2009

A Network-Aware Distributed Membership Protocol for Collaborative Defense

David Zage

Carl Livadas

Eve M. Schooler

Report Number:
09-005

A Network-Aware Distributed Membership Protocol for Collaborative Defense

David Zage
Carl Livadas
Eve M. Schooler

A Network-Aware Distributed Membership Protocol for Collaborative Defense

David Zage, Carl Livadas, and Eve M. Schooler
zagedj@cs.purdue.edu, clivadas@alum.mit.edu, eve.m.schooler@intel.com

Abstract— To counteract current trends in network malware, distributed solutions have been developed that harness the power of collaborative end-host sensors. While these systems greatly increase the ability to defend against attack, this comes at the cost of complexity due to the coordination of distributed hosts across the dynamic network. Many previous solutions for distributed membership maintenance are agnostic to network conditions and have high overhead, making them less than ideal in the dynamic enterprise environment. In this work, we propose a network-aware, distributed membership protocol, CLUSTER, which improves the performance of the overlay system by biasing neighbor selection towards beneficial nodes based on multiple system metrics and network social patterns (of devices and their users). We provide an extensible method for aggregating and comparing multiple, possibly unrelated metrics. We demonstrate the effectiveness and utility of our protocol through simulation using real-world data and topologies. As part of our results, we highlight our analysis of node churn statistics, offering a new distribution to accurately model enterprise churn.

I. Introduction

Network defense is an elusive art. The arsenal to defend our devices from attack is constantly lagging behind the latest methods used by attackers to break into them and subsequently into our networks [1]. Self-propagating malware, such as *worms*, is one of the most challenging security issues of today's Internet [2]–[5], potentially costing businesses multi-millions of dollars in lost revenue and time [6], [7].

As a worm spreads, its virility depends greatly on its ability to identify and infect vulnerable hosts. Early worms relied on large numbers of random connection attempts to identify vulnerable hosts. Due to their aggressive nature, early worms were easy to detect by singling out hosts with abnormally high connection rates. To avoid detection, newer worms have employed increasingly sophisticated techniques to identify vulnerable hosts [2], [8]. Such techniques result in fewer, more targeted connection attempts, leading to more successful worm propagation and obfuscated worm detection. Also, as the impetus behind worm creation has shifted from fun to profit [9], [10], researchers have seen increasing sophistication in the replication techniques employed by worms, including the use of code randomization and encryption designed to thwart signature-based detection schemes. In essence, the combination of these techniques has rendered once-identifiable malware into new, zero-day attacks, allowing worms to spread through once-defended networks.

To counteract these trends, recent research has developed distributed collaborative detection systems, which have been quite successful in detecting malware, while maintaining low system-wide false positive rates [11]–[13]. These systems are based on the hypothesis that if detectors are embedded in end nodes across the network, each node will be capable of assessing whether or not it is being attacked by malware based upon local information and can communicate this assessment to a dynamic but random subset of peer nodes. By aggregating the assessments of other peer nodes along with local information, individual nodes are able to infer the approximate overall health of the network with greater accuracy than when they to act alone, converting weak local hypotheses into strong global evidence. In previous work, Dash *et al.* [11] presented an approach referred to as *Distributed Detection and Inference* (DDI). DDI is capable of effectively detecting worms with connection rates that are several orders of magnitude lower than those of worms observed to date [11], [12], [14]. While individual host detectors may offer weak and inaccurate infection evidence, DDI's system-wide alarms, based on corroborated evidence, are highly accurate.

DDI's detection accuracy, however, comes at the cost of complexity due to the coordination of distributed hosts across the network. Thus, one of the most challenging aspects of implementing dynamic distributed systems such as DDI is providing an accurate membership protocol, that is, one that is able to maintain an up-to-date overlay of available nodes. In addition, a membership protocol should be designed to be scalable, benign fault-tolerant, network-aware and have low overhead. In particular, if the membership fails to take realistic network social patterns, such as churn, into account at design time, the performance of the deployed system can suffer severely [15]. Centralized solutions for managing membership are ineffective as they restrict the scalability of the system and create a single point of failure. Thus, we focus on creating an effective, distributed membership protocol.

Previous research has developed distributed membership protocols for structured overlays [16], [17], where the overlay topology has tight topological and organizational invariants, constraining the set of nodes eligible to become peers of a given node [18]. While these protocols provide valuable insights, they are not appropriate for our environment where such constraints do not exist. Other research has looked at membership protocols designed for unstructured networks, including SCAMP [19], HiSCAMP [20], Localiser [21], SwapLinks [22], [23], and HyParView [24]. While each of the previous protocols has attractive points, the majority have high overhead costs associated with them and are notably network-*unaware*, selecting peers at random rather than informed by

network measurement data or observed patterns.

In this paper, we make the following contributions:

1) We propose a network-aware distributed membership protocol, CLUSTER, which is both scalable and adaptive. CLUSTER is designed to improve the performance of DDI's collaborative defense through the use of multiple system metrics based on the social patterns of the collaborative devices and of their users. The metrics are used to bias messaging decisions towards nodes with desirable attributes, improving the ability of the DDI system to quickly and accurately correlate reports from multiple nodes.

