The Heisenbot Uncertainty Problem: Challenges in Separating Bots from Chaff

Chris Kanich Kirill Levchenko Brandon Enright Geoffrey M. Voelker Stefan Savage
University of California, San Diego
{ckanich,klevchen,voelker,savage}@cs.ucsd.edu
bmenrigh@ucsd.edu

Abstract

In this paper we highlight a number of challenges that arise in using crawling to measure the size, topology, and dynamism of distributed botnets. These challenges include traffic due to unrelated applications, address aliasing, and other active participants on the network such as poisoners. Based upon experience developing a crawler for the Storm botnet, we describe each of the issues we encountered in practice, our approach for managing the underlying ambiguity, and the kind of errors we believe it introduces into our estimates.

1 Introduction

Underlying virtually all measurement endeavors is the premise that the signal being measured can be separated from any noise produced by the environment or the measurement system itself. If this assumption is not true or if care is not taken to account for noise, then the resulting measurements may be badly skewed or meaningless. This issue has long been appreciated in the Internet measurement community, since even quiescent networks receive continual streams of bizarre traffic colloquially referred to as "Internet background radiation" [7]. However, it is less widely appreciated that variants of this problem impact our ability to meaningfully study online distributed malware such as botnets.

In particular, the practice of "crawling" as a technique for measuring the size, topology and dynamism of distributed botnets is fraught with the potential for error. At a minimum researchers must contend with temporal dynamics (e.g., address space reuse), unrelated applications (i.e., many bots piggyback on existing communications protocols), and address aliasing (e.g., via NAT). Potentially worse, however, is the impact of other *active participants*. Unlike traditional malware whose environment is largely self-contained within a single machine, distributed malware is effectively a single artifact — providing services to the entire Internet. Thus, no single group can safely presume to be measuring a botnet in isolation. In fact the activities of other researchers, miscreant competitors or well-meaning vigilantes will automatically be part of any raw measurement.

In this paper, we highlight these problems via our experiences developing a crawler for the Storm botnet. Storm uses an existing well-documented peer-to-peer protocol to implement a directory service and thus is straightforward to crawl. Unfortunately, the intuitive way to implement such a crawler — recursively querying peers for their neighbors until a transitive closure is complete — can produce a size estimate many times larger than the true size of the network. Indeed, depending on how one measures — over different time scales, using IP addresses or peer identifiers, using inbound requests or active probes — different characterizations can be reached. We believe this may explain in part the considerable diversity of claims in the security press concerning the purported "true size" of the Storm network. Even discounting the effects of flooders, naive approaches for counting the number of active Storm nodes result in estimates an order of magnitude larger than the actual number. Over the course of a typical day, simply counting unique Overnet IDs would estimate over 900,000 bots and counting unique IP addresses would estimate over 300,000, whereas the actual number is likely less than 40,000.

In the remainder of this paper we describe each of the issues we encountered in practice, our approach for managing the underlying ambiguity, and the kind of errors we believe it introduces into our estimates. In particular, we highlight the presence of other active participants in the system who, at particular points in time, can originate significant fractions of Storm-related traffic on the network. Finally, we argue that this problem is likely to be fundamental and will continue to present challenges for measuring distributed malware in the future.

2 Related Work

The combination of its effectiveness and unique architecture has focused increasing attention on the Storm botnet. Initial work has detailed the operation of the Storm malware as it transforms a host into a bot, as well as the basic bot communication patterns [4, 8]. Recent work searching the identifier space of the DHT underlying Storm reveals significant non-uniformities from

poisoning, and suggests one heuristic for pruning poisoned peers [11]. We compliment this work with a more comprehensive pruning strategy that detects and distinguishes among peers that are unresponsive, advertise incorrect addresses, or attempt to poison the network.

The size of a botnet is the most popular metric measured and reported, yet perhaps the one with the most variance. Rajab et al. [10] explore this issue in detail, highlighting the challenges of measuring botnet size and explaining why size estimates can vary substantially. Our goal is to identify and distinguish active bots in the system in the face of unrelated application traffic, address space reuse, address aliasing, and other active participants such as poisoners, polluters, and researchers. In that sense, we extend the set of techniques in [10] and [11] for refining size estimates specifically for the Storm botnet, particularly in the face of poisoning and NAT. In doing so, though, we not only obtain a more accurate estimate of botnet size, but more importantly remove noise from other analyses such as the dynamics, operation, actions, and behavior of Storm.

Address space reuse via DHCP and address aliasing via NATs [1, 2, 3] add uncertainty to bot disambiguation when using IP addresses [10]. Application identifiers can disambiguate in both instances [2], but NATs present yet another problem. Ensuring that a peer is live and functioning (beyond just appearing in a routing table) requires active communication with the peer. However, a crawler cannot directly contact peers behind NATs. Instead, analogous to [3], we rely upon communication initiated by peers behind NATs to observe and disambiguate them.

