other. This social relationship graph gives us an overview of how mobiles are related with each other and how worms might use these social relationships to propagate themselves.

We use weekly averaged traffic (WAT) to measure the relationship between two mobiles. According to what we have observed from the trace, although the number of interactions between two individuals behaves differently for weekday and weekend, the number of interactions across the two weeks remains similar. This result which is also confirmed by [5] shows that people's interaction rates are predictable on a weekly basis. Therefore, it is reasonable to use a weekly averaged traffic information to represent the interaction rate through a long period.

IV. Containing worms by Graph Theory

A. Uniform Patch and Targeted Patch

Most security patch providers such as F-Secure [7] use push-based strategy for patch distribution, that is, as soon as a new security patch is available, the notification of updates is sent to all subscribed users. Upon receiving the notification, users authenticate and verify the message, and then connect to a centralized database to download the patch updates promptly. This can be achieved by short messages through control channels. However, the time to disseminate patches to entire cellular networks could be in the order of hours or days, which is much longer than the worm propagation speed. Moreover,



(a) Balanced Graph Partitioning (b) Clustered Graph Partitioning

Fig. 2. Examples of Two Different Graph Partitioning Schemes

the bandwidth bottleneck of the control channel prevents all the mobiles from reaching the system and downloading the patches at the same time. According to [4], the total number of messages per second needed to saturate the cellular network capacity for a metropolitan area such as Washington D.C. is 240 msgs/sec and for the entire United States is $525,325$ msgs/sec. Therefore, any larger traffic volume would cause congestion or even crash the network.

Therefore, an appropriate order or scheduling of patch distribution is needed. Intuitively, the one with the highest risk to be infected or the one with the highest probability to infect others should have the highest priority for security patches. Our goal is to find a small set of nodes with the highest priority for patching, while keeping the infection rate as low as possible. We call it *targeted patching*. Under the partition based scenario, security patches do not have to reach all the mobiles if the worm could be contained in each small partition. A small set of nodes which separate all the nodes into multiple partitions is enough for our targeted patching.

With knowledge of the network topology, we partition the graph into as many separate pieces as possible and contain the worm propagation within each partition. These partitions are separated by a minimum set of key nodes called *separators*. The separators are chosen and patched by the network with the highest priority. As a result, the worm propagation can be blocked since an infected node inside its partition has to go through a separator to reach other partitions. Then, the worm containment problem becomes a graph problem and we can use graph-partition techniques to solve it. Now the question is what criteria should be used to partition the graph.

Based on the following two different partitioning strategies, there are two kinds of targeted patching: balanced patching and clustered patching (unbalanced patching).

B. Balanced Graph Partitioning

Intuitively, the significance level of each partition should be similar so that the worm damage to each partition can be balanced. As mentioned before, vertex weight and edge weight can be viewed as metrics for significance level. The vertex degree denotes how many victims an infected mobile is able to reach while the edge weight represents the probability that worms can propagate through this link successfully. Due to different ways of balancing these two metrics, we define balanced graph partitioning as follows.

Definition 2 (Balanced Graph Partitioning): Given an undirected weighted graph $G = (V, E)$, with weight $f(i)$ for each vertex $i \in V$ and $g(u, v)$ for each edge $(u, v) \in E$, a partition P cuts the vertices set V into $k (k > 1)$ subsets $V_1, V_2, \ldots, V_k$ such that $V_i \cap V_j = \phi$ for $i \neq j$, and $\cup_i V_i = V$, with the following two constraints satisfied:

- the total weights of vertices in each subset V_i are balanced.
- the total weights of all edges crossing any two subsets are minimized.

The first constraint in the definition requires the vertex weights for each partition to be balanced. Let LoadImbalance(P) denote the ratio of the highest partition weight over the average partition weight, *i.e.*, $max_i(f(V_i))/(f(V)/k)$. The first constraint minimizes LoadImbalance(P). It tries to keep the significance level in each partition balanced, so that the damage to each partition is balanced and limited. The second constraint keeps the edge weights between partitions minimized so that partitions are less related to each other. Let Edge-Cut(P) denote the total weights of all edges crossing any two partitions. Then, the second constraint minimizes Edge-Cut(P).

