A Feasibility Study on Defending Against Ultra-Fast Topological Worms[*]

Liang Xie Sencun Zhu
Department of Computer Science and Engineering, Penn State University, 16802
{lxie, szhu}@ cse.psu.edu

Abstract

Self-propagating worms have been terrorizing the Internet for several years and they are becoming imminent threats to large-scale Peer-to-Peer (P2P) systems featuring rich host connectivity and popular data services. In this paper, we consider topological worms, which exploit P2P host vulnerabilities and topology information to spread in an ultra-fast way. We study the feasibility of leveraging the existing P2P overlay structure for distributing automated security patches to vulnerable machines. Two approaches are examined: a partition-based approach, which utilizes immunized hosts to proactively stop worm spread in the overlay graph, and a Connected Dominating Set(CDS)-based approach, which utilizes a group of dominating nodes in the overlay to achieve fast patch dissemination in a race with the worm. We demonstrate through analysis and simulations that both methods can result in effective worm containment if security patches are distributed in an early stage.

1 Introduction

Recent Internet worm outbreaks indicate that hundreds of thousands of Internet servers and client machines may be infected within a few minutes. Worm threats also become imminent and devastating to distributed P2P systems, which are featured by huge population and frequent data accesses. P2P worms may first compromise client machines, and then propagate by scanning IP addresses [21], harvesting email addresses, discovering P2P neighbors from those victims, or carried in shared files.

Our focus in this paper is on the *topological worm*, which exploits neighborhood information from the overlay to locate new targets for system-wide spreading. The worm first exploits a software vulnerability and controls a host, and then explores the connectivity information from the unprotected routing table/neighbor set in this victim and chooses those active neighbors as new targets. Compared with an IP scanning worm that randomly probes IP addresses to discover new targets, a topological worm does not need such interactions. Thus, it is likely to be detected by an IDS and will be more accurate in target seeking and faster in spread. Although no instance of a topological worm has been witnessed in a real P2P network, there are strong evidences that such worms could happen. For example, worms have been reported in [1, 2] to exploit the buffer overflow vulnerability in FastTrack, KaZaA, iMesh and other P2P client programs to launch denial-of-service attacks on supernodes and potentially other machines.

The important question is: how can we combat such worms when they start to surge in a real network? Given that today's Internet worms such as Slammer may infect millions of machines in minutes and topological worms could propagate even faster, any defense based on human response is clearly too slow. In other words, we must resort to a *systematic* and *automatic* worm containment mechanism. The research challenge is: is this type of automatic containment system at all feasible, considering such a system must provide (1) automatic worm detection, (2) automatic patch generation, and (3) automatic patch dissemination, verification and application? The first two issues are being actively studied [20], and here we assume they will be completely addressed someday soon. Through the appropriate choice of the number of initially infected nodes, we may simulate the delay caused by the first two steps. Our work will focus on addressing the third problem: how can we disseminate the security patches to the vulnerable hosts before the worm reaches them? We note that although automatic update mechanisms such as Windows Automatic Updates and Symantec security updates have been in place for directly acquiring the latest security patches from vendor sites, as shown in [17], the time to disseminate patches to hundreds of millions of Windows PCs would be of the order of hours, which is much longer than what is needed for worm spread. Thus, a faster way of patch dissemination, at least comparable to the speed of worm spread, is needed. Clearly, nothing can we do if the worm has already finished its spreading, and nothing should be further done if all the hosts have received and applied the proper patch. As such, we are interested in a quantitative study of the speeds of worm and patch when they are in an arms race.

While the solution space could be huge and better solutions could exist, in this work as an initial study of this important and challenging problem, we study the feasibility of constructing a defense infrastructure within an unstructured P2P system for rapid patch dissemination and examine its effectiveness. We observe that the order of patch dissemination among hosts plays a critical role in combating worm propagation. Network-wide flooding is not necessarily the fastest and most effective way. On the other hand, security patches

[*]This work was supported in part by Army Research Office (W911NF-05-1-0270) and the National Science Foundation (CNS-0524156, CNS-0519460, and CAREER-0643906).

do not have to reach all the hosts in such a short racing period if the worm could be first quarantined in a small island. Based on these observations, we examine the effectiveness of two types of strategies. First, we propose to proactively select a set of worm-immune nodes in the overlay to block the possible worm spreads. Second, by exploiting the P2P network topology information, we may first disseminate the security patch to a select set of nodes, which will further flood it to local neighbors. Specifically, we propose a *partition-based scheme* in which immune nodes are chosen in a way that they partition the overlay graph into many nearly-balanced subgraphs and each blocks the worm spread within its area. We also propose a *CDS-based scheme* in which a dominating set of nodes are chosen from the overlay and these nodes are utilized to disseminate security alerts in a timely fashion.

Our experimental result demonstrates that both schemes could greatly help the system achieve a high immune rate if security patches could be distributed in the early stage (say when only 1% of the population have been infected). Although our work makes some assumptions, it provides some hope and guidance for combating this ultra-fast topological worm. We hope our work could attract more interests from the research community to this important issue.

Organization The rest of the paper is organized as follows. Section 2 introduces the network model and the attack model. In Section 3 and 4, we study two defense schemes, namely the partition-based scheme and the CDS-based scheme, respectively. These schemes are then evaluated through simulation in Section 5. We introduce the related work in Section 6 and conclude in Section 7.