Finally, since poisoning attacks are being waged against Storm, whether effective or not as an attack against Storm [4] any measurement efforts must take poisoning into account [11]. We describe effective heuristics for differentiating between valid and masquerading Storm nodes and a strong predicate based upon reverse-engineering the Storm OID generator.

3 The Storm Overnet

We start by providing an overview of the Storm communication architecture. Because of its implications on crawling and measuring the size of the Storm botnet, we also describe the custom pseudo-random number generator Storm uses for generating identifiers.

3.1 Architecture

The Storm botnet is organized around a Kademlia-based [6] distributed hash table (DHT) implemented using the Overnet protocol. Each node is identified by a 128-bit Overnet identifier (OID). At startup, a node chooses its OID *pseudo-randomly* and proceeds to find its neighbors (in the space of OIDs). The search is bootstrapped from a list of peers included with the binary itself. The node then advertises itself to its neighbors in the Overnet, and continues advertising itself throughout its lifetime to maintain a presence in the network as its set of neighbors changes. To insert a key-value pair into the DHT, a node *publishes* the key, which comes from the same 128-bit OID space as node identifiers, to a node with an OID close to the key. To retrieve the value associated with a key, a node searches for a node close to the key; if a node has the associated value, it advertises the value to the searching node, which then requests it.

Because every Storm node participates in Overnet, enumerating its participants is a logical means of identifying bots and determining the botnet's size and composition. However, not all participants can be trusted implicitly; Storm has both been the target of attempts to disrupt the network via poisoning, as well as the victim of buggy implementations which report participants incorrectly. As we will show, each of these properties make identifying the true population of the botnet nontrivial. Based upon the protocol and correct functionality of individual bots, we have developed a number of heuristics for removing invalid nodes from our participant list. Doing so also allows us to much more accurately estimate the size of the network.

3.2 Storm OID generator

While reverse-engineering the algorithm Storm uses to generate OIDs, we discovered that it can generate only a very small fraction of the possible OIDs (2^{15} Storm OIDs out of the full 2^{128} OID space). Although an apparent flaw in Storm's OID generator, it is very helpful for a crawler because it provides a convenient means for identifying Storm nodes. We use this property as an oracle to evaluate heuristics for differentiating Storm nodes from other Overnet nodes (Section 4.2), and to differentiate valid Storm nodes from masquerading Storm nodes (Section 4.3).

The flaw in the OID generation algorithm comes from its use of a custom pseudo-random number generator (PRNG). Storm generates OIDs using its own PRNG given by the recurrence:

$$I_{i+1} = (a \cdot I_i + b \bmod 2^{32}) \bmod m$$

with $a = 1664525$, $b = 1013904223$, $m = 32767$, and the initial value I_0 is based on the system clock. The generator appears to be based on a well-known linear congruential PRNG described in the *Numerical Recipes* series of books [9, Ch. 7], which differs from the Storm PRNG only in the absence of the second modular reduction ($\bmod m$). A pseudo-random OID is generated by concatenating the low-order bytes of successive PRNG values:

$$\mathrm{OID}_i = I_i \bmod 256.$$

where OID_1 through OID_{16} are the 16 bytes of the OID. Because there are only 32767 states of the PRNG, there are only 32767 possible Storm OIDs.

In addition to making it easier to identify valid Storm nodes, this limitation has implications for measuring the size of the Storm botnet as well. Because there are only 32767 possible Storm OIDs, collisions in the OID space have been inevitable throughout a majority of the Storm botnet's lifetime. As a result, multiple nodes will join the network with the same OID at the same time, and therefore OIDs cannot be used to estimate the size of the botnet. Furthermore, which node receives a message addressed to an OID with collisions becomes nondeterministic. It depends on which peers receive publish messages, and how those peers deal with aliasing in their routing table implementation. To mitigate the problem, a crawler should search for an OID at several different points in the network.

4 The Stormdrain crawler

We now describe the operation of Stormdrain, the crawler we developed to identify and track nodes actively participating in the Storm network. Using Stormdrain observations, we also show its estimates of the number of active Storm participants for three weeks in March. We then discuss various unique challenges of crawling a botnet like Storm, the implications those challenges have on the accuracy of identifying and tracking valid Storm nodes, and the approaches we developed for dealing with them.

4.1 Stormdrain operation

We have used Stormdrain to track the Storm botnet since June 2007. The initial purpose of Stormdrain was to identify infected nodes at our institution to maintain a clean network. The crawler bootstraps onto the network using a list of peers either from a previous crawler instance or a list of peers extracted from a Storm binary. The crawler then contacts every peer on its internal list and requests a list of additional peers to increase its knowledge of the connected peers. After communicating with all known peers, the crawler performs a pruning subroutine to determine which peers to remove from its contact list either because they are no longer responding, are incorrectly advertised addresses, or are possibly attempting to poison the network. After pruning, Stormdrain repeats the process continuously, and periodically dumps its internal state.

