Implications of Peer-to-Peer Networks on Worm Attacks and Defenses

Jayanthkumar Kannan Karthik Lakshminarayanan
{*kjk,karthik*}*@cs.berkeley.edu*

CS294-4 Project, Fall 2003

Abstract

Recently, two trends have emerged in the field of peer-to-peer networks: widespread deployment of peer-to-peer systems for file sharing and development of distributed hash tables that provide efficient lookups. In this paper, we study how to harness the power of these technologies to further the state-of-the-art in both designing and defending against Internet worms. We quantify this advance from three different viewpoints. Firstly, peer-to-peer traffic characteristics differs from traditional Internet traffic in several aspects, and we quantitatively analyze the effect of these differences on worm propagation and control. Secondly, we show that a DHT is an ideal model for coordination among worms, and design a DHT-enabled worm that is an improvement over existing worm designs in a number of aspects, mainly stealth in propagation and speed of propagation. Our DHT-based worm designs can be used to implement a variety of policies aimed at circumventing existing schemes for worm propagation control. Our results also show that a coordinated worm can spread more than twice as fast as worms such as Slammer, while halving the number of unsuccessful probes. In this way, this paper attempts to "raise the bar" in worm design, and this is essential to the development of suitable defenses. Finally, we offer some preliminary insights on how a DHT can be used to be defend against worms.

1 Introduction and Motivation

A paper on worm design and defense requires no motivation thanks to several well-publicized attacks and their global impact. The current state of the art in worm design is captured in [1] which provides the blueprint for a worm that can infect the entire vulnerable population in "under 1 hour, in possibly about 15 minutes". This worm uses a combination of a pre-collected hit list of vulnerable machines and a technique to divide the probing load among the infected machines. Concurrently, there have been several proposals for worm defense. Two mechanisms that have been suggested include content filtering and address filtering. However, [2] is a negative result that shows that neither of these is likely to work at the time scales of the worm propagation. Another mechanism that uses a identifying characteristic of worm traffic is Virus Throttling [3]. However, this requires deployment at all end-hosts in order to be effective, and does not prevent the end-host itself from getting infected. Methods that are being explored recently mainly rely on unusual traffic patterns during a worm attack. For example, Cossack [11] aims for cooperative detection, while Netbait [7] provides mechanisms for sharing information. It is currently unclear whether such schemes can control worm propagation. To summarize then, the fastest known worms can spread much faster that our best known defenses can counter them.

In this work, we attempt to show the influence of a new Internet technology, peer-to-peer networks, on this state of affairs. This technology has two aspects pertinent to this problem:

- *Widespread deployment of P2P networks*:

  File sharing networks like Kazaa have gained popularity, and have substantial user populations [5]. Traffic patterns in such networks differ substantially from those in traditional Internet applications in that a participating end-host is a client as well as a server. In acting as a

server, it is typically contacted by several machines over the Internet without any solicitation. Thus, a participant end-host has a far richer and diverse incoming traffic. This has an impact on worm propagation because passive generation of hitlists and passive propagation (infected machine attempts to infected only hosts that contact it) can be potentially much faster. This effect is well-noted in literature, in particular in [1], and we will attempt to quantify this effect. Furthermore, we also quantify how active hitlist generation can be done by crawling a deployed P2P network such as Gnutella.

- *DHT design*:

  Peer-to-peer networks can also provide an efficient implementation of the DHT abstraction, and this allows for efficient distributed coordination. We will show that worms that use DHTs for coordination during propagation can potentially be much faster and be harder to detect (this possibility was first suggested, though not explored, in [6]). The other side of the equation, worm defense, has not been well explored as well (preliminary approaches has been proposed in [7], [11], [12]). We will also offer some insights in this direction.

At this point, we also address the following question: given that known worm designs can spread very fast and no good defenses are known against worm designs, is it necessary to design better worms? For example, it has been stated in [4] that coordinated worms are not a prerequisite for fast propagation. We believe that the answer is yes, because of the following reasons. Firstly, propagation time has been the only metric that has been used so far. A coordinated worm would improve over existing designs in reducing the susceptibility to intrusion detection systems, thus possibly evading detection techniques that may be devised for uncoordinated worms. Secondly, the propagation period calculated in [1] makes several assumptions that are not true in practice, that our designs attempt to accommodate for. Finally, it is necessary to understand attacks in order to build designs that work against them.

The rest of the report is organized as follows. In Section 2, we discuss our assumptions of the way the worm operates. Section 3 enumerates techniques of harnessing the host population and the traffic patterns of deployed P2P networks and evaluates them. In Section 4, we discuss possible ways in which worms would desire coordination. We also outline how Kelips, an existing DHT can be modified to achieve this. In Section 5, we outline specific schemes that make use of the ability to coordinate, and evaluate these schemes. In Section 6, we briefly discuss how DHTs can be used for worm defenses. We discuss related work in Section 7, possible future work in Section 8, and conclude in Section 9.

2 Worm Model

We will first discuss the simple model of a worm that we will assume henceforth. In this model, the worm exploits a bug in some service running at end-hosts, then proceeds to probe the address space in some fashion in order to discover other vulnerable hosts. In the most naive form of probing, the infected host probes addresses randomly. Smarter worms [1] might choose to attempt localized infection (infect other hosts in its network) or work off hitlists embedded in the code or use sequence generators. The random probing worm is known to have a sigmoid growth curve, exponential in the beginning and then beginning to slow down, when the remaining vulnerable hosts are hard to discover.

We also assume that the probes that the worm performs at the infected host is only limited by the outgoing bandwidth at the host. UDP worms such as Slammer are limited by the outgoing bandwidth alone. Traditionally, TCP based worms use the kernel TCP implementations which places a limit on the number of connections that a host can open. Since this is not a fundamental limitation, our assumption would hold for a worm in the context of TCP-based worms too.