2 Preliminaries

2.1 System Model

Network Model There are two categories of P2P systems: *unstructured*, and *structured*. In unstructured systems, such as Gnutella and KaZaA, file placement is random and has no correlation with the topology; in structured systems, such as Chord and Pastry, placement of shared data and the topology characteristics of the network are tightly bound based on distributed hash tables (DHTs). Our main focus in this paper is on the unstructured P2P systems.

Modern unstructured systems such as Gnutella typically adopt a *two-tier* overlay [19] in which a subset of peers, called *supernodes* or *ultra peers*, form a top-level overlay while other participating peers, called *leaf peers*, are connected to the top-level overlay through one or multiple supernodes. A supernode maintains a directory of files stored at its leaf nodes. When a leaf node queries a file, it sends the request to its supernode. If the latter knows the file location (i.e., one of its leaf nodes has the file), it replies the requester directly. Otherwise, it floods the query to other supernodes.

In our work, we generalize a P2P overlay topology as an undirected random graph, in which each vertex corresponds to a peer and each edge reflects the current neighboring relationship between two peers. A P2P graph is dynamic because it updates when peers join or leave the system. However, it is not influenced by the P2P traffic, and it remains relatively static during the very short time period of topological worm spread. We note that the P2P graph is a logical concept. Physically, peers connect to an Internet routing infrastructure whose key part consists of hundreds of thousands of routers. Therefore, each edge in the topology graph has a latency, which is largely determined by the hop-count between the two end hosts.

Node States We define three security states for P2P hosts: *susceptible*, *infected* or *immune*. A susceptible host is not protected against the worm. It either gets infected when exposed to the worm attack or becomes immune when a protection is in place *before* the worm arrives. By protection we mean the host installs a security patch and activates it. We note that an infected host may be recovered when, for example, the user activates a patch and scans the machine to cleanse the infection. However, such recovery is usually too late because the worm may have caused serious damages (e.g., system crash or data loss) to the victim. We should avoid infections on the hosts if at all possible.

Attack Model Internet worms adopt three major scanning strategies in identifying new victim targets: *random, hit-list–based, and topological* scanning. A random scanning worm selects targets' IP addresses at random. It probes if a host with the specific IP address really exists; a hit-list–based scanning worm collects a preference list of susceptible nodes (based on node importance or the overlay topology) and choose new targets from the list; a topological scanning worm exploits topology information from infected hosts and accurately locate new targets. Topological worms typically combine the hit-list–based and the topological scanning strategies. Specifically, a worm starts by attacking initial targets selected from a hit-list (acquired by simply initiating a file search), it then scans the neighbor sets or the routing tables of these victims and identifies all susceptible neighbors as new targets. In this way, the worm continues its spread cycle and eventually spreads to the entire overlay. Note that for the tractability of the problem, we assume the size of a hit-list is a small fraction (e.g., 1.0%) of the population. To prevent an attacker from building a huge hit-list, we assume some mechanism has been (or will be) deployed to reject unauthorized crawling of the P2P topology.

2.2 System Overview

As described earlier in Section 1, we study the topological worms for which the patches to address the security vulnerabilities have been generated, for example, by the security vendors such as Microsoft and Symantec. We assume for security purpose, such vendors may deploy a group of *security servers* in P2P systems, which form a separate overlay. These servers are publicly known. They are well maintained by the vendors so that they keep the up-to-date worm definitions and the security patches. In addition, these servers may collaborate and share security information with each other. The number of security servers is typically determined by the size of the P2P system. As P2P networks are distributed, security servers can be deployed in various locations, each taking care of a certain proportion of the network. In our schemes, these servers also secretely construct periodic snapshots of the P2P overlay and utilize this information to help secure other hosts. On the other hand, these servers do not participate in the file-sharing process, hence they do not have to be very powerful.

We adopt two defense principles. The first is to proactively select a set of worm-immune nodes in the overlay to block the worm spread; the second is to compete with the worm so that susceptible nodes may be alerted and immunized before the worm attacks reach them. Along the first design principle, we propose a partition-based scheme, in which some worm-immune nodes (key nodes) are systematically chosen by the security servers in a way that they partition the overlay graph into as many nearly-balanced sub-graphs as possible. Thus, worm propagation can be effectively contained within these sub-graphs. Along the second design principle, we propose a CDS-based scheme, in which the security servers select a small proportion of hosts to form a *connected dominating set* for the overlay. Once a topological worm has been detected, the servers disseminate security alerts through this set to notify vulnerable hosts of the surging worm attack and urge them to launch the protection.

Previously, Zhou et al. [24] introduced a random patch dissemination scheme, which we refer to as a *baseline scheme*. The baseline scheme falls in the second category of the design principles. It assumes each node in the system has an independent probability p of being a guardian node. When a worm starts propagating and reaches a guardian, the guardian immediately detects the worm and notifies other nodes by disseminating a self-certifying alert (SCA) [8] in a flooding mode. The baseline scheme does not make use of any topological information, thus it is much less effective than ours in containing topological worms. We will make some quantitative comparison in Section 5.

3 A Partition-based Scheme

In this approach, security servers periodically take a snapshot of the overlay topology from the supernodes. According to this topological information, security servers construct a small group of worm-immune hosts (named *key nodes* hereafter) which protect the susceptible hosts by stopping the worm spread within the overlay graph. The key nodes are ordinary hosts that receive and apply security patches in the first place, and they are not assumed to have the capability of worm detection or generating self-certified alerts as the guardian nodes in [24].

