Scalable Stealth Mode P2P Overlays of Very Small Constant Degree[*]

Márk Jelasity
University of Szeged and Hungarian Academy of Sciences, Hungary
Vilmos Bilicki
University of Szeged, Hungary

October 27, 2011

Abstract

P2P technology has recently been adopted by Internet-based malware as a fault tolerant and scalable communication medium. Due to its decentralized and self-organizing nature, P2P malware is harder to detect and block, especially if it utilizes specialized techniques for hiding. We analyse a number of hiding strategies through extensive and realistic simulations over a model of the AS-level Internet topology. We show that the most effective strategy to avoid detection is to drastically reduce the maximal number of peers a node communicates with. While overlay networks of a small constant maximal degree are generally considered to be unscalable, we argue that it is possible to design them to be scalable, efficient and robust. An important implication is that stealth mode P2P malware that is very difficult to discover with state-of-the-art methods is a plausible threat. We discuss algorithms and theoretical results that support the scalability of stealth mode overlays, and we present realistic event based simulations of a proof-of-concept system. Besides the context of P2P malware, some of our results are of general interest in the area of constant degree overlays in connection with the problem of how to maintain reasonable performance and reliability with the smallest degree possible.

1 Introduction

In recent years peer-to-peer (P2P) technology has been adopted by botnets as a fault tolerant and scalable communication medium for self-organization and survival [6, 4]. Examples include the Storm botnet [6] and the C variant of the Conficker worm [27].

The detection and filtering of P2P networks presents a considerable challenge [8]. In addition, it has been pointed out by Stern [29] that the potential threat posed by Internet-based malware would be even more challenging if worms and bots operated in a "stealth mode", avoiding excessive traffic and other visible behavior.

The obvious question is whether there are P2P techniques that are powerful enough to escape state-of-the-art methods for automatic detection. In the case of P2P networks, it has been argued that the best (if not the only) way to automatically detect them is to analyse traffic patterns they generate [9, 8]. We will elaborate on this question and examine a number of P2P techniques from this point of view. Our conclusion is that the most effective method is reducing the number of peers a node is allowed to communicate with.

[*] © ACM, 2011. This is the author's version of the work. It is posted here by permission of ACM for your personal use. Not for redistribution. The definitive version was published in *ACM Transactions on Autonomous and Adaptive Systems*, 6(4):27:1-27:20, October 2011. `http://doi.acm.org/10.1145/2019591.2019596` M. Jelasity was supported by the Bolyai Scholarship of the Hungarian Academy of Sciences.

Fortunately, current infections generate considerable traffic. For example, the Storm worm contacts a huge number of peers, in the range of thousands [4], when joining the network, generating a recognizable communication pattern as well as revealing a large list of botnet members. In general, P2P clients typically contact a large number of neighbors due to maintenance traffic, and regular application traffic such as search. There are only a few notable exceptions, such as Symphony and Viceroy [21, 20], which are overlay networks of a constant degree.

It is still an open question whether it is possible to create overlay networks of a very small constant maximal degree that are efficient and scalable. Research activity concerning Symphony or Viceroy has not yet been targeted to the lower end of maximal node degree, potentially as small as 3 or 4. In fact, even negative results are known that indicate the inherent lack of scalability of constant degree networks [18, 15]. In this paper we answer this question in the affirmative and show that it is possible to build a Symphony-inspired overlay network of a very small constant degree, and with the application of a number of simple techniques, this overlay network can be made scalable and robust as well. This result calls for more research into the detection of malicious P2P networks that are potentially of a small maximal degree.

Our contribution is fourfold. First, we experimentally investigate the effectiveness of hiding techniques in P2P malware over an AS-level model of the Internet. Second, we empirically analyze known as well as new techniques from the point of view of improving the search performance in a Symphony-like overlay network. Third, we present theoretical results indicating that if we add $O(\log N)$ (or, in a certain parameter range, $O(\log \log N)$) backup links for all links (where N is the network size), then a constant degree network becomes fault tolerant even in the limit of infinite network size, while its effective degree remains constant (that is, the network can remain in stealth mode) since the backup links are not used for communication unless they become regular links replacing a failed link. This result is counter intuitive because routing in Symphony requires $O(\log^2 N)$ hops on average. Fourth, we provide event-based simulation results over dynamic and realistic scenarios with a proof-of-principle implementation of a constant degree network, complete with gossip-based protocols for joining and maintenance.

2 P2P Malware Hiding Strategies

2.1 Strategies for Detection

State-of-the-art approaches for detecting P2P botnets rely on considerable human effort: a specimen of the P2P bot needs to be captured and reverse engineered, message exchange patterns and signatures need to be extracted, the network needs to be crawled using tailor-made software, its indexing mechanism poisoned, or otherwise infiltrated, and a countless number of creative techniques has to be applied such as visualizations, identifying abnormal patterns in DNS or blacklist lookups, understanding propagation mechanisms, and so on (e.g., [28, 6, 4]). Besides, aggressive infiltration and crawling introduces lateral damage: it changes the measured object itself very significantly [14].

While creative and knowledge-intensive approaches are obviously useful, it would be important to be able to detect and map botnets *automatically*, and as *generically* as possible. Ideally, network monitoring and filtering tools installed at routers and other network components should take care of the job with very little human intervention based on the (often enormous volumes of) Internet traffic constantly flowing through them.

This automation problem has been addressed in the context of IRC-based botnets [31] and, recently, also in the context of detecting generic botnet activity [5] and, specifically, P2P traffic [9]. We examine how techniques presented in [9] perform in the case of botnets.

If we want to identify and filter P2P botnet traffic in an automated way, we can focus on roughly three kinds of activity: propagation, attacks by the botnet, and overlay traffic, which involves repairing failed overlay links, adding new nodes to the overlay, and spreading and searching commands of the botmaster.

We argue that the most promising approach is to focus on *overlay traffic*. It has a small volume, but it is arguably the most regular and reliable traffic any P2P botnet generates, since the overlay network has to be constantly repaired, and bots need regular information about commands of the botmaster, updates, and so on.

Although the propagation of bots can be detected if it involves port scanning and similar suspicious network activity (e.g., [1]) bots can also spread under the radar via email, websites, file sharing networks, ad hoc wireless networks, or even via the old-fashioned way by infecting files on pen drives and on other portable media that generate no network traffic at all [17, 33].

Certain "visible" attacks—such as DDos, spamming or brute force login—could also be detected automatically. In an optimistic scenario, we can immediately identify and block those bots that participate in these attacks. But this is still far from enough: P2P overlays can apply very cheap and simple methods that can enable them to tolerate extremely well if large portions (even 80%) of the overlay gets knocked out [12]. In addition, certain types of malicious activity, such as collecting personal data (bank account information, passwords, etc) can blend into (even piggyback) overlay traffic perfectly.

2.2 Network Monitoring with Traffic Dispersion Graphs

Approaches to automated traffic classification and filtering typically start from observing packets or flows and classify them through the application of a stack of methods ranging from simple port-based filtering to sophisticated supervised or unsupervised machine learning and classification methods over packet and flow data. A summary of such methods is given in [26].

However, P2P botnet overlay traffic does not necessarily look malicious or harmful (in fact in itself it is neither), even if isolated and classified properly. It is essential to be able to identify this traffic as part of a *network* that, as such, makes it suspicious and could trigger a warning [3].

It has already been argued that it is very difficult to detect P2P traffic using packet or flow classification methods alone [9, 8]. Most of the key characteristics of P2P traffic lie in the network defined by the flows, the so-called traffic dispersion graph (TDG). By building and analyzing TDGs of locally observable flow data after the classification phase, it is possible to extract important additional clues about the organization of an application and, for example, label them as P2P.

