The Hive Mind: Applying a Distributed Security Sensor Network to GENI

GENI Spiral 2 Final Project Report

Sean Peisert (Principal Investigator)
peisert@cs.ucdavis.edu

Steven Templeton (Lead Software Architect)
templets@cs.ucdavis.edu

Department of Computer Science
University of California, Davis

http://hivemind.cs.ucdavis.edu

September 4, 2013
Version 1.0

Overview

Security monitoring using a method inspired by ant foraging behavior

Effective security event monitoring and response, without using a significant amount of system resources or otherwise negatively impacting operations is a key challenge in computer security. Developing a tunable and lightweight method—one that autonomously and efficiently coordinates detection and response, tasking those actions where they are needed, for the needed situation, when they are needed—would be a great improvement over the many existing resource-hungry solutions such as host and network-based intrusion detection systems that in effect look for all things, at all times, in all places. Such systems consume energy, network bandwidth, disk space, and processing power that would otherwise be available to support primary business functions, and decrease overall operating costs. Such "monolithic" solutions are also particularly ill-suited to wireless sensor networks, control systems [Tem11], network testbeds such as GENI [Bis09], or any other application where system resources (e.g., processor, battery, memory, communications) are limited and disruption of operations, perhaps even slightly, unacceptable.

Biological systems in nature have been shown to find and exploit needed resources or take needed actions efficiently and to do so based on few very simple rules. These systems are decentralized; individuals operate autonomously; and are resilient to disruption. Two examples are how honeybees direct other bees to sources of pollen, or how termites build tower mounds. Communication in social insects, and in particular how ants communicate and direct other ants to sites of food, invaders or other needed ant-world tasks, all served as inspiration for the *Hive Mind* monitoring framework. However, as a wise person once stated, the most important element of using a metaphor is the ability to know when to stop using the metaphor. Therefore, some of the

key aspects of biological systems that we have researched on the way to our implementation are the ways in which strictly adhering to certain biological traits is useful at times or counter-productive at other times. We also describe our results of these analyses in this document.

Our prototype system, for which the source code and documentation are currently publicly available under the GENI Project License,[1] has allowed us to test and evaluate the behavior of such an ant-based system, study the effect of a variety of actual ant behaviors, and to gain insights into alternative configurations and design considerations. Several iterations of the system have been developed since the Hive Mind project began in 2009, and further development continues.

Unlike traditional network [HDL+90] or host-based [KFL94] intrusion detection methods, the Hive Mind project explores a distributed method based on mobile code concepts and swarm intelligence. Our goal was to provide a lightweight, tunable, distributed detection method adaptable to changing threats while allowing suspicious activity to be communicated across hierarchical layers and to humans-in-the-loop who can direct actions when needed. This differs even from network intrusion detection systems that perform hierarchical alert correlation [VVKK04, ZHR+07] in numerous ways, particularly in the way that the leaf nodes of the hierarchy can be very lightweight, specialized sensors.

Conceptually, our system is related to neural networks [Tur48] and other swarm intelligence models [BDT99], such as Ant Colony Optimization (ACO) [DS04] and Particle Swarm Optimization (PSO) [ESK01] but contains important differences. For example, our approach is more of a distributed, parallel real-time resource coordination algorithm than many existing swarm algorithms, which are closer to being optimizations of some sort: shortest path, clustering, minimum error, etc. Also, our approach is also related to artificial immune systems [FPP86, SHF97] in the way that resources are directed, however, digital ants do not currently contain immune system's primary function of identifying self from non-self.

We are incorporating expansions into the basic Hive Mind prototype based on much of what we have learned about how animal communication directs actions, and how communication methods are adapted to a particular environment. Examples include, e.g., "swarming" in wasps, and "calling" in birds or wolves. In nature, each communication method is better adapted to particular situations. One of our most significant challenges is understanding the efficiency of different animal communication methods when mapped to a virtual environment, and perhaps more importantly, how to construct and map a collection of real devices into a virtual environment in which the virtual-creatures can operate. Along with a description of the prototype system, many of the challenges and insights gained are provided in this report.

Key features/behaviors

Listed below is a brief overview of the key features and behaviors of the Hive Mind framework. Each of these are discussed in greater detail in the sections that follow.

