Netbait: a Distributed Worm Detection Service

Brent N. Chun
Intel Research Berkeley
2150 Shattuck Ave. Suite 1300
Berkeley, CA 94704
bnc@intel-research.net

Jason Lee and Hakim Weatherspoon
University of California at Berkeley
Computer Science Division
Berkeley, CA 94720
{jlee81,hweather}@cs.berkeley.edu

ABSTRACT

This paper presents Netbait, a planetary-scale service for distributed detection of Internet worms. Netbait allows users to pose queries that identify which machines on a given network have been compromised based on the collective view of a geographically distributed set of machines. It is based on a distributed query processing architecture that evaluates queries expressed using a subset of SQL against a single logical database table. This single logical table is realized using a distributed set of relational databases, each populated by local intrusion detection systems running on Netbait server nodes. For speed, queries in Netbait are processed in parallel by distributing them over dynamically constructed query processing trees built over Tapestry, a distributed object and location routing (DOLR) layer. For efficiency, query results are compressed using application-specific aggregation and compact encodings.

We have implemented a prototype system based on a simplified version of the architecture and have deployed it on 90 nodes of the PlanetLab testbed at 42 sites spread across three continents. The system has been continuously running for over a month now and has been collecting probe information from machines compromised by both the Code Red and Nimda worms. Early results based on this data are promising. First, we observe that by having multiple machines sharing probe information from infected machines, we can identify a substantially larger set of infected hosts that would be possible otherwise. Second, we also observe that by having multiple *viewpoints* of the network, Netbait is able to identify compromised machines that otherwise would have been difficult to detect in cases where worms have an affinity to certain regions of the IP address space.

1. INTRODUCTION

With hundreds of millions of machines now widely connected to the Internet and the disturbing number of remote system vulnerabilities being discovered on a routine basis, the consequences of an Internet worm epidemic today are profound. Large-scale epidemics caused by the unleashing of recent Internet worms such as Code Red [4, 6] and Nimda [5] have resulted in millions of host computer infections and over a billion dollars in financial loss. Cleanup, monitoring, and identification of Code Red infected systems alone have resulted in costs approaching $740 million dollars. In recent years, Internet worms capable of self-replication [26, 27] have been appearing with alarmingly regularity. Recent work characterizing the behavior of hypothetical Flash worms [28] makes this even more troublesome given the phenomenal infection rates Flash worms might be capable of achieving in practice.

In response to recent Internet worm attacks, there have been essentially three classes of countermeasures that have been proposed: prevention, containment, and cleanup. Prevention, while highly desirable, is still an active area of research. Part of the complexity here is coping with the massive amount of hardware and software heterogeneity found in modern computer systems. Containment, on the other hand, is a widely used technique which has been applied in response to virtually all recent epidemics. Content-based network filtering and the installation of IP address blacklists which block network traffic from infected hosts are commonplace. Cleanup is also a necessary and widely practiced technique. In the aftermath of a worm epidemic, infected machines inevitably need to be identified and purged of their underlying vulnerabilities (e.g., by applying the appropriate patches). In performing both containment and cleanup, the key problem that must first be addressed is how to identify which hosts have been infected in the first place.

This paper presents a new approach to addressing the problem of how to identify hosts in a network which have been compromised by an Internet worm. The basic idea underlying our approach is that by viewing the Internet from multiple geographically distributed vantage points and by sharing this information, we can obtain a more complete view of the extent and nature of Internet worm epidemics. We leverage the fact that worms typically replicate themselves through remote system exploits which have well-known network signatures that are easily detected by modern intrusion detection systems (IDSs). We use a set of geographically distributed machines to collect probe information using local IDSs then share that information by building an efficient distributed query processing system which exports the collective historical views of all the nodes in the system. In response to an epidemic, ISPs and network administrators might then use our system to automatically build blacklists and to identify infected machines which require cleanup to prevent further infections. Researchers could obviously also stand to benefit greatly from use of such a system.

The rest of this paper is organized as follows. In Section 2, we describe the opportunities afforded by sharing data amongst geographically distributed sets of machines to perform distributed detection of Internet worms. In Section 3, we present an architecture that realizes this sharing using a distributed query processing architecture. In Section 4, we describe a prototype implementation of a distributed worm detection system which has been deployed on 90 nodes of the PlanetLab testbed [21] and collecting data for over a month now. Results, analysis, and implications from this deployment are then presented in Section 5. In Section 6, we present related work and finally in Section 7 we conclude the paper and describe next steps for this work.

2. DISTRIBUTED WORM DETECTION

One common technique for detecting remote machines that have been infected by an Internet worm is passive detection of probe attempts. In order to replicate themselves on additional machines, Internet worms rely on two mechanisms: probing of remote machines and exploitation of well-known remote system vulnerabilities. Code Red, for example, exploits a buffer overrun bug in Microsoft's IIS web server to compromise and infect a target machine. It exploits this bug by sending IIS a request for a URL with a specific signature which causes a buffer overrun. Modern intrusion detection systems such as Snort [24] are capable of detecting these types of exploits at the packet level by doing real-time traffic analysis and packet logging of IP traffic. IDSs, more generally, are capable of performing a variety of tasks which include protocol analysis, content searching/matching, and detection of a wide assortment of attacks and probes, such as buffer overflows, stealth port scans, CGI attacks, and SMB probes. For our purposes, the key capability we are concerned with is real-time detection of probes from machines infected by an Internet worm.

One important aspect of an Internet worm's replication algorithm is how it selects target machines to be infected. In the case of Code Red v1, target machines are selected using a random IP address. Unfortunately, due to its use of a static seed in its pseudorandom number generator [17], Code Red v1 actually selected target IP addresses in precisely the same sequence by all infected machines. Code Red v2, released shortly thereafter, corrected this by using a better seed and subsequently achieved much greater infection rates. Besides Code Red variants, there is another class of worms which use more sophisticated machinery for selecting potential victims. Nimda, for example, uses the following strategy. 50% of the time an address with the same first two octets will be chosen, 25% of the time an address with the same first octet will be chosen, and 25% of the time, a random address will be chosen. Worms which have affinity for "nearby" nodes, both in geography and the IP address space, have the potential to achieve much higher infection rates due to the performance benefits of network locality.