The issue here is how to decide such a small set of key nodes from the overlay topology. If security servers can predict the worm hit-list, the choice of key nodes may be biased towards the neighbors of those initial victims. However, in real applications the worm's hit-list varies with link latency, node degree and host vulnerability. Here we consider a heuristic for choosing key nodes: *key nodes should partition the overlay graph into as many separate pieces as possible and block worm propagation within each partition*. A partition contains non-zero number of nodes and (ideally) worm propagation from an infected node inside its partition to the rest of the region has to go through a key node, which blocks the propagation. Thus, an occurrence of the worm attack will be confined within a partition, leaving other partitions of the overlay intact. This approach is optimal if we assume that servers have no knowledge on the origins of the worm attacks. Worm containment now becomes a graph problem and we may utilize graph-theory techniques to solve it.

Acquiring Overlay Topology In the partition-based scheme, security servers adopt a scalable *parallel crawling* technique [19, 18] featured by a master-slave architecture, to collect topology information in a P2P system. Specifically, a master process running in the security server coordinates multiple slave processes in a selected list of supernodes that crawl disjoint proportions of the network in parallel. Each security server is responsible for managing the list of supernodes and constructing the final topology graph. Each supernode in the list receives some initial points from the security server and start discovering the network topology around these points. We note that a distributed crawler (e.g., Cruiser [19]) is able to accurately capture a complete snapshot of a Gnutella network with more than one million hosts in just a few minutes. Moreover, the overlay graph is relatively static during the short period of worm spread and the security servers do not have to frequently reconstruct the overlay graph. Therefore, the crawling interval is largely determined by the overlay dynamics, which reflect the frequency of node joins and departures. We will evaluate the impact of overlay dynamics on the effectiveness of our schemes in Section 5.

Partitioning an Overlay Graph Once the servers have constructed the snapshot of the overlay graph, they may partition the graph and choose key nodes. We start with a simple example. Suppose we have a undirected graph $G = (V, E)$, where V is a finite set of vertices and E is a finite set of edges. We partition V into two subsets, A and B. As a result we get a sub-set C which consists of edges spanning A and B and incident vertices of these edges. We define edges in C as *cut edges* and the vertex set in C as the *vertex separator* of G. Clearly, deletion of the vertex separator will disconnect A and B. Our goal is to obtain a minimal vertex separator while maintaining an appropriate balance (e.g., the same number of vertices) between A and B. To achieve this goal, we first need to obtain a minimal set of cut edges, based on which we derive the minimized vertex separator.

The problem now becomes to partition the vertices of an overlay graph G into k subsets such that each of them has a nearly equal size of vertices, while the number of cut edges spanning the subsets is minimized (see Fig.1). We define it as a *relaxed k-way graph partitioning problem* as follows.

Definition 1. *(Relaxed K-way Graph Partitioning) Given a graph $G = (V, E)$ with $|V| = n$, partition V into k subsets $V_1,\ V_2,\ \ldots,\ V_k$ such that $V_i \cap V_j = 0$ for $i \neq j$, $|V_i| \approx n/k$, $\cup_i V_i = V$, and the number of cut edges is minimized.*

Compared with [12], we relax the strict balance requirement, so that vertex numbers in subsets do not have to be exactly equal. This allows security servers to provide an approximated overlay graph as the input. Besides, this produces a nice feature that we may always derive a good vertex separator from a minimized set of cut edges [3] (we show an efficient algorithm to achieve this later). Finding an exact minimized

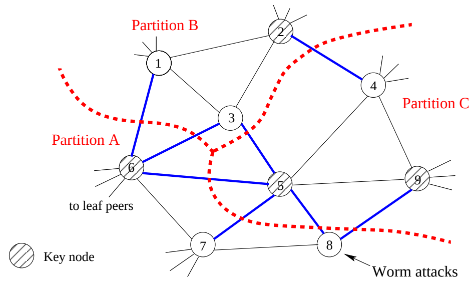


Figure 1. An overlay graph is partitioned and the worm spread is contained within partition A.

set of cut edges for a given graph is NP-hard. Fortunately, we may use some approximations because we do not require partitions to be precisely balanced. A popular technique in practice (especially when graph G is complex) is the *multi-level k-way partitioning* approach [12]. The basic structure of this approach is as follows. The graph $G = (V, E)$ is first coarsened down to a small number of vertices and a k-way partitioning of this small graph is computed, typically using the Kernighan-Lin algorithm [14] in a recursive way (i.e., divide and conquer), and then this partition is projected back towards the original graph (finer graph), by successively refining the partitioning at each intermediate level.

Now we may apply the partition-based algorithm in a P2P system. We use an undirected graph $G_O = (V_O, E_O)$ to denote the overlay graph, where V_O denotes the vertex set and E_O denotes the edge set. The proposed algorithm is executed periodically by security servers and it consists of the following four major steps:

1. collect the overlay topology and map it to an undirected graph G_O.
2. execute the multi-level k-way partitioning algorithm and obtain the cut edges of G_O.
3. run Algorithm 1 to compute a minimum vertex separator from the set of cut edges.
4. choose members from the vertex separator as key nodes and push the security alerts to them.

Algorithm 1: Minimum Vertex Separator Algorithm

```
Input: $E_c = \{e_1, e_2, ..., e_m\} \in E_O$: a set of cut edges;
Procedure:
1: $V_s \leftarrow \phi$
2: while $E_c \neq \phi$ do
3:     select $v \in V_O$ 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 $v$
Output: $V_s = \{v_1, v_2, ..., v_n\}$: the vertex separator of $G_O$;
```

We propose Algorithm 1 for further deriving a minimum vertex separator from the cut edges (obtained from step 2). Using the overlay graph in Fig.1 as an example, a typical sequence of adding nodes to the vertex separator is: node $5 \rightarrow 6 \rightarrow 2, 9$. Accordingly, the sequence of deleting cut edges (in bold line) between the partitions is: edge $(5, 3), (5, 6), (5, 7), (5, 8) \rightarrow (6, 1), (6, 3) \rightarrow (9, 8), (4, 2)$.