Figure 1 shows the state machine that Stormdrain uses to track the state of nodes in an Overnet network. Stormdrain continuously sends messages to peers to monitor their responsiveness, contacting each peer roughly every 30 seconds. How peers respond to those probes determines how they transition among states in the machine:

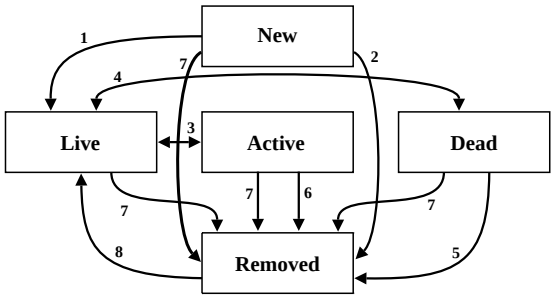

Figure 1: The Stormdrain state machine for tracking the state of nodes in Storm. At any point in time, the nodes in the active state represent Stormdrain's estimate of hosts actively participating in the Storm botnet.

1. Stormdrain learns of *new* nodes as it crawls the routing tables of peers on the network, and when it receives unsolicited messages from other peers. When a new peer replies to a probe, Stormdrain moves it to the *live* state.
2. If a new peer does not respond to any probes, Stormdrain places it in the *removed* state.
3. If a peer responds to a sufficient number and kind of probes, the peer moves from live to *active*. If an active peer falls below that threshold it moves back to the live state. We use different kinds of messages and require multiple responses to help differentiate between actual Storm nodes and masquerading active responders (Section 4.3).
4. Stormdrain moves a live peer that has not responded after a timeout expires to the *dead* state (there are many reasons why a node may not respond (Section 4.4)). A dead peer that responds before being removed moves back to the live state. Stormdrain currently uses a timeout of 15 minutes based upon our experience.
5. Stormdrain probes dead peers at a lower rate. If it does not respond after a timeout, it moves to the removed state.
6. An active peer that appears to be abusing Overnet (flooding, poisoning, broken implementation, etc.), moves to the removed state immediately and bypasses any other state; we discuss this issue in more detail in Section 4.3.
7. Any short sequence of probes to a peer that generate ICMP error responses moves that peer to the removed state, again bypassing any other state.
8. If a removed peer that was previously dead starts responding again, it moves back into the live state. Stormdrain clears all statistics and counters for the peer and treats it as if it were a new peer.

The set of nodes in each of these states captures the activity of nodes in the Storm botnet. The set of new nodes are those nodes which have been advertised to

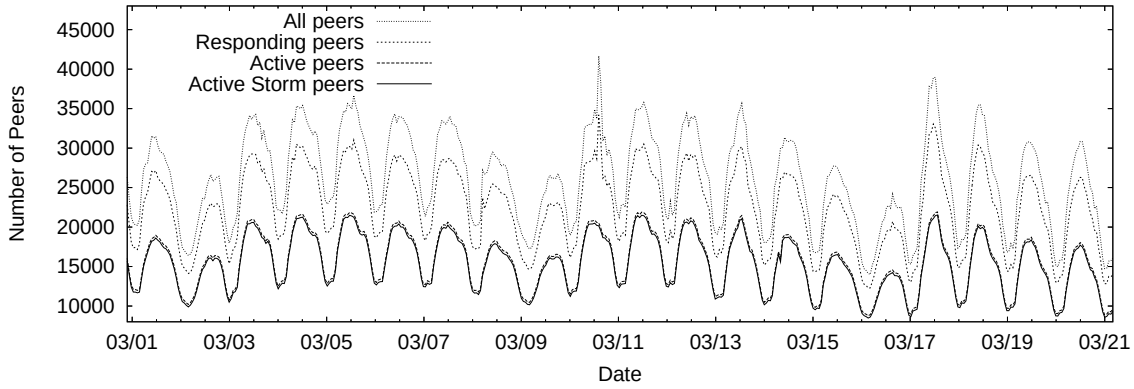

Figure 2: Estimates of the size of the Storm botnet using different notions of liveness over the first three weeks of March 2008. Note that the y-axis does not begin at zero to better separate the curves.

State	# Nodes	
New	16,042	(5.53%)
Live	12,955	(4.47%)
Active	38,630	(13.3%)
Dead	4,898	(1.69%)
Removed	217,737	(75.0%)
Total	290,262	(100%)

Table 1: Number of nodes in each of Stormdrain's states at a representative moment on February 20, 2008.

Stormdrain but have yet to respond to a probe. The set of live nodes are those nodes which are responding to Stormdrain probes, but have not responded sufficiently well to a range of messages to identify them as Storm participants (Section 4.3). The set of active nodes are those nodes which properly respond to a range of Storm messages and appear to be active participants in the Storm botnet. The sets of dead and removed nodes track expired nodes (Section 4.4) which can become live again (*dead*) or be removed from further tracking (*removed*). As one example, Table 1 shows the number of unique nodes in each of Stormdrain's states instantaneously (note that *removed* is a two hour sliding window of recently removed peers).