The aggregate coverage of the IP address space a collective worm population is able to achieve during an epidemic can be substantial. One of the key challenges faced by containment strategies and post-mortem cleanup following a worm epidemic is how to identify these infected machines. One promising approach for addressing this problem is to use the collective view of a geographically distributed set of machines running intrusion detection systems, each of which detects probe attempts from a different network viewpoint. Given a large enough number of nodes, such a scheme should allow for much greater coverage of the worm population since more probes will be received. On the other hand, simply having many hosts is not enough. During a worm epidemic, ISPs often install filtering at various places in their network. Furthermore, worms which exploit locality in geographical and network distance may infect a substantial number of machines locally before leaving its local domain to infect other machines. In both of these cases, having multiple machines is not enough; what is needed is multiple geographically distributed machines, each situated at a different network vantage point.

3. NETBAIT DESIGN

Netbait is a planetary-scale service for distributed detection of Internet worms. It allows users to pose queries that identify which machines on a given network have been compromised by Internet worms and processes these queries using a distributed query processing architecture (Figure 1) in a timely and resource efficient manner. It uses aggregate information as collected by intrusion detection systems on a geographically dispersed set of cooperating machines to obtain greater global knowledge of the extent of an Internet worm epidemic. By sharing probe information across a large number of machines, it can identify a substantially larger set of infected hosts that would be possible otherwise. In addition, by using multiple *viewpoints* of the network, it is able to identify compromised machines that otherwise would be difficult to detect when worms have an affinity to certain regions of the IP address space.

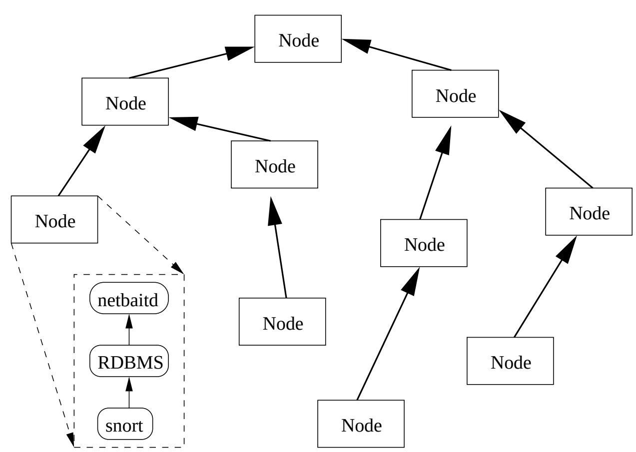


Figure 1: Netbait architecture.

At its core, Netbait is essentially a distributed query processing system on intrusion detection system data. Queries are expressed using a subset of SQL and evaluated against a single logical database table. This single logical table is physically realized as a set of tables in a distributed set of relational databases, each populated by intrusion detection systems running on a geographically dispersed set of cooperating machines. For speed, Netbait queries are processed in parallel by distributing them over dynamically constructed query processing trees built over a Tapestry DOLR [32]. For efficiency, query results are compressed using application-specific aggregation and compact encodings. Clients are load balanced across the roots of multiple overlay trees in order to avoid overloading a single root. Our performance goals are to be able to support host infection queries on a 100,000 node system on the order of a few minutes.

3.1 Data Collection and Indexing

Each node in Netbait collects probe information from infected machines using an intrusion detection system. We use the intrusion detection system to essentially observe all requests for network services on a machine and to compare the signatures of those requests against exploits on well-known vulnerabilities and log any matches. Matches are then collected by inserting them into a local relational database and indexing the results. In the proposed implementation, we intend to use Snort, a well-known open-source intrusion detection system to generate the probe information and the PostgreSQL relational database to store and index the data. For Snort, we will install rules that match signatures for all currently known exploits for all Internet worms. As new worms are discovered, we can collect two types of data. First, before the signature of the new worm is detected, we can log connection attempts. If the Internet is experiencing an epidemic, the number of such connection attempts should be very high and thus should be distinguishable from routine port scans from random hackers. Second, once the signature is known, Snort rules that match these signatures are produced fairly quickly by the open-source community. Once the new rules are obtained, they simply need to be pushed out to all the nodes. To make this distribution fast, we exploit parallelism by using an overlay tree constructed by Tapestry to push new rules out and install them on local nodes.

3.2 Overlay Construction and Maintenance

In this section, we discuss how we route the queries to the nodes and return the results in a timely and efficient manner. Specifically, we use a decentralized object and location routing (DOLR) to provide fault-tolerance, self-organization and locality.

3.2.1 Query Processing Trees

Application-level multicast research has focused on resolving issues in scalability and reliability. Such an infrastructure must clearly scale to handle our targeted performance goal of a 100,000 node system and offer probabilistic guarantees of query delivery to every participating node. Systems based on epidemic-style protocols [1, 8] have accomplished both goals, but with the assumption that duplication of messages is acceptable overhead. However, Netbait not only needs to dispatch a root's queries to all other participating nodes but requires the ability to aggregate these results back to the root. In this latter case, redundancy of aggregation information could be prohibitively expensive for certain types of queries (Section 3.3.1).

Hence we are interested in a spanning tree structure that allows both the multicasting of queries and the collection of results. The organization must reflect appropriate trade-offs between fanout and depth to achieve efficient use of bandwidth and low latency. In order to address the issues in scalability and reliability, we exploit an overlay substrate, either a Distributed Hash Table (DHT) or a DOLR, for their adaptability and robustness to construct the spanning trees at Internet scale [13]. SCRIBE [2] and Bayeux [33] have also utilized overlays to create multicast systems but they are not currently available services.

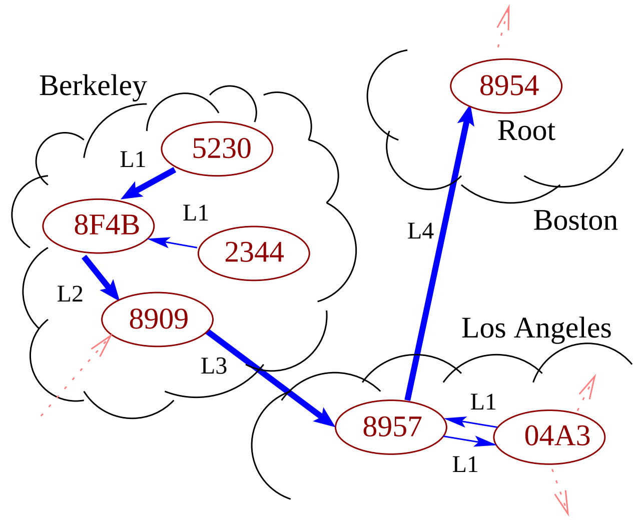


Figure 2: Tapestry. Nodes are connected to other nodes via neighbor links (→). A node can route to any other node by resolving one digit at a time, e.g. 5230 ⟹ 8F4B ⟹ 8957 ⟹ 8954. Each unique id is associated with one particular *root* node (8954).