Next, we try to find an appropriate theory to solve the above problem. Existing graph partitioning solutions [14], [24] are developed for high-performance parallel computing, circuit placement and other disciplines. All these solutions partition the vertices of the graph into equally weighted sets so that the weight of the edges crossing between sets is minimized. A new class of partitioning algorithms based on the multilevel paradigm [15], [26] has been developed and is considered to be the state-of-the-art as they provide extremely high-quality partitions. These algorithms are very fast, and can scale to graphs containing millions of vertices. The basic idea behind the multilevel approach is to first coarse down the graph G to a few hundred vertices or less. Then, some standard partitioning algorithm is used to partition the graph. Since the size of the graph is quite small, simple algorithms such as Kernighan-Lin(KL) [17] performs well. The final step is to project this partition back towards the original finer graph G. Some of these algorithms have also been incorporated into well-known software packages such as METIS [16].

These existing graph partitioning algorithms were originally designed for parallel computing, whose goal is to evenly distribute the computations over k processors by partitioning the vertices into k equally weighted sets while minimizing inter-processor communication represented by edges crossing between partitions. These two objectives exactly match the two constraints in our definition. Therefore, balanced graph partitioning can be easily solved by existing graph partitioning algorithms, for example, the multilevel KL algorithm.

C. Clustered Graph Partitioning

Balanced graph partitioning tries to maintain the significance level in each partition balanced, so that the damage to each partition is balanced and limited. However, it does not give high priority to minimize the edge-cut, therefore does not guarantee that worms can always be successfully contained within individual partitions. For example, if the weights of

the edges across two partitions are very large, the probability of worm propagation through this edge will be very high. Then, the worms may have already propagated across the two partitions before patches are distributed. Therefore, rather than partitioning the graph into balanced parts, we want to partition the graph according to the trusted social relations. This method is referred to as *clustered partitioning* where edges within each partition have higher weights compared to the edges between the two partitions.

With clustered partitioning, we keep the mobiles that are socially close to each other in the same partition, and divide nodes that are not close into different partitions. This is because closer nodes are more likely to infect each other quickly as soon as the worms breakout. We cannot do too much about it as the infection may have already happened before patching so we prefer leaving them in the same partition. On the other hand, two nodes with a low weight link may have not communicated with each other and there is a low probability for a worm to spread across the link. Therefore, keeping them in two different partitions can effectively prevent the worm in one partition from infecting the other. Note that if there is no edge between the two nodes, they will be divided into two different partitions.

Definition 3 (Clustered Graph Partitioning): Given an undirected weighted graph $G = (V, E)$, with weight $f(i)$ for each vertex $i \in V$ and $g(u, v)$ for each edge $(u, v) \in E$, a partition P cuts the vertice set V into $k(k > 1)$ subsets $V_1, V_2, \ldots, V_k$ such that $V_i \cap V_j = \phi$ for $i \neq j$, and $\cup_i V_i = V$, with the following two constraints satisfied:

- the averaged edge weights (i.e., the total edge weights divided by the number of nodes) in each subset V_i are maximized: $max(\sum_{m \in V_i, n \in V_i} g(m, n))/|V_i|$
- the total weights of all edges crossing subsets are minimized.

Figure 2 shows the node weights and edge weights for each partition by the two partitioning schemes on the social relationship graph shown in Figure 1. We can see clearly from the example that balanced partitioning has an edge-cut of 2.5 while the clustered partitioning achieves an edge-cut of 0.9. As a result, it takes longer time for worms to propagate between partitions under clustered partitioning, which leaves itself more response time.

Unfortunately, this problem is NP-hard and these two constraints cannot be achieved at the same time. Thus, we can only apply heuristics to generate approximate solutions. We define a new concept called *Connectivity* and propose a recursive clustered partitioning algorithm based on this definition.

Definition 4 (Connectivity): We define the connectivity C recursively as follows:

Connectivity between two nodes: If i and j are two nodes and the edge between them has a weight of $w(i, j)$, then the connectivity between node i and j is $C(i, j) = w(i, j)$. If there is no edge between i and j, $C(i, j) = 0$.

Connectivity between a node and a set: S is a set with more than one node in it, and i is a node outside of S. Then the connectivity between node i and set S is $C(i, S) = \sum_{j \in S} C(i, j)$.

Connectivity between two sets: S_1 and S_2 are two sets in the graph, the connectivity between set S_1 and S_2 is $C(S_1, S_2) = \sum_{i \in S_1} C(i, S_2)$.