An immediate consequence of Stormdrain's node tracking is that we can estimate the number of active participants of the Storm botnet. Figure 2 shows estimates for various definitions of participation for the first three weeks of March 2008. Each count of peers corresponds to an estimate of Storm's "live population" [10], the number of peers active at that point in time.

"Active" shows the number of nodes in the *active* state, "Responding" shows the number of nodes in the *active* and *live* states combined, and "All" shows the number of nodes in the *new*, *active*, *live*, and *dead* states combined. The "Active Storm" curve is a strict subset of "Active" showing Stormdrain's best estimate of the number of true participants in the Storm botnet using knowledge of the valid set of Storm OIDs (Section 3.2). Nodes in "Active" pass Stormdrain's heuristic for responding sufficiently well to be labeled as active Storm participants. However, this heuristic may fail if other parties use a sufficiently rich emulation of a Storm node, or the Storm implementation itself. Assuming those parties have not discovered the flaw in Storm OID generation, we can use the Storm OIDs as an oracle to reject nodes as active Storm nodes if their OID is not in the known Storm OID set. Considering how close the "Active" and "Active Storm" curves are, the promotion heuristic works quite well. We discuss this issue in more detail in Section 4.3.

These different curves correspond to estimates based on different definitions of participation. "All" includes those nodes that have responded to probes within a long timeout (30 minutes), "Responded" includes those within a shorter timeout (15 minutes), and "Active" and "Active Storm" include those that appear to be valid nodes and have responded immediately. The "All" and "Responding" estimates include a large number of false positives (Sections 4.3) and nodes that are expiring (Section 4.4). As a result, we consider "Active" to be a more accurate estimator.

Note that Stormdrain actually tracks two independent variants of the Storm network simultaneously, the remnants of an older version of the Storm implementation and the botnet formed by the current Storm implementation that we refer to as the "encrypted" network (Section 4.2). Figure 2 shows results for nodes in the encrypted network only, since those are the nodes which are available, e.g., for Storm spamming activities [5].

4.2 Protocol aliasing

One unique challenge to tracking the Storm botnet is that Storm and other applications use the same overlay

network protocol, creating a protocol aliasing problem. Before October 2007, Storm shared its overlay network with users of Overnet-based file-sharing programs such as MLDonkey. As a result, a crawler needs to differentiate between nodes participating in the Storm protocol and other applications. Simply identifying participants in the network as Storm would include the file-sharing nodes, thereby overestimating the botnet population.

During the evolution of Stormdrain, we explored two heuristics for differentiating Storm nodes from nodes using other applications. One technique is to classify nodes by their usage of the protocol and by the content hashes they publish and search for. Overnet publish messages can contain metadata tags attached to key-value pairs. File sharing applications can publish content hashes with metadata, but all but a few versions of Storm do not. Consequently, a crawler can mark the nodes publishing or searching for these hashes as non-Storm nodes. The heuristic is transitive: when a node is marked as non-Storm, the crawler can mark other hashes that the node searches for and publishes as non-Storm hashes. Any node that searches for or publishes those hashes is non-Storm as well. With this heuristic, the more time spent crawling the network, the more comprehensive the set of known hashes becomes.

Conversely, we can also positively identify Storm nodes by their use of hashes specific to Storm. When a newly infected Storm node starts, for example, it searches for well-defined hashes to rendezvous with the rest of the Storm botnet. With the assistance of Joe Stewart, we reverse-engineered the algorithms used to encode and decode Storm key-value pair content hashes. Stormdrain can therefore identify hashes as known Storm content with very high probability, and mark any node searching for or publishing those hashes as Storm nodes.

Once we reverse-engineered the Storm OID PRNG (Section 3.2), these heuristics were no longer necessary. An Overnet message containing one of the 32K Storm OIDs immediately identifies that node as a Storm node. For evaluating our methodology, however, we can use the set of Storm OIDs as an oracle to estimate the accuracy of the above heuristics.

For instance, we examined a snapshot of 24 hours of nodes observed on the Overnet network that are subject to the protocol aliasing problem between Storm and file-sharing applications. Using the known Storm OID set as an oracle, 35% of those nodes are infected with Storm. The metadata heuristic greatly overestimates the set of Storm nodes as 94% of all nodes. The known Storm hash heuristic, however, severely underestimates the set, only identifying one node as a Storm node. Neither heuristic effectively resolves the protocol aliasing problem, motivating the other heuristics we developed below.

In October 2007, Storm began to "encrypt" its messages using a simple transformation with a static key. With this change, Storm bifurcated into "encrypted" and "unencrypted" networks: newly infected nodes communicated with each other using encrypted messages, and previously infected nodes were stranded. This older version of Storm remains active today, however. At any one point in time, nearly 5,000 hosts remain infected with the older form of Storm communicating using "unencrypted" traffic. Stormdrain actively tracks nodes in both variants of the Storm network.