2) We design an extensible method for aggregating multiple, possibly unrelated metrics into a single Proximity function. The individual metrics are discretized based on distributions known *a priori* or calculated during runtime if not already known and the results are subsequently aggregated. Our method allows CLUSTER to be easily extended in the future to include other metrics defining node desirability.

3) We demonstrate the effectiveness of our solution through simulation using real-world enterprise traffic data, topologies, and churn rates in mitigating the effects of distributed system dynamics and maintaining high reachability with low overhead.

4) We highlight unexpected results from the analysis of the node churn statistics; we offer a new distribution to model enterprise churn to more accurately simulate traffic patterns when actual data is not available.

This paper is organized as follows: We present an overview of the DDI system in Section II. Section III focuses on enhancing the efficiency and effectiveness of the DDI system as a whole by incorporating a lightweight, metric-driven distributed membership protocol. We present our experimental methodology in Section IV and evaluate our new, network-aware membership protocol through simulations based on real-world data sets in Section V. We discuss related work in Section VI. Finally, in Section VII, we conclude by summarizing the results and highlight possible future research directions.

II. DDI System Model

In this section, we provide an overview the current DDI architecture and its components.

A. The Distributed Detection and Inference System

A simplified depiction of DDI is shown in Figure 1. Hosts participating in the DDI system are embued with anomaly-detecting sensors producing measurement data known as local detectors and aggregating sensors that consume this data to supply inference across time, sensors, and/or machines known as global detectors. Along with these two types of detectors, the DDI end-node architecture also includes a back-end agent that supplies membership tracking of available DDI nodes, a messaging library to support gossip-style communication between nodes, and the capability to track the performance of components in the system.
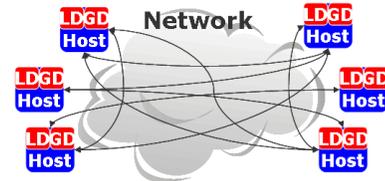
Fig. 1. A simplified depiction of Distributed Detection and Inference. On each of the hosts resides one or more local detectors and global detectors. Nodes communicate through periodic sending of messages to subsets of neighbors provided by the system membership.

B. Local Detectors (LDs)

A host may run one or more local anomaly detectors, with each detector issuing binary *local infection reports* (LRs). A LR indicates whether the host on which the local detector resides is believed to be infected. Local detectors issue their LRs based on the host's operational state, such as network traffic and processor/memory characteristics. While a number of sophisticated heuristics may be used to issue LRs, the most commonly used is based on the host's connection count. A local detector monitors the number of outgoing connections within its reporting period and if this number exceeds a particular connection threshold, it issues a positive report (*i.e.*, an infection has been detected). Otherwise, it issues a negative report (*i.e.*, a infection has not been detected).

C. Local Report Dissemination

Local detectors disseminate their LRs to one or more global detectors using a gossip dissemination scheme parameterized by a fanout $f_L \in \mathbb{N}^+$ and a scope $s_L \in \mathbb{N}^+$. The gossip scheme is closely linked to the membership provided by the back-end of DDI as the membership supplies the potential gossip receivers. Upon generating a LR, a local detector sends the LR message to f_L random nodes with global detectors provided by the membership component. The LR messages contain the local detector's binary infection status, a timestamp corresponding to the time at which the report was issued, and the scope s_L of the report. Without a correctly functioning membership protocol, LR messages will not propagate through the system and the performance will be severely degraded.

D. Global Detectors (GDs)

Global detectors issue *global infection reports* based on the set of recently received LRs. Each detector maintains a cache C_g of all of the LRs received within a particular time window of $W_g \in \mathbb{R}^{\geq 0}$ seconds. Let A_g to denote the size of this cache and A_g^+ and A_g^- to denote the number of positive and negative LRs in C_g, where $A_g = A_g^+ + A_g^-$. In DDI, global detectors issue global infection reports using the *PosCount* inference algorithm. Thus, a global detector g issues a positive global report when $A_g^+ > d_g$, where $d_g \in \mathbb{N}^+$ is the detector's PosCount threshold. Any positive global infection report constitutes a system-wide infection alarm. For more details, see [11], [12]

E. Current DDI Membership

As mentioned earlier, the DDI membership component is critical to the proper functioning of DDI. Without an accurate membership, the LRs will not reach the global detectors, minimizing the data received during a particular time window and decreasing the probability of making correct inferences on the state of the system. The DDI system currently has two methods of providing accurate membership at each node: a centralized scheme that provides each node with entire view of the network or a decentralized scheme which provides each node with a partial view of the network.

1) Centralized Membership: The most basic implementation of DDI utilizes a membership scheme in which a pre-defined rendezvous node stores and disseminate membership information to all network participants. Whenever a node joins or leaves the network, the rendezvous node is contacted with the updated information which then disseminates the information to the rest of the network. This simple membership management scheme provides a consistent, global view of the membership. It is helpful for small networks and testing new DDI functionality since it provides a baseline of system performance and eliminates a source of randomness inherent to most decentralized membership schemes. However, this approach does not scale.

