On the Effectiveness of Internal Patching against File-sharing Worms

Liang Xie*, Hui Song‡, and Suncun Zhu*†

*Department of Computer Science and Engineering, The Pennsylvania State University
†College of Information Sciences and Technology, The Pennsylvania State University
‡Computer Science Department, Frostburg State University
lxie@cse.psu.edu, hsong@frostburg.edu, szhu@cse.psu.edu

Abstract. File-sharing worms have been terrorizing Peer-to-peer (P2P) systems in recent years. Existing defenses relying on users' individual recoveries or limiting users' file-sharing activities are ineffective. Automated patching tools such as Microsoft Windows Update and Symantec Security Update are currently the most popular vehicles for eliminating and containing Internet worms, but they are not necessarily the best fits for combating P2P file-sharing worms, which propagate within a relatively smaller community. In this paper, we propose a complementary P2P-tailored patching system which utilizes the existing file-sharing mechanisms to internally disseminate security patches to those participating peers in a timely and distributed fashion. Specifically, we examine the effectiveness of leveraging the file downloading or searching process to notify vulnerable end hosts of the surging worms and push corresponding security updates to these hosts. We show through in-depth analysis and extensive experiments that both methods are scalable and effective in combating existing P2P worms.

1 Introduction

P2P file-sharing programs such as KaZaA, iMesh, Morpheus are popular Internet applications that allow users to download and share electronic files. As one of the most popular networks, KaZaA has four million simultaneous users. The powerful data access feature, however, brings about unique privacy and security threats in these systems. In addition to adware and spyware, P2P hosts are placed at the risk of various viruses and malicious codes [13].

Our focus in this paper is *file-sharing worms*[1], which are malware spreading through file-sharing activities within P2P systems. Specifically, a user searches for a file in the network and acquires a list of accessible targets among which there could be a disguised one provided by some infected machine. Unwittingly she downloads the file and opens it, resulting in her own machine being infected. Recently, many file-sharing worms have been reported, e.g., Benjamin.a, Franvir, Bare.a, Darby.m, and Duload worm that are actively attacking KaZaA, Gnutella, and eDonkey2000 networks [3, 7]. One experimental study reported that 44% of the 4,788 executable files downloaded through a KaZaA client program contain malicious code [25]; another experimental study revealed that

[1] Like mass-mailing worms, file-sharing worms also require human's operations to propagate. As such, they are sometimes referred to as viruses or malware.

12% of the KaZaA client hosts were infected by over 40 different worms in February and May, 2006 [21]. Some disastrous consequences of attacks from file-sharing worms include opening backdoors, changing system registries and client configurations, and collecting clients' confidential data [7].

It would not be surprising that worms will increasingly exploit file-sharing applications as their major infection vector. However, research on these imminent threats has just started and most existing work focused on modeling worm behavior such as infections and propagation in file-sharing environments [11, 17, 23]. The problem of quarantining these worms has not been adequately addressed. Currently the best defense mirrors the strategy against Internet computer viruses with the inception of security patches from vendors (e.g., Microsoft, Symantec, McAfee). The method of automated security update is widely employed by security servers to automatically push the latest security patches to Internet hosts[24]. This generic solution certainly helps protect Internet hosts at the earliest stage of worm spreads, but it is not P2P-oriented. That is, security servers either have to blindly deliver P2P patches to all the Internet hosts (including those non-P2P machines and those who have installed P2P software but are not executing it), or have to scan for currently running P2P client programs within each Internet host before sending a patch to it. In both cases, system resources and network bandwidth could be greatly wasted. Moreover, unnecessary security updates could cause annoying machine reboots (sometimes required for a complete security update), which unavoidably interrupt those non-related users' on-going tasks running on their hosts.

Contribution: In this paper, *we study the feasibility of utilizing the existing file-sharing infrastructure to internally push security updates to the participating nodes in P2P systems*. We propose two BitTorrent-like mechanisms for distributing the security patches. In file-sharing networks such as Gnutella, a very small fraction (5%) of hosts usually provide a large fraction of the shared files (70%) [14]. Exploiting this asymmetry in file-sharing, we consider first disseminating the security patches to these popular hosts, such that most of the other participating hosts can receive the patches from these popular hosts when they actively download files from them. Our second approach is based on the belief that P2P users as a community should help each other in combating worm attacks. Therefore, when a host detects worm infection from a downloaded file, it first re-performs a search on the infected file to identify those hosts possessing the same file, and then it collaboratively notifies these hosts of the worm information as well as the security patch. Based on a modified fluid model, we analyze worm spreads and evaluate the effectiveness of our approaches in unstructured networks. Our result demonstrates that both schemes can help a file-sharing system with 20, 000 hosts achieve a high immunity rate (90%) within a few dozens of hours after the initial worm surge.

Our solutions are not a substitution for the existing automatic patching systems but rather a nice complement to them. Our proposed techniques are not necessarily very complex, but our work, backed up with solid analytic modelling and extensive experiments, makes a concrete movement towards solving the important security problem facing many P2P users. Also, our solution is scalable and easy to deploy by leveraging the existing P2P infrastructure, without involving a dedicated Content Distribution Network (CDN) (e.g., Akamai).

Organization: The rest of the paper is organized as follows. Section 2 describes an attack model for the network, as well as the design principles and the assumptions. Section 3 and 4 introduce two internal patching schemes for securing participating hosts in P2P systems. We evaluate the schemes in Section 6 and describe the related work in Section 7. We conclude in Section 8.

2 Preliminaries

Network Model Many P2P file-sharing systems are actively running in these days (a comparison can be found in [2]). The most popular ones include eMule, KaZza, Gnutella, and BitTorrent. For concreteness, however, our discussion will focus on those unstructured networks such as Gnutella and KaZza. In Section 5 we will briefly mention how our approaches can be extended to other P2P systems such as BitTorrent.

We use Gnutella as an example to describe the file-sharing process. Specifically, each node uses a *shared folder* to store those files it wishes to share. When a requesting node initiates a download request for a specific file, it places a search for the target node(s) responsible for the given file identifier. The search request is routed through a two-tiered system of ultra-peers and leave nodes in the Gnutella overlay. In response, the requester collects a list of peers, each of which contains a file copy (probably with different versions). The requester then connects to one target node in the list and downloads the copy. Finally, she opens the downloaded file for use.

Attack Model A file-sharing worm usually copies itself to a host's shared folder and publishes it with an attractive name, for example, as a popular song or movie. Sometimes attackers replace real movie or sound files with their malicious copies or add executable extensions to such files. When a host searches for some file and finds an match from an infected machine, it downloads and opens the file without being aware of the threat. Consequently, the worm is activated and it copies/attaches itself to all the files in the shared folder(s) of this new victim. In this way, a file-sharing worm continues its spread cycle.