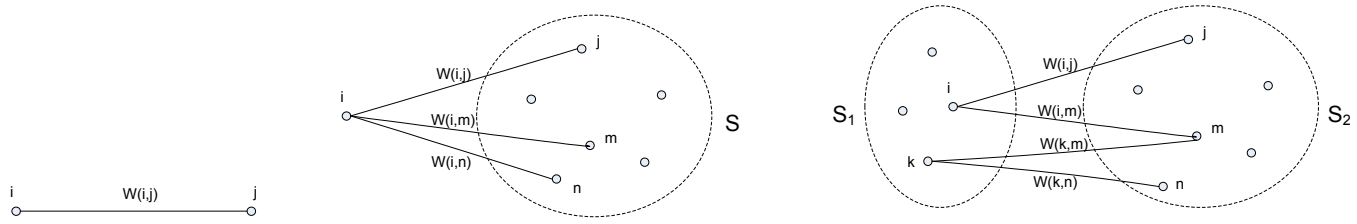
Connectivity of a set: The connectivity of set S is defined as the expected connectivity of any node i in the set S to the set $S_{\_i}$, $S_{\_i}$ is the set S excluding node i. Then $C(S) = \frac{\sum_{i \in S} C(i, S_{\_i})}{n}$, where n is the number of nodes in S.

The connectivity C denotes the connectivity level or closeness between two objects. For example, consider the closeness between a node i and a set S. Node i has one or more edges connected to set S, with weight $p_1, p_2, \cdots p_k$ respectively. As each edge weight p_i denotes the probability that a message is successfully delivered from i to S through that particular edge i, the probability that a message is successfully delivered from i to S can be computed by $1 - (1 - p_1)(1 - p_2) \cdots (1 - p_k)$. After ignoring the product items, it can be simplified as $p_1 + p_2 + \cdots + p_k$, which is the connectivity $C(i, S)$ between i and S.

Consider the connectivity of a set S. According to the definition of $C(S)$, each edge weight in S would be counted twice. Therefore, the connectivity of S can also be presented as $C(S) = \frac{\sum_{i \in S, j \in S} 2w(i,j)}{n}$. Without losing generality, we can rewrite it as $C(S) = \frac{\sum_{i \in S, j \in S} w(i,j)}{n}$, which is exactly the same as our fist constraint. Therefore, to satisfy the first constraint of clustered partitioning, we just need to maximize the connectivity C for each partition. Based on the definition of connectivity, we propose a heuristic algorithm to separate a graph into no more than k clustered partitions. k is a predefined threshold for the number of partitions.

The basic idea behind this algorithm is to enlarge each partition from individual nodes based on the metric of connectivity; i.e., a new node which has the largest connectivity with the current partition is chosen and added to the partition. This process stops until any node's joining could not increase the connectivity for the partition. Then another partition expanding process is started from a remaining node. When there is no more partition growing, the graph has been partitioned to clusters, which is called a round. If the number of partitions is still larger than k, a new round is started, where each partition is contracted to a node and the partition expanding process is performed on the updated graph. The detailed algorithm includes the following three stages:

1) **Expanding Stage**
   - Sort all edges in graph G by their weight w. Pick the edge with the largest weight and put its two end nodes into one partition P.
   - Partition P grows as follows: for all neighboring nodes of this partition, choose node i which has the largest connectivity with partition P and add node i to form a new partition P'. Update $C(P')$. Repeat the above step on the new partition P' until there is no neighboring node that can achieve $C(P'') \geq C(P')$.
   - Pick the edge with the largest weight from the rest of the edges and perform the above expanding process. The expanding stage stops when every node has been added to a partition.
2) **Contracting Stage**
   - Based on the resulting partitions from the expanding



(a) $C(i,j) = w(i,j)$ (b) $C(i,S) = w(i,j) + w(i,m) + w(i,n)$ (c) $C(S_1,S_2) = w(i,j) + w(i,m) + w(k,m) + w(k,n)$

Fig. 3. Examples of Connectivity

stage, contract G to a condensed graph G' such that each partition P_i in G becomes a node i in G' and all the interconnection edges between two partitions P_i and P_j become an edge e(i,j) between the two corresponding nodes i and j in G'. $w(i,j) = C(P_i, P_j)$.

- Recursively apply the Expanding stage and the Contracting stage on graph G'. It stops when the number of partitions falls below the specified value k.