Protecting Key Nodes In the partition-based scheme, security servers require a key node to launch the protection once it has received the security alert. Each alert typically contains the specific worm information, a vendor patch and the vendor's signature for verification. A key node is aware of its importance in the overlay and is urged to launch immediate protection to protect itself and others. However, in some special cases when a selected node refuses to be a key node or ignores the patch, several unseparated partitions could be merged into a larger one and a worm will eventually escape from a single partition. To address this problem, we propose that security servers pre-define a threshold, say S_t, as an upper-limit of node population in each partition. Based on the responses from those key node candidates, security servers leverage the graph knowledge to estimate the population s in each merged partition. Once they find out that s exceeds S_t, the servers will repartition this unbalanced piece and set up new key nodes to launch the protection.

Security and Performance Analysis We analyze the effectiveness of the partition-based scheme. We assume every node in the vertex separator becomes a key node once selected by the security server. We consider the case that once a worm sets its initial victim(s) in a partition, all susceptible nodes in this partition will eventually be infected.

We use I to denote *immune rate*, the final fraction of immune nodes over the entire population. We also use m to denote the size of the hit-list, k to denote the number of partitions, and α to denote the fraction of nodes as key nodes. We model the graph partitioning algorithm as a process, which takes G_O and k as the input, and generates $N \cdot \alpha$ key nodes as the output. The overlay graph and the partition information should be kept from attackers. In the worst case, we assume the worm knows about m distinct partitions in the overlay O and chooses one initial victim from each of these m partitions. Thus, the partition-based scheme has an immune rate: $I_{worst} = 1 - \frac{m}{k} \cdot (1 - \alpha)$, which indicates when $m = k$, the worm eventually floods the entire overlay and $I_{worst} = \alpha$. We compute the average immune rate as follows.

Lemma 1. *Suppose the overlay graph G_O has k equal partitions separated by $N \cdot \alpha$ key nodes, and the attacker later randomly chooses m targets from these partitions to inject worms. The averaged immune rate in the system is $\alpha + (1 - \frac{1}{k})^m \cdot (1 - \alpha)$.*

Proof. Let X_i denote whether or not partition i is chosen by the attacker, $X_i = 1$ means chosen while $X_i = 0$ means not. We have $P\{X_i = 0\} = (\frac{k-1}{k})^m$, which is the probability that partition i is not chosen, and $P\{X_i = 1\} = 1 - (\frac{k-1}{k})^m$, the probability that partition i is chosen. Thus, the average number of distinct partitions chosen by the attacker can be computed as $X = \sum_{i=1}^{k} X_i = k(1 - (\frac{k-1}{k})^m)$. Finally, we derive the averaged percentage of partitions that are eventually not infected by the worms: $I_{avg} = 1 - \frac{X}{k}(1 - \alpha) = \alpha + (1 - \frac{1}{k})^m \cdot (1 - \alpha)$. □

Fig.2 shows the immune rate (the worst and the average case) of the scheme. It indicates that the immune rate goes down as there are more initial victims. Clearly, when the initial hit-list is too large, there is no way to protect the overlay. When the fraction of key nodes α is low, a larger k (more

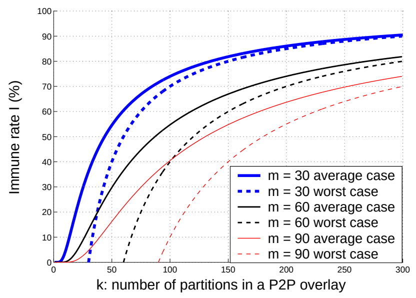


Figure 2. Immune rate vs. no. of partitions and hit-list size

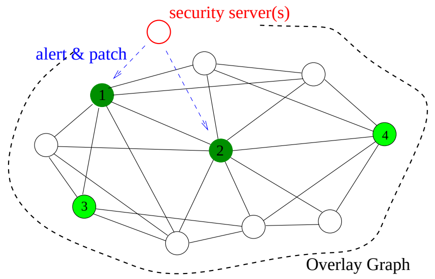


Figure 3. CDS node 1, 2 are found in a well-connected overlay. Node 3, 4 form a weakly connected dominating set.

graph partitions) usually results in a better containment result. This is consistent with Lemma 1.

According to [12], the running time of the multi-level partitioning algorithm is $O(|E_O|)$, linear to the number of edges in the overlay. Algorithm 1 has a time cost $O(|E_c|)$, linear to the number of cut edges in the overlay. Note that $|E_c| << |E_O|$, hence the running time of the partition-based scheme is $O(|E_O|) + O(|E_c|) \approx O(|E_O|)$. Messages incurred in setting up the key nodes is $O(\alpha \cdot |V_O|)$, which is far less than the number of vertices in the overlay. Having a time cost $O(|E|)$ and a message overhead $O(\alpha \cdot |V|)$, the scheme is lightweight. This ensures a low cost on graph repartitioning (as the snapshot updates) and an easy setup for the key nodes. In our experiments, graphs with over 1,000,000 vertices can be partitioned into 256 parts in a few seconds on a PC with Intel Pentium 4 2.5GHz CPU.