[1] Hive Mind source code repository: https://github.com/UCDavisHiveMind

Monitoring and response: The "Hive Mind" method monitors activity and responds to events across a defined collection of "entities." In our GENI implementation entities represent independent hosts (either physical or virtual). In practice these could be any physical or logical objects that have observable attributes. Examples could include PLCs and other devices in an industrial control system, nodes in a wireless sensor network, computer files, database records, or even people.

Decentralized, autonomous system. The Hive Mind is a fully decentralized, autonomous framework. Local peer-to-peer communication drives coordination of detection and response resources. Once initialized, the Hive Mind system operates without requiring direct user control. Alerts may be sent to administrative personnel keep them informed of significant findings and when specific responses must be externally managed or when the system may require manual retuning of key parameters. New detection and response algorithms can be added to the system and will automatically spread throughout.

Efficient resource use. A common property of most security monitoring systems is to look for "all things, at all places, at all times." While this can result in fast detection times, it can result in extreme consumption of system resources. Alternatively, the Hive Mind method directs detection resources to where they are needed when they are needed. This helps the system avoid spending much of the time looking for problems where they are not occurring.

Mobile sensors. In the Hive Mind system, mobile sensors, "Ants," pass between peer entities (e.g., monitored computers). Each Ant represents a sensor function to be executed. Ants move through the system checking for specific conditions to report or to trigger actions, to leave information to direct other Ants to a particular region, or to be so directed. In this way the mobile sensor Ants are directed to different locations.

Lightweight sensors. Each time an Ant moves, it executes its sensor function it is imperative that the sensor functions be lightweight—that is, it should not use a significant amount of system resources or to otherwise disrupt operations. One of the challenges to the system is to decompose detection tasks into a collection of simple, lightweight tasks.

Detection speed. Because the Hive Mind is designed for low and/or tunable resource use, on average the time required to detect a problem will be longer that other systems that monitor "all things, all places, all times." The Hive Mind is not intended to detect attacks as (or before) they occur, but rather to detect, report, and respond to them after they occur. Because zero-day and other attacks are not detected by intrusion detection or anti-malware software, and "advanced persistent threats" are common, it is a good assumption that computers will be compromised, and that efficient after the fact detection of value.

Non-determinism. The Hive Mind framework coordinates random[2] actions. Because of this, the time to detect a particular activity is not deterministic, but will have an estimated time based on how aggressively the system is configured, and other activities occurring in the system.

[2] Although we use the term "random" we acknowledge that when referring to software generated actions or values these are pseudorandom.

Document Organization

Theory of Operation

The Hive Mind method uses "cybernetic" Ants to monitor state and events across a collection of "entities." In our implementation, these entities represent independent hosts (either physical or virtual). In practice they could be any physical or logical objects that have observable attributes. Examples include PLCs and other devices in an industrial control system, nodes in a wireless sensor network, computer files, database records, or even people. Each entity is referred to as a *Node*. The collection of Nodes is referred to as the *Hive*. The Hive is the world in which the Ants operate.

In order form a virtual environment for the Ants, each Hive Node is assigned a fixed position in a hexagonal mesh. Each Node will have 6 adjoining Nodes, the Node's neighbors. A Node's position in the Hive is determined by a best-effort similarity function that attempts to place Node's near other Nodes that are similar, where similarity is determined by a set of attributes that characterize the underlying host entity.

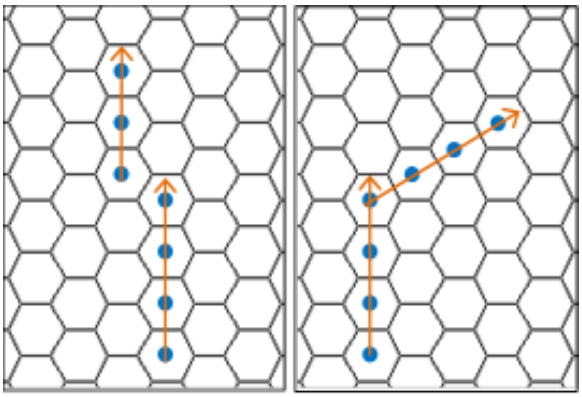


Figure 1: ***Ant foraging – directed random walk***

(left) Drift: Ant steps to the side, but doesn't alter direction vector. (right) Wander: Ant changes direction vector.