We define two states for a file in P2P systems: *normal* and *abnormal*. A file is normal when it is valid and clean, and it becomes abnormal once malicious codes have been injected or attached to it. Also, we define three states with respect to a surging worm for each host in the systems: *vulnerable*, *infected*, and *immune*. A vulnerable host is not well-protected against the worm, hence it gets infected when exposed to the attack. For example, when a user opens an downloaded file which is abnormal, all files inside the shared folder(s) of this infected machine consequently become abnormal. A vulnerable/infected node becomes immunized once the protection (e.g., a patch) has been in place. Fig 1 illustrates the node state transitions. We note that in real applications, some P2P users could voluntarily install the vendor patch on their machines. Therefore, these nodes are initially immunized to the worm. For simplicity, we assume no such individual recoveries occur during the period of defense.

We notice that a few elaborated worms such as Worm.Win32.Hofox were recently reported to be able to block the anti-virus protection services or kill anti-virus programs on P2P hosts. Clearly, at the system level, some local countermeasures will be devised to protect defense tools from being eliminated, and the arms race will continue. In this paper, however, we assume that P2P worms cannot disable the patching protocol

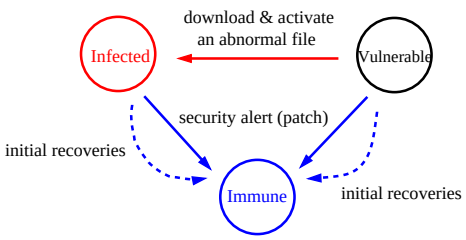


Fig. 1. Node state transition during the defense of internal patching. Initial recoveries include individual updates from security vendors; its percentage is relatively low as a new worm surges.

deployed in end hosts, so that an infected host can receive patches and become immunized as it is expected. Note that this assumption will not affect the correctness of our approaches, and our analysis model presented later can be slightly changed for the case when this assumption does not hold.

System Design Overview To effectively combat file-sharing worms, we cannot merely rely on users' precaution and worm elimination skills; rather, we need *an automated and systematic approach to disseminate security patches to the P2P users*. Existing automated patching systems can be utilized to secure P2P hosts as well by simply treating them as normal Internet hosts; however, they are not necessarily the best-fit choices because not all Internet hosts are equally exposed to those P2P worms. For P2P users who often download files, their machines are more likely to be affected by P2P worms, whereas for non-P2P users, their machines will not be affected by worms exploiting P2P applications. Moreover, a traditional centralized model of patch distribution could cause single-point failure or overloading on the patch servers.

As such, we are motivated to study a P2P-tailored automated patching mechanism as a supplement to existing solutions and examine its effectiveness. Our approach utilizes the existing file-sharing infrastructure to internally push security updates (alerts) to the participating peers. It has several good features. First, it is customized for P2P environments and delivers security updates to P2P hosts only. This avoids unnecessary consumption of network/computer resources. Second, it adopts a distributed manner to disseminate security patches to those vulnerable peers in need and no longer strains the central servers. Third, our push-based scheme delivers security updates more promptly than the traditional once-a-day update adopted by existing patching systems.

Internal patching should leverage the existing file-sharing infrastructure for distributing security patches. Approaches utilizing IP address scanning or topology exploration to locate alive patching targets bring extra computation and communication overhead to P2P systems. Moreover, these methods could be easily exploited by malicious users and used as the vehicle for rapid worm spread and denial-of-service attacks.

In this paper, we study two push-based patching mechanisms for P2P systems. We first examine a download-based approach, in which a small fraction of popular nodes (also referred as *key nodes*) act as early patch distributors and a node which downloads a file from a key node will also be offered with a security patch/alert. Thus, the patch is propagated to many active hosts along with the file-download process. We then examine a search-based approach, in which once a key node detects worm infection in a

downloaded file, it re-performs a file search to identify those active hosts that possibly possess the abnormal file and disseminates the patch to immunize/disinfect them.

3 A Download-based Approach

In our download-based approach, a small set of key nodes internally push the security patches to participating peers through the file-downloading process. Nodes that are notified of the approaching threat hence have a good chance of being immunized/disinfected against the worm. Our design of the scheme answers the following questions.

- Which hosts are determined as the key nodes so that they can distribute patches to others in a most efficient and timely manner? Key nodes cannot be determined in a centralized mode because no node in the system holds a global knowledge of file-sharing activities of others.
- What is a user's strategy to choose the download source and what is the impact on patch dissemination? How should the existing P2P file transfer protocol be adapted to support the patch dissemination?
- How does a recipient authenticate a patch that it receives? In a distributed environment, even if a public key infrastructure (PKI) is deployed to provide sender authentication, it cannot prevent malicious peers from injecting worm codes instead of security patches. In other words, a node cannot fully trust others in the P2P system. Moreover, how does the recipient deal with the patch and how does her decision affect the immunity level of the system?

3.1 Scheme Description

Bootstrapping Key Nodes The first important issue is the choice of key nodes. In most decentralized systems such as Gnutella and KaZaA, downloading traffic is highly focused around a small minority of popular targets and these popular files tend to be gradually concentrated in a small set of providers. For example, in Gnutella, 50% of all files are served by just 1% of nodes and 98% of all files are shared by the top 20% nodes [14]; in KaZaA, 10% most popular files generate 60% of the download traffic and 70% of the highly popular files will remain popular for at least 10$\sim$15 days [18]. These are strong indications that a small fraction of popular hosts sharing the most interesting files could be conveniently leveraged as the early distributors, which effectively push security patches to active downloaders in the system.

We consider a distributed algorithm for bootstrapping such key nodes in the P2P systems. Specifically, a small set of key nodes are individually decided according to a predefined policy. These key nodes then automatically pull (download and launch) the patch from vendor, so that they become immunized against the surging worm. To describe this algorithm in detail, we first introduce a node parameter named *file-offering rate* ϕ_O, which is defined as the number of files a node offers to its requesters in unit time. Note that this parameter reflects the popularity degree of the node. Each node calculates its ϕ_O based on its own file-sharing history. For example, node i may derive $\phi_O(i) = D_{out}(i)/T_f$, where $D_{out}(i)$ denotes i's out-degree in its file-access graph

within its neighborhood time window T_f. Thus, we can adopt the following policy to bootstrap the key nodes: *key nodes are selected from a subset of popular nodes with the highest file-offering rates in the system.* Specifically, each candidate node i refers to its recent file-offering rate $\phi_O(i)$ and decides to be a key node only if $\phi_O(i) \geq H_O$ satisfies. Here H_O is a globally defined threshold, which controls the fraction of key nodes. This policy can be automatically enforced through the client program. Once a node decides to become a key node, it should automatically fetch the latest security updates (if there are any) from the trusted vendor(s) and immediately launch the protection on the local machine (another option is they register to the vendors so that the vendors may push the latest security patch to them once available). Thus, key nodes get immunized against the surging worm and are ready to secure other file requesters. We note that as an active holder of more popular files, the user has to sacrifice a little bit convenience (patch activation if needed) and bandwidth (patch transfer) for the sake of the security of the entire system. On the other hand, a key node could be malicious or a regular node may claim to be a key node. We will discuss the related security issue shortly.