2) Decentralized Membership: For larger systems, the DDI system currently uses the decentralized, gossip-based membership management protocol SCAMP [19] because it offers a self-organizing subscription mechanism that is well suited to the dynamic nature of the membership. SCAMP provides each node in a system of size N with a membership view of size $O(logN)$. As shown in [19], SCAMP is able to maintain high messaging reliability similar to that of a centralized global-knowledge scheme. Our implementation of SCAMP uses several of the enhancements suggested in [19], including message indirection to help randomize new node subscriptions and a leasing mechanism to help re-balance view sizes. The major drawback of SCAMP is that it is agnostic to any underlying metrics or topology and thus unable to leverage the knowledge available in the network to improve the system performance. Secondly, SCAMP has a high overhead cost associated with maintaining the membership.

III. CLUSTER: NETWORK-AWARE MEMBERSHIP

In this section, we formulate an improved membership protocol, CLUSTER, which offers a network-aware approach to decentralized membership management. In general, many distributed membership algorithms provide a node with a random subset of the nodes in the entire network, known as a partial view. We build on the ideas proposed by Chee-tancheri *et al.* [12] and Li *et al.* [14]. In CLUSTER, instead of relying on a partial view chosen completely at random, a system-defined percentage of choices for a node's view are biased based on additional *Proximity metrics* towards nodes with desirable attributes. The impetus for our protocol design lies in the fact that, by influencing the construction and maintenance of the partial view of membership, we can significantly improve the performance of the collaborative defense provided by DDI.

In this paper, we use the terms *Proximity* and *Proximity metrics* to represent one or more metrics used to bias the neighbor selection of the membership protocol. For example, consider node selection biased by network latency. Each node's view of the membership becomes biased towards local information, allowing it to draw more accurate conclusions of what is occurring locally, as well as decreasing the need to propagate information to distant nodes. As a result, the overhead incurred by maintaining distributed membership can be greatly reduced, making the system more scalable.

In this paper, we use the Proximity metrics of network latency and node connectivity (duration of a node's time connected to the network), but could easily augment the selection with other metrics such as resource availability (power, bandwidth, cpu, memory, network interfaces), resource consumption (power, storage), social trends (usage patterns, time of day/week), node believability (to generate low false alarms), or persistence (level of continued social interactions between pairs of nodes). As the DDI framework continues to mature, we fully expect metrics representing additional social patterns and reputations (of devices and their users) to be incorporated into the system.

A. Algorithm Overview

CLUSTER is a light-weight distributed membership protocol that provides each node in the system with a partial view of the global membership. This view is maintained at each node in the form of two lists, a *SendList* of neighbors a node is responsible for sending messages to and a *ReceiveList* of neighbors from which the the node receives messages. The protocol consists of three main components: node join, node leave, and a node maintenance algorithm.

1) Join: When a node n initially joins the system, it contacts a well-known rendezvous node from which it receives a list of potential neighbors. The node n will then contact a random subset of the potential neighbors, which reply with further potential neighbors. Piggy-backed on all of the membership messages are the network metrics used (*e.g.*, latency and connectivity) and a freshness period denoting how long the node information is valid (discussed in Section III-A.2). Once a node n has received a set of potential neighbors, it chooses a subset of these nodes as neighbors. During this selection process, a system-defined percentage of nodes is chosen based on Proximity and the remaining are chosen at random. For each selected neighbor node, n will send a request to receive information from the node. Once a positive acknowledgement is received at n, the neighbor node will be added to n's *ReceiveList* and inversely n will be added to the neighbor's *SendList*. The node n will select neighbors until it has reached the system prescribed minimum *ReceiveList* size.

2) Maintenance: During normal system operation, each node will periodically check the *freshness period* for each node on its *ReceiveList*. The freshness period represents the duration of time for which the information on the *ReceiveList* is valid. If

the freshness period of a *ReceiveList* neighbor elapses without being renewed, the neighbor is removed and a replacement neighbor is found using a technique similar to that used during node join. To renew the freshness of the *ReceiveList*, the node maintaining the *ReceiveList* will explicitly poll the nodes on it, updating the period when a response is received. In order to decrease the overhead associated with maintaining fresh views, any message received between the nodes (not just membership messages) acts as a poll message and is used to update the freshness of the nodes. This allows the majority of CLUSTER nodes to remain quiescent during the periodic checks and greatly reduces the overhead seen in the system.

Along with periodically checking the neighbor freshness, each node checks the percentage of nodes in its *ReceiveList* selected based on Proximity. If this percentage is less than the desired system-defined percentage, the node probes its peers for possible additional neighbors and replaces a neighbor that was chosen at random with a one based on Proximity. This mechanism allows the node to optimize its *ReceiveList* over time to contain the desired fraction of nodes selected based on Proximity even as the system undergoes churn.