Each Ant is assigned a *Task*. In general, Tasks involve looking for some condition that requires an action, e.g., reporting the condition, responding to the condition, or updating a configuration value. Ants move through the Hive by moving from Node to neighbor Node using a direction-biased random walk. This gives each Ant a basic heading, yet allows a level of wandering about.

The level of wandering is determined by two parameters, *drift* and *wander*. Drift affects the likelihood that an Ant will step sideways, but without changing direction, while wander affects the likelihood the Ant will shift its direction. The Ant will perform its Task at each Node it enters. If an Ant's Task is successful, it selects a new heading and continues to travel in the Hive, but now on each Node it passes, it places a "marker" serving to direct other Ants toward the Node (and neighbors) where it successfully performed its Task. This is inspired by stigmergic communication in ants that leave pheromone trails. After a period of time, the Ant will stop placing markers and return to foraging.

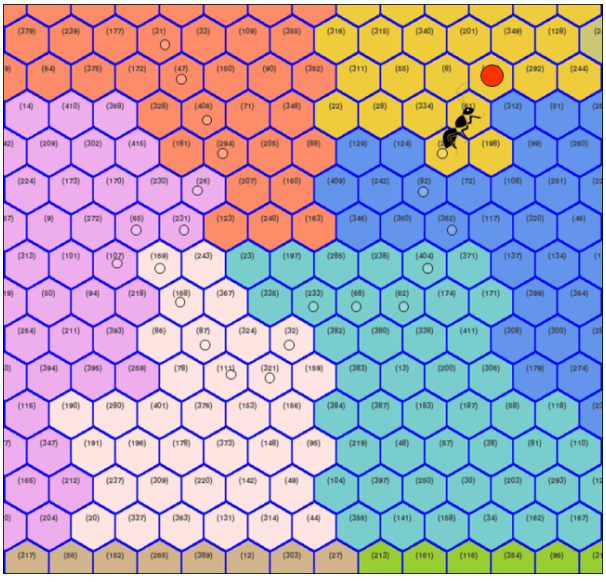


Figure 2: After a period of time foraging (light dots), an Ant finds a Node where it will successfully perform its Task (red dot).

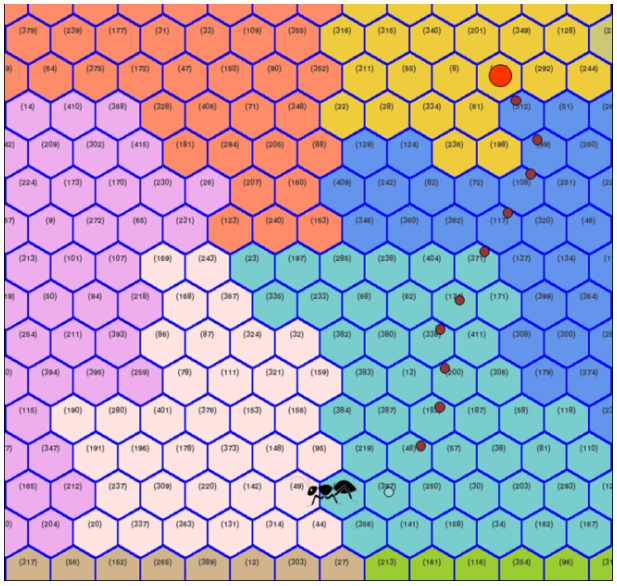


Figure 3: Ant leaves marker trail then returns to foraging.

Our Ants can also employ "Lévy flights" [Man83] when foraging. This is a common behavior seen in foraging creatures. Foraging occurs in a small region, and if nothing of interest has been found the creature moves some distance away and forages in this new area. If still unsuccessful, the creature again moves some distance away to forage, but the distance is increase over the prior movement. Each time the creature in unsuccessful, they move a greater distance away before foraging in an area. This behavior supports moving out of areas of low value. These "flights" are included in our foraging behavior.