3 Discovering Vulnerable Hosts

We discuss and quantify in this section how the diverse traffic in P2P networks allow for faster worm propagation.

3.1 Hitlist Generation

Passive hitlist generation before the attack can considerably speed up worm propagation time because the worm attack takes time to get up to speed. If the worm exploits a bug in the peer to peer application itself, then a hitlist can be obtained by deploying a number of clients that log the IP address of every machine that contacts them. These clients can also crawl the network actively to discover as many hosts as possible in the network. Active probing has the advantage of being faster and of yielding a more complete topology information, as compared to passive probing. On the other hand, if the worm exploits a bug in some other application, the above hitlist still offers a useful starting point: tools like NMap [8] can be used to probe the hosts to discover whether they are susceptible to the worm.

3.2 Worm Propagation

Assuming no hitlist is generated beforehand, and the bug is in the P2P application itself, worm propagation in a P2P network can potentially be very rapid, due to the frequent communication among peers. Moreover, such networks are known to follow the power-law model [9], and such graph are known to have a tightly knit core that connects the other nodes. Techniques used to optimize lookup distances, such as a biased random walk [9], can also be used by the worm to optimize its infection strategy. Such a walk biases itself towards this richly connected core, ensuring that a large part of the network is discovered quickly.

3.3 Evaluation

We deployed Gnutella clients on 25 Planetlab nodes that performed active crawling of IP addresses, in order to assess the effectiveness of this scheme. The implementation was a modification performed to the Limewire implementation to perform appropriate packet logging. The crawlers were run for about 2 hours and the crawl logs were post-processed.

3.3.1 Hitlist Generation

Figure 1 shows the number of IP addresses crawled as a function of time. The different lines on the graph correspond to varying the number of crawlers used to generate the IP addresses. As time increases, the rate

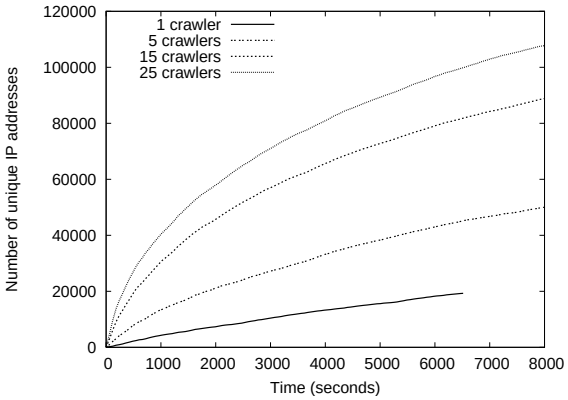


Figure 1: Number of IP addresses crawled as a function of time. Different lines correspond to different total number of crawlers.

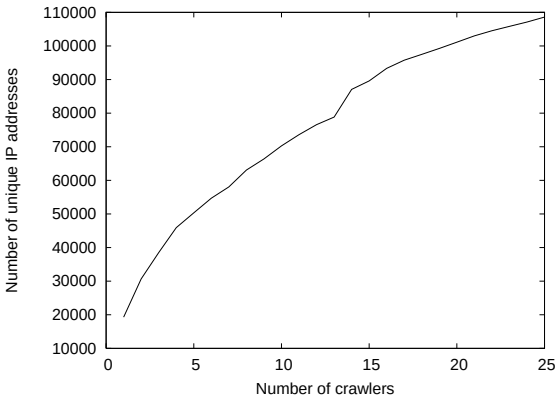


Figure 2: Number of IP addresses crawled as a function of total number of crawlers used.

at which new IP addresses are seen decreases rapidly. After a certain point, the number of new IP addresses seen increases only linearly at a small rate. This can be attributed to the churn that the Gnutella system typically sees combined with the aggressive crawling technique. The other phenomenon evident from the graph is the diminishing returns in using additional nodes for crawling. In particular, with 15 crawlers, we are able to gather $90,000$ IP addresses, and with 10 more crawlers, we add only $20,000$ more new IP addresses. This effect of diminishing returns is further demonstrated in Figure 2 which plots the number of IP addresses seen as a function of number of crawlers. In the plots, a random subset of crawlers is chosen for each point corresponding to a certain number of crawlers. We note that changing the set of crawlers, in spite of affecting the actual numbers, had

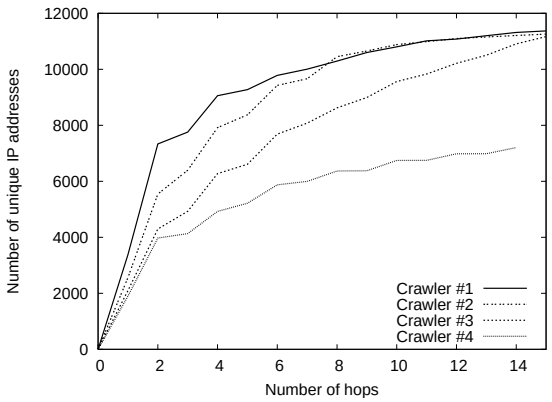


Figure 3: Cumulative number of nodes as a function of depth from a super-peer in Gnutella.

little effect on the overall trends.

We also study the limitations of hitlist generation in a peer-to-network by studying how fast such a list goes stale given the typical host availabilities. On some initial studies, we found that about 57% of the list is stale after a period of about 1 week.

3.3.2 Worm Propagation

Using the edges that were seen during the crawl of the Gnutella network, we regenerated the Gnutella graph structure. Since many edges were not reported because the clients did not send responses, the graph we obtained is a proper subgraph of the actual Gnutella network. In fact, in some cases, we could successfully construct the edges for only about half the nodes that the Gnutella crawler encountered.