4 A CDS-based Scheme

In this section, we consider flooding the security patch to all the hosts in a race with the worm. Clearly, if the patch starts from one host, it is unlikely to win the race because the worm started earlier and at many nodes. Randomly selecting a small set of key nodes to start the propagation is not optimal either because the distance (in terms of number of hops) of a node from a key node is not bounded–some nodes may have to wait for a long time to receive the patch. To achieve the best performance, we must exploit the P2P network topology information to start the dissemination process, so that the patch latency could be upper bounded.

We propose that the security servers compute a small group of nodes named *dominating set* each time when the overlay graph has been constructed. A group of nodes is a dominating set if every node not in the subset is adjacent to at least one node in the subset. Also, security servers prefer nodes in this subset to be *connected* for the ease and reliability of alert deliveries. These selected hosts are called *CDS nodes* (or key nodes) hereafter. The CDS nodes are *only* derived by the servers, which securely construct and maintain the topology graph. We note that the P2P overlay usually has rich node connectivity, hence the set of CDS is relatively small. An example in Fig.3 shows that, when the security server pushes a patch into the CDS nodes $\{1, 2\}$, everyone receives it within 1-2 hops from the server.

Finding CDS Nodes in the Overlay We show how security servers derive the set of nodes to which they directly push the security alert. We first define an *Overlay CDS* problem, which aims at finding a feasible set of CDS nodes based on an overlay graph. We give the following definitions.

Definition 2. *(Dominating Set) Given an overlay graph $G_O = (V_O, E_O)$, a dominating set D of G_O is a subset of V_O such that for every node $v \in V_O$, either $v \in D$ or there exists a node $u \in D$ such that $(u, v) \in E_O$.*

Definition 3. *(Overlay CDS problem) The Overlay CDS problem is to find a minimal subset S of nodes, such that the subgraph induced by S is connected and S forms a dominating set in the overlay graph G_O.*

The CDS problem was originally defined in [11], and it has already been shown that finding the CDS in a general graph is a NP-complete problem [10]. However, a good approximation can always be found for an overlay graph G_O, which typically has high connectivity among the supernodes. We adopt the following two-phase algorithm [11] to derive the CDS nodes:

1. Initially all nodes are colored *white*. Each time we include a vertex in the dominating set, we color it *black*. Dominated nodes are colored *gray* (once adjacent to a black node). In the first phase, we pick a node at each step and color it black, coloring all adjacent white nodes gray. A *piece* is defined as a white node or a black connected component. At each step we pick a node to color black that gives the maximum (non-zero) reduction in the number of pieces.

2. In the second phase, we have a collection of black connected components. We recursively connect pairs of black components by choosing a chain of vertices, until there is only one black connected component. The final solution is the set of black vertices that form the connected component.

We further adapt the solution to the P2P environment. Due to the complexity and the randomness of an overlay graph, sometimes the size of the resulting CDS nodes could be unreasonably large. To be more scalable, we propose security servers push the alert to a reasonable and random subset of the CDS nodes to save their bandwidth. We note that CDS nodes are well connected in the P2P overlay. Therefore, our strategy only slightly increases the patching latency (i.e., nodes receive the alert through a few more hops). Currently we do not consider alternatives such as the *k-dominating set* [16] or the *weakly connected dominating set* [6] in which the resulting nodes typically are less connected.

Table 1. Notation for the worm spread model

Note	Explanation
d	average degree of the P2P graph
N	total number of hosts in the overlay
v_i	number of vulnerable hosts in time tick i
f_i	number of infected hosts in time tick i
e_i	number of newly infected hosts in time tick i
h	number of infected hosts when patching starts (hit-list size)
T_{pat}	average patching time (patch dissemination + activation)

Security and Performance Analysis We first study the worm spread process using a dynamic epidemic model [7, 13]. Specifically, we adopt discrete time to conduct recursive analysis and deterministic approximation to describe topological worm propagation. We assume a random graph for the overlay and a worm takes one time tick to complete its infection on a vulnerable host. We refer to Table 1 for notation.

Lemma 2. *In a P2P overlay with no defense, if there are v_i susceptible hosts at time tick i, then in the next time tick there will be $e_{i+1} = v_i \cdot (1 - (1 - \frac{1}{N})^{d \cdot f_i})$ newly infected hosts.*

Proof. We know that $v_i + f_i = N$ at any time i. We want to prove $e_{i+1}(k) = v_i \cdot (1 - (1 - \frac{1}{N})^k)$ for k attacking attempts on neighbors. By induction, there are v_i susceptible hosts when $k = 1$ and the probability that this attempt results in a newly infected host is: $\frac{v_i}{N}$. Suppose the lemma is true for k attempts, we want to show it also holds for the $(k+1)$'th. Let p denote the probability the $(k+1)'$th attempt eventually results in an infection, we have

$$\begin{aligned} e_{i+1}(k+1) &= (e_{i+1}(k) + 1) \cdot p + e_{i+1}(k) \cdot (1-p) \quad (1) \\ &= v_i \cdot (1 - (1 - \tfrac{1}{N})^k) + \tfrac{v_i - e_{i+1}(k)}{N} \\ &= v_i \cdot (1 - (1 - \tfrac{1}{N})^{k+1}). \end{aligned}$$

We know that at time i, the total number of infected hosts is f_i and there are totally $k = d \cdot f_i$ attacking attempts. Therefore, we get $e_{i+1} = v_i \cdot (1 - (1 - \frac{1}{N})^{d \cdot f_i})$. □

Next, we study the effectiveness of the CDS-based scheme. We examine how long it takes the system to contain the worm spread and the final severity level of attacks the worm causes in the overlay. We derive the following lemma.