3) Leave: As with any dynamic system, nodes will eventually leave the system due to many factors such as laptop movement or shutting down at the end of the work day. When a node n leaves the system it sends a leave message to all nodes in its *ReceiveList*, informing the neighbors data no longer needs to be sent to n. No messages are required to be sent to the *SendList* as the node will be removed once its freshness period has elapsed. In order to handle ungraceful leaves, a node will also periodically check the freshness of its *SendList* (with less frequency than its *ReceiveList*), to avoid sending messages to nodes that have left the network.

B. Multi-Metric Proximity Weighting

In order to make our technique extensible and allow for the inclusion of multiple, possibly non-correlated metrics (*e.g.*, latency and node connectivity), we discretize each of the metrics into a series of values (bins) based on its distribution. Using Equation 1, where B stands for the discretization function, m_n stands for the nth metric, and w_n stands for the weight associated with the nth metric, the discretized values are aggregated to form a singular Proximity value which can easily be compared between nodes.

$$B(m_1) \times w_1 + B(m_2) \times w_2 + \cdots + B(m_n) \times w_n \quad (1)$$

For metrics which have a definable distribution such as the node connectivity (see Section IV-B), the discretization bins can be formed *a priori* based on the known distribution. For metrics such as latency, which are more variable due to their dependency on the network and network conditions, we use an adaptive scheme in which the estimated metric mean and standard deviation are used to determine the width of the bins. The width of the bins is set to the standard deviation while the initial starting bin is set to the estimated mean. In this manner, the metric comparison process adapts based on current system performance without the need for user input.

C. Protocol Benefits

Our protocol has several desirable features to highlight. First, it is straightforward to understand, making it easy to implement and extend. While the DDI framework is currently used for intrusion detection, the inclusion of other sensors and system components (such as a reputation service) necessitate a network-aware membership component that is easily adaptable. It should be noted that CLUSTER utilizes the "social" interaction patterns between devices to bias membership and this facet of our protocol will become all the more important as the system is extended to utilize ideas such as reputation between infrequently communicating nodes. Not only is our protocol extensible, it is easy to configure, only requiring a desired minimum *ReceiveList* size and percentage of nodes selected based on Proximity. In future work, we intend to have these parameters determined autonomically by the system (*i.e.*,derived through measurement and historical observation). Secondly, as we show in Section V, our protocol is lightweight, imposing very little control overhead. Finally, the CLUSTER protocol does not need extra infrastructure, such as landmarks for locality, maintaining a fully decentralized membership.

IV. EXPERIMENTAL METHODOLOGY

In this section, we present our methodology for the DDI simulations. We employ the ns2 simulator [25] to test the performance of the DDI system with different membership components. We discuss real background traffic patterns, churn statistics, and topologies, all derived from a large-scale corporate enterprise network. We describe the malware that is synthetically deployed. Finally, we present the experimental configuration of the DDI system, the configuration of the membership components, and the metrics we use to evaluate the effectiveness of the membership protocols.

A. Background Traffic Patterns

Our simulations are driven with real background traces collected from over 400 end-hosts in the Intel corporate enterprise network spanning 5 weeks of time. During the simulations, each node chooses a random trace file from which to replay outbound traffic. The destination addresses of the packets in the traces are mapped to nodes in the simulated topology and the packets are sent to them. To preserve realism, the starting points of the trace files are aligned (*e.g.*, an experiment may begin at 9am on a Monday in all trace files), to preserve the diurnal patterns found in the enterprise.

B. Modeling Enterprise Node Churn

Along with realistic traffic patterns, node joins and leaves are based on the real connectivity patterns of the users participating in the network traces. As DDI is designed to be a persistent system in the distributed environment, we analyzed the traffic patterns for periods of 1 week and greater. The node connectivity patterns form heavy-tailed distributions, with the average amount of time a node is connected to the network being just under 6 hours, but with a median connection time of only 117 minutes. This causes a moderate amount of churn

in the system, with an average of ten nodes per minute joining or leaving the network.

Similar to other research, we find that the join pattern of enterprise end-host nodes can be accurately modeled as a Poisson process with a $\mu = 6$, irrespective of the length of time. However, contrary to popular assumptions, we find that the Pareto distribution that is often used to model node connectivity [26] does not accurately fit our enterprise data. We find, irrespective of the number of weeks, that node connectivity can be accurately represented by a Weibull distribution, seen in Figure 2, with the scale parameter $\eta = 12900$, the shape parameter $\beta = 0.56$, and the location parameter $\gamma = 0$. This implies other simulations can accurately model the social patterns of end-nodes in a large enterprise network by using a Weibull distribution instead of the Pareto distribution for node connectivity.
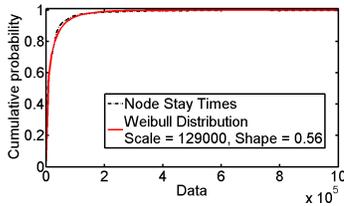

Fig. 2. Distribution of Actual Node Connectivity

C. Topology

For the majority of our experiments, we use a "star" topology incorporating 100 nodes that communicate with each other through a central router. In order to determine the effects of real-world latencies on the membership protocols, the roundtrip time (RTT) between nodes is modeled on latencies seen in the high performance network of the NLANR Active Measurement Project [27]. Every node is infectable when it is connected to the network. This simplified topology helps to isolate the differences between memberships and provides comparable baseline numbers.
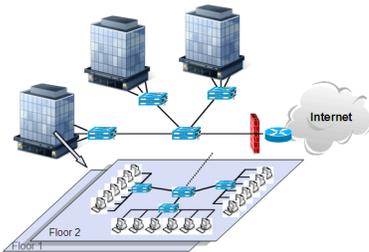
Fig. 3. Example of an enterprise campus network.