3) **Restoring Stage**
   - Restore the original graph G by replacing each condensed node in each partition with its original nodes in the corresponding partition created in the contracting stage. Then, graph G is cut into less than k partitions.

Figure 4 illustrates how this algorithm works on a clustered graph. The distance between any two nodes denotes the closeness relationship between these two nodes. Thus, two nodes that are closer to each other in the plane would have higher connectivity and should be partitioned together.

The time complexity of this algorithm can be easily analyzed. At the beginning, there are n nodes in the graph which can be viewed as n individual partitions. In the end, the number of partitions is lower than k. As each partition expanding adds at least one node or one subset to a partition, there are at most $n-k$ times of partition expanding. For each partition expanding, a node with the largest connectivity to the partition is searched. This takes time $O(n_p * C)$, where n_p is the number of nodes in the current partition and C is the average degree of each node. In the worst case, the partition is as large as the entire graph and n_p becomes n. Therefore the total time for the algorithm is $O(n^2)$.

D. Worm Containment and Patching

In this section, we propose a systematic method to contain worms within different partitions. There are four steps to achieve it:

1) build an undirected weighted graph G representing the mobiles' social relationship in the cellular network from a real trace.
2) apply either balanced partitioning or clustered partitioning algorithm to graph G to obtain a partitioning and the corresponding cut edges.
3) use the Minimum vertex Separator Algorithm shown in Algorithm 1 to compute a minimum vertex separator from the set of cut edges.
4) send the security patches to separator nodes to block the worm propagation between partitions. These separator nodes could be responsible for forwarding the patches to other nodes in the same partition.

Algorithm 1 Minimum Vertex Separator Algorithm

Input: E_C: the set of cut edges;

1: $V_S \leftarrow \phi$
2: **while** $E_C \neq \phi$ **do**
3: Select $v \in V'$ which is shared by the most number of cut edges in E_C
4: Add v to V_S
5: Remove from E_C any cut edge whose end point is v
6: **end while**

Output: V_S: the set of vertex separators;

To obtain a set of separator nodes from the set of cut edges has been shown to be NP-complete [12]. We propose Algorithm 1 to approximately solve this problem by the Greedy paradigm, in which the next vertex selected for the Minimum Separator Set is the vertex that covers the most uncovered elements.

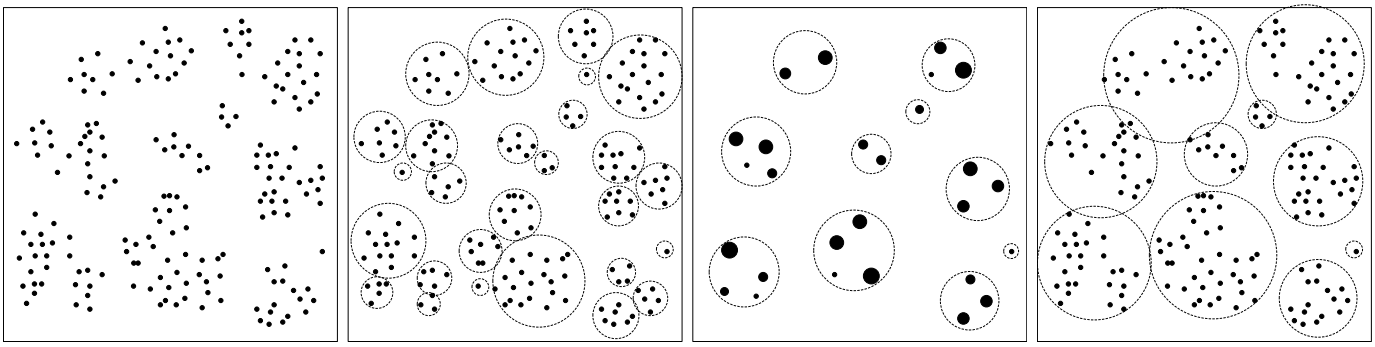
V. Performance Evaluations

In this section, we evaluate and compare three different patching strategies, random patching, balanced patching and clustered patching based on the worm infection rate and the number of separator (patched) nodes.

A. Simulation Setup

Our experiments are based on the social relationship graph generated from a real network traffic trace from one of the largest cellular networks in the US. Compared to related works that are based on cellular network traces, our trace analysis includes not only MMS and SMS messaging service, but also other popular services such as SIP based voice services as all of these services and interactions are equally likely to be exploited by worms.