Using this graph structures generated, we studied how an infection would spread along a Gnutella graph. Figure 3 plots the cumulative number of nodes at a particular depth starting from the root node. Here, the root node is the first node that is infected by the worm. The different lines in the graph correspond to multiple starting points for the worm, ie. a set of machines are infected and then start propagating it in the Gnutella network.

As Figure 3 indicates, most of the nodes that are visible from any vantage point are only a few hops away. In fact, for crawler 1 (in figure), about 80% of the $11,593$ hosts whose edges we were able to reconstruct were found within 4 hops away. In fact, by having multiple infected hosts, similar phenomenon persists indicating benefit in such a technique.

4 Coordinated Worm Attacks

In this section, we outline several ways in which a worm might wish to coordinate during its propagation, and then discuss why a DHT is the ideal solution to provide that coordination.

4.1 Desirable Coordination

In the traditional worm design, an infected machine begins to probe independently of other infected machines. Schemes in [1] attempt to provide some coordination by sharing information when a new machine is infected. However, coordination during propagation allows for more powerful designs. The following factors which deter a traditional worm can be solved using coordination:

- *Avoiding detection*: As worm defense technology improves, it might become necessary for worms to follow certain policies during propagation to avoid detection. Following is an simple example of such a policy: if a machine inside an access network is infected, then no other machines should probe the network so as to minimize worm traffic seen by vigilant firewalls that interpose themselves in the path from external hosts. Another example of such a policy could be that a infected machine avoids probing the same access network more than x times in order to counter a firewall that uses a threshold of x attempts from the same source to detect an attack.

- *Uneven IP address space allocation*: The occupation of the 32-bit IP address space is far from even as has been pointed out in [13]. This implies that random probing is not the ideal infection strategy. One would like to direct probes towards likely targets, by allowing nodes to share information about heavily occupied IP prefixes etc.

- *Locality-based infection*: It is clearly desirable for machines to attempt to infect a close-by host rather than a distant host. Given that round trip times can vary widely in the Internet (for one such study, see [14]), it is conceivable that choosing close-by targets for infection can speed up infection for TCP-based worms. Also, since delay is correlated to number of ASes in the

path, this ensures the probes cross lesser number of ASes.

- *Probing rate control*: It has been pointed out in [15] that the CodeRed worm could have spread faster if it had avoided congesting links due to its own probe traffic. Coordination can be used to achieve such rate control.

- *Avoid needless probing*: Infected hosts can share information in order to avoid attempt to re-infect the same end-host.

At a high-level, coordination can achieve two goals. The obvious one is that worm propagation can be speeded up. A more subtle and important point is that all of these factors aid evasion of detection mechanisms. For example, if a worm avoids probing an unoccupied IP prefix, it can avoid detection by schemes such as backscatter analysis [16]. A worm that chooses close-by targets is likely to encounter fewer core routers in its path to the target, thus improving evasion against defense mechanisms deployed by ISPs (say, content filtering). There are other situations where such coordination is useful. It is especially useful in getting the worm off the ground, i.e. if the initial startup phase has been identified as the slow phase, we can speed it up using coordination. It is also useful for worms targeting a small population of machines running a rare installation (which for some reason is interesting to the attacker). In such a case, the start-up phase will be more of a limiting factor, and our schemes will considerably speed it up.

4.2 Coordination Using a DHT

The coordination of infected machines is a challenging problem because the set of infected machines increases rapidly during the initial stages of the propagation. Based on the desirable goals of the previous section, the scheme we use for coordination should provide the following services:

- Fast routing table maintenance to deal with the rapid joining of new nodes. Information should not be lost pathologically in the face of nodes leaving.

- Load balancing properties to help divide probing load equally across all infected machines.

- Ability to answer such questions such as "Is any node from IP address prefix x.y.w.z/16 infected?" efficiently.

- Ability to answer nearest neighbor nearest queries. Accuracy is a secondary requirement.

It is clear that DHTs can provide the above services a worm would require to coordinate, though it is questionable whether DHTs can handle the high churn rates of such an environment. We chose a churn-resistant DHT, Kelips [17] to explore this question. Also, note that accuracy of lookups is not of paramount importance – if consistency is delayed by a few seconds, it would not affect the coordination phase.

Before delving into using Kelips to achieve coordination, we first discuss how a DHT can be made to work in our environment with rapid joins. Firstly, our overlay mechanism is organized as a two-level hierarchy, utilizing the well-known heterogeneity of peer-to-peer hosts ([18]). Each worm joins the super-peer level only if it has sufficient bandwidth to do. Otherwise, it joins a super-peer which shields it from DHT control traffic. We expect that each worm will find a super-peer nearby which maintains worm state on its behalf.

4.3 Kelips-Based Coordination

In this work, we use Kelips, a hybrid DHT for maintaining information across all infected hosts.

4.3.1 Introduction

We first give a brief introduction to Kelips. Kelips is a hybrid of an unstructured network and an DHT designed to handle high churn. Kelips nodes are organized into a set of k affinity groups (based on a hash function). Keys are hashed using the same hash function to determine which affinity group they should be stored in. Note however that a key can be stored at *any* node in its affinity group. Each node maintains the following state:

- Affinity Group Pointers: Pointers to *every* node in its group.

- Foreign Contacts: A set of c pointers to nodes in every other group.

- Home Pointers: For all keys that map into its group, a pointer to the node in its group that is responsible for maintaining the value.
- Data: (Key,Value) pairs for data stored on itself.