Fortunately, the use of encrypted communication conveniently solves the protocol aliasing problem: all nodes using encryption use the Storm protocol. With the reverse-engineered key, we could readily separate Storm nodes from other nodes in the Overnet network. However, although nodes on the encrypted network communicate using the Storm protocol, not all of those nodes are actually hosts infected with the Storm bot.

4.3 Adversarial aliasing

Another unique aspect of crawling a botnet like Storm is the challenge of differentiating between actual Storm nodes and other nodes that masquerade as Storm nodes. We call this effect adversarial aliasing. Over time Storm has received significant attention from other parties, and the actions of these parties substantially influence the act of measuring Storm. Just as we crawl and monitor Storm, so do other groups. Storm has also been the target of attempts to poison or pollute the network to prevent it from functioning efficiently, or at all. As one example, earlier in Storm's life cycle, updates to the binary were disseminated by publishing URLs to a set of hard-coded keys. By advertising non-existent nodes close to those keys, it was theoretically possible to prevent nodes from finding the published URLs and thus prevent them from downloading updates [12]. As another example, the connectionless nature of UDP allows messages with spoofed source addresses to reach the application layer and force peers to consider these IPs as possible nodes in the network. Finally, when a node trusts unsolicited publish messages, an attacker can overwrite the value stored at the key and pollute the network.

Network poisoning, for example, remains an active phenomenon. Figure 3 shows Stormdrain's estimates of live, active, and dead peers on March 10, 2008. At 14:00 there is a sudden influx of messages from several so-called unique hosts. Since these hosts contacted Stormdrain directly, it considers these hosts as live. These hosts do not respond to any subsequent probes, however, and Stormdrain quickly moves them to the dead state and eventually removes them. Without these responsiveness heuristics, those 5,000 spoofed hosts would have substantially inflated the estimate of the size of Storm.

Given the presence of other nodes masquerading as

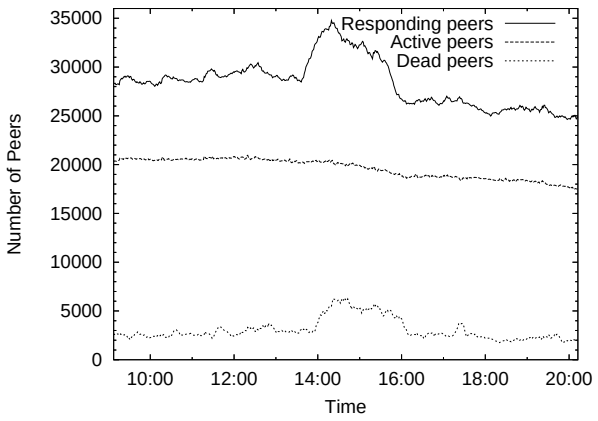

Figure 3: Live, active, and dead peers on the encrypted network on March 10, 2008. Note the anomalous increase in the number of live peers at 14:00 due to source address spoofing by other parties.

Location	Hallmarks
Germany	Random OIDs with lower 10 bytes constant. Floods the Storm network aggressively with thousands of fake node IPs.
Iran	Random OIDs biased to upper half of space (first bit always set).
Sweden	Random OIDs biased to upper half of space (first bit always set). Does not appear in routing tables of any other peers.
France	One fixed OID, relatively passive crawler, appears to just be sampling Storm.
East Coast, US	257 OIDs evenly distributed in ID space behind one IP, port number used as upper two bytes of the OID.
East Coast, US	Uniform random OIDs, both a Storm implementation and crawler behind the same IP, does not report other peers.
West Coast, US	Random OIDs biased to upper half of space 100:1. Does not report IPs in response to queries.

Table 2: Other parties participating in the "encrypted" Storm network on April 4, 2008.

Storm nodes, not all of the nodes that Stormdrain discovers are from hosts infected and under control of the Storm botnet. We use four heuristics to identify masquerading Storm nodes.

First, Stormdrain queries all nodes it learns about. If a node never responds to a query, Stormdrain removes it from further consideration.

Second, we identify and remove peers which suspiciously send information about too many peers into the network too rapidly. The implementation of the Kademlia algorithm that Storm uses is not particularly aggressive. After the bootstrap phase, it does not learn about new peers very rapidly. As the crawler nears transitive closure and goes into steady-state, new peers should only be learned at approximately the rate at which new peers come online. We do not observe valid nodes disseminating information about many new nodes far above the anticipated rate, so any node that does is flagged as a flooder and removed from our peer list.

Third, if a peer reports more than a small proportion of bogon addresses (RFC 1918, Multicast, IANA Reserved, etc.), Stormdrain removes it from consideration. The cutoff proportion is strictly above zero, though, as Storm nodes do not perform any sanity or reachability checks before inserting a peer into its own peer list, and bogon addresses routinely show up in communication with valid peers.