Disseminating Security Patches Next, we discuss the message format of a security patch generated by the key node. This patch is used to notify the receivers of the worm threat and to provide the source of the security update. As illustrated in Fig 2, a patch message MSG_a typically contains two parts: a message header which contains the key nodes's identifier, and a message payload which contains (1) the worm alert (name, type, severity level, etc..), (2) the security patch itself (e.g., a Microsoft XP patch in binary delta compression format [6]) or simply a link to the URL of that patch (e.g., the Microsoft Security Bulletin), (3) a vendor signature of (1) and (2). We note that for a specific worm (identified by a specific vendor), the payload of its security patch message is unique. In addition, the security patch is *self-verifiable*: either the signature from a well-known vendor is attached to the patch, or the patch link can be directly verified through the vendor's website. This mechanism does not require a recipient to authenticate the intermediate patch distributors. Instead, it verifies the authenticity of the message content with the vendor or through its web site – these are considered more reliable and trustable.

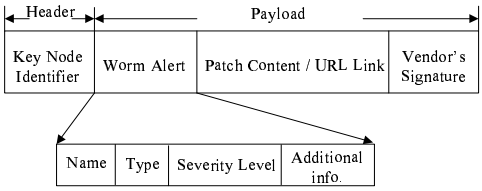


Fig. 2. Message format of security patch MSG_a

The next issue is how key nodes leverage the existing P2P file transfer protocol to internally distribute the security patch to file downloaders. Using Gnutella 0.6 system as an example, search results are directly delivered to a client (requester) through UDP packets. If the client chooses a resulting node for file download, it typically sends

an HTTP Request to the provider and reads the bit stream of file content that follows the HTTP Response [1]. We propose that the client also includes its latest patch version in the HTTP Request. Thus, a key node verifies the request before its file transfer and decide whether to distribute the latest security patch. To notify the recipient of the patch existence, the key node simply sets an indicator and the patch length in its HTTP Response. Both sides may establish an additional channel or use the existing control channel to perform patch transfer. In the case when the provider is behind a firewall, a PUSH process is executed to establish the connection [1] and the remainder of the file download and patch dissemination is similar as we have described.

Client Strategy and User Behavior In response to a file search, a client receives a set of replies pointing to different file providers. The main decision that the client needs to make is which one of these node to ask for a copy of the file. This choice clearly has some influence on the defense. We consider the following three major selection strategies:

Random. The client selects a random node, independent of the node's advertised resources. In this mode, when there are α percentage of key nodes in the system, every client has an equal chance of α to download the file from a key node. This also implies that every client will eventually receive the security patch from key nodes.

Best. The client selects the node that advertises the best performance, i.e., the node with the lowest estimated delay (the node's queue length times the file size times the maximum number of simultaneous uploads divided by the access link bandwidth). Unlike the random mode, in this mode every client has a higher chance of downloading files from key nodes (i.e., those popular file holders). However, a small fraction of nodes which are not interested in the popular files may not have a chance to receive the security patch from key nodes.

Redundant. The client performs redundant download from either randomly chosen C nodes or C nodes with the lowest estimated delay. Once the first download finished and the content is verified for correctness, the other downloads are stopped. When the file download from a key node is aborted, the client cannot receive the patch that follows.

Next, we discuss how recipients react to the security patch. Although our scheme effectively leverages the internal infrastructure to expedite patch dissemination and ensures most participating pees receive the update as the file downloads proceed, the immunity level of the system is still in some degree dependent on individual users' responses to the patch. Upon receiving a patch message MSG_a, the client program first examines the payload and compares it with the existing version in order to discard out-dated or duplicated patch content. An accepted patch notifies the user of a surging worm and reminds her to launch immediate protection. If the user accepts the patch, the client program authenticates the patch payload either by directly examining the vendor signature or by visiting the trusted vendor site and verifying if the patch link provides consistent worm information. The program applies the new patch (a download is possibly needed) on the local machine immediately after a successful verification. Thus, the local machine gets immunized/disinfected against the worm and consequently all the files in its shared folder are/become normal. However, when a user declines the offer, either unwilling to follow the link or failing to activate the patch, her machine remains

vulnerable to the worm. We quantitatively analyze the impact of user behavior on the system immunity in Section 3.2.

Table 1. Notation for worm propagation model

Note.	Explanation
N	the total number of hosts in the network
$V(t)$	the number of vulnerable hosts
$I_f(t)$	the number of infected hosts
$I_m(t)$	the number of immune hosts
$F(t)$	the total number of files
$h(t)$	the proportion of abnormal files in the system
$s(t)$	the average size of a shared folder
λ_d	the average rate of file download (files/hour)
λ_a	the probability a user activates the downloaded file
α	the percentage of key nodes in the system
β	the probability at which a user accepts a patch

3.2 Security and Performance Analysis

We derive a new fluid model for worm propagation and analyze the security and performance of the download-based approach. We refer to notation in Table 1.

A Fluid Model for Worm Propagation We first consider the case when no defense has been deployed in the system. Each node is either in vulnerable or immune state, i.e., relation $N = V(t) + I_f(t)$ always satisfies. We show the evolution status of the system under the worm threat. The vulnerable population decreases as some nodes unfortunately download abnormal files, activate these files and get infected. We have

$$\frac{dV(t)}{dt} = -\lambda_d \lambda_a \cdot V(t) \cdot h(t). \tag{1}$$

Here $1/\lambda_d$ is the average time a node takes to download a file, and $h(t)$ reflects the percentage of abnormal files at time t. Solving the above differential equation, we get

$$V(t) = N - I_f(t) = V(0) \cdot e^{-\lambda_d \lambda_a \int_0^t h(\tau) d\tau}, \tag{2}$$

where $V(0)$ denotes the initial number of vulnerable hosts. This equation indicates that the vulnerable population in the system decreases exponentially as there are more file downloads and activations; the increase of the proportion of abnormal files accelerates the worm spread. We further derive the file state.