We also utilize the enterprise topology shown in Figure 3, which is based on the construction of a typical Intel enterprise campus network. Inside the enterprise network, there are one or more buildings linked by a networking device (*e.g.*, switch). Inside each of the buildings are multiple networked floors, with each floor containing several subnetworks. These subnetworks consist of a set of end-user nodes, each running the DDI system. Our simulation topology consists of 3 buildings with 2 floors per building. Each of the floors contain 3 subnetworks

with 6 end-user nodes for a total of 108 DDI nodes and 56 networking devices (e.g. routers or switches) in the network.

D. Worm Model

We presume the background traffic is "clean" and introduce worm traffic by superimposing an actual (albeit synthetic) worm whose propagation follows a discrete SI epidemic model [28]. At a random point in time, a randomly selected node is *infected* with a worm instance. Thereafter, every 10 seconds each infected node attempts to infect other nodes by randomly generating a target address. The success of finding another infectable node in the network is determined by an *address density* parameter. For our simulations, we use an address density of $1/10000$; thus, 1 in 10000 addresses is a valid infectable node.

E. DDI Configuration

In all of our simulations, each node is instrumented with a local detector and global detector. Each local detector issues local reports based on its background and worm traffic. In particular, each local detector inspects outgoing packets and keeps a count of the number of outgoing connections within a 50 second sliding window. Every 10 seconds, the local detector issues a local report by inspecting the number of outgoing connections. When the node is not infected, the local detector issues a positive local report when the number of outgoing connections is greater than 4; otherwise, it issues a negative local report. Thus, a node issues a false positive local report when the number of connections in the background traffic exceeds this connection threshold. Once the node becomes infected, the local detector always issues positive local reports, irrespective of the number of connections. In our simulations, we use a scope $s_L = 1$ and fanout $f_L = 1$ to disseminate reports to neighboring nodes provided by the DDI membership.

Global detectors cache the local reports received within a sliding time-based observation window of $W_G = 20$ seconds. Every 10 seconds, each global detector inspects its cached local reports. Instead of running one simulation for each PosCount threshold d_g, we instrument global detectors to report the fraction of positive to total local reports within their cache. Thus, post-simulation, we can determine the time-to-detection and the infection level at detection for any threshold d_g chosen *a posteriori*. We obtain the empirical global false positive rate (GFPR) for a threshold d_g by running a very long simulation without a worm; we estimate the system FPR as the ratio of the number of system alarms to total number of global detector periods simulated.

For each experimental configuration, we run 20 simulations. Each simulation is 4 weeks in length, with the worm beginning prorogation during the 3rd week. In one of the simulations, the worm is disabled in order to obtain an empirical system FPR. The remaining 19 simulations have identical configurations, but their results differ due to the variability in the formation of the membership, variability in how the infection spreads, and the randomness inherent in the local report dissemination.

F. Membership Configuration

We configure all of the membership protocols to use a freshness period of 10 seconds to allow for fair comparison between the membership types. CLUSTER is configured to use a desired minimum *ReceiveList* size of 10 and we vary the percentage of nodes chosen based on Proximity as described in the results below. As discussed in Section III-B, we use the equation $B(l) \times 1.0 + B(nc) \times 1.0$ to compute the Proximity value of a node, where latency (l) and the node connectivity (nc) are each weighted by a factor of 1.0.

G. Performance Metrics

In order to quantitatively compare the performance of the DDI system and its respective membership components, we use the following metrics:

- **Infection Level** is the average percentage of nodes that is infected by the worm at the time-of-detection.
- **Global False Positive Rate** (*GFPR*) is the average number of times per week a system-wide infection alarm is issued when the system is *not* infected.
- **System Overhead** is the average amount (in Mb) of membership control data sent and received at each DDI node per second.
- **Reachability** is the percentage of nodes that can be contacted (possibly through more than one hop) from a randomly chosen DDI node. Ideally, the membership protocol should have a reachability of 100.