Lemma 3. *Given average patching time T_{pat}, a P2P system protected by the CDS-based scheme reaches a stable stage with immune rate $I_{avg} = 1 - \frac{f_{T_{pat}}}{N}$, where the infected population f follows the recursion $f_{i+1} = v_0 - (v_0 - f_i)(1 - \frac{1}{N})^{d \cdot f_i}$, where $v_0 = N - h$.*

Proof. We consider the time when the server starts to disseminate security alerts as the starting time point 0. At this time, $f_0 = h$ (i.e., hit-list size) nodes in the system have been infected by the worm. As the worm spreads, we may derive that the infected population on the next time tick $i + 1$ will be $f_{i+1} = f_i + e_i$. According to Lemma 2, we have

$$\begin{aligned} f_{i+1} &= f_i + v_i \cdot (1 - (1 - \tfrac{1}{N})^{d \cdot f_i}) \\ &= v_0 - (v_0 - f_i)(1 - \tfrac{1}{N})^{d \cdot f_i}, \qquad (2) \end{aligned}$$

where $v_0 = N - h$. This recursion stops at a stable stage when the worm cannot infect more targets and its spread has been contained, i.e., nodes in the system either have been infected or have received and applied the security patches.

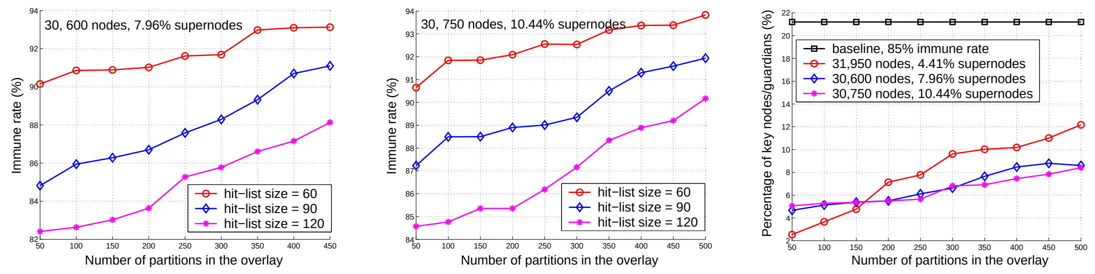
Considering the fact that the server keeps the security patch and the CDS nodes have already been computed before the server is notified of the worm infection, the average patching time T_{pat} typically contains the time to deliver message from servers to end hosts through CDS nodes and the patch activation time. The former is determined by the average hop-count a patch travels (basically decided by the overlay topology). Since the worm spreads in an average life time T_{pat} before it is stopped, we have the average immune rate $\bar{I}_{avg} = 1 - \frac{f(T_{pat})}{N}$. □

5 Evaluation of Effectiveness

Default Environmental Setting To construct overlay graphs for P2P systems such as Gnutella 0.6 and KaZaA, we implemented two distributed crawlers [18, 19]. The crawlers walk over the network and adopt the membership protocol (PING/PONG mechanism) to collect topology information. They also make use of the two-tier overlay structure to reduce the crawling time. In our experiments, we set up a security server (a PC with Intel Pentium 4 2.5GHz CPU) to run a server thread, which coordinated the client crawler threads launched on 8 different PCs (with the similar configuration as the server PC). Snapshots of the overlay graph were assembled on the server side. According to our observations, in Gnutella 0.6, about 15% of the supernodes depart from the overlay in $3 \sim 4$ hours. To reflect network dynamics, we used a 4-hour crawling interval. Note that if the interval is too large, the topology graph in the security servers will lose fidelity; if it is too short, much overhead will be added to the crawling hosts and the normal P2P traffic.

To obtain various overlay networks, we selected different subnets (each has a population of 30,000 $\sim$ 40,000 nodes) from the constructed topology graphs. Note that these subnets depict the logical connections among the participating hosts. We then randomly placed the hosts in a subnet into an Internet environment with 5050 hierarchical routers (generated using Georgia Tech's Transit-Stub Internet Topology Generator [23]). The latency between each pair of hosts was computed according to the routers between them and is between 10ms and 1s. Initially, a number of supernodes are chosen by the attacker to form a hit-list. These initial victims accept an incoming worm message and forward the same message to their neighbors. To evaluate the schemes on their tolerance against node dynamics, we keep the graph knowledge unchanged while performing a number of rewirings on the underlying topology, according to the speed in which the network evolves. Each of our tests takes 50 runs. We report the mean of test data.

Partition-based Scheme Our results show that the server divides an overlay (about 30,000 hosts) into $300 \sim 500$ nearly-balanced pieces in one second and with an averaged CPU usage below 17%. Fig.4 (a)$\sim$(b) illustrates the results when



(a) Topology I (7.96% supernodes) (b) Topology II(10.44% supernodes) (c) Fraction of key nodes needed

Figure 4. Test results when applying the partition-based scheme in Gnutella 0.6 overlays (partial snapshots). $(a) \sim (b)$ each shows the immune rate as a function of the number of partitions and the worm hit-list size; (c) shows the percentage of key nodes needed to achieve an immune rate $\geq 85\%$ (hit-list size = 90). The baseline scheme [24] requires more guardians.