3.2.2 DOLR & DHT

Overlay substrates, DHTs (e.g., CAN [23], CHORD [29]) and DOLRs (e.g. Tapestry, Pastry [25]), offer the following three properties: fault-tolerance, self-organization and locality. Fault-tolerance should robustly handle the unexpected deletion of nodes and loss of these nodes' information during queries. The overlay must adapt to a continuously changing topology, e.g. insertion and deletion of nodes as well as changes in network latency, and self-organize accordingly. Finally, the query processing tree is bound by the constraints of bandwidth and network latencies, which in turn determine the resulting shape, i.e. the fanout and the depth, of the tree itself. Locality allows for efficient use of resources to construct such an optimal tree and adjust for efficient bandwidth and parallelization.

In particular, Tapestry, a wide-area DOLR substrate of OceanStore [14], provides a routing mesh upon which Netbait could construct its query processing trees. Each Tapestry node is assigned uniformly at random a unique 40-digit hexadecimal address, called a *node-ID*. These nodes are then connected in this overlay network via neighboring links to over node who share node-ID prefixes. A neighboring link's level is calculated as one plus the number of matching digits in the node-IDs.

Routing of messages is thus handled digit by digit over these neighbor links. For example, Figure 2 shows the path from node 5230 to node 8954. At every node, we find the neighbor whose node-ID matches the value of the destination node's next digit, and this process is repeated digit by digit. The resulting path for this example would be 8*** $\rightarrow$ 89** $\rightarrow$ 895* $\rightarrow$ 8954, where *'s represent wildcards. This prefix-routing mechanism has been inherited from the original work by Plaxton, Rajaraman and Richa [22]. In other words, every destination node could be considered the root node of its own unique spanning tree. This routing mechanism guarantees, by the reachability property [12], that every leaf node can route to the root via the appropriate branches in the spanning tree.

3.2.3 Tree Construction and Maintainence Algorithm

Tapestry has an affinity to choose neighbors that are close in terms of network latency. Each Tapestry node is the root of a unique spanning tree. We leverage both the natural locality and spanning tree properties of Tapestry to construct a unique query processing tree to disseminate and aggregate distributed queries. Specifically, Tapestry acheives its locality probabilistically from the number of nodes that match a given prefix. That is, the number of nodes that match a given prefix decreases geometrically with the number of digits in the prefix being matched. The probability of matching d digits is $1/(base^d)$, e.g. 1/16 for matching one digit with an address base of 16. Therefore for a given node, level-1 links connected it to its *closest*, in terms of network latency, 16 neighbors. The highest possible level labels the neighboring links that connect a node to its furthest neighbors. (Tapestry currently utilizes network latency in these calculations for closest neighbors because of the strong correlation between bandwidth and round trip time [20].)

We then construct the tree with links near the root to be further apart than the branches near the leaves, which improves bandwidth and exploits aggregation for the long hauls to distant nodes. Once a node has been chosen as a root of a query processing tree, its neighboring node(s) with the greatest level are identified as possible children. The prospective children are then asked whether they are already participating in a tree with id matching the root and to confirm with a (N) ACK if they accept this new parent. This process is repeated for these confirmed children until no such child exists that is not already participating in this tree. As the construction approaches the leaves, the only possible level for neighboring links will be the Level-1 links, thus assuring localized organization of the leaves. Moreover, this construction supports the participation of nodes in multiple trees and efficiently delivers the queries to the leaves and results to the root. Finally, this construction is probabilistically guaranteed to efficiently cover all the nodes in the Netbait network.

Nodes can dynamically insert and delete themselves from these query processing trees. If a given node chooses to depart, even unexpectedly, server heartbeats [31] will detect the failure at its children and its parent. The parent will remove this deleted node from its list of children, and every orphaned child will use its redundant backup pointers to find another parent whose link level matches the missing node. The orphans will then ask these nodes to become their children, and by the reachability property of Tapestry, there will always be a path from these orphaned children to the root via their new parent. A node insertion alerts the new neighbors who will then contact this new node and ask for it to be a child using the same scheme as in the construction of the tree itself. (After the adoption of this new node, other nodes can then decide whether to change parents if this node has a lower network latency, higher bandwidth, etc.)

3.2.4 Optimizations

Although we currently construct the query processing trees with an emphasis on localized leaves, we are exploring other configurations. Each node can determine a minimum and maximum limit for the number of children Sending data down the tree is obviously a different request than aggregating the results; we would like to determine if two specialized trees (i.e. one serving multicast and the other pushing the data back to the root) would improve bandwidth and latency performances. Finally, we would like to incorporate some of the features, e.g. limited membership, adaptation of some gossip paradigms [10] for propagating the multicast message, etc., found in other multicast systems, such as SCRIBE, Bayeux, HiScamp [9], etc.

3.3 Distributed Query Processing

Netbait uses distributed query processing to evaluate queries on host infection data. Probe data is stored and indexed locally on the nodes and queries are pushed out to the nodes for evaluation. The key reason for adopting a distributed query processing scheme is scalability and response time. One of the goals of Netbait is to ultimately support systems comprised of thousands of Netbait servers and to do this in a resource efficient manner. In the common case, queries will mainly be set membership queries. For example, an ISP such as AT&T might wish to find which of the 16,777,216 hosts in their 12.0.0.8/8 network are infected with any of the known worms (e.g., Nimda, Code Red, Code Red II, etc.). This common case of membership queries on a particular set of network addresses can be heavily optimized. First, since all nodes have their own local database of probe information, the query can be evaluated in parallel as all of the nodes execute the query against probe data stored in their local relational database. Second, since the common case is a membership query, the space complexity of a query result can substantially reduced by aggregating the results of a set of nodes by simply taking the union of each node's membership matches.

3.3.1 Target Workload

In Netbait, we envision two classes of queries in our workload. Each of these queries has different requirements in terms of response time and is targeted to a different class of users. The first type of query is a query that might be issued by an ISP or network administrator of an organization (e.g., a university). These queries are expected to be membership queries against a range of IP addresses over some time interval against some set of worms, which often might simply be all the worms we know about. These queries require fast response times. For example, the results of such a query might be used by an ISP to help construct an address blacklist to filter out traffic from infected nodes to mitigate the spread of a worm epidemic. Another common use we envision is using these queries to discover which nodes are infected within an organization so those nodes can be cleaned up by a system administrator, potentially in some automated fashion.