This state is synchronized using periodic gossip with nodes in its group and other groups. During each gossip round, only a constant amount of data is exchanged, which means that complete consistency within a group is achieved in $O(\log n)$ rounds. The insertion of a (key,value) pair is accomplished by first routing to the group, then choosing a random node within the group as the home for that item. Lookups are done in a similar fashion. In our application, data for a particular prefix is inserted using the prefix as the key.

4.3.2 Modifications to Kelips

We modified Kelips in a few respects in order to adapt it for our purpose, which we outline in this section. Firstly, the parameter k in Kelips is chosen to be $O(\sqrt{n})$, so as to optimize state maintenance. In our case however, because of our admission control procedure, the Kelips node are well-provisioned, thus we chose k to improve consistency at high churn. A high value of k means there are lesser nodes in each affinity group, which implies that data consistency is reached sooner. However, this also means that the fault-tolerance within a group is poorer. In our case, the population of Kelips nodes is around a few thousand (in Peer-to-peer networks like Gnutella, about 5000 super-nodes handle the queries of a hundred thousand nodes, thus this value is in the right ballpark). We chose $k = 40$ to ensure quick consistency (we found in our simulations that with these parameters, consistency was reached in less than a minute). Also, we chose to exchange *all* inconsistent state during gossip, which is reasonable given the small group size.

Secondly, during high churn, it is possible that two simultaneous inserts can lead to more than one node being appointed as the homenode for the same key. This could happen, for example, when these lookups arrive at two different nodes in the same group, which chose different homenodes and start propagating inconsistent homepointers. This inconsistency is discovered when two nodes in the same group attempt to synchronize. We chose an application-specific form of conflict resolution: the data stored in the two home nodes is combined, and one of them (the one with the higher address) is chosen as the homenode. The important point here is that the data inserted during every *put* operation is incorporated into the eventual homenode, even during churn. This is possible since the information that we store in Kelips is an aggregate of all updates.

Thirdly, we modified the matching algorithm for key matching to be a longest-prefix match. This means that given a lookup or a insert for a IP address prefix, a Kelips node performs a longest prefix match in its homepointers table to find the homenode. This allows some flexibility in relocating certain keys, which we will use in our locality-aware infection schemes.

4.3.3 Introduction Service

We assume that the attacker acquires control over a small number of well-connected zombies. These zombies provide the introduction service into the network. An infected machine first contacts these zombies in order to obtain a list of other infected machines that it can add as neighbors.

5 Schemes

In this section, we discuss how each of the goals we outline in the previous section can be attained using a DHT.

5.1 Policies to Evade Detection

In this section, we discuss how DHTs can be used to implement specific policies that can help worms evade existing schemes. Note that we do not attempt to hide the worm propagation itself, we only seek it to make it more difficult for existing schemes to stop worm propagation.

Honeypots ([10]) are monitored machines that are deliberately exposed to external attack. As has been pointed out in [2], address filtering can be used to throttle worm propagation. Honeypots provide a way of harvesting IP addresses of infected machines, so that any probes from those addresses are dropped. This can be solved by using address spoofing, which

however does not work for worms that propagate using TCP. One policy that worms might use to circumvent such schemes is to not let more than x machines probe a given prefix. This ensures that the honeypot sees only a small subset of the infected machines. Such a scheme can be implemented as follows: the value stored for each prefix is the number of currently infected hosts probing that prefix. Before probing an IP prefix, each host issues a lookup into the DHT for that prefix. The homenode for that prefix grants access until the number of probers exceeds a threshold value. This scheme can be made resilient to failure by setting conservative thresholds. This method can also be used to circumvent Cossack, FloodWatch, and other propagation control schemes outlined in [12].

5.2 Uneven IP Address Space Allocation

Broadly, the scheme attempts to maintain a probability distribution over IP address prefixes to reflect the fraction of the vulnerable hosts that are present in each prefix. Let an infected host scan a set of x_i IP addresses in a prefix, of which x_i' are vulnerable and x_i'' are then newly infected ($x_i'' - x_i'$) were already infected). It sends back (x_i, x_i', x_i'') back to the home node of the prefix which aggregates the information. For each prefix, the home node maintains the fraction of vulnerable hosts and the extent of the prefix that is scanned.

Thus, the home node for a prefix has knowledge of total probes made to a prefix ($P = \Sigma x_i$), total number of vulnerable hosts that were probed including multiple probes ($V = \Sigma x_i'$), and the total number of unique infected hosts ($I = \Sigma x_i''$). The expected number of vulnerable hosts is then $f = V/P$, and the expected probability that a new probe will hit a vulnerable host, $p = (f.X - I)/X$, where X is the size of the prefix.

When a host is infected, it chooses a new prefix to probe based on the following algorithm. It generates K IP addresses randomly (K is a simulation parameter, set to 10 in our simulations) and probes the DHT about the occupancy information of the prefixes. In return, it receives the "p" value for each of the prefixes. If the prefix is probed for the first time (i.e., the Kelips home node does not have any information about the prefix), p for that prefix is initialized to the global average value of p over all prefixes probed thus far. The global average is performed by simply piggybacking average occupancy information along with Kelips' gossip protocol. Upon receiving, p for all the K prefixes, the infected host generates random IP addresses chosen from the K prefixes in proportion to the p values. Thus, a prefix that is estimated to have more vulnerable hosts is probed more. The host probes the set of K prefixes for a small period T. After this phase, it updates the information back into the DHT, and then chooses a new set of K prefixes and continues the same operation.