When a foraging Ant enters a Node that has an existing marker, the Ant will tend to change its direction to follow the marker trail back to where the originating Task was successful. The Ant continues foraging (performing its designated Task) as it follows the path. If it is successful along the path, the Ant will also stop foraging and start creating its own marker trail.

This behavior results in Ants being directed to the region of the Hive where a Task was successful. This has the effect of directing Ant "resources" into this region. We assume that an entity that has one condition requiring action may have others, and that similar entities will have similar conditions requiring action. For example, servers located at the same site, running the same operating system version and application software, managed by the same system administrators, are likely to have common vulnerabilities, be targeted by the same attackers, or be compromised by the same malware. By locating similar Nodes near each other in the Hive, and directing additional Ants to that region (and the original Node), we are tasking resources to where they are more likely to be needed, enhancing efficiency and the success of detecting problems by intelligently coordinating resources.

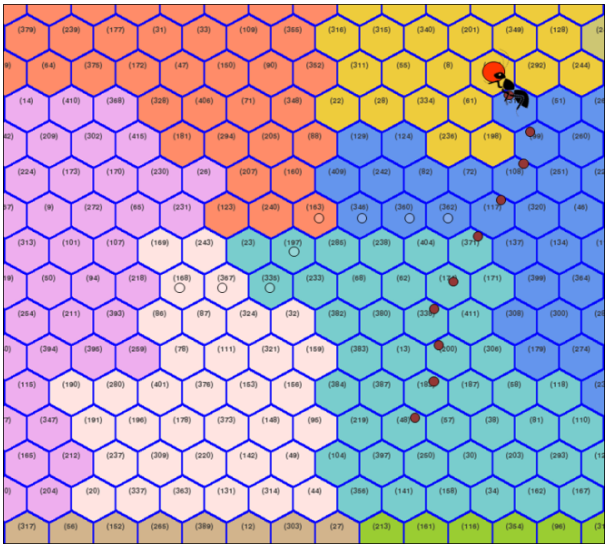


Figure 4: Another foraging Ant intersects the marker trail and follows it to the Node. If successful, it will in turn change state and begin "dropping".

Over time the markers are removed, much as actual ant pheromone evaporates. This limits the time that Ants will be directed to the target region, freeing them to investigate other regions of the Hive.

Task Selection

Real ants do not have a single assigned job. Although individuals develop a tendency to do a particular job, they will change and do any task that is needed (i.e., is not being sufficiently done) even across different morphological types (e.g., workers and soldiers). Our Ants have a similar behavior. If an Ant has reached the end of a marker trail and has not found its activity of interest, there is a probability that the Ant will be recruited to change its Task to that of the marker. Similarly, if an Ant has unsuccessfully searched for its Task for an extended period, it may change its Task to some other Task. These behaviors have the benefit of assuring that all Tasks receive some coverage and that a problem that has been observed receives enhanced monitoring.

An early version of the code, prior to the implementation of recruitment, increased the presence of selected Ant types by probabilistically creating new Ant instances upon a successful Task. This both increased the Ant population and the number of Ants to perform this Task. We reasoned that if a problem such as a worm was in the system, it would be valuable to generate an increased response – more Ants to find and respond to the threat.

Population Control

The monitoring system uses a metric of local population density to affect the birth and death rate of the Ants. In general, a region with a low population density will have an increased chance of Ant birth, while a region with a high population density will have an increased chance of death. This has a number of benefits including balancing the overall population size (which affects system resource use), prevents swarming of Ants to a particular region of depriving other regions of monitoring, and prevents swarming from overwhelming local system resources. Ants are born with a random Task, but based on Task selection as described above, will change based on system needs.

Automated Response

Each Task function may optionally have an associated mitigation or response function. If a response has been defined and the Ant's Task was successful, the Ant will initiate the response. For example, if an unauthorized port has been identified, the associated process can be killed. If an unauthorized user account has been detected, the account can be disabled. If a service with a known vulnerability has been detected, that file may be patched.