V. EXPERIMENTAL RESULTS

A. DDI System Performance Using CLUSTER

The CLUSTER protocol can use multiple metrics to determine node Proximity. We begin by using only latency and vary the percentage of node selected based on Proximity. We can see from Figure 4(a), that while the infection level is low for all of the CLUSTER percentages, the results are indiscriminate and do not show any percentage of CLUSTER to be superior. This is due to the churn inherent in the environment which causes the views of the nodes to be in constant flux, making it harder for the nodes to accurately correlate local reports. Based on this assessment, we incorporate node connectivity into the Proximity calculation.
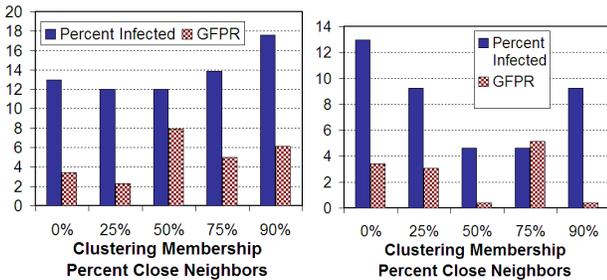
(a) Proximity Using Latency Only (b) Proximity Using Latency and Node Connectivity

Fig. 4. DDI system performance on a star network using different percentages of nodes chosen by different Proximity measures.

As seen from Figure 4(b), using CLUSTER with the inclusion of node connectivity in the neighbor Proximity calculations significantly improves the system performance. The inclusion of the extra network knowledge is able to improve DDI system performance compared to random neighbor selection (*i.e.*, CLUSTER 0%). As the percentage of neighbors chosen based on Proximity increases, the DDI system is able to detect the worm at lower infection levels while maintaining a low *GFPR*. Intuitively, as the percentage increases, nodes have a greater focus on their local area while keeping a modicum of random connections farther out in the network to maintain global perspective. There are diminishing returns for higher percentages (75% and 90%) of Proximity nodes, in which nodes focus almost exclusively on local information, missing global context and incurring higher *GFPR* and infection levels than the median value of CLUSTER.

B. Comparison of CLUSTER and other Membership Protocols

We now look at the system performance of DDI using each of the different membership protocols and their associated overhead. Based on the results presented in Section V-A, we compare the existing protocols versus CLUSTER 50%, where half of the nodes are chosen based on Proximity, since this setting exhibited up to 3 times fewer nodes infected at the time of detection and an order of magnitude fewer false alarms.
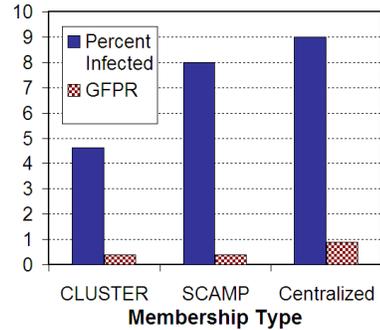

Fig. 5. DDI System performance on a star network comparing CLUSTER to other membership types.

We can see from Figure 5 that the CLUSTER-based system has the lowest infection level while maintaining a *GFPR* of less than one false alarm per week. By taking advantage of both latency and node connectivity to bias the node selection, the system is able to accurately correlate the local reports, detecting the malware at a lower infection than the DDI system using the other membership types.

TABLE I

DDI MEMBERSHIP OVERHEAD

Membership Type	Overhead (Mb/sec)	Reachability (%)
Centralized	1.2	100
SCAMP	4.1	100
CLUSTER	.05	100

Next, we analyze the overhead required to maintain the different membership schemes for the DDI system. From Table I, we can see that each scheme is able to maintain a reachability of 100%. In order to achieve this high level of

connectivity, both SCAMP and the centralized membership protocol require a significant amount of network bandwidth. The overhead in SCAMP comes from the constant need to renew the partial views at each node while the overhead for the centralized scheme results from the aggregation of information at the centralized server and the subsequent dissemination of it to all the nodes for each membership change. As CLUSTER is designed for dynamic systems, it requires just over 50Kb of membership overhead due to its ability to minimize the need for liveness messages and to take advantage of locality to minimize the number of hops messages traverse.

C. Effect of Network Topologies on DDI System Performance

We have shown that CLUSTER is able to improve the performance of the DDI system on an artificial network topology. To more accurately simulate the deployment of DDI in an enterprise environment, we study the impact of real-world inspired enterprise network topologies, such as that depicted in Figure 3, on the performance of the DDI system using different membership mechanisms.
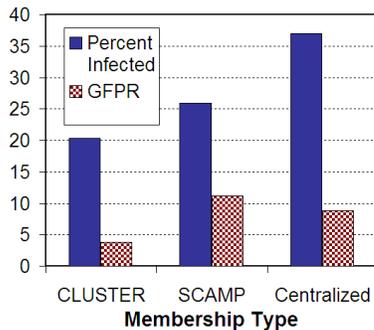

Fig. 6. DDI system performance on a realistic enterprise network.

We can see from Figure 6 that network topology has a large impact in the function of the DDI system. The non-uniform topology makes it easier for the malware to spread through the network while avoiding detection, which can be seen as a general increase in infection level regardless of the membership type utilized. With this in mind, we note the CLUSTER-based system is able to outperform both of the other membership schemes, resulting in 24% and 46% fewer nodes infected and 67% and 58% lower *GFPR* when compared with SCAMP and Centralized, respectively.