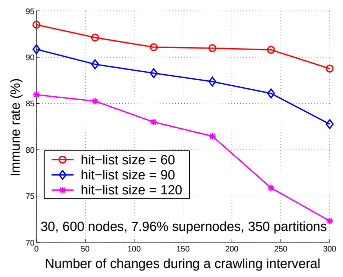


Figure 5. Impacts of node dynamics on the partition-based scheme (Gnutella 0.6)

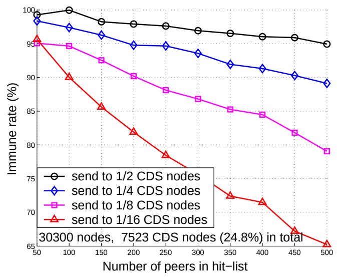


Figure 6. Immune rate vs. hit-list size and proportion of CDSs used in the CDS-based scheme

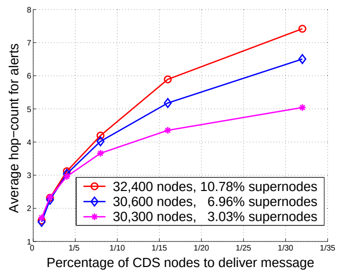


Figure 7. Average hop-count of alert dissemination as a function of percentage of CDSs selected

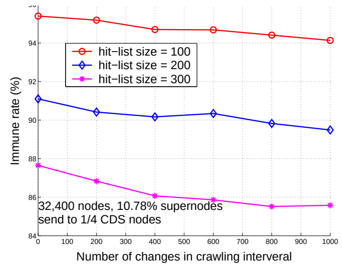


Figure 8. Impacts of node dynamics on CDS-based scheme. This scheme tolerates more dynamics

the partition-based scheme is applied in various Gnutella 0.6 overlays. The figures indicate that a higher immune rate can be achieved when there are more partitions in the overlay. They also demonstrate that the size of a worm hit-list has a negative influence to the defense (Section 3). A further comparison reveals that an overlay with a higher percentage of supernodes achieves a higher immune rate. Fig.4(c) illustrates the proportions of key nodes needed in various overlays to achieve the immune rate of $\%85$. The result indicates that we may nicely partition the overlay graph and achieve a high immune rate by simply utilizing less than 10% of the population as key nodes.

Fig.5 shows the impacts of node dynamics on the partition-based scheme. Here the number of changes (e.g., node joins, departures, reconnections) reflects a degree of overlay dynamics. We notice that in general the partition-based scheme demonstrates good performance even when the servers' topology information becomes outdated. However, as the worm hit-list size increases, the tolerance level deteriorates more or less. In our tests, 250 changes of supernodes typically happens within a 3-hour time period and the figure shows a good tolerance in the case of adopting a 3-hour crawling interval.

CDS-based Scheme We evaluated the performance of the CDS-based scheme in Gnutella 0.6 systems. Our results show that using the Guha and Khuller's algorithm [11], a security server can derive the CDS nodes of an overlay (40,000 hosts) in 0.5 second, with its averaged CPU usage below 23%. Fig.6 illustrates the containment result when security servers choose different proportions of CDS nodes to deliver worm containment messages. Clearly, a larger subset results in a higher immune rate. For example, when 1/2 CDS nodes (12% of the total population) are selected, a worm with a hit-list of 500 nodes can be successfully contained (with a final infection rate below 5%). However, considering the bandwidth limit and the impact to the normal P2P traffic (this becomes worse when the messages contains heavy payload), security servers may want to push the message directly to a smaller node set. Fig.6 indicates that choosing 1/4 CDS nodes keeps a good balance between immune rate and overhead.

Fig.7 shows the average hop-count (logical) through which security patches are forwarded. Different subset of CDS nodes were tested and the result demonstrates that a full set of CDS nodes results in the lowest hop-count (less than 2). As the subset becomes smaller, the hop-count increases, i.e., a security message has to go through more forwarding nodes before reaching the destination. When the overlay has a higher fraction of supernodes, more forwarding nodes exist between a pair of end hosts and the average hop-count increases. Figure 8 illustrates the impacts of network dynamics on the CDS-based scheme. Compared with the partition-based scheme which depends more heavily on an exact overlay topology (Fig.5), the CDS-based scheme is more tolerance to the topology distortions: it keeps good performance even when there are 1000 changes among the supernodes. The crawling interval, therefore, can be larger, say 12 hours.

Comparison of Schemes We used the scheme [24] as the base-line approach and compared it with our schemes in different overlays (Gnutella and KaZaA). The comparisons were based on the same fraction of key nodes (or guardians in [24]) and the same size of initial worm hit-list. Fig.9 demonstrates that in a KaZaA system, both our schemes achieve a higher immune rate than the baseline approach. When the worm hit-list is relatively small, tho schemes attain the similar immune

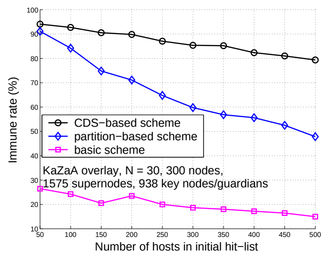


Figure 9. Immune rate vs. hit-list size

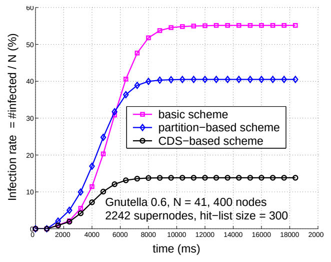


Figure 10. Infection rate vs. time

rate. As the hit-list size increases, the CDS-based scheme outperforms the partition-based scheme. Fig.10 shows the change of the infection rate when applying these schemes in a Gnutella system. Initially the infected population increases as the worm launches the attacks, it is then slowed down as the defenses take effects. Eventually the system reaches a stable state. Both our schemes outperform the basic approach in that they help the system retain a lower infected population.