Finally, after we determined that the Storm PRNG would produce only a small set of node OIDs, Stormdrain now marks any peer as non-Storm whose OID is not in that set. As discussed in Section 4.1, these OIDs allow us to validate Stormdrain's heuristics of which nodes are actual Storm hosts. The "Active" curve in Figure 2 is the estimate based upon the above heuristics, and the "Active Storm" curve includes just the set of active nodes with valid Storm OIDs. The proximity of these curves demonstrates how well the heuristics work.

In addition, given the set of rejected nodes we can trace their IP addresses to determine where in the network they originate. Further, based upon observing the traffic generated by those nodes, we can roughly characterize their behavior as part of the Storm overlay. Table 2 lists some of the other parties we observed participating on the Storm network, where in the Internet they originate, and observations of their behavior. Note that their presence can skew estimates of the size of the Storm botnet far more than the number of OIDs they use, particularly if they flood the network with fake node IPs (and they do). The Storm authors also have an Overnet presence used to facilitate the proper functioning of bootstrapping botnet command and control via Overnet.

Just as Stormdrain has to take into account the presence of other active participants interacting with the Storm network, Stormdrain itself has an effect on the network, too. Stormdrain selects a random OID on every query and response that it sends, and these OIDs will enter the routing tables of other nodes participating in the Storm Overnet that Stormdrain interacts with directly or indirectly. As such, Stormdrain will appear to be millions of bots with random OIDs over time. Although the impact might appear substantial, other participants tracking nodes in the network can easily identify Stormdrain as an invalid node. Stormdrain does not spoof its source IP, and manual inspection of the source of such hashes would identify Stormdrain as an incorrectly functioning node. In fact, another instance of Stormdrain on the current network would be quickly removed for flooding OIDs and would not have an impact on our estimate of the network size.

Cause	Number	Fraction
Invalid IP	49	0.02%
Dead	54,020	24.8%
ICMP Error	61,853	28.4%
Unresponsive	101,812	46.8%
Total	217,734	100%

Table 3: Number of peers removed by Stormdrain and the cause of removal during a representative 2 hour interval on February 20, 2008.

4.4 Temporal dynamics

The active node population of Storm is constantly changing over time: hosts arrive and leave continuously. Learning about new hosts is straightforward. Stormdrain actively crawls the routing tables of peers in the network, and continuously receives unsolicited messages advertising hosts. However, when tracking the active population of bots, it is crucial to also track when nodes become inactive.

Peers can become inactive, or dead, for a wide variety of reasons: they can be NATted nodes behind silent drop firewalls, previously valid peers which have been cleaned or shut down, currently active peers which have been restarted and have chosen a different port for communication, or invalid addresses sent by a buggy or malicious poisoning node. In a similar vein, nodes which had previously responded to Stormdrain's probes and fell silent could be due to rebooted or cleaned machines. It is also common that a NAT device in front of the peer has lost the mapping for communication between the infected machine and Stormdrain.

The Storm Overnet implementation includes no method of removing bad hosts from its internal peer list. As a result, Stormdrain has to determine when peers expire and much of the development effort put into Stormdrain has been to improve the accuracy of pruning dead nodes from the list of active nodes. Stormdrain repeatedly sends messages to its list of known nodes and tracks which nodes respond. The data structure for connected peers includes a counter of unanswered messages for every peer. Peers which have never replied are removed after a handful of unanswered messages, whereas peers which had previously responded and were considered connected and in good standing are given more leeway in terms of missed replies. If messages sent to a peer result in more than a few ICMP errors of any sort, Stormdrain considers the host unlikely to be able to communicate and discards it. Table 3 shows the number of nodes removed by Stormdrain over a representative two-hour time period and the reason why Stormdrain removed the nodes.

4.5 Bridging middleboxes

While the removal of false positives like poisoned data and stale mappings is an important facet of gaining an accurate view of the network population, ensuring that the crawler does not remove valid peers is equally important. The problem is that valid peers may not be able to always receive and, hence, respond to probes from Stormdrain. Lacking responses to probes, Stormdrain may then decide that those peers are dead when in fact they are still active participants in the botnet. While we do not have any evidence that valid peers will refuse to respond to valid queries, NATs and firewalls can refuse to forward our traffic to nodes which are behind them. This dropping can either be because the node has never contacted us before, or the mapping between the external port and the internal node has expired and communication is no longer possible.

When a node, even one which is not responding, is added to the Stormdrain state machine, we keep track of how many times this node has been advertised to us, and whether it has been advertised by an active node. If a node is behind a NAT and has not been informed of the presence of our crawler by one of its peers, we will be informed of its existence by active nodes but will not be able to communicate with it directly. Conversely, if a node is behind a NAT which times out UDP ports rapidly, it will be continually reported to us as active, but our crawler's mapping in the NAT table might be lost and, with it, our ability to communicate with this node.