The second class of queries we expect Netbait to see are queries from researchers (and potentially ISPs) who are interested in analyzing the spread of a worm epidemic. These types of queries require gathering more data than simply membership queries. More likely, query results for these queries will need to return detailed information on each probe, including the time that each probe occurred. Time information might be used, for instance, to correlate probes from different targets in an attempt to reverse engineer how a new worm is selecting target IP addressees for additional hosts to infect. Another possible use for these types of queries is to study how a worm propagates over time in relation to geographical extent by mapping IP addresses to longitude and latitude coordinates and by studying where probes appear over time in relation to these coordinates. These types of detailed queries do not require real time response. Netbait supports these types of queries, but its main performance optimizations are focused primarily on the first class of queries.

3.3.2 One Logical Table

Queries in Netbait are evaluated against a single logical database table. The attributes in this table include data such as the time of the probe (UTC), the probe host IP's addresses, the target host's IP address (i.e., the PlanetLab node), and the worm whose signature the probe matched. Each node has a local relational database that contains a table with these attributes. When processing a query in Netbait, we use a tree formed by Tapestry to distribute the query to all the nodes. Each node then executes the query locally against its database. Finally, the results are aggregated up the tree starting from the leaves and working their way back up to the root of the tree and eventually to the client. Aggregation at each stage of tree involves performing applications-specific optimizations and using compact encodings of query results. These optimizations are further described in Section 3.3.4.

3.3.3 Load Balancing

Netbait uses multiple query distribution trees and load balances client requests to different trees to avoid having a central bottleneck in the system. To do this, Netbait builds multiple trees on top of Tapestry and constructs the trees such that the roots of the trees are geographically distributed around the world. Clients then direct their requests to a particular tree via Tapestry. As shown in Figure 3, the root of such a tree, labeled as the "Netbait root", advertises its location and service by sending a publish message to the "Tapestry root." (Because this Tapestry root is a single point of failure, Tapestry publishes to multiple roots for fault-tolerance [32].) During the routing of this publish message, intermediate nodes store a backpointer to the Netbait root. Clients would then route towards the Tapestry root for the desired service, and if they encounter such a backpointer, would immediately be directed towards the service provider.

In another approach, clients could instead choose a tree based on external means. One possibility is to use DNS

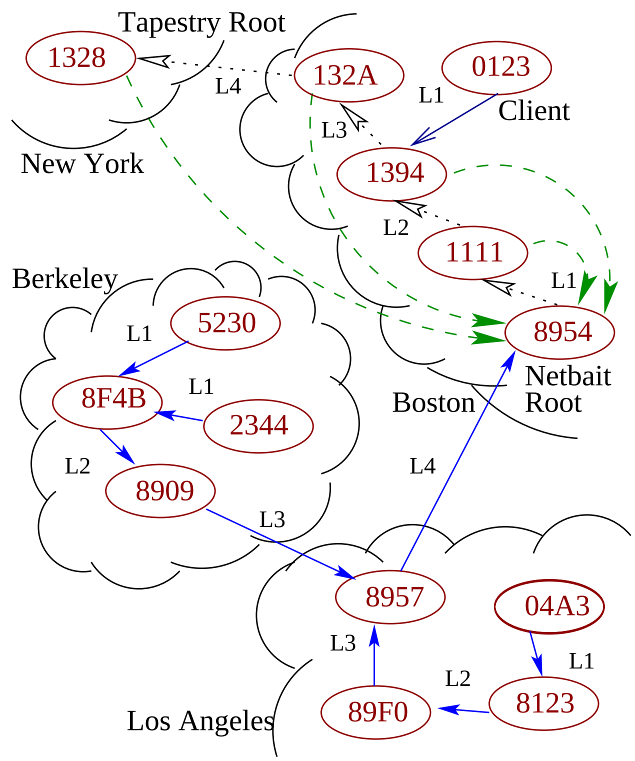


Figure 3: Tapestry and Netbait. A server/service provider (e.g. "Netbait root) *publishes* the location of the service by sending a message towards the "Tapestry root", leaving backpointers (dotted arrows) at each hop. Clients locate a service by sending a message towards the tapestry root until they encounter a pointer. The client 0123 can locate a Netbait service provider after only one hop in this example: 0123 → 1394.

round robin, dynamically update the set of root IP addresses, and have DNS return a different IP address each time. Another approach might be to leverage the Ganglia distributed monitoring system [16], currently deployed on PlanetLab, to publish IP addresses and port numbers. In both of these cases, the choice of a root still does not involve performance considerations between the client and the root of the tree. In the case of Ganglia, for example, it would be prudent to query some external network monitoring service in order to estimate the quality of the path from the client to each of the roots when choosing which tree to choose from. The load on the particular root might be another consideration here. That information could be obtained from Ganglia, which already publishes a variety of metrics including CPU load, available memory, and so forth.

3.3.4 Aggregation and Encoding

One of the keys to scalability and processing of distributed queries in an efficient manner is aggregation and compact encoding of query results as they are propagated back up the tree to the root. In the common case of a membership query, consider a client which issues the following query:

```
SELECT distinct(probeIP) LIKE '12.0.%.%'
```

```
WHERE time > '2003-01-01' AND
      time < '2003-01-30' AND
      (worm = 'Code Red' OR worm = 'Nimda')
```

In this query, the client is requesting all IP addresses in the 12.0.0.0/16 network that appear to be infected with either Code Red or Nimda as determined by probes which were recorded by one or more PlanetLab nodes during January 2003. This is essentially a membership query on the set of infected machines for all IP addresses on the 12.0.0.0/16 network. The query result is simply a list of IP addresses of infected machines. Since the query in this case is just a membership query, one significant optimization that we can use is aggregation. For each node in the tree, we take the union of the node's results and its children's results and pass the aggregated result back up the tree. Furthermore, since the address list consists of IPV4 addresses and since we know that this list is likely to be sparse in many cases, there are additional encoding optimizations we can use to obtain further savings in representing query results.

One simple way of representing the query results is to simply to return a sorted list of 32-bit integers, each of which represents an infected IP address. For large networks (e.g., the entire 2^{32} potential hosts on the Internet), the extent of infections even during a severe worm epidemic is likely to be small relative to the entire IP address space. For example, Code Red during its peak infected 359,000 computers on the Internet. Relative to the entire IP address space this is only 0.008% of the total number of IP address. Returning a query result simply as a list of IP addresses in 32-bit binary form would have required only 1.37 MB of data. Furthermore, given that infected IP addresses are not completely randomly distributed and the fact that usable IP addresses are not uniformly spread across the range of addresses from 0.0.0.0 to 255.255.255.255, compressing the results would add further savings.