Lemma 1 *In a P2P file-sharing system, the percentage of abnormal files can be computed as* $h(t) = h(0) \cdot e^{\frac{\lambda_d \cdot \lambda_a}{N} \int_0^t V(\tau) d\tau}$*, where* $h(0)$ *is the initial abnormal rate. An approximation of* $h(t)$ *can be computed as* $h(t) \approx \frac{I_f(t)}{N}$*, assuming* $\lambda_a \to 1$.

This lemma is proved in Appendix A. It shows that user behavior has significant impact on the percentage of abnormal files: more file downloads and activations lead to more infections. However, as the amount of files increases and the vulnerable population decreases, worm infection is gradually slowed down.

Analysis of the Download-based Defense Time Performance Next, we examine the download-based defense. For simplicity, we assume users always adopt the random strategy to choose file providers (see 3.1) and all infected hosts have a patch acceptance probability β. We define *immunity rate* $i(t)$ as the fraction of immune nodes $i(t) = I_m(t)/N$, and let the initial immune population be $I_m(0)$. Note that these nodes, including the key nodes, either have applied the patch or does not expose the software vulnerability to the worm.

To study how long it takes to achieve a certain level of immunity rate, we formalize the problem as finding a lower bound t_0 for time t, so that we have $i(t) \geq \Psi$ when $t \geq t_0$, where $\frac{Im(0)}{N} \leq \Psi \leq 1$ is a predefined threshold.

Lemma 2 *In a file-sharing system which adopts the download-based defense, the number of immune nodes is* $I_m(t) = N + (I_m(0) - N) \cdot e^{-\lambda_d \alpha \beta t}$ *and the system takes at least* $t_0 = \frac{1}{\alpha\beta\lambda_d} \ln \frac{N - I_m(0)}{N(1-\Psi)}$ *hours to achieve an immunity rate* Ψ.

Proof. From the state diagram in Fig 1, we know that $N = V(t) + I_f(t) + I_m(t)$ always holds. Each time when a node downloads a file from a key node, it also receives a patch and the user decides whether or not to accept it. Note that only infected and vulnerable $(I_f(t) + V(t))$ nodes are immunized/disinfected in this process. We derive the change of immunity rate.

$$\frac{dI_m(t)}{dt} = (I_f(t) + V(t))\lambda_d \alpha\beta = (N - I_m(t)) \cdot \lambda_d \alpha\beta. \tag{3}$$

Here α also denotes the probability that each client selects a key node as the provider. Solving this differential equation for $I_m(t)$, we get the number of immune nodes in the system

$$I_m(t) = N + (I_m(0) - N) \cdot e^{-\lambda_d \alpha\beta \cdot t}. \tag{4}$$

From the given condition $i(t) = I_m(t)/N \geq \Psi$, we may further derive $t \geq t_0 = \frac{1}{\alpha\beta\lambda_d} \ln \frac{N - I_m(0)}{N(1-\Psi)}$.

Fig 3 illustrates the change of immunity rate $i(t)$ when the percentage of key nodes (α) varies from 5% to 15%. Clearly, as there are more patch distributors, the system takes less time to reach a certain level of immunity rate (in our case 90%). For example, when $\alpha = 5\%$, it takes 60 hours for 90% of nodes to receive the patch, whereas it takes 20 hours when $\alpha = 15\%$. This figure also shows that in the random selection mode, each downloader (including those not interested in the popular files) will eventually receive the patch from a key node.

System Evolution Status We also examine the evolution status of the system which adopts the download-based defense. During the worm containment, a vulnerable host either (1) becomes infected when it downloads and activates an abnormal file from a

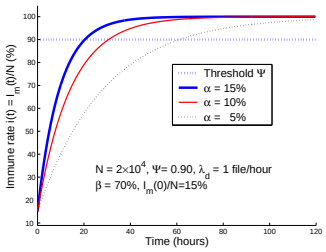


Fig. 3. Immunity rate as a function of time an and key nodes

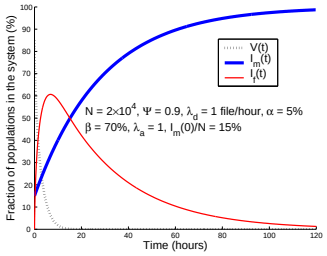


Fig. 4. System evolution status under the download-based defense

non-key node, or (2) gets immunized when it downloads a file from a key node and accepts the patch. Hence, we set up the following equation for the change of the vulnerable population.

$$\frac{dV(t)}{dt} = -\lambda_d \lambda_a (1-\alpha) \cdot V(t) \cdot h(t) - \lambda_d \alpha \beta \cdot V(t). \tag{5}$$

Note that here $\lambda_d \lambda_a (1-\alpha) \cdot V(t) \cdot h(t)$ computes the reduction caused by (1) ($1-\alpha$ denotes the probability of downloading from a non-key nodes) and $\lambda_d \alpha \beta \cdot V(t)$ computes the reduction caused by (2). We use the approximation $h(t) \approx \frac{I_f(t)}{N}$ and the solution in Equ.4 to solve this differential equation for $V(t)$. We also compute $I_f(t)$ using the relation $N - I_m(t) - V(t)$. Fig 4 illustrates the evolution status of a file-sharing system. Initially, the percentage of infected nodes increases as the worm surges. However, when more and more file downloaders receive the patch, worm infections are gradually cleansed from the network and the infected population starts to decrease. Eventually, immune nodes become the major population. The figures also indicate that the immune time t_0 is determined by several factors: the fraction of the key nodes (α), the file downloading rate (λ_d), patch acceptance rate (β) and the initial immunity rate. Our analytical result has been validated in Section 6 (Fig 8).

4 A Search-based Approach

This section proposes a search-based approach, in which once a key node detects worm infection in a file it has just downloaded from other participating peers, it immediately infer from a new search result a set of suspicious targets, to which it directly pushes the security patch and disinfect/immunize them. Given the latest vendor updates, we assume key nodes are able to detect on-going worm attacks based on techniques such as worm signature matching, taint analysis or anomaly detection [9, 16, 20, 19]. We answer the following questions in our design.

– Which hosts in the system should be chosen as the key nodes, so that they detect file anomalies in the system and distribute patches to others in a most efficient and timely manner? Key nodes should be bootstrapped in a distributed way.

– Once a key node detects file anomaly, how does it infer a set of suspicious nodes by examining the query response and how to deal with network dynamics?
– How does a key node disseminate the patch to those suspicious nodes? To be scalable, how should a key node limit its bandwidth for patch delivery?
– What is the user's reaction towards the patch and how does it influence the immunity level of the system?