In [8], the TDG is defined on top of a set of flows S that have the usual format <srcIP, srcPort, dstIP, dstPort, protocol>. The TDG is the directed graph $G(V, E)$ where V, the set of vertices, contains the set of IPs in S, and E, the set of edges, contains the edges (a, b) such that there is a flow in S with srcIP $= a$ and dstIP $= b$.

To be generous to the TDG method, we assume that *traffic that belongs to a given P2P botnet can be isolated in an unlabeled way*. That is, we assume that there are methods available to group a set of flows together that belong to the P2P botnet, but that we cannot determine whether the identified class of traffic is in fact botnet traffic. Note that this is an extremely strong assumption; in reality, classifying flows is very difficult in itself.

2.3 Our P2P Overlay Model

There is a huge number of design options for P2P overlays [19], so selecting a suitable model that allows us to draw conclusions on the detectability of P2P botnet overlay traffic in general is highly non-trivial.

Unstructured networks are extremely robust, and, due to their lack of structure, they can be harder to discover as well. However, command and control operations are more expensive; and, most importantly, communication cannot be localized (an important technique we describe below) since we have no structure to map on the underlay.

Although we do not rule out unstructured networks as a potential architecture for botnets, here we focus on structured networks; indeed, existing P2P botnets are known to be DHT-based [27].

As a model we use an ordered ring with exponential long range links, a simplified version of the Chord topology [30]: We have N nodes with IDs $0, 1, \ldots, N-1$. Node i is connected to nodes $i-1 \pmod N$ and $i+1 \pmod N$ to form the ring. In addition, node i is connected to nodes $i + 2^j \pmod N$ for $j = 1, 2, \ldots, (\log_2 N) - 1$, which are the long range links.

It is important to make a distinction between the overlay and the flows that exist in the overlay. Two nodes a and b are connected in the overlay if a "knows about" b. This, however, does not imply that a will ever actually send a message to b. For example, a might remember b simply in order to increase robustness in case of a failure. On the other hand, a node a might send a message to b even though b is not the neighbor of a in the overlay (that is, for the overlay to function properly, a does not need to remember b after sending it a message). For example, in Kademlia if a wants to find the node of ID x then a will actually contact all nodes on the route to x. This is why Storm bots generate so many messages locally as part of the overlay traffic [6].

In short, we want to model the flows and not the overlay *per se*, so our model refers to the *flows* we can potentially observe. In the actual overlay there would probably be links to the 2nd, 3rd, etc, neighbors in the ring as well that are learned from direct neighbors.

In the following we describe two fairly straightforward techniques: clustering and locality, that future botnets could use to hide their traffic. The key point is that, using these techniques, the functionality of the overlay can be preserved while using far fewer links and traversing fewer routers.

2.3.1 Clusters for Sharing Long Range Links

Clustering is a technique to reduce the actual number of neighbors a node actually communicated with, without changing the logical structure of our original overlay model, at the cost of a slightly increased hop-count.

In the ring every node has two neighbors that it actually communicates with at any given time, but it has $\log N$ long range links all of which are frequently used for communication to achieve as few as $O(\log N)$ hops in overlay routing (where N is the network size). Evidently, the ring would be sufficient for communication but then sending a message from a node to another random node would require $O(N)$ hops in expectation.

There is a middle ground: we can reduce the number of long range links to a constant number, and still have relatively efficient routing: $O(\log^2 N)$ hops [21] or even $O(\log N)$ hops [20].

However, let us remember that we are interested in the flows and not the overlay. In fact, we can modify our model to have a *single* long range flow per node, and still have $O(\log^2 N)$ hops for routing messages in expectation. The trick is to create clusters of $\log N$ consecutive nodes in the ring, and allow each node to actually use only one of its long range links. Routing proceeds as usual: but when a node decides to send a message over a long range link, it first has to locate the node in its cluster that is allowed to use that link and send the message to that node along the ring. Note that nodes that are in the same cluster can rely on an identical set of long range links since clusters can be interpreted as replicas of a node in an overlay of size $N/\log N$. In sum, from the point of view of flows all nodes now have two ring flows (one in and one out) and two long range flows on average (one out and one in on average).

It is important to note that clustering in our model is statistically very similar to applying a Viceroy topology with one long range link. We apply clustering in this section due to technical reasons.

2.3.2 Locality

We can also optimize the ring by trying to assign IDs to nodes in such a way that the resulting ring has links which touch the smallest possible number of routers. Several algorithms are known for achieving such optimized topologies that could be adapted to this application, for example [23, 10].

2.4 Simulation Experiments

To examine the partial TDGs as seen locally from several points of the Internet, we (i) created a static AS-level model of the Internet topology and routing, (ii) mapped the overlay network to this AS-level topology, (iii) and we analyzed the local TDGs that are defined for each AS by the set of traversing flows in our model. We will now elaborate on these steps.

2.4.1 The AS-level Underlay

As our AS-level underlay we used an AS link dataset from CAIDA [7] that we cleaned by deleting uncertain links (around 3% of all links). We are aware of the methodological problems with collecting AS-level links and simulating protocols over them. However, for the purposes of this study, the main goal was not to achieve perfect low level realism but to capture the important structural properties of the Internet as a complex network, a level that even a good topology generator could provide.

We calculated the shortest paths for each pair of nodes in the topology after assuming that edges have equal weights. As a simple model of BGP routing we assumed that flows actually follow these shortest paths. Shortest paths also define the betweenness centrality of each node, that is, the number of shortest paths that touch a given node. This is a very important metric from the point of view of TDGs, since an AS with a high betweenness value is likely to be able to capture a more complete view of the TDG of the application.

2.4.2 Mapping the Overlay to the Underlay

Based on notions described in Section 2.3, we experiment with two kinds of mappings: random and localized. First we describe the common settings for these two mappings, and then we discuss the specifics of both.

In all our experiments the overlay contains 100,000 nodes. Note that we do not expect our results to be sensitive to increasing the overlay size, since the overlay localization techniques we discussed in Section 2.3 essentially cause the problem to depend only on the AS-level graph.

The AS topology contains 14,630 nodes. With each type of overlay we map the overlay nodes onto the AS nodes in such a way that the number of overlay nodes in each AS is proportional to the size of the AS, but each AS has at least one overlay node. That is, in our model we do not take into account the geographical, social, or cultural bias that is known to affect botnet distribution [6]. The size of an AS is approximated based on the IP-prefix-to-AS mapping available from CAIDA[1]. It is interesting to note that the size and the betweenness of an AS seem to show no correlation,

In the *random mapping* we assign overlay nodes to ASes at random, keeping only size-proportionality in mind, as outlined above.

To create a *localized mapping* as described in Section 2.3.2, we first define a traveling salesperson problem (TSP) over the AS topology and, using a simple heuristic algorithm, we produce a “good enough” tour over the ASes. Finally, we assign the overlay nodes to ASes in such a way that the overlay ring is consistent with this tour as illustrated in Figure 1.

We define the TSP as follows: find a permutation of the ASes such that if we visit all the ASes exactly once in the order given by the permutation, but assuming a closed tour that returns to the origin, and assuming also that for each transition from one AS to the other we follow the shortest paths in the AS-level topology, then the *sum of the hops* in the AS-level topology is minimal.

The heuristic we applied is nearest neighbor tour construction [13]. We start the tour with a random AS, and iteratively extend the tour by adding an AS that has the smallest shortest path length among those ASes that have not yet been visited. Ties are broken at random.