Future Internet worms such as hypothetical Flash worms are capable of achieving significantly greater penetration into the Internet host population due to its clever use of techniques to rapidly spread an epidemic. Should such an epidemic arise, the number of infected hosts in the aftermath could be enormous and vastly exceed the 359,000 infected hosts that were observed as a result of Code Red, by comparison a relatively simple worm. To be able to handle these types of future cases, Netbait could use other encoding tricks. For example, rather than representing each infected machine with 32-bits, we might simply encode the "distances" between infected machines in the IP addresses space and leverage the fact that we will usually require much less than 32 bits. Another promising approach is to represent the set of infected machines as a trie, a data structure popular in the information retrieval and IP routing communities.

Using either a simple array of binary 32-bit IP addresses or the distances between IP addresses of infected machines results in reasonable time and space complexity. Each node in the query processing tree essentially performs the following tasks. The node first executes the query against its local database of host infection information. This data is then sorted and merged with the data collected from the node's children in the query processing tree and passed up the tree towards the root. Let n be the number of Netbait servers in a k-ary query processing tree. Let m be the number of infected hosts on some network of IP addresses. The worst case, per-node time complexities are then $O(m \cdot lg(m))$ to sort the list of infected machines and $O(km)$ to merge the results from the sort and the results from the node's k children. Since merging must be done at each level of the tree, the overall time complexity for these steps, excluding local database queries (which should be fast given proper indexes), is thus $O(lg_k(n) \cdot (k \cdot m + m \cdot lg(m))$.

4. PROTOTYPE

We have implemented a prototype of the Netbait system based on a simplified version of our architecture. The primary goals of the prototype were to gain some early experience with building and deploying a distributed worm detection system and to gather some initial data to try and assess the efficacy of sharing probe information across multiple machines distributed across the wide-area. The prototype system (Figure 4) consists of a `netbaitd` daemon that runs on each node of the PlanetLab testbed and a central aggregator daemon that collects the data and indexes it. Each `netbaitd` daemon collects probe information from machines infected with either the Code Red or Nimda worms. Probes are detected through HTTP requests which carry payloads that match well-known signatures for Code Red and Nimda exploits via Microsoft IIS web server vulnerabilities. In our system, such requests are observed by running simple, multithreaded web servers that listen on TCP port 80 and process each incoming request in an identical fashion by returning a static page and logging the request. Logfiles generated by the web servers are filtered against a set of pattern matchers which determine whether a request corresponds to a probe from an infected machine. Matches are then collected by the `netbaitd` daemons and periodically aggregated using XML-RPCs from an aggregator daemon running on `www.planet-lab.org`. While our implementation currently only detects Code Red and Nimda probes based on web server logfiles, our prototype in principle is capable of supporting detection of arbitrary worms through pattern matching on logfiles generated by multiple services.

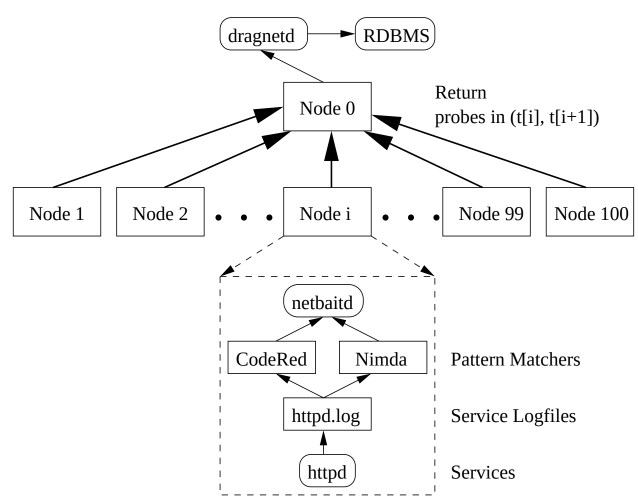


Figure 4: Netbait prototype.

5. RESULTS

In this section, we present an analysis of the data collected by the Netbait prototype over January 2003. The prototype has been running continuously for over a month now on approximately 90 nodes of the PlanetLab testbed (Figure 5). The nodes were located at 42 different sites and scattered across three continents: North America, Europe, and Australia. During the month of January, we observed a total of 4195 probes from 3495 distinct IP addresses corresponding to machines infected by either the Code Red or Nimda worms.

Figure 5: The PlanetLab testbed as of January 2003. The testbed currently consists of approximately 100 nodes at 42 sites scattered across three continents.

Figure 6a plots the total number of unique Code Red probes detected by Netbait on 90 PlanetLab nodes. Consistent with previous work [4], we observe Code Red's cyclic behavior with three distinct phases of scanning and replication (1st to the 19th), an attempted distributed denial of service (DDoS) attack on `www.whitehouse.gov` (20th to the 27th), and an indefinite sleep phase beginning on the 28th. Because no probes are attempted from the 20th of the month and on, we cannot directly observe the transition from the DDoS attack phase to the sleep phase. However, we can observe a rise in probes from infected machines at the beginning of the month. This resurgence following a lull of activity at the end of the previous month is due essentially to machines with bad clocks continuing to probe and replicate when they ought to be sleeping. The lack of global clock synchronization is basically what restarts the Code Red cycle and causes Code Red to perpetuate. On the 20th, for example, we observed two Code Red probes and on the 21st, we observed an additional probe.

Similar to Code Red, Figure 6b plots the total number of unique Nimda probes detected by Netbait as a whole running on 90 PlanetLab nodes. The Nimda worm [5] propagates through a variety of exploits including email, open network shares, client browsing of compromised web sites, Microsoft IIS vulnerabilities, and back doors previously created by the Code Red II [6] and sadmind/IIS [3] worms. In the results shown here, we only account for Nimda probes which attempt to exploit Microsoft IIS vulnerabilities. As with Code Red, we identify a probe by running a web server on each PlanetLab machine and by matching the URL requested against well-known probe signatures, here ones corresponding to Nimda IIS exploits. Unlike Code Red, the data shows that Nimda infected machines are probing PlanetLab constantly with an average of approximately 90 probes a day. This constant probing is consistent with Nimda's replication strategy, which does not exhibit a cyclical pattern as with Code Red.