As far as we know, there does not exist any realistic models for worm propagation using SMS/MMS services in cellular network. Although the work by Fleizach *et al* [11] models the mobile worm propagation, it is only based on US census data and estimated address book degree distribution. We construct



(a) original graph (b) partitioned graph (c) contracted graph (d) restored graph

Fig. 4. An example of the clustered partitioning algorithm. (a) to (b) shows the partition expanding process. (b) to (c) shows contracted and partitioned graph. (d) restores to the original graph with 10 partitions.

a MMS worm propagation model as follows. We assume worms are able to exploit the social relationship information for propagating. We model the probability that a mobile will activate a worm received from another mobile as directly proportional to the connectivity level between them and model the time taken for the worm code to propagate from one mobile to another as inversely proportional to the connectivity level. This time includes the latency for worm transmission as well as the delay from the time the receiver receives the worm and activates it. Once the worm has infected a new mobile, it starts to propagate to its neighbors after t time units.

We choose a number of (0.02%) nodes in the network by uniform distribution as the seed set of worm sources to initiate the infection process at the very beginning. This would provide the most pessimistic scenario as under a uniform distribution worm sources are more likely to be distributed across different clusters. We use a *Patching Threshold* α to control when the patching procedure starts. It is measured as the percentage of infected users in the network. This parameter represents the time delay since the worm starts propagating till it is detected by the network and a patch is generated. Once the percentage of infected users reaches this threshold α, the network would start to distribute patches to the chosen separator nodes. Each run of the simulation lasts for 2000 time units.

B. Effects of the Patching Threshold

We assume that some kind of systematic detection system is deployed across the network to observe the abnormal traffic and detect any worm outbreak. The time when to start the patching procedure depends on the strength of the detection system. Obviously, early detection can achieve better effects on worm containment but consumes more resources on monitoring and computation, while later detection, though less resource intensive in terms of monitoring could significantly delay worm containment. We use the parameter of patching threshold, α to simulate the time delay for worm detection and patch generation. Once the infection rate reaches this predefined threshold α, the network would start to distribute patches. Figure 5 compares the performance of three patching schemes: random patching, balanced patching, clustered patching under various α. As expected, the longer we wait to begin patching (higher patching threshold), the more number of nodes need to be patched for balanced or clustered patching to achieve the same infection rate. Interestingly, for random patching, the infection rate does not change irrespective of when to start the patching. Moreover, balanced patching has similar infection rate as random patching when patched nodes are under 2% in Figure 5(b) and 2.6% in Figure 5(c) due to the lack of enough separator nodes for effective partitioning.

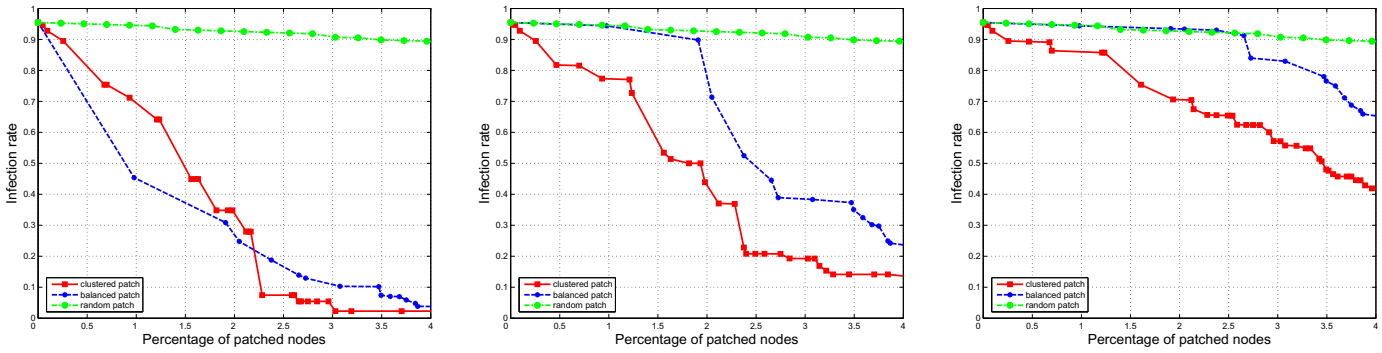
As shown in Figure 5, clustered patching requires much less patched nodes than balanced patching to achieve a certain infection rate in most cases. This is because clustered partitioning always cuts the graph from the least connected part, which results in less separator nodes, whereas balanced partitioning sometimes has to separate a strongly connected cluster apart and thus results in more separator nodes. Even in Figure 5(a), to achieve a low infection rate such as lower than 0.2, clustered patching requires much less patched nodes than balanced patching.