4.1 Scheme Description

Bootstrapping Key Nodes To address the first issue, we consider a distributed algorithm to bootstrap key nodes in our search-based scheme. Similar to the algorithm in Section 3.1, key nodes automatically pull (download and install) the latest vendor patch so that they become immunized against the surging worm. However, here we adopt a different policy for determining key nodes in a distributed way. We first introduce a node parameter named *file-downloading rate* ϕ_I, which is defined as the number of files a node downloads from others in unit time. This parameter reflects a node's activity level of file downloads. Each node i derives $\phi_I(i) = D_{in}(i)/T_f$, where $D_{in}(i)$ denotes the number of files i has downloaded within the time window T_f. Now we can adopt the following policy to bootstrap the key nodes: *key nodes are selected among a subset of nodes with the highest file-downloading rates in the system.* Specifically, each candidate node i refers to its recent file-downloading rate $\phi_I(i)$ and decides to be a key node only if $\phi_I(i) \geq H_D$ satisfies. Here H_D is a globally defined threshold which controls the fraction of key nodes. This bootstrapping policy is automatically enforced through the client program within an end host.

The above policy chooses those active file requesters as the key nodes because these nodes keep actively acquiring files from various origins, hence their chance of being infected is relatively higher than hosts with relatively low downloading rate. Keeping these nodes updated with the latest vendor patch also enables them to explore more worm infections from file providers. Our search-based scheme requires a key node P immediately examines the file state after it finishes downloading a file f_p. Once an anomaly has been identified, the key node composes a security patch message Msg_a.

Distributing Security Patches The next issue is to which nodes the security patch should be distributed. Pushing the patch directly to the provider who has uploaded the abnormal file is effective. However, this is not efficient because the key node has a good reason to suspect that other file-owners may have also been infected. On the other hand, simply flooding the patch or locating the targets by IP address scanning or topology exploration is not scalable. Our solution is to let the key nodes exploit the file search list to locate those *suspicious* file providers and push the patch to these targets. However, there exist a time gap (could be in hours) between the original search and the worm detection. During this period, nodes frequently join and leave the network. A good strategy for the key node is to re-perform a file search once it has detected a file anomaly.

We propose a distributed algorithm for patch dissemination, as illustrated in Fig 5. Specifically, once a key node P has detected worm infection in a downloaded file f_p, it immediately re-perform a search on f_p and consequently receives multiple QueryHit responses, based on which it sorts the destination nodes according to the *activity level* and constructs a *ranked* search list S_p. Here a node i's activity level $L_a(i)$ is derived from

three parameters: the bandwidth of its access link $Spd(i)$, the queue length $QLen(i)$ and the number of simultaneous uploads $N_{up}(i)$. These parameters for node i are included in the QueryHit message, i.e., we have $L_a(i) = f(Spd(i), QLen(i), N_{up}(i))$, where f is a monotonically increasing function. The application in the key node then computes an activity lower bound $H_L(P)$ based on the bandwidth $Spd(P)$ and the current number of connections (ongoing P2P traffic) in the local machine. Finally, the key node chooses from S_p top k target nodes whose activity level satisfies $L_a \geq H_L(P)$ and establishes a direct HTTP connection with each of these suspicious targets to push the security patch. Note that such patch transfers are out-of-band (not through the Gnutella overlay).

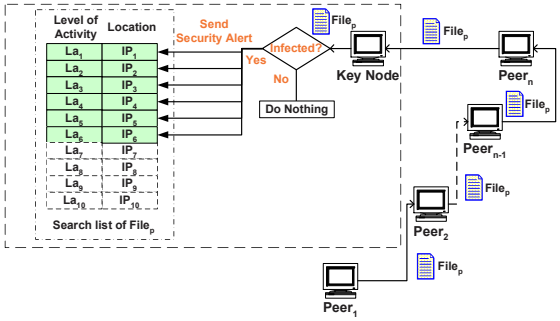


Fig. 5. An illustration of the search-based approach. In this example, key node P detects an infected file f_p and delivers patch MSG'_a to $k = 6$ suspicious nodes in the search list S_p.

Here a security patch MSG'_a also contains the identifier of the infected file f_p. Upon receiving the patch, each node j needs to verify the existence of f_p and then displays a warning in the local machine. If the user accepts the patch, the application first authenticates the patch content/link. Once the patch has been successfully applied, node j becomes immunized to the worm and its shared folder will be immediately scanned and cleansed. We quantitatively analyze the impact of user behavior on the system immunity level in Section 4.2.

4.2 Security and Performance Analysis

Time Performance We analyze the effectiveness of the search-based defense. Let k denote the average number of suspicious targets to which a key node distributes the security patch, we first derive how long the system takes to achieve an immunity rate Ψ. According to the state diagram in Fig 1, $N = V(t) + I_f(t) + I_m(t)$ always holds. In the search-based scheme, the increased immune population comes from either vulnerable nodes or infected nodes. Hence, the rate at which the immune population increases can be computed as

$$\begin{aligned} \frac{dI_m(t)}{dt} &= N\lambda_d \cdot \alpha h(t)\left(\frac{N - I_m(t)}{N}\right) \cdot k\beta \\ &= a(N - I_m(t)) \cdot h(t), \end{aligned} \tag{6}$$

Where $a = \alpha\beta\lambda_d k$. Note that $\alpha h(t)(\frac{N-I_m(t)}{N})$ computes the probability that a key node downloads an abnormal file from those non-immune hosts.

Also, the decrease of the vulnerable population could be caused either by (1) worm infections or (2) by host immunizations (or disinfections). Therefore, the rate at which vulnerable nodes become either infected or immunized is

$$\begin{aligned}\frac{dV(t)}{dt} &= -\lambda_a\lambda_d V(t)h(t) - N\lambda_d \cdot \alpha h(t)(\frac{V(t)}{N}) \cdot k\beta \\ &= -(a+b)V(t)h(t). \end{aligned} \tag{7}$$

where $b = \lambda_a\lambda_d$. Here $\lambda_a\lambda_d V(t)h(t)$ computes the reduction caused by (1); $N\lambda_d \cdot \alpha h(t)(\frac{V(t)}{N}) \cdot k\beta$ computes the reduction caused by (2), in which $\alpha h(t)(\frac{V(t)}{N})$ denotes the probability a key node downloads an abnormal file from a vulnerable host (not infected yet). In this case, the latter receives the patch and could be immunized.