Although second-hand knowledge of NATted nodes is helpful, if we were able to convince these nodes to initiate contact (for the former case) or continually contact Stormdrain (in the latter case), we can form a much more informed estimate of which nodes are behind such NATs. For the latter case of rapidly remapping NAT devices, the problem is that nodes do not query Stormdrain often enough to maintain the connection within the device's NAT table. One key deficiency of earlier versions of Stormdrain in this case is that, while it sent queries and listened for responses, it did not send responses of its own to nodes which query it. Having Stormdrain respond to nodes which query it refreshes NAT and firewall timeouts, thereby enabling Stormdrain to probe and track those nodes more reliably.

For the former case, we devised another heuristic to elicit new connections from nodes with which we have never had any direct contact. This heuristic takes advantage of semantics of Storm node behavior. Over time, we found that certain regions in the ID space (with locality defined by the XOR metric) would become very popular for a short period of time. To make it more likely that NATted nodes contact Stormdrain, whenever Stormdrain noticed these popular hashes it chose for itself an OID in that region while crawling the network.

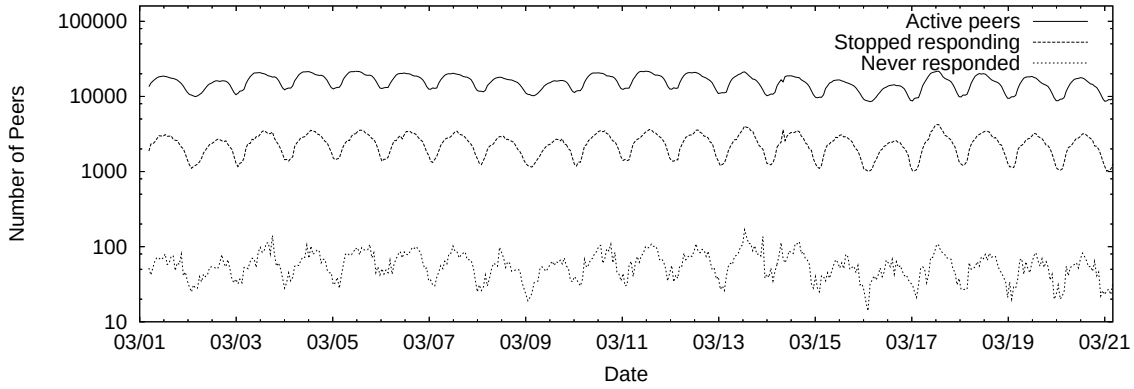

Figure 4: Upper bound error estimates of the number of NATted nodes with which Stormdrain could not communicate consistently over the first three weeks of March 2008. Note that the y-axis is log scale.

These two techniques had a noticeable improvement on Stormdrain's ability to track NATted nodes. Table 4 shows the effect of implementing these techniques and deploying them on January 28, 2008. The table shows two columns, each counting the number of nodes in the live, active, and dead states, and the number of nodes that never responded. The first column shows counts for the day before deploying the techniques, and the second column for the day after. These techniques reduced the number of unresponsive nodes and live nodes that failed to promote to active, overall increasing the fraction of active nodes by 9%.

Even with these heuristics, there could still be errors in Stormdrain's estimates of active nodes. To estimate the potential magnitude of this error, we can use counts of the number of nodes that stopped responding and never responded as upper bounds on the error — note, though, there are many reasons why nodes stop responding or never respond.

Figure 4 shows the number of active Storm nodes from Figure 2 together with the number of nodes that stopped responding and never responded for the first three weeks of March 2008. These measurements are of the encrypted network only, and exclude the effects of flooders advertising fraudulent IP addresses as identifiers (Section 4.3). Also note that the y-axis is in log scale. As an upper bound, Stormdrain could potentially lose track of thousands of nodes behind NATs that had been responding but stopped. Our experience, however, is that any error is much less than this number since it has to include nodes that actually leave the network in that time period. The number of nodes that never respond is much smaller. Even if all of these nodes were actual Storm nodes that Stormdrain could not communicate with, the error estimate of the number of active nodes would be less than 1%.

	Before		After	
Live	30,567	(49.6%)	22,963	(34.3%)
Active	22,833	(37.1%)	37472	(55.9%)
Dead	7,915	(12.8%)	6403	(9.56%)
Never resp.	283	(.459%)	162	(.242%)

Table 4: The number of nodes in the live, active, and dead states in Stormdrain one day before and one day after deploying the query responsiveness and popular hash locality heuristics on January 28, 2008.