Detection of Local vs. Global Activity

The Hive Mind method exclusively uses local assessment. For a network of hosts, this implies host-based detection. The monitoring functions only can determine information about the local node. Without a central site overseeing activity, detection of global events is problematic. Independently detecting that the nodes are being used in a distributed denial-of-service, is part of a botnet, or is being compromised by worm-like activity requires global information. We address this problem using a hierarchy of Task functions.

The node manager function on each node can use a history of which types of markers have been deposited on the node to support detecting global activity. When Ants report the same type of activity via marker dropping, Ants with Tasks querying multiple reports of the same activity type, can transmit this situation to other nodes by dropping markers for this Task. Similarly, if Ants using this Task identify 'multiple report type activity" they can in turn report the same, but with a greater level indicator. The more that reports of activity with higher-level indicators are observed, the higher the likelihood that this is a global event will become. At some point the level will be high enough to infer that the base activity is widespread in the system and should be reported.

We are also currently evaluating an alternative to this model in which there is a Special Ant that simply patrol the Hive, paying attention to the marker trails it crosses, but not changing its state to "following." These Special Ants in effect count the number of times they see a particular Task being marked, and upon reaching some threshold, becomes "excited" and will emit an alert that a particular condition needing attention is being widely seen.

Layered Organization

Our ant-inspired method can be applied to any monitoring or sampling task. This requires analysis of the problem domain using a multi-layered approach.

Physical layer

The physical layer is composed of all the monitored devices or other entities, and all contextual information about the devices and relations between the devices. For example, the nodes in a wireless sensor network, device characteristics, performance characteristics, operational requirements, type of data collected, and peer relations (e.g., which nodes are in radio contact with which others).

Logical layer

Assigning and maintaining logical neighbor relations such that entities with similar, significant characteristics are near each other is a key requirement for system performance. These relations are defined in the logical layer. This is the virtual space in which the Ants forage. The logical layer is constructed using (a) a mesh specification, (b) similarity/dissimilarity metrics, and (c) a neighbor assignment method. These can vary based on the particular application. We typically use a hexagonal (6-way) mesh specification. It has better mathematical properties than 4 or 8 way meshes that suffer respectively from unequal radial path lengths, and non-intersecting paths (black/white queens problem). The specific metrics used to determine similarity and assign neighbors can be as varied as those used in data clustering and should be selected to reflect the significant characteristics of the monitored entities. Neighbor assignment is somewhat different from clustering in that here it involves arranging entities in the cells of a matrix rather than at arbitrary coordinates in some space.

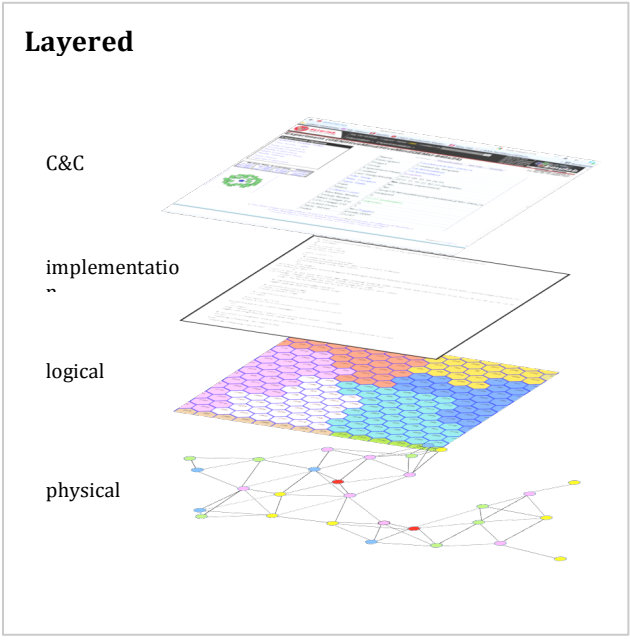


Figure 5: Layered Organization

The physical layer is the actual collection of objects to include in the Hive and any natural relations between them.

The logical layer reflects a mapping of the objects to a "topography" that places objects with similar characteristics near each other.

The implementation layer instantiates the logical layer. This will reflect the

The Command and Control layer is composed of the user interfaces and software tools used to monitor, control, and maintain the implementation.