We made this basic technique robust to statistical variations during the probing process. During the initial phase of the worm propagation, few vulnerable hosts will be discovered in sparsely populated prefixes, and such prefixes will have a lower chance of being probed further. Also, the estimate of the fraction of vulnerable hosts using the formula $f = V/P$ can be very inaccurate, especially for sparse prefixes. To avoid this problem, we added confidence estimates for the vulnerability fraction, and used the high end estimate as the metric. This high end estimate is biased towards the sparsely populated prefixes. The estimate is calculated using a 95% confidence estimate interval.

5.2.1 Extensions

This scheme can also be extended to a multi-vector worm that can maintain different address spaces for its different payloads, since each prefix has a different vulnerability index for its payloads.

As such, this scheme serves another purpose as well: it directs the search to vulnerable networks, not simply dense networks. In most cases, the worm targets a bug in a set of specific versions of an application. In practice, if a host is running a specific version, it is not unlikely that other hosts within its network (and hence with the same prefix) are running the same version. Thus, this scheme exploits the locality factor as well.

5.3 Locality-Aware Infection

The goal in this scheme is to ensure that each infected host probes nearby prefixes. This is advantageous because this ensures that each probe traverses fewer ASes (in general, delay is correlated to the number

of ASes traversed). This in turn, decreases the probability of detection or dropping, when ASes deploy defenses within their network.

To implement such a scheme, we modified Kelips to make it locality-aware for our purposes. When choosing a home pointer for a particular prefix, the Kelips node chooses m random nodes within its affinity group, and chooses the node closest to the prefix as the homenode for that prefix. Each infected host, that is not in the Kelips DHT (due to admission control), also tries to adaptively find a closer parent. This is accomplished by having each host query its parent periodically for a better parent within the *same* group. The reason we restrict this search to nodes within the same group is that the foreign contacts of a node are chosen so as to minimize latency. This means that using the foreign contacts leads to a biased choice.

Under these above modifications, we have the following properties: the homenode for a prefix is close to that prefix and each infected host is close to its parent. Now, we make the following assumption: if node A is close to node B, and node B to C, we assume A is also close to C. This is not universally true in the Internet, but is a reasonable approximation. Along with this assumption, it is easy to see how an infected host can choose a prefix close to it. It uses the prefixes stored at its parent and those at the foreign contacts of its parent (which are close to its parents). The host now samples from these prefixes and begins probing that prefix. This scheme can also be combined with the uneven IP address scheme, by letting the host sample from these prefixes according to the metric for each of those prefixes.

5.3.1 Extensions

In our simulations, we have assumed that all prefixes have the same length. This assumption is not true in practice, since the Internet has a wide range of address prefixes. Thus, all IP addresses belonging to the same prefix need not have similar latencies with respect to a given host. This issue can be addressed by incorporating longest prefix matching.

If a homenode for a given prefix finds that there are distinct clusters within that prefix of widely different latencies (for example, using the GeoCluster technique in [19]), then it can choose to split that prefix into sub-prefixes. Each sub-prefix can then be migrated to other suitable home nodes using the earlier probing technique. Other nodes in the group will now use longest prefix matching on their homepointer tables in order to find the homenode for a given prefix.

5.4 Simulations

We used a discrete-event simulator to simulate the above schemes and evaluate their effectiveness. Simulating the wide-area Internet is a hard problem due to insufficient data about delay, bandwidth etc, and to do so while also simulating thousands of end-hosts is computationally expensive (for example, see [20] for a discussion of these issues). We did not simulate any background traffic for scalability; we only simulated vulnerable hosts. We also assume that route changes do not occur during worm propagation since modeling such changes is a hard problem in its own right. We chose a worm payload size of $10K$ bytes in our simulations. We simulated $25,000$ vulnerable hosts living in $2,000$ /20 prefixes (non-vulnerable or non-existent hosts are indistinguishable for us). This prefix length was chosen so that the size of the prefix was large enough to allow information sharing among multiple users. Hosts in a given prefix are assumed to be connected to the the AS-level graph through an access router with an access link bandwidth of 20 Mbps. The access bandwidth of the hosts were set as follows: 20% of the hosts had bandwidth assigned uniformly from the range $10 - 100$ Kbps, 40% from $1,000 - 4,000$ Kbps, and the rest from $4,000 - 100,000$ Kbps. Last-hop delays were ignored. The admission control bandwidth for Kelips was set at 5 Mbps. Each node gossiped every 1 second with a randomly chosen node from its group and with a randomly chosen node from 5 other groups. Each node chooses an IP prefix every 5 seconds using lookups into the DHT, and then reports statistics for that prefix at the end of that time period. Also note that since our simulated system is an order of magnitude smaller than the Internet, our results should only be interpreted as comparisons between our schemes and a strawman scheme (random probing) in an Internet-like distribution, and not as any indication towards actual performance in the Internet.

In our discrete simulator, at the start of each time slot, each infected host is allocated a fixed number of probes, that is calculated based on its access band-

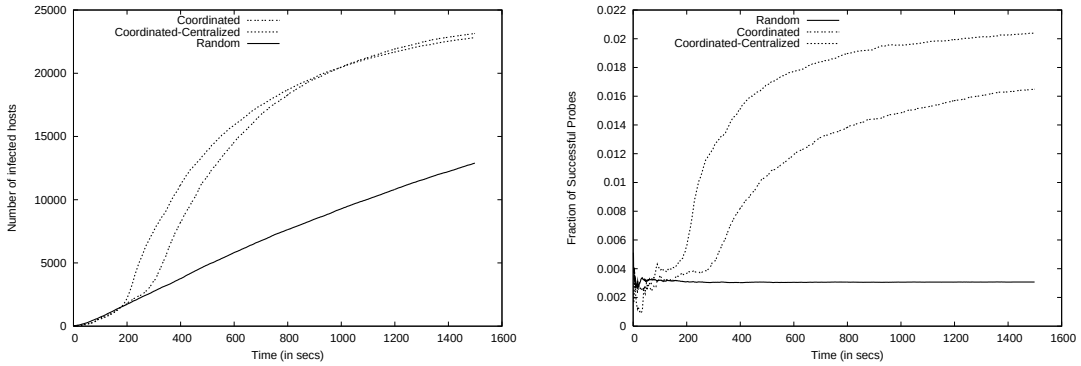


Figure 4: Impact of uneven IP address space allocation: (a) Rate of infection, (b) Fraction of probes that failed

width and the current number of infected hosts in its network (we model congestion due to worm probing only in the access link). The host now proceeds to send these probes to chosen IP address (either random or based on our schemes). Since we only simulate bandwidth-limited worms, TCP timeouts etc are not an issue.