Finally, we know that the infected population (1) increases when some vulnerable nodes get infected, and (2) decreases when some victim nodes have been disinfected. Hence we derive the following differential equation.

$$\begin{aligned}\frac{dI_f(t)}{dt} &= \lambda_a\lambda_d V(t)h(t) - N\lambda_d \cdot \alpha h(t)(\frac{I_f(t)}{N}) \cdot k\beta \\ &= bV(t)h(t) - aI_f(t)h(t). \end{aligned} \tag{8}$$

Note that here $\lambda_a\lambda_d V(t)h(t)$ computes the increase of infected nodes caused by (1); $N\lambda_d \cdot \alpha h(t)(\frac{I_f(t)}{N}) \cdot k\beta$ computes the reduction of infected nodes caused by (2), where $\alpha h(t)(\frac{I_f(t)}{N})$ denotes the probability a key node downloads an abnormal file from an infected host. In this case, the latter receives the patch and could be disinfected. To solve these differential equations, we derive the immune population $I_m(t)$. We divide Equ.6 by Equ.7 and get

$$V(t) = V_0^{-\frac{b}{a}} \cdot (N - I_m(t))^{\frac{a+b}{a}}, \tag{9}$$

where $V_0 = V(0)$ denotes the initial vulnerable population in the system. We then apply the approximation $h(t) \approx \frac{I_f(t)}{N} = \frac{N-I_m(t)-V(t)}{N}$ and substitute Equ.9 into Equ.6. Thus, we have

$$\frac{du}{dt} = -\frac{a \cdot V_0}{N} \cdot u^2(1 - u^{\frac{b}{a}}) \tag{10}$$

where $u = (N - I_m(t))/V_0$. We further solve this equation for $I_m(t)$ and illustrate the change of $I_m(t)$ in Fig 6. This figure indicates that under an average size $k = 30$, the search-based approach only needs to deploys 5% nodes as key nodes and help the system achieve a 90% immunity rate within 60 hours.

System Evolution Status Adopting the similar method as above to solve Equ. 6, 7, 8, we derive the following

$$\frac{dV(t)}{dt} = -\frac{a+b}{N}(c \cdot V^{1+\frac{a}{a+b}}(t) - V^2(t)), \tag{11}$$

where $c = V_0^{\frac{b}{a+b}}$. Hence we may compute $V(t)$. Using $I_f(t) = N - V(t) - I_m(t)$, we may further derive the infected population $I_f(t)$. We illustrate the system evolution

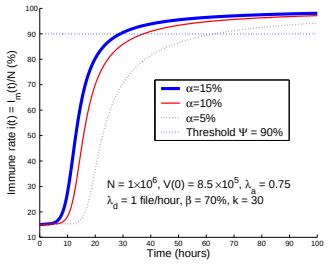


Fig. 6. Immune population vs. time and key nodes

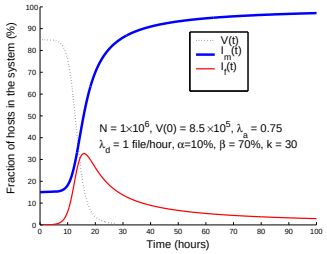


Fig. 7. System evolution status in search-based scheme

status in Fig 7. The figure shows that the infected population initially increases due to the surging worm. However, this triggers the defense and many suspicious nodes receive the patch. Eventually worm infections are eliminated and immune nodes become the major population. The above analytical result has been validated in Section 6 (Fig 8).

5 Security Analysis

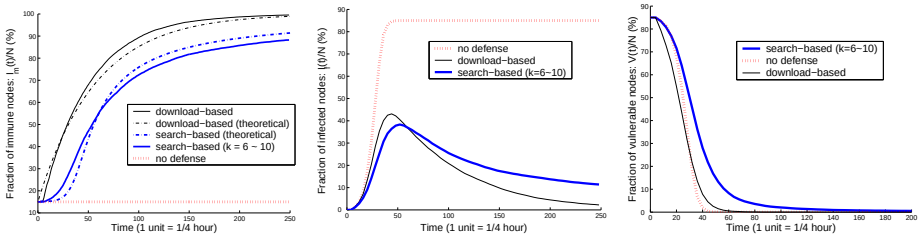
We discuss two attacks that may happen in both schemes.

Fake Security Alerts A malicious node, either a key node or a regular node claiming to be a key node, may replace security patches with worms and deliver them to other hosts. This attack will fail because our signature-based mechanism allows a receiver to verify if the patch truly comes from a trusted vendor or the link to the patch is correct. On the other hand, we notice that a lot of false messages may cause a DoS attack to other hosts. Since we do not assume a PKI, P2P nodes may not be able to authenticate each other. Indeed, even a PKI is available, it does not solve this type of insider attacks. A simple solution is that a node blacklists the nodes reporting false alerts based on their IP addresses. To prevent IP spoofing, before a node accepts a security alert, it challenges the source.

Patch Suppression Attack A malicious (or selfish) candidate key node may not propagate security patches. That is, in the download-based approach, it does not offer the security patches to downloaders and in the search-based scheme it does not care about other susceptible nodes. This patch suppression attack will degrade the effectiveness of our schemes. However, it only decreases the actual α. As long as they are not a lot, our schemes will still work. Otherwise, we should increase the value of α.

6 Evaluation

Environmental Setup We evaluated and compared our schemes in a variety of file-sharing systems. For unstructured networks we implemented a Gnutella simulator based on Gnutellasim from limewire.org; for structured networks we used P2PSim ([5]) to



(a) Fraction of immune nodes vs. time (b) Fraction of infected nodes vs. time (c) Fraction of vulnerable nodes vs. time

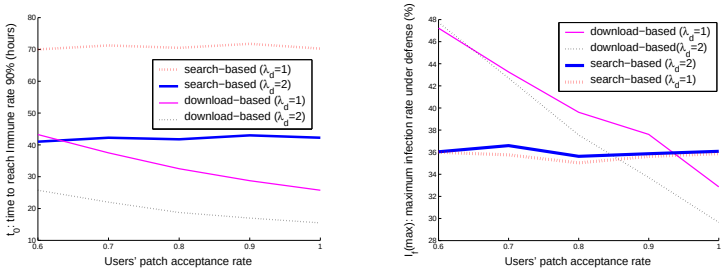
Fig. 8. Comparing time performance of the approaches in Gnutella 0.6; N=20k nodes, M=1k files, λ_d=1 file/hour, λ_a= 1.0, α=10%, β=0.7

construct a basic Chord [22] infrastructure for routing queries and responses. We implemented a protocol similar as NeuroGrid [15] to generate large-scale file-sharing traffic on top of the routing infrastructures. We studied the case when a file-sharing worm Benjamin.a [4] surges in the network and evaluated the effectiveness of our countermeasures.