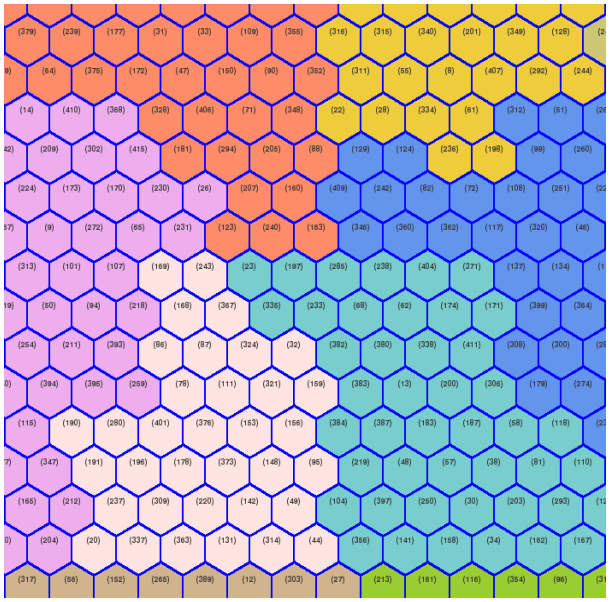


Figure 6: Logical layer maps monitored entities to Nodes arranged to maximize similarly between nearby entities, i.e., Nodes are grouped based on similarity.

Implementation layer

Implementation details will vary based on the operational requirements of the monitored system. For example, this may be done using actual network messages passed between entities, or as requests for sensor data and the results sent from/to a central process controlling the activity. An implementation must consider resource constraints of the monitored system: minimize use of battery power, network bandwidth, local memory, processor power, etc. This layer also includes the collection of Tasks to assign to the Ants (i.e., what data is to be collected), and the ratios of different types of Ants.

C&C layer

Although the monitoring method can be implemented as decentralized and autonomous, there is often a need to start/stop, change configuration options, push out new sensor functions, etc. Also, there is often a need to receive information about significant activity that has been detected and any actions taken by the system (e.g., remediation of detected problems). The command and control layer represents that collection of software/hardware used for this function. This master process and associated server is referred to as the Queen.

The Hive Mind Prototype

The prototype system developed for this GENI[3] Spiral 2[4] security project was designed to monitor either a ProtoGENI "slice" (i.e., an experiment), or to monitor the "slivers" provided by an Aggregate Manager.[5] Our testing primarily used the DeterLab testbed [BBK+06] on QEMU-based virtualized nodes [Bel05]. This is an Emulab-based system [WLS+02] and is a federated with the ProtoGENI testbed [Flux]. Our tests typically used 100 to 400 virtual nodes; the maximum size tested was 1024 virtual nodes.

The Hive Mind system is primarily a host-based event monitoring system, each monitored node being part of the "Hive." Once initiated, it operates autonomously, detecting, reporting, and responding to events without direct supervision. Through a "Queen" process, the system is initiated, and new Ant types can be created. Tools for visualization of Ant behavior and administrative support are also provided. These are described below. Additional information or installing and running the monitoring system can be found in the Hive Mind User Guide [HMDoc].

All primary Hive Mind code is written in the Python language. Some supporting tools are written in Bash shell scripts or Perl.

The "Hive"

The aggregate of monitored nodes (e.g., GENI experiment hosts) is referred to as the Hive. A process to manage Ant movement and execution of the Ants' sensor functions is initiated on each node.

Ants

The messages passed between Hive members are called Ants. Each Ant is assigned a designated sensor function that determines what specific actions the Ant takes when it arrives at a new node.

The "Queen"

The Queen refers to a designated host where telemetry from the Hive nodes will be collected, and typically where administrative processes are run. Typically this will be a different node from those monitored.

The Queen's primary functions are to initiate the monitoring processes in the Hive, collect telemetry from the Hive nodes, and to issue new Ant-types into the system.

When nodes are initiated, their neighbors (the other nodes with which the node will directly communicate) are assigned. Neighbor assignment is based on an algorithm that determines the relative similarity/dissimilarity between the nodes. Nodes that are more similar to each other will tend to be neighbors. This is a static assignment, however if node configurations changed significantly, the Queen process could be rerun to assign new neighbors.