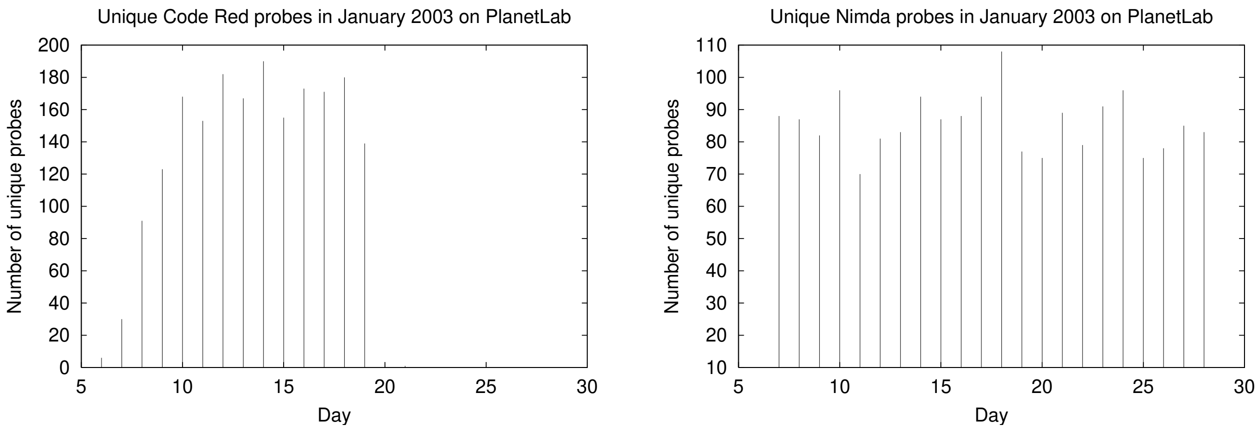
5.1 Benefits of Sharing

In Figures 7a and 7b, we show the distribution of unique Code Red and Nimda probes for each of the 90 PlanetLab machines running the Netbait server. The key observation here is that by having multiple machines sharing probe information, the system is capable of identifying a substantially larger set of infected hosts compared to a single machine. For Code Red, the benefits achieved through sharing of information are substantial. We observe that out of 1971 probes from Code Red machines spread across 90 PlanetLab machines, 1851 of those probes were from unique IP addresses. Having more machines in this case is clearly beneficial, especially given that Code Red v2 fixed the pseudo-random number generator seed problem in Code Red v1 and hence probes target machines more uniformly across the IP address space. At the same time, however, we observe that, despite a full three weeks of Code Red activity, there was very little overlap in the probes detected by different PlanetLab machines. We conjecture that filtering of Code Red traffic at ISPs and the effect of DHCP for ADSL and cable modem users may be a contributing factor here. For Nimda, the benefit of having multiple machines is also clear. In this case, however, we see that the probe attempts are much more localized compared to Code Red. We hypothesize this is due to a combination of Nimda's affinity to certain regions of the IP address space and post-epidemic ISP filtering of Nimda traffic.

5.2 Benefits of Multiple Viewpoints

In Figures 7c and 7d, we plot the distribution of unique Code Red and Nimda probes for each of the 42 PlanetLab sites. The key point of these graphs is that by observing an epidemic from multiple, geographically distributed vantage points, we can identify infected machines that would otherwise have been difficult to detect. One case where this advantage comes into play is in the detection of machines infected with worms that have affinity to certain regions of the IP address space (e.g., to speed the rate of infections). Another case is the detection of infected machines post-mortem after widespread filtering of worm replication traffic at ISPs has already been deployed. In both cases, having machines that are diverse in both geography and the IP address space is beneficial. For example, despite three full weeks of Code Red probing and replication, each PlanetLab site still observed a large number of probe attempts from distinct IP addresses. If no filtering was being done, we would expect that sites would experience much greater overlap in the set of Code Red infected machines probing them.

In the case of Nimda, we observe a combination of what is likely two effects. First , when Nimda chooses target machines, it has an affinity for certain regions of the IP address space. 50% of the time an address with the same first two octets will be chosen, 25% of the time an address with the same first octet will be chosen, and 25% of the time, a random address will be chosen. From Figure 7d, we see that the effect of this affinity to "nearby" IP addresses is fairly pronounced. A second factor that also is likely also coming into play here is ISP filtering of Nimda probe attempts. In the



(a) Code Red

(b) Nimda

Figure 6: Total unique Code Red (a) and Nimda (b) probes detected by Netbait on 90 PlanetLab nodes in January 2003.

aftermath of the Nimda epidemic, many ISPs installed filters in their routers to filter Nimda probe traffic in an attempt to mitigate the spread of the infection. Two years later, we observe that these filters appear to still be in place. Our data shows that out of the 42 PlanetLab sites, half of the sites observed three or fewer Nimda probes from distinct IP addresses while a small number of sites accounted for nearly all of the probes detected. In the latter case, having those handfuls of extra machines situated nearby infected machines allowed us to observe infections that the majority of the other PlanetLab sites were unable to detect.

5.3 Summary

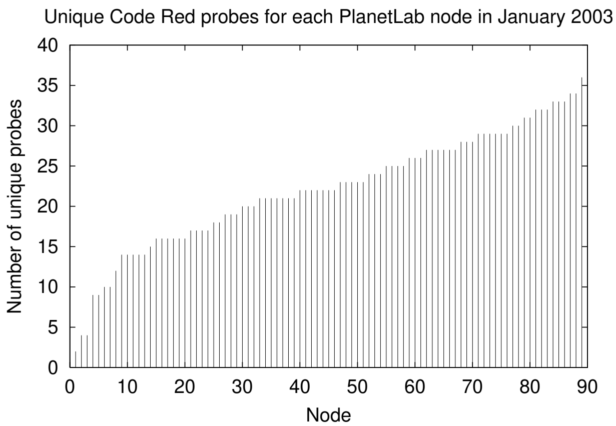
In summary, we observe that Netbait's approach of having multiple geographically dispersed machines sharing probe information from machines infected with Internet worms is beneficial. These benefits come from a combination of two effects. First, by sharing the results across multiple machines, Netbait is able to identify a substantially larger set of infected hosts simply by having more distinct IP addresses available to be probed. Second, by having spreading these machines out over the world on many different networks, we achieve greater coverage both in terms of the IP address space and in geographical extent. The former is important for identifying infected machines which have affinity to certain regions of the IP address space (e.g., to speed infection). The latter is important for identifying infected machines during a response to a worm epidemic and during the inevitable post-mortem cleanup phase. Even after a worm is largely contained (e.g., by blocking its traffic at all major ISPs), infected machines within various organizations still need to be identified and purged by system administrators to make them usable again and to reduce potential financial liability.