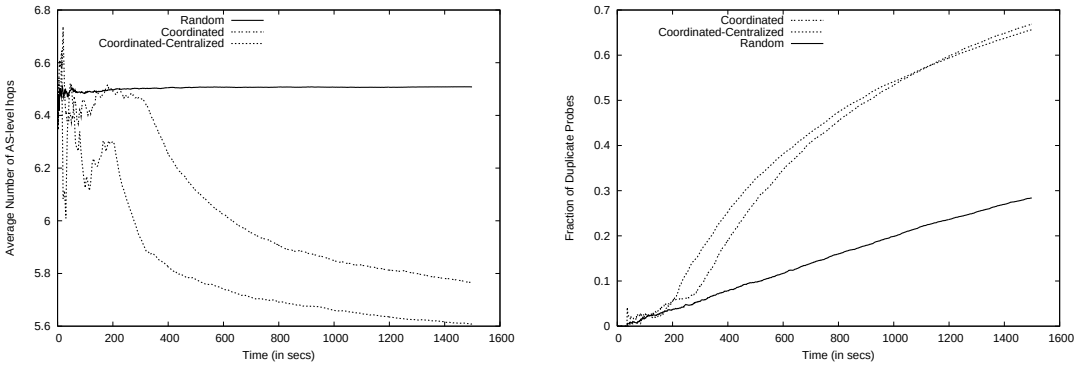
5.4.1 Kelips Consistency and Overhead

In this section, we attempt to evaluate how well our Kelips-based scheme scales to the Internet. In our simulations, we used 40 Kelips groups ($k = 40$) and found that Kelips reached consistency in the order of seconds, for 4000 nodes in the DHT. This is reasonable because the number of nodes in an affinity group is on the average 100. With a gossip period of 1 sec, consistency is reached in $O(\log(n))$ time. The overhead for gossiping is also manageable for the high bandwidth nodes. This overhead consists of two main parts: synchronization with nodes in the same group and with nodes in 5 other groups. The first term counts the overhead of exchanging lists of nodes in the same group (for 100 nodes, this is about 500 bytes) and homepointer information (about $100,000$ thousand prefixes corresponds to about $25,000$ prefixes per group, which is about $200K$ bytes of data). Thus, since gossip occurs once per sec, this requires a bandwidth of 1.6 Mbps. The second term is minimal overhead since it involves contacting 5 foreign contacts and sampling from their groups. Thus, the maintenance overhead is about 1.6 Mbps.

The lookup and insert overhead is also simple to calculate. Assuming an infected population of $100,000$ nodes which perform a lookup or insert every 5 seconds, and observing that each lookup involves contacting a maximum of 3 nodes (the parent, his foreign contact, the homenode), this overhead is about 15 operations per second per node, which is quite reasonable.

5.4.2 Evaluating Coordinated Worms

In these simulations, each node looked up 10 prefixes in the DHT before choosing one to infect. The $2,000$ prefixes were populated in a exponential fashion (with the highest one being populated by 15% vulnerable hosts). We compare the DHT-based coordinated worm with two baseline cases – (a) random worm (b) coordinated centralized worm, which represents the case where coordination is achieved through a central entity. The first baseline (random) provides insight into how much better our design performs, and the second (centralized coordinated) tells us how much worse we perform by using a DHT for sharing information. For locality-based simulations, it was necessary to simulate a Internet-like delay distribution between ASes. For this purpose, we obtained the AS-level graph was obtained from [21] and shortest paths and distances were computed in compliance with the inferred policies. Then, the distance matrix was sampled to obtain the required distribution. All prefixes were equally populated in this simulation. When choosing a prefix for infection, each node picks up 10 prefixes from its parents and its parent's foreign contacts. When choosing the homenode for a prefix, the best of 10 random choices is chosen as the home.



(a) (b)

Figure 5: Impact of uneven IP address space allocation: (a) Number of AS hops traversed on an average (extent of network utilization), (b) Fraction of addresses probed that were already infected

Figure 4(a) plots the rate of infection of the worm in terms of the number of infected hosts as a function of time. The graph clearly shows that the coordinated worm is able to infect the entire population in about half the time as the random worm. What is more interesting is the fact that the coordinated worm performs much better in the initial startup phase where it performs more than thrice as well as the random worm. Furthermore, using a DHT to coordinate information performs only marginally worse than the case in which coordination information is maintained at a central point. This is because, as discussed in Section 5.4.1, Kelips achieves consistency in order of seconds and the staleness of the information used for sampling is not very high.

Figure 4(b) shows the fraction of probes that succeeded in hitting a vulnerable host. Due to the very low fraction of vulnerable hosts that we used in our simulations ($f = 0.003(0.3\%)$), the number of successful probes is very small. We improve the fraction of successful probes by a factor of upto 5. Note that the fluctuations during the initial phase of worm propagation are due to inaccuracy of estimates caused by the small number of samples obtained.