Before moving on to the analysis of TDGs, some comments are in order. First, our simulation is completely indifferent to the way a solution for the TSP problem is generated (i.e., a P2P

[1] `http://www.caida.org/data/routing/routeviews-prefix2as.xml`

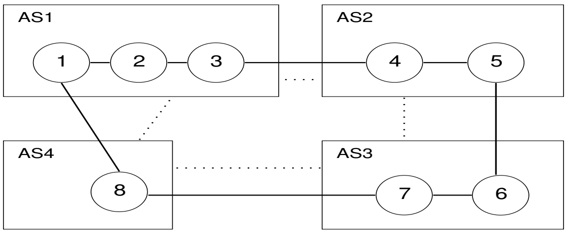


Figure 1: Localized mapping: alignment of the overlay ring (solid lines and circles) with the AS tour (dotted lines and rectangles) that is output by the nearest neighbor heuristic.

algorithm or some other arbitrary heuristic method). What we would like to focus on is what happens when the mapping is sufficiently well localized.

Second, the heuristic mapping we produce is most likely quite far away from the optimal localization. The actual optimal mapping is prohibitively expensive to calculate since the TSP problem is NP-hard in general, and we have a very large instance. Moreover, the definition of the localization problem itself could be refined as well, taking the requirements of the P2P botnet into account more directly in the objective function, and, for example, minimizing the sum or the maximal number of flows that can be seen at the ASes.

For these reasons our results should be interpreted as an upper bound on the amount of information that is available at local nodes.

2.4.3 Analysis of TDGs

We experimented with four overlay models that are given by the two kinds of mappings described in Section 2.4.2 (random and localized) with or without the clustering technique described in Section 2.3.1.

For these models we simply collected the flows that traverse a given AS, created the TDG, and collected statistics. The statistics we collected were the following: number of nodes, number of edges, number of weakly connected components, size of largest weakly connected component, average node degree (where we count both incoming and outgoing connections) and finally, a metric called InO, introduced in [9]. InO is the proportion of nodes that have both incoming and outgoing connections.

The results are shown in Figure 2. The first observation we can make is that the more efficient factor for hiding the overlay traffic is clustering. Recall, that the main effect of clustering is to reduce the flows each node participates in from $O(\log N)$ to 4 on average. The effect of localization is significant as well, but it is less dramatic overall. There is one exception: the largest connected component, where localization results in a value that is two orders of magnitude smaller for the two most central ASes.

Let us first compare these results to those found in [9] for existing P2P networks in real traces. There, it was concluded that P2P traffic can be characterized by a high InO value (larger than 1%) and a high average degree (larger than 2.8). From this point of view, the TDGs we observe do *not classify as P2P traffic*, because the average degree is extremely low: in fact less than 2 in the case of the localized and clustered network even for the most central ASes. It is interesting that even for the random mapping without clustering the threshold is crossed only at the most central ASes, although by a large margin.

On the other hand, the InO values are high. This is simply because we did not pay any attention to deceiving this metric explicitly. The reason is that in practice determining the direction of a flow is not very reliable, is prone to errors and quite possible to manipulate. We predict that the InO value could also be manipulated by a botnet using techniques that cannot be captured by the relatively high level model we apply that ignores flow details and dynamics.

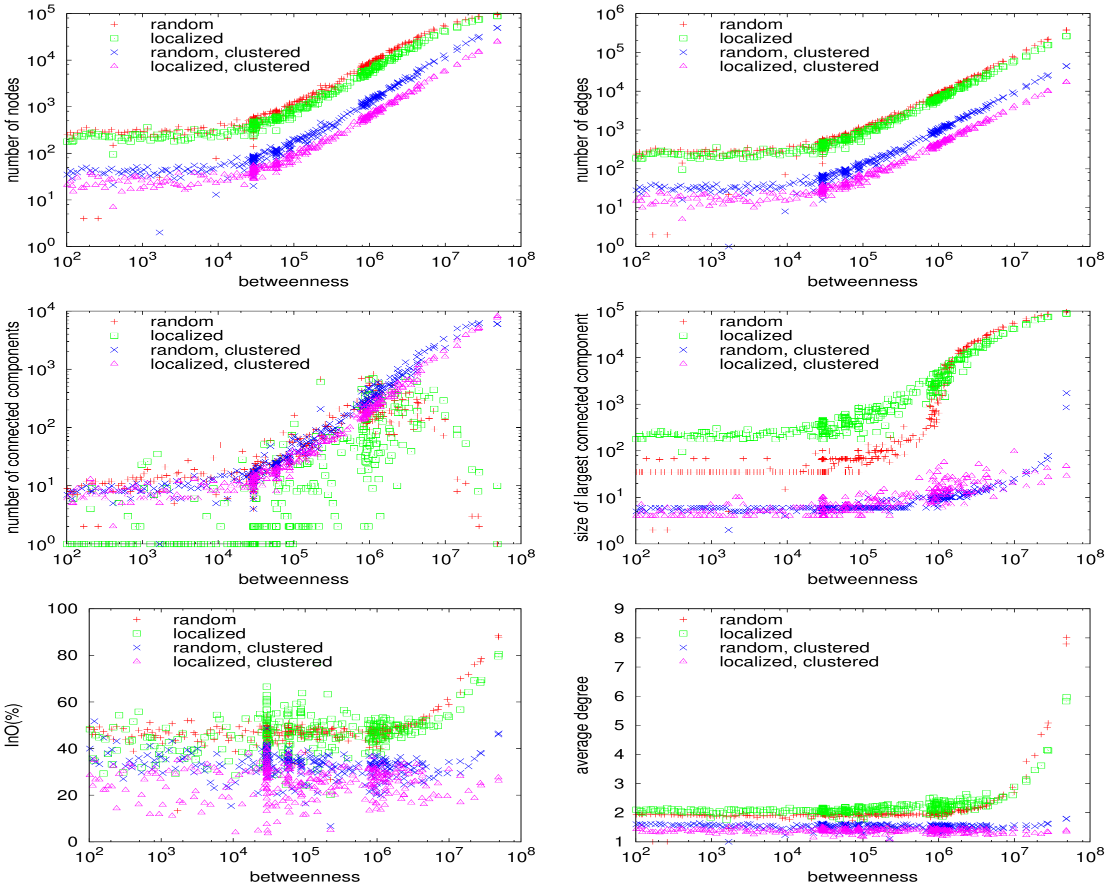


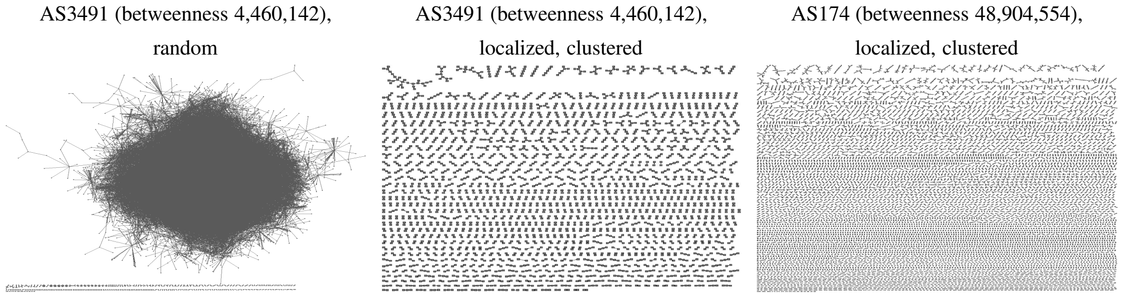
Figure 2: Different characteristics of the TDG as seen from different ASes of a given betweenness.

In addition, in [9] some applications with high InO and low average degree have been found: one example is FTP, where the server initiates connections to the client as well, which further complicates detection and offers the botnet other opportunities for camouflage.