6. RELATED WORK

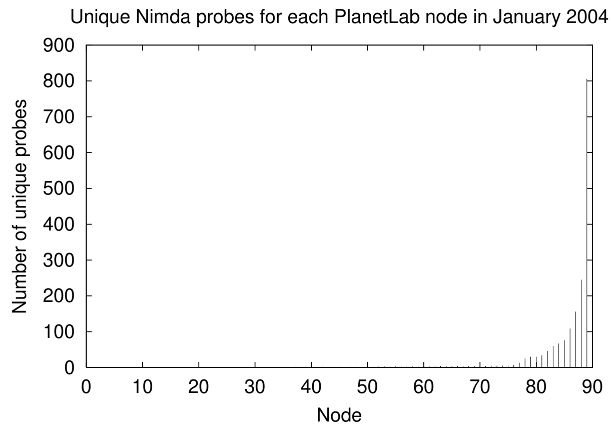
Dshield [7] is a platform that allows firewall users to share intrusion detection information. Users contribute intrusion detection information by running client programs that collect local firewall logfiles and submit them to the DShield team via email or a web-based interface. Contributed logfiles are stored in a centralized database which can then be queried to determine if a particular machine has been compromised and to help produce summary reports of various Internet worm epidemics. Compared to Netbait, Dshield shares similar goals. Architecturally, however, Netbait differs in several key respects. First, Netbait is a based on a distributed query processing architecture that supports "real time", resource efficient queries, as opposed to queries on data manually aggregated into a centralized database. Second, Netbait completely automates data aggregation in response to queries by dynamically discovering which nodes are available and contributing and by aggregating their contributions using an efficient overlay network (i.e., no manual submission or parsing of logfiles is necessary). Finally, since all machines are currently trusted, Netbait does not need to take malicious users who submit false reports into account [1], thereby simplifying its design.

A number of studies have been done on characterizing the macroscopic behavior of Internet worm epidemics using analytical modeling and simulation. In [28], Staniford, Paxson, and Weaver develop analytical models characterizing the rates of infection for recent Internet worms including Code Red, Code Red II, and Nimda, as well as hypothetical Flash worms. In [18], Moore et al. develop similar models based on infection models used in the public health community and use them to quantify the effectiveness of different worm containment techniques, such as content filtering and address blacklisting. The detailed and timely analyses provided by researchers at CERT and CAIDA in response to recent Internet worm epidemics also fall into the analysis space. Compared to Netbait, these efforts are largely complementary. One possibility we see as a potential synergy

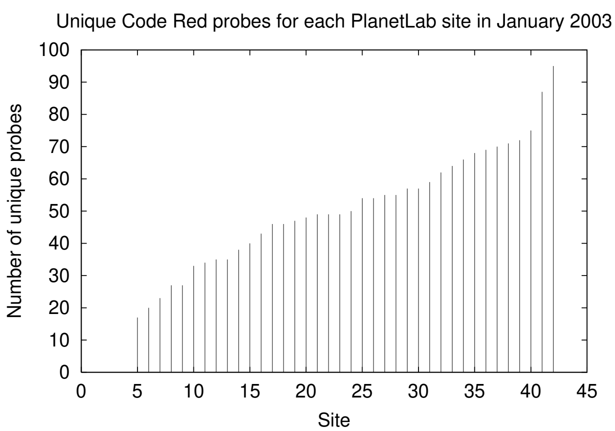
[1]Being able to support faulty and malicious nodes is an interesting avenue for future work. The results of such efforts would likely be applicable to both Netbait and systems like DShield.



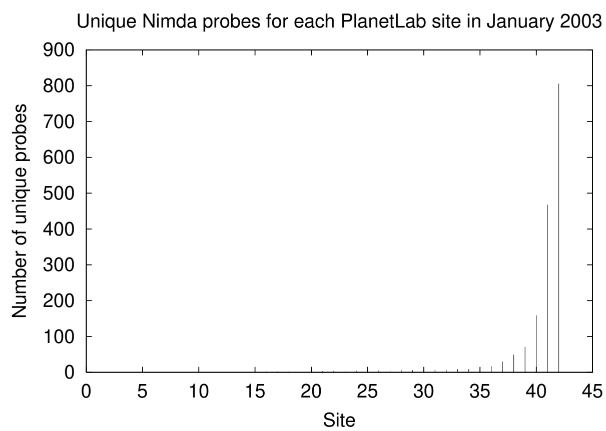
(a)



(b)



(c)



(d)

Figure 7: Unique Code Red (a) and Nimda (b) probes detected by each of the 90 PlanetLab nodes running Netbait in January 2003. Unique Code Red (c) and Nimda (d) probes detected by each of the 42 PlanetLab sites in January 2003.

is to use Netbait's historical observations of Internet worm behavior to drive the validation of various analytical models and simulations of large-scale Internet worm behavior.

Honeypots are systems that are specifically configured to be either probed, attacked, or fully compromised by remote attackers. Their main purpose is to acquire knowledge of a remote attacker's techniques as a means for developing better defenses. In their probe-only and attack configurations, honeypots bear a striking resemblance to the individual data collection components in Netbait. In these configurations, honeypots detect probes and attacks from remote machines through use of an intrusion detection system but do not allow the system to be actually compromised. Such a configuration is precisely what the data collection components in Netbait accomplish. Indeed, one way one to characterize Netbait might be to view it as a distributed query processing system over honeypot data as collected by a set of cooperating machines spread over the wide-area.

Distributed query processing systems capable of performing queries over large collections of nodes in a scalable, robust, and resource efficient manner have been a topic of much interest lately. In the realm of sensor networks, TinyDB [15] processes distributed queries over networks of low-power, wireless sensors in a time and energy efficient and fault tolerant manner. IRISNet [19] is distributed query system for querying and mining Internet-scale systems composed of large collections of sensors, such as webcams. Compared to TinyDB, IRISNet focuses on sensors which are richer in computational capabilities and are not power constrained. PIER [11] is a distributed dataflow query engine based on distributed hash tables. It supports a dataflow-diagram style scripting language, with built-in support for DHT-based joins (including Bloom filters), grouping, filtering and aggregation. Finally, Astrolabe [30] is a distributed information management system intended for large-scale, highly dynamic Internet applications. It uses peer-to-peer gossiping, hierarchy, and makes heavy use of aggregation for robustness against failures and for scalability. One interesting area for future work might be to build Netbait on a general-purpose system like PIER or Astrolabe and to compare it with our specialized implementation in terms of performance, resource usage, functionality, and system complexity.