Not only is the CLUSTER-based system able to provide the lowest infection level, it is able to maintain 100% reachability using 0.053 Mb/s membership overhead, which is only a .7% increase when compared to the star topology overhead. This is not the case for the SCAMP-based system, which is only able to maintain reachability to 90% of the network. When the scope of the messaging and the frequency of updates for SCAMP were updated to better match the enterprise environment as opposed to the star, the SCAMP-based system was able to maintain full reachability. However, the infection level did not improve and the membership overhead was over 8Mb/s, two order of magnitude greater than CLUSTER.

By using a membership that is network-aware and takes into account the effects of node connectivity when selecting poten-

tial neighbors, the DDI system is able to accurately correlate the information from local peers and identify the malicious activity. This is especially evident in the more realistic setting of the enterprise network where multiple subnetworks and longer paths are present.

D. Churn in the DDI System

As we noted earlier, taking into account node connectivity when designing and testing systems is critical to successful real-world deployment [15]. As we have been working with the DDI system, one of the most interesting conclusions we draw, similar to that of Varun *et al.* [29], is that when design for, churn can actually *improve* the performance and reliability of the system. In DDI, churn acts as a randomization factor, allowing nodes to slowly discover new, potentially better neighbors as nodes re-join the network. As nodes join and establish peer connections, the messages exchanged in the network between nodes act to update the freshness of the partial views of the network, avoiding the need for explicit polling for liveness to be performed. In this manner, the system performance can be improved while much of the overhead associated with churn is converted into useful work. Secondly, in the context of malware and worm propagation, churn actually makes the network more robust to attack. This is due to the intuitive fact that nodes which are not currently connected to the network are not vulnerable to attack. While tangential to this research, this finding has implications for enterprise computer management. For example, if an employee is not needing remote access to a computing resource, its is in their best interest to remove the machine from the network.

VI. RELATED WORK

In this section, we provide an overview of the previous research in two areas related to our work: membership protocols and modeling node behavior in distributed systems.

A. Distributed Membership Protocols

Recently, there has been increasing interest in membership protocols for unstructured distributed systems. SCAMP is a self-organizing peer-to-peer membership service which provides each node in a system of size N with a membership *view* of the network with a size $O(logN)$. SwapLinks [22], [23] is a membership service that uses random walks to build unstructured overlays designed to balance the load experienced at each node. While both systems are able to create reliable distributed memberships, they differ from our system since they are agnostic to network-locality information. The authors of SCAMP have also proposed HiSCAMP [20], a hierarchical membership protocol in which close nodes are grouped into clusters and clusters are themselves recursively grouped into clusters, with each cluster running SCAMP. HiSCAMP has high overhead-costs stemming from the organization and synchronization of clusters. Localiser [21] is a maintenance protocol which uses the Metropolis-style algorithm to achieve node balance and localization on unstructured membership overlays. Localiser is only applicable to existing overlays and has high overhead due to adjustment operations performed to

optimize the overlay. Also, our work differs from the previous works as it is easily extensible to accommodate multiple network metrics in the selection of nodes.

B. Modeling Node Behavior in a Distributed System

As the number and variety of distributed systems continues to grow, research has been conducted in developing methods of accurately simulating the environments in which these systems appear. Multiple empirical studies have been conducted for peer-to-peer systems [30]–[34], confirming that these systems are highly dynamic. However, the results of each study vary widely and none are representative of a large enterprise system. Other work has generalized the join and leave patterns seen in peer-to-peer systems as statistical models such as the Poisson process [35] or on the Pareto distribution [26]. Our work differs in the fact it is able to use real-world enterprise data as well provide a generalization of the empirical data to suggest a more accurate model of node connectivity.

VII. CONCLUSIONS

In this paper, we present a network-aware distributed membership protocol, CLUSTER, that offers significant performance improvements over other state-of-the-art membership management schemes. Moreover, CLUSTER improves the performance of collaborative defense algorithms, such as DDI, which depend on robust, scalable membership tracking in a dynamic operational setting. CLUSTER biases membership decisions toward nodes with desirable attributes as a consequence of using the data gained from observing multiple system metrics and network social patterns (of devices and their users). Additionally, we design an extensible method for aggregating multiple, possibly unrelated metrics into a single Proximity function and show its utility in simulation. Using real-world traffic patterns, topologies, and churn rates, we demonstrate through simulations that our protocol, at minimum, is able to decrease the infection rate of the system by 24% and the false positive rate by 58%, while reducing the overhead by 2 orders of magnitude. Finally, our analysis of node churn in the enterprise network unexpectedly reveals that a Weibull distribution can be used to accurately simulate enterprise traffic patterns when actual data is not available.

VIII. ACKNOWLEDGMENTS

REFERENCES

[1] U. Bayer, I. Habibi, D. Balzarotti, E. Kirda, , and C. Kruegel, "A view on current malware behaviors," in *Proc. of LEET*, 2009.

[2] N. Weaver, V. Paxson, S. Staniford, and R. Cunningham, "A taxonomy of computer worms," in *Proc. of WORM*, 2003.

[3] D. McPherson and C. Labovitz, "Worldwide infrastructure security report, volume ii," tech. rep., Arbor Networks, Inc., 2006.