Of course it is possible that other metrics could help characterize these TDGs as belonging to a P2P network. Let us look at the TDGs using other metrics in order to have a more precise idea of what information is visible locally. Out of the 200,000 edges in the overlay, even the most central AS can see only 16,814 edges. The number of nodes in the TDG is 24,985 which is much larger than the number of edges: indeed, the connected components are mostly of size 2 (pairs) and 3. There are 8,172 clusters, the maximal of which contains only 29 nodes. A visualization of the TDG that belongs to the most central AS is shown in Figure 3. The information available at the less central ASes is significantly less, as shown in Figures 2 and 3. Finally, the maximal node degree we observed in any TDG we have generated is no more than 4.

It is important to emphasize that results presented here are based on the assumption that within one AS transit traffic traces can be aggregated and treated in a unified way. Although not impossible, this is a rather strong assumption, especially for the most interesting ASes with high betweenness centrality, that handle enormous volumes of transit traffic. In practice, the information visible locally could be even more fragmented.

Overall, then, we may conclude that when localization and clustering are applied, the overlay network traffic is almost completely hidden. Clustering has a larger effect, which means that keeping the effective node degree mininal is the key technique to hide the network. A non-trivial proportion of the traffic can be seen only at the most central ASes, but even there, what is visible



AS3491 (betweenness 4,460,142), random

AS3491 (betweenness 4,460,142), localized, clustered

AS174 (betweenness 48,904,554), localized, clustered

Figure 3: Visualizations of TDGs at various ASes. AS174 had maximal betweenness in the dataset.

is predominantly unstructured.

3 Performance of Small Constant Degree Topologies

Motivated by the previous section, we would like to understand whether a reasonable routing performance can be achieved in a network that operates in stealth mode; that is, where the maximal node degree is as small as 3 or 4. We will base our discussion on Symphony, a simple constant degree network [21]. We explore several (existing and novel) simple techniques for improving the routing performance of Symphony at the lower extremes of maximal degree. To the best of our knowledge, this problem has not been tackled so far in detail by the research community.

Symphony was proposed by Manku et al. [21] as an application of the work of Kleinberg [16]. Like many other topologies, the Symphony topology is based on an undirected ring that is ordered according to node IDs. Node IDs are drawn uniformly at random from the interval $[0, 1]$ when joining the network. Apart from the two links that belong to the ring, each node draws a constant number of IDs with a probability proportional to $1/d$, where d is the distance from the node's own ID. Subsequently, each node creates undirected long-range links to those peers that have the closest IDs to the IDs drawn. (We note that an implementation needs an approximation of the network size N for normalizing the distribution. A rough, but practically acceptable, approximation exploits the fact that the expected distance of the closest neighbor in the ring is $1/N$.)

Symphony applies a greedy routing algorithm: at each hop the link is chosen that has the numerically closest ID to the target. Due to the undirected ring, the procedure is guaranteed to converge. It can also be proven that routing takes $O(\log^2 N)$ hops on average; the idea of the proof is to show that it takes $O(\log N)$ hops to halve the distance to the target.

3.1 Techniques for Improving Routing Performance

3.1.1 Lookahead

Greedy routing can be augmented by a lookahead procedure where nodes store the addresses of the neighbors of their neighbors locally as well, up to a certain distance. This way, route selection is based on the best 2, 3, etc., hop route planned locally as opposed to a 1 hop route. Routing with a single hop lookahead has been studied in detail [22, 25]. Since small constant degree networks have small local neighborhoods that can easily be stored and updated, we study 2 hop lookahead as well.

3.1.2 Degree balancing

To enforce a strict small upper bound on node degree, nodes that are already of the maximal degree have to reject new incoming long-range links. To make sure that most joining nodes can create

network size	2^i, $i = 10, 11, \ldots, 20$
maximal degree (k)	3 or 4 (1 or 2 long-range links)
lookahead	0, 1, or 2 hops
stratification	yes or no
join attempts	1, 2, or 4 attempts
degree balancing	1, 5, 10, or 20 neighbors checked
short link avoidance (m)	0, 1, 2, 3, or 4

Table 1: The parameter space of the experiments.

long-range links, we need to introduce balancing techniques. In addition to the usual technique of repeated join attempts, we propose degree balancing: when a node of maximal degree receives a join request, it first checks its closest neighbors in the direction of increasing node ID to see whether they have free slots for a link. This need not require extensive communication as neighbor information is available locally (and, for example, the lookahead mechanism described above also requires local neighborhood information).

3.1.3 Stratification

Since each node has only a small constant number of long-range links (1 or 2 in our case), many hops will follow the ring. It is therefore important that neighboring nodes in the ring have different long-range links. We propose a stratified sampling technique that involves dividing the long range links into a logarithmic number of intervals $[e^i/N, e^{i+1}/N]$ $(i = 0, \ldots, [\ln N] - 1)$. All the nodes first choose an interval at random that is not occupied by a long-range link at a neighbor, and then they draw a random ID from that interval with a probability proportional to $1/d$, where d is the distance from the node's own ID.

3.1.4 Short link avoidance

Interestingly, if the average route is long, then it might be beneficial to exclude long-range links that are too short. This way we introduce some extra hops at the end of the route, when routing follows the ring only. However, we save hops during the first phases due to the longer long-range links. As we will see later, this technique works well only in very small degree networks where routes are long, but in such cases we can obtain a significant improvement. We implement short link avoidance based on the same intervals defined for stratification above. We introduce a parameter m: the number of shortest intervals that should be excluded when selecting long-range links. For example, for $m = 2$, the first possible interval will be $[e^2/N, e^3/N]$.

3.2 Experimental Evaluation of the Proposed Techniques

We performed experiments using the parameter space in Table 1. For all parameter combinations, we first constructed the network and subsequently we selected 10,000 random node pairs and recorded the hop count of the routing.

A main methodological tool we apply to evaluate the large parameter space is drawing scatter plots to illustrate the *improvement* in the hop count as a function of a varying parameter. In these plots the points correspond to different combinations of the possible values of a subset of parameters. The remaining free parameters are the ones we are interested in; they are used to calculate the coordinates of the points as follows. The hop count for a specified setting for the free parameters is the horizontal coordinate of a point, whereas the vertical coordinate is the ratio of the horizontal coordinate and the hop count that belongs to another (typically baseline) setting of the same parameters.

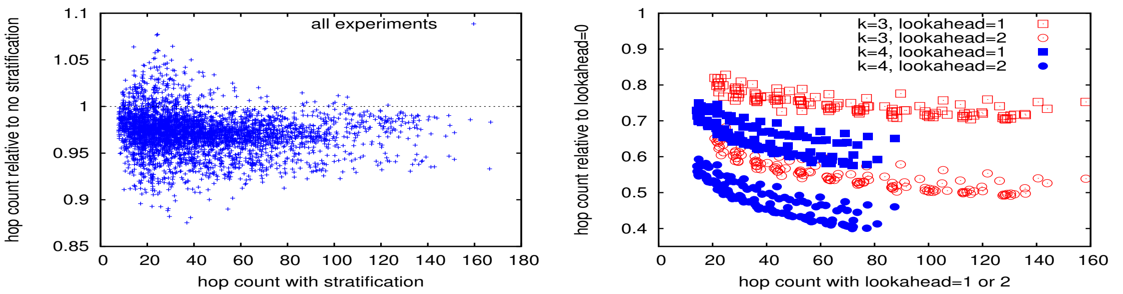


Figure 4: The improvement in hop count as a result of stratification (left) and lookahead (right).