6 Related Work

Recently some attention has been drawn to addressing various security problems in P2P networks. For example, there are schemes for addressing Sybil attacks [9], providing secure routing [5], increasing resilience of topology in the presence of a powerful adversary [15], detecting message dropping attacks [22], or key management [25]. However, The most deadly and imminent topological worm threats in P2P systems have not received due attention. Vojnovic et al. [17] demonstrated that effective automatic patching is feasible for an overlay network if combined with mechanisms to bound worm scan rate and with careful engineering of the patching system. Their work reveals a race between the worm spread and the patch dissemination. Zhou et al. [24] proposed an end-host based defense infrastructure for containing topological worms. Their scheme adopts the self-certifying alerts [8] and their preliminary results have shown effectiveness in protecting various overlays. Cai et al. [4] suggested a collaborative way of worm containment in a DHT-based overlay. This approach automatically generates worm signatures by analyzing payload contents and address dispersions to identify alerting traffic.

7 Conclusions and Future Work

Based on recent malware outbreaks in P2P systems, we predict that in the near future, topological worms will be increasingly destructive to users. Security agencies and the alike must be well prepared for the inevitable by employing effective defenses to combat the imminent threat. Current defenses are inadequate and we must keep one-step ahead and adopt new containment techniques to better protect P2P systems. In this paper, we have proposed two countermeasures based on automated secure patching. The first scheme proactively stops worm spreading in the graph topology and the second scheme reactively helps win the race with the worm. Our solutions have not touched issues such as impact of host diversity, feasibility of dynamic patching, and applicability to structured P2P networks, which we will address in our future work.

References

[1] http://www.symantec.com/avcenter/security/Content/7680.html.

[2] KaZaA and Fasttrack P2P network client buffer overflow vulnerability. http://secunia.com/advisories/8868/.

[3] B. Anthony and G. Blelloch. http://www.cs.cmu.edu/afs/cs/project/pscico-guyb/realworld.

[4] K. Cai, Y. Kwang, S. Song, and Y. Chen. Collaborative internet worm containment. In *IEEE Security and Privacy'05*, 2005.

[5] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and S. Wallach. Secure routing for structured peer-to-peer overlay networks. In *OSDI'02*, 2002.

[6] Y. Chen and A. Liestman. Approximating minimum size weakly-connected dominating sets for clustering mobile ad hoc networks. In *ACM MOBIHOC'02*, 2002.

[7] Z. Chen, L. Gao, and K. Kwiat. Modeling the spread of active worms. In *IEEE INFOCOMM'03*, March 2003.

[8] M. Costa, J. Crowcroft, M. Castro, A. Rowstron, L. Zhou, L. Zhang, and P. Barham. Vigilante: End-to-end containment of internet worms. In *SOSP*, 2005.

[9] G. Danezis, C. Lesniewski-Laas, M. Kaashoek, and R. Anderson. Sybil-resistant dht routing. In *OSDI'02*, 2002.

[10] M. Garey and D. Johnson. Computers and intractability: A guide to the theory of np-completeness. Freeman, New York, 1979.

[11] S. Guha and S. Khuller. Approximation algorithm for connected dominating sets. In *Algorithmica*, Apr. 1998.

[12] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. In *SIAM Journal on Scientific Computing*, 1998.

[13] J. Kephard and S. White. Directed-graph epidemiological models of computer virus. In *Proc. of 1991 Computer Society Symposium on Research in Security and Privacy*, May, 1991.

[14] B. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graghs. In *Bell Systems Technical Journal*, 1970.

[15] F. Kuhn, S. Schmid, and R. Wattenhofer. A self-repairing p2p system resilient to dynamic adversarial churn. In *OSDI'02*, 2002.

[16] S. Kutten and D. Peleg. Fast distributed construction of k-dominating sets and applications. In *PODC'95*, 1995.

[17] M.Vojnovic and A. Ganesh. On the effectiveness of automatic patching. In *WORM'05*, 2005.

[18] M. Ripeanu, I. Foster, and A. Iamnitchi. Mapping the gnutella network: Properties of large-scale peer-to-peer systems and implications for system design. In *IEEE Internet Computing Journal*, 2002.

[19] D. Stutzbach, R. Rejaie, and S. Sen. Characterizing unstructured overlay topologies in modern p2p file-sharing systems. In *Internet Measurement Conference*, 2005.

[20] H. Wang and et al. Shield: Vulnerability-driven network filters for preventing know vulnerability exploits. In *ACM SIGCOMM*, 2004.

[21] N. Weaver, V. Paxson, S. Staniford, and R. Cunninggham. A taxonomy of computer worms. In *USENIX Security'04*, 2004.

[22] L. Xie and S. Zhu. Message dropping attacks in overlay networks: Attack detection and attacker identification. In *Proceedings of International Conference on Security and Privacy in Communication Networks (SecureComm)*, 2006.

[23] E. Zegura, K. Calvert, and S. Bhattacharjee. How to model an internetwork. In *IEEE INFOCOMM'96*, March 1996.

[24] L. Zhou and et al. A first look at peer-to-peer worms: threats and denfenses. In *IPTPS'05*, 2005.

[25] S. Zhu, C. Yao, D. Liu, S. Setia, and S. Jajodia. Efficient security mechanisms for overlay multicast-based content distribution. In *Proc. of Applied Cryptography and Network Security (ACNS) conference*, 2005.