7. CONCLUSION

In this paper, we presented Netbait, a planetary-scale service for distributed detection of Internet worms. Netbait is based on a distributed query processing architecture. It processes queries over dynamically constructed query processing trees built on the Tapestry DOLR and uses application-specific aggregation and compact encodings of query results to process queries in a resource efficient manner. We have implemented a prototype implementation of Netbait based on a simplified version of the architecture and have deployed it on 90 nodes of the PlanetLab testbed at 42 sites spread across three continents. Preliminarily results based on one month's worth of data collection shows that Netbait's approach of sharing probe information across multiple machines in the wide-area is quite promising. By sharing information across machines, we observe that Netbait is able to identify a substantially larger set of infected hosts than would be possible with only a limited set of network viewpoints. Further, we also observe that by spreading these machines over many different networks, we are also able to identify infected machines which have affinity to certain regions of the IP address space.

Our long-term goal is to build a Netbait system that is capable of scaling to 100,000 hosts and processing distributed queries on the order of a few minutes such that ISPs and network administrators can rapidly respond to an Internet worm epidemic. Towards this end, our immediate plans are first to build a full implementation of the Netbait distributed query processing system as described in this paper, to deploy it on PlanetLab, and to measure its performance in response to various workloads. Key research problems that will require further investigation include scalability, robustness, and how to cope with faulty and malicious hosts. Even at the scale of hundreds of machines in early deployment, Netbait will still provide a valuable service to the Internet community. For example, despite the fact that Code Red and Nimda have been largely contained, their presence on the Internet still remains, as observed by the probes we received from thousands of Code Red and Nimda infected machines. For worms that have yet to emerge, Netbait will offer even greater benefits to ISPs, allowing for significant automation in the cleanup process during a post-worm epidemic, thereby drastically reducing the financial costs incurred by such activities.

8. REFERENCES

[1] K. P. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, and Y. Minsky. Bimodal multicast. Technical Report 98-1665, Cornell University, Department of Computer Science, 1999.

[2] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron. Scribe: A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communications*, 20(8), October 2002.

[3] CERT. Cert advisory ca-2001-11: sadmind/iis worm, May 2001.

[4] CERT. Cert advisory ca-2001-19: Code red worm exploiting buffer overflow in iis indexing service dll, July 2001.

[5] CERT. Cert advisory ca-2001-26: Nimda worm, September 2001.

[6] CERT. Cert incident note in-2001-09: Code red ii: Another worm exploiting buffer overflow in iis indexing service dll, August 2001.

[7] DShield.org. Distributed intrusion detection system. http://www.dshield.org, November 2000.

[8] P. Eugster, S. Handurukande, R. Guerraoui, A.-M. Kermarrec, and P. Kouznetsov. Lightweight probabilistic broadcast. In *Proceedings of The International Conference on Dependable Systems and Networks*, July 2001.

[9] A. J. Ganesh, A.-M. Kermarrec, and L. Massoulie. Hiscamp: self-organizing hierarchical membership protocol. In *Proceedings of the SIGOPS European Workshop 2002*, September 2002.

[10] I. Gupta, A.-M. Kermarrec, and A. J. Ganesh. Efficient epidemic-style protocols for reliable and scalable multicast. In *Proceedings of the 21st Symposium on Reliable Distributed Systems*, October 2002.

[11] M. Harren, J. M. Hellerstein, R. Huebsch, B. T. Loo, S. Shenker, and I. Stoica. Complex queries in dht-based peer-to-peer networks. In *Proceedings of the 1st International Workshop on Peer-to-peer Systems*, March 2002.

[12] K. Hildrum, J. D. Kubiatowicz, S. Rao, and B. Zhao. Distributed object location in a dynamic network. In *Proceedings of the Fourteenth ACM Symposium on Parallel Algorithms and Architectures*, August 2002.

[13] J. Jannotti, D. K. Gifford, K. L. Johnson, M. F. Kasshoek, and J. James W. O'Toole. Overcast: Reliable multicasting with an overlay network. In *Proceedings of the Fourth Symposium on Operating System Design and Implementation*, October 2000.

[14] J. D. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. Oceanstore: An architecture for global persistent storage. In *Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems*, November 2000.

[15] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. Tag: a tiny aggregation service for ad-hoc sensor networks. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, December 2002.

[16] M. L. Massie, B. N. Chun, and D. E. Culler. The ganglia distributed monitoring system: Design, implementation, and experience, 2003. Submitted for publication.

[17] D. Moore, C. Shannon, and J. Brown. Code-red: a case study on the spread and victims of an internet worm. In *Proceedings of the SIGCOMM Internet Measurement Workshop 2002*, August 2002.

[18] D. Moore, C. Shannon, G. Voelker, and S. Savage. Internet quarantine: Requirements for containing self-propagating code. In *Proceedings of the 2003 IEEE Infocom Conference*, April 2003.

[19] S. Nath, A. Deshpande, P. B. Gibbons, and S. Seshan. Mining a world of smart sensors. Technical Report IRP-TR-02-05, Intel Research Pittsburgh, August 2002.

[20] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling tcp throughput: A simple model and its empirical validation. In *Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, October 1998.

[21] L. Peterson, D. Culler, T. Anderson, and T. Roscoe. A blueprint for introducing disruptive technology into the internet. In *Proceedings of the 1st Workshop on Hot Topics in Networks (HotNets-I)*, 2002.

[22] C. G. Plaxton, R. Rajaraman, and A. W. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *Proceedings of the 9th Annual Symposium on Parallel Algorithms and Architectures*, June 1997.

[23] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proceedings of the ACM SIGCOMM '01 Conference on Communications Architectures and Protocols*, August 2001.

[24] M. Roesch. Snort - lightweight intrusion detection for networks. In *Proceedings of the 13th Systems Administration Conference (LISA '99)*, 1999.

[25] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms*, November 2001.

[26] J. F. Shoch and J. A. Hupp. The worm programs: Early experiences with distributed systems. *Communications of the ACM*, 25(3), March 1982.

[27] E. H. Spafford. The internet worm program: An analysis. Technical Report CSD-TR-823, Purdue University, 1988.

[28] S. Staniford, V. Paxson, and N. Weaver. How to 0wn the internet in your space time. In *Proceedings of the 11th USENIX Security Symposium*, August 2002.

[29] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the ACM SIGCOMM '01 Conference on Communications Architectures and Protocols*, September 2001.

[30] R. van Renesse, K. P. Birman, and W. Vogels. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM Transactions on Computer Systems*, 2003.

[31] H. Weatherspoon and J. D. Kubiatowicz. Efficient heartbeats and repair of softstate in decentralized object location and routing systems. In *Proceedings of the SIGOPS European Workshop 2002*, September 2002.

[32] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report CSD-01-1141, University of California, Berkeley, Computer Science Division, 2000.

[33] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. H. Katz, and J. D. Kubiatowicz. Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemintation. In *Proceedings of the Eleventh International Workshop on Network and Operating System Support for Digital Audio and Video*, June 2001.