C. Infection Rate vs. Time

Figure 6 shows how infection rate changes over time under different patching strategies. Clustered patching achieves the best performance as it limits the infection rate within a certain bound much faster than the other two Also, the infection rate can be bounded to a much lower value if the patching threshold is lower, i.e. patching is started earlier. We can observe from the figure if the operator were to begin patching the network after 2% of mobile devices had already been infected, then clustered partitioning bounds the infection rate to 0.025 within 30 time units. Balanced partitioning is only able to bound the infection rate to 0.1 with a longer 450 time units. However, both schemes perform significantly better than random patching, which leads to 0.9 of nodes getting infected after 900 time units.

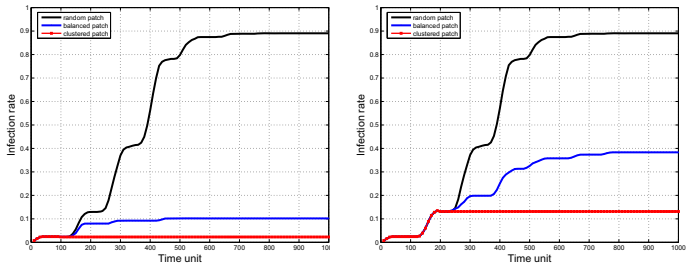
D. Effect of Dynamic Graph Topology

The trace we collected for social interaction analysis may not always be up to date unless it is frequently updated. To avoid the update overhead, there will be a gap between the time when the social relationship graph is generated and the time of worm breakout. For example, a few new users may register and join the network, and start to build their social relationship. This may result in inaccuracies in our patching schemes. Figure 7 shows the effect of dynamic topology change on the



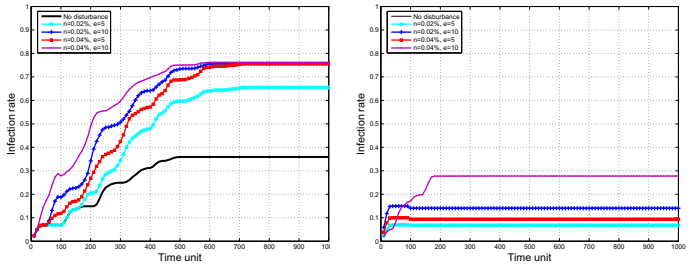
(a) $\alpha = 2\%$ (b) $\alpha = 10\%$ (c) $\alpha = 20\%$

Fig. 5. Effect of patching threshold α



(a) $\alpha = 2\%$ (b) $\alpha = 10\%$

Fig. 6. Infection rate vs. time (percentage of patched nodes = 4%)



(a) Effect on balanced patch (b) Effect on clustered patch

Fig. 7. Effect of dynamic topology under various disturbance levels (percentage of patched nodes $= 4\%, \alpha = 2\%$)

two patching schemes under various disturbance levels, where n denotes the percentage of new users joining the network and e denotes the number of edges for each new user connecting to other users. Notice that we cannot differentiate the curve of no disturbance and curve of $n = 0.02\%, e = 5$ for clustered patching because they behave the same. From the figure we can see that clustered patching is always behaving robuster to network disturbance than balanced patching. This is because new users usually join a certain cluster and only communicate with users in this cluster. Therefore, clustered patching can tolerate this disturbance more effectively.

VI. Discussion

Our worm containment strategy assumes the presence of a detection system to detect and generate an alert for a newly propagating worm. There are several works for detecting mobile worms at the network-level such as SmartSiren [3] and at the host-level using behavior anomaly detection [2] or energy anomaly detection [18]. These mobile worm detection systems can detect a worm within a reasonable latency and hence could serve as the initial trigger for our worm containment via patch distribution mechanism. Moreover, as a game between the worm designer and the patch system designer, a deterministic solution between balanced and clustered patching is required for a worm designer to circumvent.