Figure 5(a) suggests that the number of AS hops traversed by the coordinated worm is about 10% lower than the random worm. This represents a marginal improvement in the utilization of network resources. We believe that incorporating locality aware probing mechanisms would improve this significantly.

Figure 5(b) plots the fraction of probes that hit hosts that are already infected. Here, the coordinated worm hits an already infected hosts far more than the random worm. There are two reasons for the trends seen: (i) random worm infects far fewer hosts in the same time, and hence has lot more hosts left to infect, (ii) coordinated worm probes "better" areas and hits infected hosts rather than non-existent addresses. In the future, we plan to explore how to share information about what hosts are probed using techniques such as Bloom filters.

6 Using DHTs for Worm Defense

We now discuss some preliminary ideas for using DHTs for worm defense. We believe that the ideal point in the network to fortify against worm attacks are firewalls, the traditional point where most network intrusion detection systems are placed. Deployment at firewalls is easier to achieve than deployment at millions of end-hosts, and also provide a way to implement organization-wide policies. In our model, there are a number of cooperating firewalls that exchange information in order to detect ongoing worm attacks and alert one another other about such attacks. We believe DHTs can aid in worm defense in at least the three following ways.

Two recently proposed schemes for DDoS attack detection, Cossack and [12], rely on each participating firewall to inform all other firewalls of any events by multicast etc. Such state maintenance might be too expensive for every router to maintain. DHTs offer an efficient way to maintain such state in a balanced and robust fashion. Robustness is easily provided on top

of the DHT abstraction using replication. There are some additional security advantages as well: because structured DHTs impose constraints on the keyspace that a node is responsible for, it is not easy for a node to corrupt selective data.

Secondly, a more subtle point worth noting is that, in the likely case of partial deployment, DHTs offer an additional advantage. The state maintained by firewalls typically includes models of outgoing or incoming traffic at itself. In the partial deployment scenario, each firewall has to maintain state on behalf of non-participating firewalls in some indirect fashion. This state could be a model of outgoing traffic from that firewall for example. As another example, the state could be a model of incoming traffic, though maintaining such state would require some form of feedback by the core as well. Multiple vantage points offer more information and exchange of observed information will improve observation accuracy. DHTs provide a way of aggregating information about non-participating firewalls in a simple fashion: information about each non-participant firewalls is maintained by one of the participants. Information about such firewalls can be observed at any vantage point, and can be made available to others by maintaining it using a DHT.

The third observation here is that we would like this exchange of information to proceed during a worm attack, indeed, it is during the attack that such exchange of information (such as signatures of worms etc) might be crucial. During worm attacks, it has been noted in several studies (for example, see [22]) that Internet routing can suffer due to link congestion. These firewalls can use a DHT as an efficient routing mechanism that works in the face of Internet failure. Even if a firewall is unreachable directly, multi-hop routes by relaying through other firewalls might be available, and the ability of a DHT to provide redundant routes has been pointed out in [23]. Thus, DHTs can be used for connectivity when the Internet routing itself fails.

7 Related Work

In the last few years, there has a rapid rise in the number of worms in the Internet (such as Nimda, Code Red, Slammer). Most of these worms are very simple in their probing mechanism – once a host is infected, it scans random IP addresses for vulnerability and tries to infect it. Multi-vector worms (such as Nimda) look for multiple vulnerabilities on an end-host for break-in. [1] presents many techniques that can be used in worm design, and shows that it might be possible to infect the entire vulnerable population in "under 1 hour, in possibly about 15 minutes". The techniques suggested include a pre-collected hitlist of vulnerable machines and a technique to divide the probing load among the infected machines. However, none of the techniques suggested in literature attempt to achieve coordination among worms during the propagation phase. They do not explicitly try to circumvent the policies of intrusion detection systems. In this paper, the ability to share information efficiently using DHTs makes it possible to achieve these goals.

The best defenses that are known today have been explored in [2], COSSACK [11], Netbait [7], [12]. We have shown that carefully designed worms can circumvent such detection techniques. Again in designing worm defenses, the idea of sharing information efficiently using a DHT has not been explored in literature.

DHTs have been originally proposed as an alternative to Gnutella, Napster for efficiently sharing files across the wide-area. But since then, they have been used for a variety of purposes [1]. The original designs (such as Chord, CAN, Pastry, Tapestry) have been improved upon to provide better resilience in the presence of churn (such as Kelips, Bamboo). We exploit this ability to efficiently share information in the presence of churn. Such an overlay system not only allows for efficient data sharing, but also provides redundant routes (as suggested in [23].

8 Future work

Evaluation of our schemes under a more comprehensive model of the wide-area Internet is necessary in order to understand how they would work in realistic scenarios. This is a hard problem since several aspects of the wide-area Internet have not been characterized so far.

[1] As an aside, it is interesting to note that the original goal still remains elusive!

In our schemes, we have not considered the effect of patching machines during the infection. Patching can lead to nodes leaving the DHT and requires replication for robustness. This can conceivably be added into our schemes with minor modifications.

We have barely scratched the surface in attempting to use DHTs for defense against worms. We believe that it offers a lot of promise in providing a mechanism for defending against worm propagation.

9 Conclusion

The new emerging wave of P2P systems has led to tremendous amount of research in many areas in the field of file-sharing, distributed filesystems, networking and so on. But it also carries along with it the ability to perform malicious activities efficiently. In this paper, we have shown how P2P systems can be used to design better coordinated worms. We have used the power of deployed P2P systems for gathering IP address hit-lists and propagation within the system, and efficient DHTs for coordinating information across infected nodes. The primary challenge in using DHTs for coordination is achieving consistency in the face of a large arrival rate of nodes into the system. We show that, with minor modifications, Kelips can achieve consistency in maintaining data within a few seconds. This DHT abstraction allows for the design of worms that implement policies to circumvent existing schemes for throttling worm propagation. It also allows for improving other aspects of worm propagation, by exploiting the distribution of the IP address space and the delay distribution of the Internet. Our simulations support our thesis that such coordination can help design more stealthy worms that also spread more rapidly than current day Internet worms.