5 Conclusion

In this paper, we have described our experience trying to measure the membership dynamics of the Storm botnet. What seemed an easy task, given Storm's use of a well-documented protocol (Overnet), was in fact significantly complicated by a range of factors including innocent applications using the same protocol, the impact of address translation, and the challenges presented by active network spoofing and poisoning by outside parties (such as other research groups). We have characterized these problems in this context and described the set of heuristics and accommodations we have developed to maintain accurate estimates in spite of their influence. However, the entire endeavor begs the question, "How can one be sure?", in such an environment. It seems likely that the same kinds of problems we have encountered will continue to arise with new botnets, although they will undoubtedly have their own intricacies and complexities. Unfortunately, it seems overly optimistic to hope that the techniques we have developed will prove universally applicable and thus our community should have considerable skepticism about the veracity of botnet measurement results going forwards.

Considering this we offer a number of suggestions, ranging from the grounded and pragmatic to the wildly

naive and optimistic. First, when evaluating botnets using existing protocols for their organization or command and control, researchers should be required to explain how they differentiate between bots and aliasing from other protocol users. Simply claiming that the botnet does not overlap with other applications is not sufficient since this can happen both intentionally (from the botmaster using the existing infrastructure to jump start their own) or inadvertently (via aliasing from NAT or simply a bot host that also happens to have the application on it).

Second, researchers should be expected to document the key assumptions underlying their measurements. Several of these assumptions (e.g., bot ID persistence, bot ID uniqueness, IP address persistence and IP address uniqueness) should be considered fundamentally suspect in the same way that assumptions of Poisson packet arrival times trigger eye rolls in the networking community. Only when such assumptions are independently validated should results based on them be permissible.

Finally, care must be taken not to inadvertently measure the activities of other researchers. Addressing this problem is perplexing because there is nothing fundamental that allows one to make this determination; a sufficiently sophisticated researcher can design software that is indistinguishable from a bot at the network layer or simply infect a large number of honeypots with the bot itself and manipulate their behavior. Conversely, a bot herder wishing to conceal their activities might make subsets of their bots behave anomalously and thereby be mistaken for other researchers (and hence not be measured). In the end we have no perfect proposal. Life was decidedly easier when one could study malware in isolation. However, since network-persistent malware like today's botnets seems likely to stay, it behooves us to be aware of these problems and do our best to manage them.

6 Acknowledgments

Our thanks to Vern Paxson and Christian Kreibich for detailed discussions and feedback on investigating the Storm botnet, Joe Stewart of Secureworks for offering his insight into the workings of Storm, Erin Kenneally for advising us on legal issues, and to Gabriel Lawrence and Jim Madden for supporting this activity on UCSD's systems and networks. Our data collection was made possible by generous support from Cisco, Microsoft Research, Intel and UCSD's Center for Networked Systems, for which we are very grateful. This work was made possible by the National Science Foundation grant NSF-0433668. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors or originators and do not necessarily reflect the views of the National Science Foundation.

References

[1] S. M. Bellovin. A Technique for Counting NATted Hosts. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet Measurment*, pages 267–272, November 2002.

[2] R. Bhagwan, S. Savage, and G. M. Voelker. Understanding Availability. In *Proceedings of the International Workshop on Peer To Peer Systems (IPTPS)*, pages 256–267, Berkeley, CA, February 2003.

[3] M. Casado and M. J. Freedman. Peering Through the Shroud: The Effect of Edge Opacity on IP-Based Client Identification. In *Proceedings of the 4th USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI'07)*, April 2007.

[4] J. Grizzard, V. Sharma, C. Nunnery, B. Kang, and D. Dagon. Peer-to-Peer Botnets: Overview and Case Study. In *Proceedings of the First USENIX Workshop on Hot Topics in Understanding Botnets*, April 2007.

[5] C. Kreibich, C. Kanich, K. Levchenko, B. Enright, G. M. Voelker, V. Paxson, and S. Savage. On the Spam Campaign Trail. In *First USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET'08)*, April 2008.

[6] P. Maymounkov and D. Mazières. Kademlia: A peer-to-peer information system based on the XOR metric. In *Proceedings of the 1st International Workshop on Peer-to-peer Systems*, 2002.

[7] R. Pang, V. Yegneswaran, P. Barford, V. Paxson, and L. Peterson. Characteristics of Internet Background Radiation. In *Proceedings of the ACM Internet Measurement Conference (IMC 2004)*, October 2004.

[8] P. Porras, H. Saïdi, and V. Yegneswaran. A Multi-perspective Analysis of the Storm (Peacomm) Worm. Technical report, Computer Science Laboratory, SRI International, October 2007.

[9] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in C*. Cambridge University Press, second edition, 1992.

[10] M. Rajab, J. Zarfoss, F. Monrose, and A. Terzis. My Botnet is Bigger than Yours (Maybe, Better than Yours). In *Proceedings of the First USENIX Workshop on Hot Topics in Understanding Botnets*, April 2007.

[11] S. Sarat and A. Terzis. Measuring the Storm Worm Network. Technical Report HiNRG 01-10-2007, Johns Hopkins University, 2007.

[12] M. Steiner, T. En Najjary, and E. W. Biersack. Exploiting KAD: possible uses and misuses. *Computer communications review, Volume 37 No 5, October 2007*, 2007.