Service providers usually take a multi-pronged strategy towards containing a zero-day worm - they start rate-limiting or filtering outbound traffic from hosts that are infected and also start extracting the signature for the worm so that the uninfected hosts can be protected. Developing a patch typically takes a substantial amount of time and manual effort. The time scale required to generate security patches or signatures is up to 2 hours [6]. Regardless of whether the service provider is able to develop the patch within hours of a worm outbreak, our proposed mechanism could also be used for rate-limiting in the following way. Once the service provider has identified the set of key nodes via our partitioning algorithms, he can generate *stricter* filtering or rate-limiting rules for outbound traffic from these nodes so that the damage due to the worm can be contained more effectively.

An effective patch distribution strategy needs to make sure that the patches do not compete for the scarce bandwidth resources [27]. In this regards, our patch distribution mechanisms take a hierarchical approach and instead of flooding out the patch to *all* nodes, we determine an optimal set of nodes to which the patch must be sent to obtain a bounded infection rate. These patches involve nothing about broadcasting and only bring limited traffic into the network.

As discussed above, it is up to the network operator to decide which approach to use to distribute the patches: push, pull or traffic controlling. In the case of pull, some mobile users may refuse to install the patch since they may not trust the source of the patch. An efficient way is needed for the network operator to have the patch messages authenticated. Fortunately, many mobile operators have some 'Wake Up' mechanism to directly distribute software or patches to devices

without any intervention from the users to make patches activated.

Cellular network bandwidth usually places constraints on worm propagation and patch dissemination. However, we can skip this influence in our simulation since once our patching strategy is deployed, the worms should have been contained and stopped at the very early stage before saturating the network capacity. For patch distribution, as only a limited number of mobiles are patched while no broadcasting is introduced, the limited patching traffic is far away from saturating the cellular network bandwidth.

VII. Related Work

Defense techniques against Internet worms include rate limiting [28] or filtering [23]. Vojnovic *et al* [25] studied the efficacy of automatic patching countermeasure in protecting the Internet against scanning worms. Zou *et al* [31] used a Kalman filter to detect Internet worm's propagation at its early stage in real-time. However, these techniques are not directly applicable to the mobile network scenario.

There is limited work on mobile viruses/worms modeling and containment in literature. Yang *et al* [29] applied a software diversity approach to deal with worm attacks in wireless sensor networks. Mickens and Noble [21] proposed a probabilistic queuing framework to model the propagation of mobile viruses over short-range wireless interfaces. Fleizach *et al* [11] evaluated the effects of malware propagating using communication services like VOIP and MMS in mobile phone networks. However, they do not use real traffic data in their worm propagation model. Bose and Shin [1] applied two commonly used mechanisms: rate limiting and quarantine to the dynamically generated list of vulnerable clients in the mobile messaging network. Miklas *et al* [5] used a trace-driven simulator to study the interactions between Bluetooth devices. They conclude that Bluetooth based worms would spread more widely by exploiting contacts between 'strangers' instead of 'friends'. While our focus here is on worms which spread via MMS or SMS, the hypothesis driving our work is analogous - that to contain a worm, we must first detect and patch the devices which bridge social clusters. Meng *et al* [20] investigated the reliability of SMS by analyzing traces collected from a nationwide cellular network over a period of three weeks. Here, we exploit the social relationship graph from a real cellular network trace that includes a variety of services and use it to develop a worm containment mechanism. Other security issues such as DoS attacks in the 3G network scenario are also studied in [4], [30].

VIII. Conclusions and Future Work

This paper proposed a methodology for effectively limiting the spread of MMS and SMS based worms via a graph partitioning approach. In our solution, mobile devices are divided into multiple partitions based on the social relationships among them. Two patching schemes, namely balanced and clustered patching are designed and their performance is evaluated using simulations based on data collected from real cellular networks. Through extensive evaluations, we demonstrate that our partitioning strategy can effectively contain worms.

Further research in this area includes dealing with hybrid worms which can make use of both cellular network interface and Bluetooth interface to propagate, and looking into worms and users roaming between cellular networks operated by different service providers.

References

[1] K.G. Shin A. Bose. Proactive security for mobile messaging networks. In *Proceedings of the 5th ACM workshop on Wireless security*, 2006.

[2] Abhijit Bose, Xin Hu, Kang G. Shin, and Taejoon Park. Behavioral detection of malware on mobile handsets. In *MobiSys*, 2008.