Understanding the potential power of Internet worms is of paramount importance, especially with emerging technologies that make sharing information easy and efficient. We believe that this possibility that we have explored here would guide the research in designing better and smarter defenses for protecting hosts against Internet worms.

10 Acknowledgments

We thank Boon Thau Loo for providing the implementation of Gnutella crawler, and helping us with the same. We also thank Lakshminarayanan Subramanian for discussions regarding using DHTs for worm defense. We are also grateful to Indranil Gupta and Linga Prakash for helping with parameter choice for Kelips.

References

[1] Stuart Staniford, Vern Paxson and Nicholas Weaver, “How to 0wn the Internet in Your Spare Time”, Proc. USENIX Security Symposium 2002.

[2] David Moore, Colleen Shannon, Geoffrey Voelker and Stefan Savage’, “Internet Quarantine: Requirements for Containing Self-Propagating Code”, Proceedings of the 2003 IEEE Infocom Conference, San Francisco, CA, April 2003.

[3] Matthew M Williamson, Jamie Twycross, Jonathan Griffin, Andy Norman, “Virus Throttling”,HPL-2003-69, Technical Report, HP Labs.

[4] Nicholas Weaver, “Warhol Worms: The Potential for Very Fast Internet Plagues”, http://www.cs.berkeley.edu/ nweaver/warhol.html.

[5] Stefan Saroiu, P. Krishna Gummadi, and Steven D. Gribble, “A Measurement Study of Peer-to-Peer File Sharing Systems”, Proceedings of Multimedia Computing and Networking, January 2002 (MMCN’02), San Jose, CA, USA.

[6] Brandon Wiley, “Curious Yellow: The First Coordinated Worm Design”, http://blanu.net/curious_yellow.html.

[7] Brent N. Chun, Jason Lee, and Hakim Weatherspoon, “Netbait: a Distributed Worm Detection Service”, Unpublished manuscript, February 2003.

[8] Network Mapper, http://www.insecure.org/nmap/.

[9] Lada A. Adamic, Amit R. Puniyani, Rajan M. Lukose, and Bernardo A. Huberman, “Search in Power-Law Network”, Appears in Phys. Rev. E, 64 46135 (2001).

[10] Christian Kreibich, Jon Crowcroft, “Honeycomb - Creating Intrusion Detection Signatures Using Honeypots”, Proc. Hotnets-II , 2003.

[11] Christos Papadopoulos, Robert Lindell, John Mehringer, Alefiya Hussain, Ramesh Govindan, “COSSACK: Coordinated Suppression of Simultaneous Attacks”, DARPA Information Survivability Conference and Exposition - Volume II, April 2003.

[12] Jiang Wu, Sarma Vangala, Lixin Gao, and Kevin Kwiat, “An Effective Architecture and Algorithm for Detecting Worms with Various Scan Techniques”, Network and Distributed System Security Symposium 2004.

[13] Eddie Kohler, Jinyang Li, Vern Paxson and Scott Shenker, “Observed Structure of Addresses in IP Traffic”, Proc. ACM SIGCOMM Internet Measurement Workshop, November 2002.

[14] Skitter measurement tool, http://www.caida.org/tools/measurement/skitter/.

[15] Cliff Changchun Zou, Weibo Gong, Don Towsley. “Code Red Worm Propagation Modeling and Analysis”, 9th ACM Conference on Computer and Communication Security (CCS’02), Nov. 18-22, Washington DC, USA, 2002.

[16] David Moore, Geoffrey M. Voelker, and Stefan Savage, “Inferring Internet Denial-of-Service Activity,” Usenix Security Symposium, 2001.

[17] Indranil Gupta, Ken Birman, Prakash Linga, Al Demers, and Robbert van Renesse. Kelips, “Building an efficient and stable P2P DHT through increased memory and background overhead”, IPTPS, 2003.

[18] Qin Lv, Sylvia Ratnasamy, Scott Shenker, “Can Heterogeneity Make Gnutella Scalable”, The 1st International Workshop on Peer-to-Peer Systems (IPTPS).

[19] Venkata N.Padmanabhan and Lakshminarayanan Subramanian, “Determining the Geographic Location of Internet Hosts”, Extended Abstract in ACM SIGMETRICS 2001, Boston, MA, June 2001.

[20] Michael Liljenstam, Yougu Yuan, BJ Premore, David Nicol, “A Mixed Abstraction Level Simulation Model of Large-Scale Internet Worm Infestations”, in MASCOTS, IEEE Computer Society Press, Fort Worth, TX, Oct 2002.

[21] Lakshminarayanan Subrmanian, Sharad Agarwal, Jennifer Rexford and Randy H.Katz, “Characterizing the Internet Hierarchy from Multiple Vantage Points”, IEEE INFOCOM 2002 , New York, June 2002.

[22] James Cowie, Andy Ogielski, BJ Premore, and Yougu Yuan, “Global routing instabilities during Code Red II and Nimda worm propagation”,http://www.renesys.com/projects/bgp_instability, September 2001.

[23] Ben Y. Zhao, Ling Huang, Jeremy Stribling, Anthony D. Joseph, and John D. Kubiatowicz, “Exploiting Routing Redundancy via Structured Peer-to-Peer Overlays”, The 11th IEEE International Conference on Network Protocols(ICNP) Atlanta, Georgia, November 2003.