We adopted the following metrics in our evaluations: t_0, time takes to reach an immunity rate $\Psi = 90\%$; $h(t_0)$, the the percentage of abnormal files at time t_0, which also reflects the infection rate $I_f/N(t_0)$ when $\lambda_a \rightarrow 1$; $I_f(max)$, the maximum infection rate which indicates how severe the system has been attacked. For each scheme, we also investigated the system evolution status, the impacts from user behavior and the message overhead. To examine the schemes' tolerance against node dynamics (joins/departures), our implementation followed the observations from Gnutella 0.6, i.e., 45% of the nodes quit the network in less than $4 \sim 5$ hours, and 22% persistent node tend to stay in the network for longer than 24 hours. Each of our experiments takes 100 runs. We report the mean of the measurement results. Unless otherwise indicated, in all our tests, the total population $N = 20,000$ nodes. The number of files (with different contents) varies from $1,000$ to $10,000$ and the average size of shared folders ranges from 5 to 50 files. We set the initial the percentage of abnormal files $h(0) = 1.5\%$, the initial infection $I_f(0)/N = 0$ and the initial immunity rate $i(0) = 15\%$. Among these immune nodes, $\alpha = 5 \sim 10\%$ of the entire population were bootstrapped as key nodes and each of them obtained the latest security updates from vendors.

Scheme Effectiveness We compared the time performance and the system evolution status of different approaches, using the same set of parameters (e.g., λ_a, λ_d, α and β). We also used the no-defense case as the base line. Our test results are shown in Fig 8. Fig 8(a) illustrates the change of immune population over time. Without any defenses, the system keeps a low immunity rate and has to rely on individuals' patch updates. The download-based approach and the search-based approach both significantly increase the immunized population. The former takes around 35.5 hours to achieve a 90% immunity rate while the latter takes around 62.5 hour due to its reactive nature. The download-based approach largely depends on the activity level of file downloads and the search-based approach is triggered by worm detections. Fig 8(b) shows the change of the infected population over time. Without any defenses, the worm spreads

in a relatively high speed and infects all the vulnerable hosts within 9.5 hours. Both our schemes effectively help the system reduce the infected population by internally pushing the security patch to disinfect those victims. A further comparison indicates that the search-based approach has a relatively slower disinfection speed; it takes 62.5 hours to reduce the overall infection rate to below 10%. However, it keeps a lower maximum infection rate ($I_f(max)/N$ = 37%). On the contrary, the download-based approach takes 45 hours to reach an infection rate below 5%, but it yields a higher maximum infection rate ($I_f(max)/N$ = 44%) in the system. Fig 8(c) illustrates the change of vulnerable population over time. Without any defenses, the vulnerable population quickly drops to zero (within 10 hours) as more and more nodes get infected during file downloads. Our schemes effectively slow down this process by either immunizing the vulnerable hosts or disinfecting the victims. We can see that after around 62.5 hours, there remain few vulnerable nodes and victims in the system and the immunity rate exceeds 90%.



(a) t_0 vs. (β, λ_d) (b) $I_f(max)$ vs. (β, λ_d)

Fig. 9. Impact from user behavior on the defense scheme (Gnutella 0.6)

User Impacts and Overhead We evaluated the impacts of system parameters and user behavior on the defense. Fig 9 illustrates the test result in Gnutella 0.6 system. Fig 9(a) shows that both schemes take less time (t_0) to achieve an 90% immunity rate as the speed of file download (λ_d) increases. For the download-based approach, a higher download speed results in a faster patching process; for the search-based approach, a higher download speed leads to more worm infections and this in turn speeds up the patching process. The figure also indicates that in the download-based approach, t_0 gets reduced as users become more willing to accept the patch (β increases). However, this is not distinct in the search-based scheme due to its reactive nature. When more users are patched, the defense also gets slowed down. Fig 9(b) shows that in the download-based scheme, the severity level of worm attacks ($I_f(max)$) quickly drops as β increases. When $\beta \geq 0.85$, the maximum infection rate in the system is below that of the search-based scheme.

Fig 10 illustrates the message overhead of the defense schemes. Using Microsoft XP as an example, the patches during SP2 are in binary delta compression format [6] and the mean patch size is 32.9 KBytes [12]. This patch and its vendor signature (typically around 300 Bytes) constitute the main part of the payload in an alert message. Thus,

the average length of a patch message is 33.2 KBytes. The figure shows that when β increases from 0.6 to 1.0, the message count of the download-based approach decreases until finally it reaches around $20,000 \times (90\% - 15\%) = 15,000$. We examined three cases for the search-based scheme: (1) *worst case* in which each key node simply delivers the patch to its targets. Patch messages could be duplicated and the message count is above 50, 000; (2) *average case* in which a key node does not deliver a patch to the same target and the message count is above 28,000; (3) *optimal case* when key nodes collaborate to avoid patch duplicates or each node indicates its current patch version in the QueryHit response. Hence, there are few patch duplicates and the message count approaches 15, 000 when $\beta \rightarrow 1$.

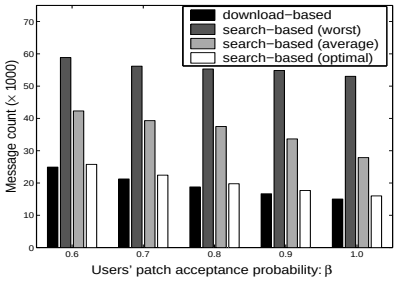


Fig. 10. Message overhead of the defense scheme (N = 20,000 nodes, $\Psi = 90\%$); the download-based approach has less overhead than the search-based one due to no alert duplicates.

7 Related Work

File-sharing worms have recently attracted much attention in research community. The initial studies mainly focused on understanding the threats and modeling the behavior of such imminent worms. Dimitriu et al. [11] demonstrated that worm spreads highly depend on user behavior, such as willingness to share files and quickness in removing infected files. Kumar et al. [17] developed a suite of fluid models that characterize pollution proliferation in file-sharing systems. Thommes et al. [23] derived deterministic epidemiological models for file-sharing viruses spreading in file-sharing systems.

The problem of throttling these worms have not been adequately addressed. Existing defenses mirror the strategy against Internet computer viruses. Generic automated patching tools (e.g., Microsoft Window Update, Symantec Update, McAfee VirusScan) are widely adopted to launch protection on P2P hosts. Vojnovic et al. [24] studied the effectiveness of automatic patching and quantified the speed of patch dissemination required for worm containment. Gkantsidis et al. [12] provided general guidelines on how to design a fast planet-scale patching system based on their studies on Window Update. They also suggested alternative patching strategies such as caching. Costa et al. [10] proposed Vigilante, an end-to-end approach in which hosts run instrumented software to detect worms and broadcast self-certifying alerts (SCA) upon worm detection. Zhou et al. [26] further applied Vigilante in P2P systems to contain fast-spreading topological worms. Our work differs from the above in that we provide internal patching mechanisms exclusively for file-sharing systems. Our focus is not on generating anti-worm

code on-the-fly to combat zero-day worms, but on studying good patching schemes which both save network bandwidth and avoid unnecessary host interruptions.