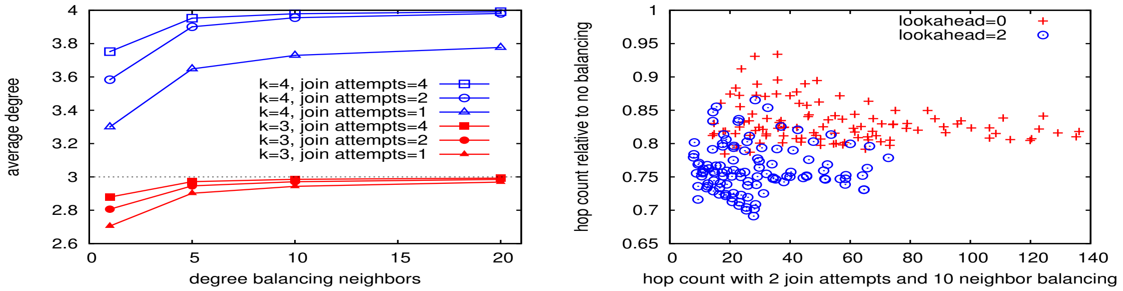


Figure 5: Average degree for $N = 2^{20}$ (left) and the performance improvement achieved by a good balancing strategy (right). The average degree is practically identical with all the other network sizes too (not shown).

The improvement brought about by stratification is illustrated in Figure 4 (left). Clearly, for almost all parameter settings, stratification is a better choice (most values fall below 1). The experiments with values higher than 1 were performed on the smallest networks, with no apparent additional common features. The lack of improvement in these cases is most likely due to the larger noise of random sampling in smaller networks. From now on, we restrict our discussion to experiments with stratification.

The improvement brought about by lookahead is shown in Figure 4 (right). We can see that lookahead helps more if the degree of the network is larger. This is plausible since the local neighborhood is exponentially larger in a network of a larger degree. We also notice that lookahead is more useful in larger networks where the routes are longer.

Let us now have a look at the average degree of the networks (Figure 5). The main observation here is that it is important to approximate the maximal degree because in some cases we can observe a performance improvement of almost 30% relative to the baseline approach (that is, when no balancing efforts have been made), especially if lookahead has been applied as well. The setting of 2 join attempts with degree balancing over 10 neighbors appears to be a good compromise between cost and performance.

Figure 6 illustrates the effects of short link avoidance. When $m = 2$ (left), performance is improved with each parameter setting, except for $k = 4$ and $lookahead = 2$, where routing is so efficient that even for the largest networks there are too few hops, so short link avoidance does not result in a net gain in hop count. For $m = 3$ (right) the same effect is amplified: for parameter settings with a large hop count the relative improvement is larger, but for short routes the relative cost is larger as well. All in all, the effect of this technique depends on the other parameters, but $m = 2$ appears to be rather robust and results in a slight improvement in most settings. We note that $m = 4$ was not the best setting in any of the experiments, so the maximal reasonable value was $m = 3$.

Lastly, Figure 7 shows hop count as a function of network size. Theory predicts an $O(\log^2 N)$

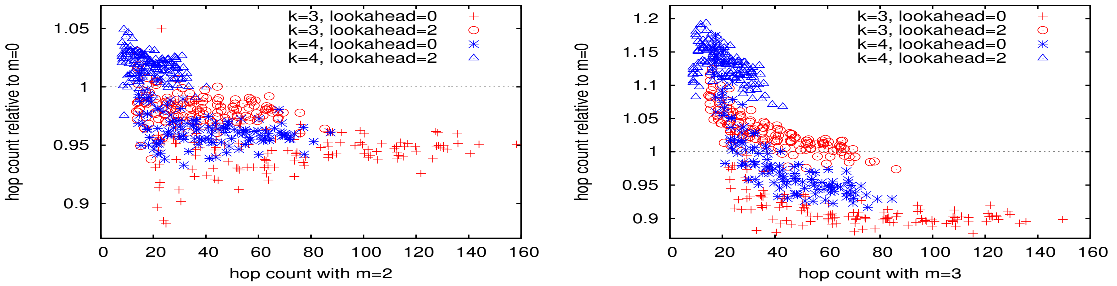


Figure 6: The improvement in hop count as a result of setting $m \neq 0$.

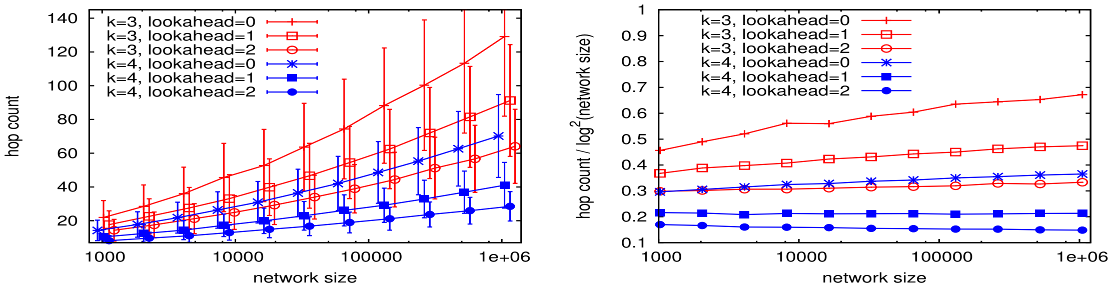


Figure 7: Scalability of routing. Statistics over 10,000 random node pairs are shown. Stratified sampling was applied, along with 2 join attempts with a degree balancing over 10 neighbors, and we set $m = 2$.

hop count complexity; to a good approximation we can observe this scaling behavior, especially for $k = 4$. The very large difference between the best and the worst setting is also worth noting. Moving from $k = 3$ to $k = 4$ results in a very significant improvement: 2 long-range links instead of 1 causes the speed of routing to double, as predicted by theory [21].

We may conclude that routing in networks of a very small constant degree is feasible if certain techniques are applied. We found that the most effective technique is lookahead based on locally available information about the neighborhood of the nodes. In addition, degree balancing is very important as well. Further techniques such as short link avoidance and stratification also result in an additional 5-10% improvement, depending on the parameters. With these techniques we can route in around 30 hops in a network of size $N = 2^{20} \approx 1,000,000$ with a maximal node degree of only 4.

4 Scalability of Fault Tolerance

In the above sections we discussed several aspects of scalability. Our focus in this section will be whether the *fault tolerance* of the network diminishes as the network grows. This is crucial from the point of view of P2P networks (in particular, botnets), which have to tolerate a considerable node churn, as well as other types of failures.

We first touch on some important issues regarding the scalability of constant degree topologies, and then we propose the simple technique of using backup links to increase their fault tolerance. We present theoretical results to show that the proposed technique indeed turns constant degree networks scalable in a well defined sense in the presence of node failures.

We consider properties of networks of size N as $N \rightarrow \infty$. This means that the results presented here are mainly of theoretical interest, since in practice an upper bound on network size can easily be given, and the algorithm designer can set protocol parameters according to the upper bound even if the algorithm is not scalable in the present sense. Still, our results are somewhat counter intuitive, and as such increase our insight into the behavior of constant degree networks.

The Achilles' heel of constant degree networks is fault tolerance and not performance. Performance is not a problem if the network is reliable. It is well-known that a constant number of neighbors is sufficient to build a connected structure. Not only the trivial constant degree topologies such as the ring or a tree are connected, but also there exist random topologies of constant degree such as the random k-out graphs. In such graphs each node is connected to k random other nodes. It has been shown that for $k \geq 4$ a k-out graph is connected with high probability [2]. It is also well-known that a constant degree is sufficient for an efficient routing algorithm. In the Symphony network routing takes $O(\log^2 N)$ hops while in Viceroy, the optimal $O(\log N)$ hop-count is achieved [20, 21].