[3] J. Cheng, S.H.Y. Wong, H. Yang, and S. Lu. SmartSiren: virus detection and alert for smartphones. *MobiSys*, 2007.

[4] W. Enck, P. Traynor, P. McDaniel, and T. La Porta. Exploiting open functionality in SMS-capable cellular networks. *CCS*, 2005.

[5] A. Miklas et al. Exploiting social interactions in mobile systems. In *UbiComp 2007: Ubiquitous Computing*, 2007.

[6] F-SECURE. Close the zero-hour gap: Protection from emerging virus threats, *http://www.f-secure.com/f-secure/marketing/white_papers*.

[7] F-SECURE. F-secure deepguard - a proactive response to the evolving threat scenario, *http://www.f-secure.com/f-secure/marketing/white_papers*.

[8] F-SECURE. F-secure malware information pages: Sms-worm:symbos/feak, *http://www.f-secure.com/v-descs/sms-worm_symbos_feak.shtml*.

[9] F-SECURE. F-secure virus information pages: Cabir, *http://www.f-secure.com/v-descs/cabir.shtml*.

[10] F-SECURE. F-secure virus information pages: Commwarrior, *http://www.f-secure.com/v-descs/commwarrior.shtml*.

[11] C. Fleizach, M. Liljenstam, P. Johansson, G.M. Voelker, and A. Mehes. Can you infect me now?: malware propagation in mobile phone networks. *Proceedings of the 2007 ACM workshop on Recurring malcode*.

[12] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. WH Freeman & Co., 1979.

[13] JC Haartsen, E.R.S. BV, and N. Emmen. The Bluetooth radio system. *IEEE Wireless Communications*, 2000.

[14] B. Hendrickson and T.G. Kolda. Graph partitioning models for parallel computing. *Parallel Computing*, 26(12):1519–1534, 2000.

[15] G. Karypis and V. Kumar. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM*, 1999.

[16] G. Karypis, K. Schloegel, and V. Kumar. ParMeTiS: Parallel Graph Partitioning and Sparse Matrix Ordering Library, Version 3.0. 2002.

[17] B.W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal*, 1970.

[18] Hahnsang Kim, Joshua Smith, and Kang G. Shin. Detecting energy-greedy anomalies and mobile malware variants. In *MobiSys*, 2008.

[19] S. Keshav M. Ghaderi. Multimedia messaging service: System description and performance analysis. In *Proceedings of the First International Conference on Wireless Internet*, 2005.

[20] X. Meng, P. Zerfos, V. Samanta, S.H.Y. Wong, and S. Lu. Analysis of the Reliability of a Nationwide Short Message Service. *INFOCOM 2007*.

[21] J.W. Mickens and B.D. Noble. Modeling epidemic spreading in mobile environments. *The 4th ACM workshop on Wireless security*, 2005.

[22] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver. The Spread of the Sapphire/Slammer Worm, 2003.

[23] D. Moore, C. Shannon, GM Voelker, and S. Savage. Internet quarantine: requirements for containing self-propagating code. *INFOCOM 2003*.

[24] K. Schloegel, G. Karypis, and V. Kumar. Graph Partitioning for High Performance Scientific Simulations. *Computing Reviews*, 2004.

[25] M. VojnoviĆ and A. Ganesh. On the effectiveness of automatic patching. *Proceedings of the 2005 ACM workshop on Rapid malcode*, 2005.

[26] C. Walshaw and M. Cross. Parallel optimisation algorithms for multilevel mesh partitioning. *Parallel Computing*, 26(12):1635–1660, 2000.

[27] Nicholas Weaver and Dan Ellis. White Worms Don't Work. *USENIX*.

[28] C. Wong, S. Bielski, A. Studer, and C. Wang. Empirical Analysis of Rate Limiting Mechanisms. *RAID 2005*.

[29] Y. Yang, S. Zhu, and G. Cao. Improving sensor network immunity under worm attacks: A software diversity approach. *ACM mobihoc*, 2008.

[30] B. Zhao, C. Chi, W. Gao, S. Zhu, and G. Cao. A Chain Reaction DoS Attack on 3G Networks: Analysis and Defenses. *IEEE INFOCOM 2009*.

[31] C.C. Zou, L. Gao, W. Gong, and D. Towsley. Monitoring and early warning for internet worms. *CCS 2003*.