8 Conclusions and Future Work

File-sharing worms are becoming the most dominating and devastating security threats to P2P systems. Current defenses relying on individual recoveries or limiting file-sharing activities are not adequate. As a complement to the existing centralized patching mode, we proposed internal patching mechanism which conveniently leverages file-sharing infrastructure to disseminate security updates to participating peers in an automated and distributed way. We studied a download-based approach which exploits the file download process and a search-based approach which exploits the file search process for notifying P2P hosts of the worm attack and pushing the security patch to them. In spite of some remaining issues such as host diversity and user diversities, the free-rider problem [8] in patching, our study suggests some interesting directions for designing countermeasures against worms in distributed environments. We address remaining issues in our future work.

Acknowledgements This research was supported in part by the National Science Foundation CAREER-0643906. We thank the anonymous reviewers for their helpful comments.

References

1. The gnutella protocol specification. http://www.the-gdf.org.
2. http://en.wikipedia.org/wiki/comparison_of_file_sharing_applications.
3. http://www.facetime.com/securitylabs/imp2pthreats.aspx.
4. http://www.viruslist.com/en/viruslist.html?id=49790.
5. P2PSim: a simulator for peer-to-peer protocols. http://pdos.csail.mit.edu/p2psim.
6. Using binary delta compression technology to update windows operating systems. Microsoft online White Paper.
7. www.viruslist.com/en/virusesdescribed?chapter=153311928.
8. P. Biddle, P.England, M.Peinado, and B. Willman. The darknet and the future of content distribution. In *ACM Workshop on Digital Rights Management*, 2002.
9. D. Brumley, J. Newsome, D. Song, H. Wang, and S. Jha. Towards automatic generation of vulnerability-based signatures. In *Proceedings of the 2006 IEEE Symposium on Security and Privacy*, 2006.
10. M. Costa, J. Crowcroft, M. Castro, and A. Rowstron. Can we contain internet worms? In *Proc. of the 3rd Worshop on Hot Topics in Networks (HotNets-III)*, Nov. 2004.
11. D. Dumitriu, E. Knightly, A. Kuzmanovic, I. Stoica, and W. Zwaenepoel. Denial-of-service resilience in peer-to-peer file sharing systems. In *Sigmetrics'05*, 2005.
12. C. Gkantsidis, T. Karagiannis, P. Rodriguez, and M. Vojnovic. Planet scale software updates. In *Proc. of SIGCOMM'06*, 2006.
13. N. Good and A. Krekelberg. Usability and privacy: a study of kazaa p2p file-sharing, 2002. In *http://www.hpl.hp.com/shl/papers/kazaa/index.html*.
14. D. Hughes, G. Coulson, and J. Walkerdine. Free riding on gnutella revisited: the bell tolls. In *Proc. of IEEE Distributed Systems Online*, 2005.

15. S. Joseph. NeuroGrid: Semantically routing queries in peer–to–peer networks. In *International Workshop on Peer-to-Peer Computing*, 2002.
16. G. Kc, A. Keromytis, and V. Prevelakis. Countering code-injection attacks with instruction-set randomization. In *ACM CCS'03*, Oct. 2003.
17. R. kumar, D. Yao, A. Bagchi, K. Ross, and D. Rubenstein. Fluid modeling of pollution proliferation in p2p networks. In *Sigmetrics'06*, 2006.
18. N. Leibowitz, M. Ripeanu, and A. Wierzbicki. Deconstructing the kazaa network. In *Proc. of IEEE IWAPP'03*, 2003.
19. D. Mulz, F. Valeur, C. Kruegel, and G. Vigna. Anomalous system call detection. In *ACM TISSEC'06, 2006*.
20. J. Newstone and D. Song. Dynamic taint analysis: Automatic detection and generation of software exploit attacks. In *Proc. of the 12th Annual Network and Distributed System Security Symposium (NDSS'05)*, 2005.
21. S. Shin, J. Jung, and H. Balakrishnan. Malware prevalence in the kazaa file-sharing network. In *ACM Internet Measurement Conference'06*.
22. I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, M. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup protocol for internet applications. In *IEEE Trans. on Networking*, 2002.
23. R. Thommes and M. Coates. Epidemiological modeling of peer-to-peer viruses and pollution. In *Infocom'06*, 2006.
24. M. Vojnovic and A. Ganesh. On the race of worms, alerts and patches. In *ACM Workshop on WORM*, 2005.
25. K. Zetter. Kazza delivers more than tunes. In *The Wired Magazine*, 2004.
26. L. Zhou, L. Zhang, F. McSherry, N. Immorlica, M. Costa, and S. Chien. A first look at p2p worms: threats and denfenses. In *IPTPS*, 2005.

A Percentage of Abnormal Files

Proof. We refer to Table 1 for notation. Let $A(t)$ denote the number of abnormal files at time t, we have $h(t) = A(t)/F(t)$. The change rate of the total number of files $F(t)$ is

$$\frac{dF(t)}{dt} = \lambda_d N \quad i.e.,\ F(t) = F_0 + \lambda_d N \cdot t, \tag{12}$$

where F_0 is the initial number of files in the system. A newly added abnormal file could be caused either by a vulnerable host activating another abnormal file in the same folder, or by a node directly downloading an abnormal copy from others. Therefore, we have the change rate of the number of abnormal files

$$\frac{dA(t)}{dt} = \lambda_d \cdot \lambda_a \cdot V(t) \cdot h(t) \cdot (s(t) - 1) + N\lambda_d h(t). \tag{13}$$

Considering $s(t) - 1 \approx F(t)/N$ and $A(t) = F(t) \cdot h(t)$, we solve equation 13 and finally get

$$h(t) = h(0) \cdot e^{\frac{\lambda_d \cdot \lambda_a}{N} \int_0^t V(\tau) d\tau}, \tag{14}$$

where $h(0)$ is the initial percentage of abnormal files.

We may compute an approximation for $h(t)$. Assuming all the nodes have a similar size of $s(t)$ for their shared folders and the user parameter λ_a approaches 1, which means every client usually activates (opens) the file he has just downloaded, all the abnormal files should gradually be kept by those infected hosts in the system. Hence, we may derive $h(t) \approx \frac{I_f(t)}{N}$.