Unfortunately, constant degree networks do not tolerate node failure very well. We will examine the case when each node is removed with a fixed constant probability q (that is, the expected number of nodes remaining in the network is $(1-q)N$). For example, the 4-out random graph is no longer connected with high probability in this model. In fact, in order to get a connected random topology in spite of node failures, one needs to maintain $O(\log N)$ neighbors at all nodes [15]. Similarly, it has been shown by Kong et al. [18] that—in this failure model—DHT routing is not scalable in Symphony, while it is scalable on other topologies that are able to find more alternative routes via maintaining $O(\log N)$ links at all the nodes. In the following, we summarize the results of Kong et al. for completeness and extend them to show how to achieve an effectively constant degree, yet scalable, topology.

Kong et al. examined the *success probability* of routing $p(h, q)$, the probability that in a DHT a node h hops away from a starting node will be reached by the routing algorithm under a uniform node failure probability q [18]. Their criterion for scalability is

$$\lim_{N\to\infty} p(h,q) = \lim_{h\to\infty} p(h,q) > 0, \quad 0 < q < 1-\epsilon, \tag{1}$$

where $\epsilon > 0$, and h is the average routing distance in the topology under study ($h = O(\log^2 N)$ for Symphony). This expresses the requirement that increasing network size should not increase sensitivity to failure without limit. Given this criterion, the proposed methodology consists of finding the exact formula or a lower bound for $p(h, q)$ for a topology of interest, and then calculating the limit to see whether it is positive. To calculate $p(h, q)$, one can create a Markov chain model of the routing process under failure, and determine the probability of reaching the failure state.

Kong et al. proved that Symphony is not scalable. They showed that for each step the probability of failure is a constant (C), so

$$\lim_{h\to\infty} p(h,q) = \lim_{h\to\infty} (1-C)^h = 0. \tag{2}$$

However, if we assume that there are *backup* links for each link in Symphony, then the situation changes dramatically. We do not go into detail here about how to collect the backup links; Section 5 discusses an actual algorithm. From our point of view here the important fact is that the backup links are such that if a link is not accessible, then the first backup is the best candidate to replace it. If the first backup is down as well, then the second backup is the best replacement, and so on.

Recall that the Symphony topology consists of a ring and a constant number of shortcuts. For the ring, the notion of backup should be clear. A shortcut link is defined by a randomly generated ID: we need to find the numerically closest node in the network to that ID. The first backup in that case is the second closest node in the network, and so on. This notion can be extended to all routing geometries as well.

The backup links do not increase the *effective* degree of an overlay node: a DHT can use the original links if they are available, even if some of the backups were closer to the target. In fact, backup links are *never used* for communication, not even during maintenance or any other function, except when they become regular links after replacing a failed regular link. In addition, as we explain in Section 5, backup links can be collected and updated during regular DHT maintenance without any extra messages.

It seems clear that backup links can turn Symphony scalable. However, the question is how many of them do we need? In the following we show that $O(\log N)$ backup links are sufficient, and under some circumstance even $O(\log\log N)$ links will do.

Lemma 4.1. *If in a DHT routing network all the links have $f(N)$ backup links then $p(h, q) \geq (1 - q^{f(N)})^h$.*

Proof. The probability of being able to use the best link in the original overlay is $1 - q$. Considering the backups this probability becomes $1 - q^{f(N)+1} > 1 - q^{f(N)}$. Now, if we follow only the optimal link in each step then the probability of success is not smaller than $(1 - q^{f(N)})^h$. Clearly, $p(h, q)$ is no less than this value since it accounts for methods for routing around failed links as well. □

Lemma 4.2. $\lim_{N\to\infty}(1 - q^{\log N})^{\log^k N} > 0$ *if* $0 \leq q < 1 - \epsilon$ *and* $k \in \mathbb{R}$.

Proof. For $k <= 0$ the lemma is trivial. For $k > 0$, according to Theorem 1 in [18] we need to prove that

$$\lim_{N\to\infty} q^{\log N} \log^k N < \infty$$

and the lemma follows. The convergence of the above expression can be proven by applying the l'Hospital rule on $(\log^k N)/q^{-\log N}$ a suitable number of times. □

Lemma 4.3. $\lim_{N\to\infty}(1 - q^{\log\log N})^{\log^k N} > 0$ *if* $0 \leq q < \min(e^{-k}, 1 - \epsilon)$.

Proof. We again need to prove that

$$\lim_{N\to\infty} q^{\log\log N} \log^k N < \infty.$$

Substituting $x = \log N$ we get

$$q^{\log x} x^k = x^{\frac{1}{\log_q e}} x^k = x^{\frac{1}{\log_q e}+k}$$

This means that we need $\frac{1}{\log_q e} + k \leq 0$ for convergence. Elementary transformations complete the proof. □

Theorem 4.4. *The Symphony topology is scalable, that is,* $\lim_{h\to\infty} p(h, q) > 0$*, if (i) all the links have $O(\log N)$ backup links, or if (ii) all the links have $O(\log\log N)$ backup links and $q \leq e^{-2} \approx 0.135$.*

Proof. Straightforward application of the previous lemmas for Symphony where $k = 2$, that is, $h = O(\log^2 N)$. □

To sum up, we have shown that Symphony-like topologies can be made scalable by adding only $O(\log N)$ backup links for all the links, and under moderate failure rates even $O(\log\log N)$ suffices. This is rather counter-intuitive given that routing still takes $O(\log^2 N)$ steps. These results suggest that collecting good quality backup links can dramatically improve scalability at a low cost.

5 Experimental Results

In this section we present proof-of-principle experiments with a simple implementation of a small constant degree network in realistic churn scenarios. Our goal is not to present a complete optimized implementation but rather to show that it is indeed possible to achieve acceptable fault tolerance and performance in realistic environments.

We performed the experiments using the PeerSim event-based simulator [24]. In our system model nodes can send messages to each other based on a node address. Nodes have access to a local clock, but these clocks are not synchronized. Messages can be delayed and nodes can leave or join the system at any time. The statistical model of node churn is based on measurement data [32], as we describe later.

Our goal was to design a protocol to construct and maintain a Symphony topology in a fault tolerant way, with backup links (see Section 4). To this end, we applied T-Man, a generic protocol for constructing a wide range of overlay topologies [11]. Here we briefly outline the protocol and the specific details of the present implementation. The reader is kindly requested to consult [11] for more information.

In our experiments each node has a single long-range link; that is, the maximal effective degree is 3. Each node has three local caches: long-range backups, ring backups and random samples. We set a maximal size of 80 for both the long-range and ring backup caches, and 100 for random samples. These values were chosen in an ad hoc way and were not optimized.

The caches contain node descriptors that include the ID and the address of a node. Each node periodically sends the contents of all its caches to all its neighbors. The period of this communication is called the *gossip cycle*, and was set to 1 minute in our experiments.

As described in Section 4, the two backup caches should ideally contain those nodes from the entire network whose IDs are closest to the node's own ID (for the ring neighbors), and the ID of the long-range link, respectively. When receiving a message containing node descriptors, a node updates its own local caches. It also updates the random sample cache, using a stratified sampling approach: the ID space is divided into 100 equal intervals, and each cache entry is selected from one of these intervals. If the random sample cache can be improved using any of the incoming node descriptors, the cache is updated.

In addition, if a node receives a message from a node it should not receive messages from (for example, because the sender has inaccurate knowledge about the topology) the node sends its caches to the sender of the misdirected message as well, so that it can improve its backup caches.