[4] F. Akujobi, I. Lambadaris, and E. Kranakis, "Endpoint-driven intrusion detection and containment of fast spreading worms in enterprise networks," in *Proc. of MILCOM*, 2007.

[5] B. Stone-Gross, M. Cova, L. Cavallaro, B. G. andMartin Szydlowski, R. Kemmerer, C. Kruegel, and G. Vigna, "Your botnet is my botnet: Analysis of a botnet takeover," tech. rep., UC Santa Barbara, 2009.

[6] C. Economics, "Malware Report: The Economic Impact of Viruses, Spyware, Adware, Botnets, and Other Malicious Code," tech. rep., Computer Economics, 2007.

[7] K. J. Higgins, "Smbs often hit hardest by botnets."

[8] Z. Chen and C. Ji, "A self-learning worm using importance scanning.," in *Proc. of WORM*, 2005.

[9] J. Franklin, V. Paxson, A. Perrig, and S. Savage, "An inquiry into the nature and causes of the wealth of internet miscreants," in *Proc. of CCS*, 2007.

[10] P. Gutmann, "The commercial malware industry," in *DEFCON*, 2007.

[11] D. Dash, B. Kveton, J. M. Agosta, E. M. Schooler, J. Chandrashekar, A. Bachrach, and A. Newman, "When gossip is good: Distributed probabilistic inference for detection of slow network intrusions.," in *Proc. of AAAI*, 2006.

[12] S. G. Cheetancheri, J. M. Agosta, D. H. Dash, K. N. Levitt, J. Rowe, and E. M. Schooler, "A distributed host-based worm detection system," in *Proc. of SIGCOMM LSAD*, 2006.

[13] K. J. Hall, *Thwarting network stealth worms in computer networks through biological epidemiology*. PhD thesis, Virginia Polytechnic Institute & State University, 2006.

[14] J. Li, D.-Y. Lim, and K. Sollins, "Dependency-based distributed intrusion detection," in *Proc. of DETER*, 2007.

[15] C. Livadas, "Collaborative malware detection: From theory to practice." December 2008.

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

[17] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *Proc. of SIGCOMM*, 2001.

[18] M. Castro, M. Costa, and A. Rowstron, "Debunking some myths about structured and unstructured overlays," in *Proc. of NSDI*, 2005.

[19] A. Ganesh, A. Kermarrec, and L. Massoulie, "Scamp: Peer-to-peer lightweight membership service for large-scale group communication," *LNSC*, vol. 2233, pp. 44–55, 2001.

[20] A. J. Ganesh, A.-M. Kermarrec, and L. Massoulié, "Hiscamp: self-organizing hierarchical membership protocol," in *Proc. of EW*, 2002.

[21] L. Massoulie, A.-M. Kermarrec, and A. Ganesh, "Network awareness and failure resilience in self-organizing overlay networks," in *Proc. of SRDS*, 2003.

[22] V. Vishnumurthy and P. Francis, "On heterogeneous overlay construction and random node selection in unstructured p2p networks," in *Proc. of INFOCOM*, 2006.

[23] V. Vishnumurthy and P. Francis, "A comparison of structured and unstructured p2p approaches to heterogeneous random peer selection," in *Proc. of ATEC*, 2007.

[24] J. Leitao, J. Pereira, and L. Rodrigues, "Hyparview: A membership protocol for reliable gossip-based broadcast," in *Proc. of DSN*, 2007.

[25] "The network simulator - ns-2 (ver. ns-2.29)," 2006. http://www.isi.edu/nsnam/ns/.

[26] J. S. Kong, J. S. A. Bridgewater, and V. P. Roychowdhury, "Resilience of structured p2p systems under churn: The reachable component method," *Comput. Commun.*, vol. 31, pp. 2109–2123, 2008.

[27] "Nlanr active measurement project." http://amp.nlanr.net.

[28] H. W. Hethcote, "The mathematics of infectious diseases," *SIAM Review*, vol. 42, pp. 599–653, 2000.

[29] T. C. Varun, V. Kacholia, S. Sankararaman, J. M. Hellerstein, and P. Maniatis, "Induced churn as shelter from routing-table poisoning," in *Proc. of NDSS*, 2006.

[30] S. Saroiu, P. K. Gummadi, and S. D. Gribble, "Measurement study of peer-to-peer file sharing systems," in *Proc. of MMCN*, 2001.

[31] R. Bhagwan, S. Savage, and G. Voelker, "Understanding availability," *LNCS*, vol. 2735, pp. 256–267, 2003.

[32] F. Bustamante and Y. Qiao, "Friendships that last: Peer lifespan and its role in P2P protocols," in *Proc. of IWCW*, 2003.

[33] S. Sen and J. Wang, "Analyzing peer-to-peer traffic across large networks," *IEEE/ACM Trans. Netw.*, vol. 12, pp. 219–232, 2004.

[34] D. Stutzbach and R. Rejaie, "Understanding churn in peer-to-peer networks," in *Proc. of IMC*, 2006.

[35] D. Liben-Nowell, H. Balakrishnan, and D. Karger, "Analysis of the evolution of peer-to-peer systems," in *Proc. of PODC*, 2002.