The join procedure starts by generating the node's own ID at random, as well as the ID for the long-range link. The caches need to be initialized as well, using a set of known peers; we applied 50 fixed descriptors for the initialization. Once the caches contain at least one link, the gossip protocol sketched above can start, and all the caches will fill and improve gradually.

When handling a routing request, a node applies greedy routing using the three links: the two ring links and the long-range link. However, before using the currently active ring links or long-range link, the node always checks the best candidate in the backup caches for availability (sending a ping message). Note that we do not check the best candidate for the message to be routed; we check the best candidate for the given link slot (ring or long-range). This way, it is guaranteed that the right links are used based on the current state of the network at any given time.

The scenario we experimented with involves node churn. Applying appropriate models of churn is of crucial importance from a methodological point of view. Researchers have often applied an exponential distribution to model uptime distribution, which corresponds to a failure probability independent of uptime. Measurements of a wide range of P2P networks in [32] suggest that a Weibull distribution of uptime is more realistic, with a shape parameter around $k = 0.5$. In this case the failure rate decreases, that is, the more time a node spends online, the less likely it is to fail. This favors longer sessions, but the Weibull distribution is nevertheless not heavy-tailed.

We applied the Weibull distribution with $k = 0.5$ to model uptime, and scaled the distribution, so that around 30% of the nodes live longer than 30 minutes [32]. The downtime distribution was

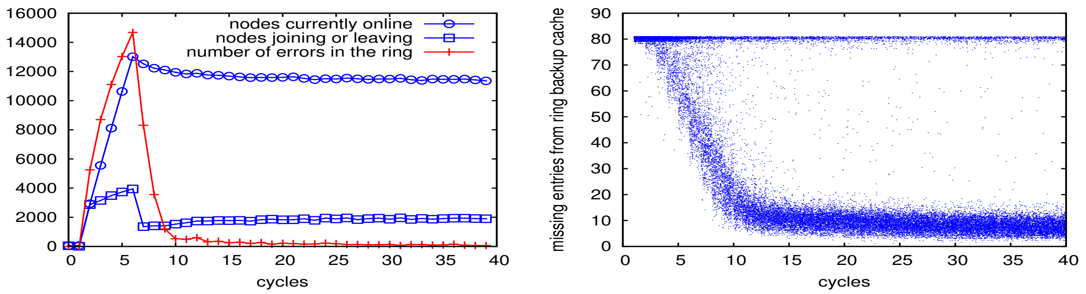


Figure 8: The evolution of the topology and the backup links for $N = 2^{14}$. In the figure on the right points belong to individual nodes and are randomly shifted so as to visualize the density.

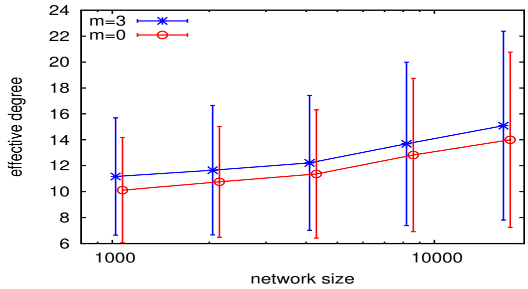


Figure 9: The observed effective degree by the end of the experiments.

modeled by a uniform random distribution, with an average downtime of 2 minutes. This average is very short; however, longer downtimes result in a relative increase in the proportion of nodes in the network that have long session lengths. Paradoxically, if the downtime is long, then the network is almost completely stable in the time range we are interested in (around 30 minutes).

As noted in [32], the lengths of the online sessions of a node correlate: there are nodes that tend to be available and nodes that are not. We assigned each node a fixed session length from the distribution above, that remained fixed during the experiment.

We applied a 1 minute gossip cycle. Each experiment lasted for 40 cycles. The network gradually grew to its final size during the first 10 cycles, when we added each node at a random time. During the remaining 30 cycles churn was applied. The network sizes we tested were $N = 2^i$, $i = 10, \dots, 14$. Parameter m (which controls short link avoidance) was set to $m = 0$ or $m = 3$. Other features such as lookahead, stratification, and degree balancing were not implemented at the time of writing.

Figure 8 illustrates the speed at which the ring topology is being formed despite of the continuous churn. The improvement of the backup links (80 links per node) for the ring links is also illustrated. It can be seen that most nodes collect good quality backups, but some of them seem to have no usable backups at all; these nodes have a very short session time and spend very little time in the network.

One of our main goals was to show that the effective degree—the number of nodes an average node actually communicates with—can be kept low. Figure 9 shows that indeed this can be accomplished. Despite heavy churn, which results in a constant fluctuation of the ring neighbors, the effective degree is small and seems to scale well. Recall that, for example, the Storm worm has been observed to communicate with thousands of neighbors [4].

Finally, let us examine the reliability and the efficiency of routing (see Figure 10). Recall that we work with a baseline implementation with no lookahead, stratification, or any other techniques. Only short link avoidance is implemented. When testing routing we pick IDs and not nodes as

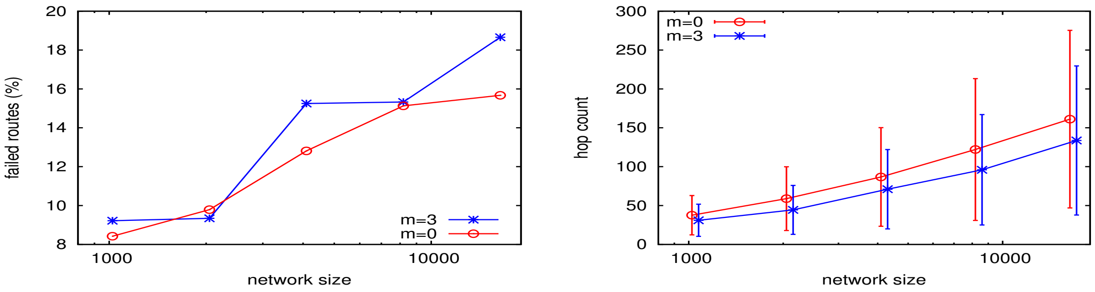


Figure 10: Routing performance. The figure on the right corresponds to successful routes.

targets, and consider routing successful if the closest node receives the message at the time of reception. That is, it is possible that the optimal target is different at the start of the routing and at the end of the same routing.

We can see that short link avoidance improves the hop count by a large margin. Overall, we observe almost twice the hop count as in the ideal case shown in Figure 7. However, in our hostile scenario with heavy churn this can be considered acceptable for this baseline approach. We also note that the actual routing performance observed in real botnets can be significantly worse; for example, the success rate of queries has been found to be extremely low in the Storm botnet [4].

6 Conclusions

In this paper we argued for the effectivity and feasibility of P2P systems that operate in a stealth mode via allowing nodes to communicate only with a very limited number of peers during their lifetime.

Our results have at least two implications. First, they are a strong indication that P2P botnets need to be taken seriously by the P2P community. We showed that stealth mode P2P networks are practically invisible for state-of-the art methods for P2P network detection. Current botnets do not exploit P2P technology to its full potential, and by the time they learn how to do that, they will be very difficult to detect and remove.

The second implication is not related to malware. There can be other applications where it is important to utilize very few connections because of a large associated cost. Detailed arguments for a constant degree design can be found in related works as well [20]. For this reason, the research issue that we raised, that is, the investigation of networks of a very small constant degree, and our related results are relevant to non-malicious applications as well.

References

[1] S. G. Cheetancheri, J. M. Agosta, D. H. Dash, K. N. Levitt, J. Rowe, and E. M. Schooler. A distributed host-based worm detection system. In *Proc. 2006 SIGCOMM workshop on Large-scale attack defense (LSAD'06)*, pages 107–113, New York, NY, USA, 2006. ACM.

[2] C. Cooper and A. Frieze. Hamilton cycles in random graphs and directed graphs. *Random Structures and Algorithms*, 16(4):369–401, 2000.

[3] D. Dagon. Botnet detection and response: The network is the infection, 2005. OARC Workshop presentation.

[4] J. Grizzard, V. Sharma, C. Nunnery, B. Kang, and D. Dagon. Peer-to-peer botnets: Overview and case study. In *Proc. 1st USENIX Workshop on Hot Topics in Understanding Botnets (HotBots'07)*, 2007.

[5] G. Gu, R. Perdisci, J. Zhang, and W. Lee. BotMiner: Clustering analysis of network traffic for protocol- and structure-independent botnet detection. In *Proc. 17th USENIX Security Symposium (Security'08)*, 2008.

[6] T. Holz, M. Steiner, F. Dahl, E. Biersack, and F. Freiling. Measurements and mitigation of peer-to-peer-based botnets: a case study on storm worm. In *Proc. 1st USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET'08)*. USENIX, 2008.

[7] Y. Hyun, B. Huffaker, D. Andersen, E. Aben, M. Luckie, k claffy, and C. Shannon. The IPv4 Routed /24 AS Links Dataset – 2008-01-02, 2008. `http://www.caida.org/data/active/ipv4_routed_topology_aslinks_dataset.xml`.

[8] M. Iliofotou, P. Pappu, M. Faloutsos, M. Mitzenmacher, S. Singh, and G. Varghese. Network monitoring using traffic dispersion graphs (TDGs). In *Proc. 7th ACM SIGCOMM conf. on Internet measurement (IMC'07)*, pages 315–320, New York, NY, USA, 2007. ACM.

[9] M. Iliofotou, P. Pappu, M. Faloutsos, M. Mitzenmacher, G. Varghese, and H. Kim. Graption: Automated detection of P2P applications using traffic dispersion graphs (TDGs). Technical Report UCR-CS-2008-06080, Dept. CS and Eng., Univ. California, Riverside, 2008.

[10] M. Jelasity and O. Babaoglu. T-Man: Gossip-based overlay topology management. In S. A. Brueckner, G. Di Marzo Serugendo, D. Hales, and F. Zambonelli, editors, *Engineering Self-Organising Systems: Third Intl. Workshop (ESOA 2005), Revised Selected Papers*, volume 3910 of *LNCS*, pages 1–15. Springer-Verlag, 2006.

[11] M. Jelasity, A. Montresor, and O. Babaoglu. T-Man: Gossip-based fast overlay topology construction. *Computer Networks*, 53(13):2321–2339, 2009.

[12] M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec, and M. van Steen. Gossip-based peer sampling. *ACM Trans. on Computer Systems*, 25(3):8, August 2007.

[13] D. S. Johnson and L. A. McGeoch. The traveling salesman problem: A case study in local optimization. In E. H. L. Aarts and J. K. Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 215–310. John Wiley and Sons, 1997.

[14] C. Kanich, K. Levchenko, B. Enright, G. M. Voelker, and S. Savage. The heisenbot uncertainty problem: Challenges in separating bots from chaff. In *Proc. 1st USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET'08)*. USENIX, 2008.

[15] A.-M. Kermarrec, L. Massoulié, and A. J. Ganesh. Probablistic reliable dissemination in large-scale systems. *IEEE Trans. on Parallel and Distr. Syst.*, 14(3):248–258, March 2003.

[16] J. Kleinberg. The small-world phenomenon: an algorithmic perspective. In *Proc. 32nd ACM Symposium on Theory of Computing (STOC'00)*, pages 163–170, New York, NY, USA, 2000. ACM.

[17] J. Kleinberg. The wireless epidemic. *Nature*, 449:287–288, 2007. News and Views.

[18] J. S. Kong, J. S. A. Bridgewater, and V. P. Roychowdhury. A general framework for scalability and performance analysis of DHT routing systems. In *Proc. Intl. Conf. on Dependable Systems and Networks (DSN'06)*, pages 343–354, Washington, DC, USA, 2006. IEEE CS.

[19] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim. A survey and comparison of peer-to-peer overlay network schemes. *IEEE Commun. Surveys & Tutorials*, 7:72–93, 2005.

[20] D. Malkhi, M. Naor, and D. Ratajczak. Viceroy: A scalable and dynamic emulation of the butterfly. In *Proc. 21st ACM Symposium on Principles of Distributed Computing (PODC'02)*, pages 183–192, New York, NY, USA, 2002. ACM.

[21] G. S. Manku, M. Bawa, and P. Raghavan. Symphony: Distributed hashing in a small world. In *Proc. 4th USENIX Symposium on Internet Technologies and Systems (USITS'03)*, 2003.

[22] G. S. Manku, M. Naor, and U. Wieder. Know thy neighbor's neighbor: the power of lookahead in randomized p2p networks. In *Proc. 36th ACM Symposium on Theory of Computing (STOC'04)*, pages 54–63, New York, NY, USA, 2004. ACM.

[23] L. Massoulié, A.-M. Kermarrec, and A. J. Ganesh. Network awareness and failure resilience in self-organising overlay networks. In *Proc. 22nd Symposium on Reliable Distributed Systems (SRDS 2003)*, pages 47–55, Florence, Italy, 2003.

[24] A. Montresor and M. Jelasity. Peersim: A scalable P2P simulator. In *Proc. 9th IEEE Intl. Conf. on Peer-to-Peer Computing (P2P 2009)*, pages 99–100, Seattle, Washington, USA, September 2009. IEEE. extended abstract.

[25] M. Naor and U. Wieder. Know thy neighbor's neighbor: Better routing for skip-graphs and small worlds. In *Peer-to-Peer Systems III*, volume 3279 of *LNCS*, pages 269–277. Springer, 2005.

[26] T. T. T. Nguyen and G. Armitage. A survey of techniques for Internet traffic classification using machine learning. *IEEE Communications Surveys and Tutorials*, 10(4):56–76, 2008.

[27] P. Porras, H. Saïdi, and V. Yegneswaran. A foray into Conficker's logic and rendezvous points. In *2nd USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET'09)*. USENIX, 2009.

[28] A. Ramachandran, N. Feamster, and D. Dagon. Revealing botnet membership using DNSBL counter-intelligence. In *Proc. 2nd Workshop on Steps to Reducing Unwanted Traffic on the Internet (SRUTI'06)*, 2006.

[29] H. Stern. Effective malware: The trade-off between size and stealth. In *2nd USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET'09)*. USENIX, 2009. invited talk.

[30] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. 2001 Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, pages 149–160, San Diego, CA, 2001. ACM, ACM Press.

[31] W. T. Strayer, D. Lapsely, R. Walsh, and C. Livadas. Botnet detection based on network behavior. In W. Lee, C. Wang, and D. Dagon, editors, *Botnet Detection: Countering the Largest Security Threat*, volume 36 of *Advances in Information Security*, pages 1–24. Springer, 2008.

[32] D. Stutzbach and R. Rejaie. Understanding churn in peer-to-peer networks. In *Proc. 6th ACM Conf. on Internet Measurement (IMC'06)*, pages 189–202, New York, NY, USA, 2006. ACM.

[33] N. Weaver, D. Ellis, S. Staniford, and V. Paxson. Worms vs. perimeters: the case for hard-LANs. In *Proc. 12th Annual IEEE Symposium on High Performance Interconnects (HOTI'04)*, pages 70–76, Washington, DC, USA, 2004. IEEE CS.