---

[3] The GENI Project: http://www.geni.net
[4] GENI Spiral 2: http://groups.geni.net/geni/wiki/SpiralTwo
[5] GENI Aggregate Manager API: http://groups.geni.net/geni/wiki/GAPI_AM_API

Because the system runs autonomously, the telemetry data is not needed for normal operations, however it is required for visualization of the Ant and Hive activity, and for debugging purposes. Telemetry data is sent from each Hive node to a remote "syslog" server [Lon01] running on the Queen. This is captured to a designated Hive Mind log file. This log data is available for real-time or batch review of the system.

Task functions

We developed and tested a collection of Ant Task functions applicable for a variety of scenarios. Our primary focus in using these scenarios is to detect changes to a baseline configuration of a UNIX host. If no baseline exists, the Ant Task function will create one. Functions include:

- Unexpected user account
- Unexpected change in account privileges
- Unexpected process
- Unexpected interface
- Unexpected open network port
- Unexpected service
- Free disk space less than a defined value
- Change in free space in excess of a defined value
- Determination of a particular file exists on the Node

Visualization

We developed two visualization tools as well: one to display Ant movement and the state of the Hive nodes, and a second to show Ant coverage in heat-map format.

The Hive Mind Visualizer displays a grid of hexagonal shaped nodes representing the Hive nodes. The color of the hexagon indicated group membership of the node, for example the GENI Aggregate Manager providing the nodes, or the sliver provided. Ants are displayed as centered dots in the hexagon assigned to the node in which the Ant currently exists. The color of the Ant represents its state: foraging, dropping, following, or idle, also if the Ant is newly born, died, or has been reassigned to a different task. Small red dots at the bottom of the hexagons represent targets (i.e., events for the Ants to detect).

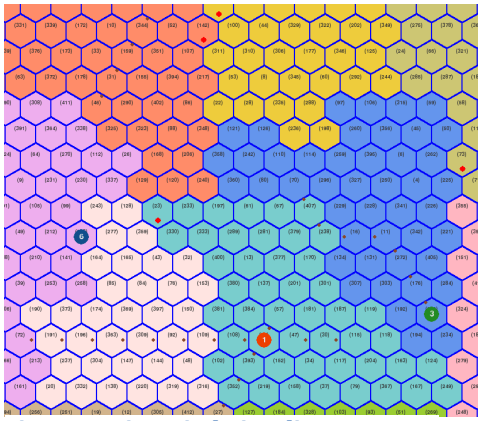


Figure 7: Hive Mind Visualizer

The visualizer will play through a saved trace extracted from the collected telemetry data. Typically all activity is viewed, but it is also possible to view the trace of an individual Ant. The display can have its speed increased, decreased or paused. A video clips of the viewer in action is available at http://www.youtube.com/user/UCDavisHiveMind

Similar in operation to the visualizer, the Heat Map Viewer

shows the relative distribution of Ants over time. The darker the cell color indicates the more times the node has been visited. From this it is easy to see how balanced the Ants are in visiting each Hive node.

Additional details on the visualizer, the heat map viewer and their operation can be found in the Hive Mind User's Guide [HMDoc].

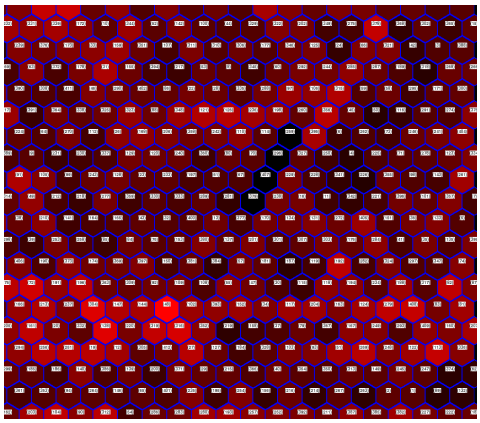


Figure 8: Hive Mind Heat Map Viewer

Support tools

We created a large collection of administrative support tools as well. The most significant are those that check to see if Hive node processes are running, to start and stop them, and to allow user manipulation of Hive node process parameters, for example whether or not the node should monitor local Hive Ant density and create/destroy Ants to manage population levels (vs. running with the existing set of Ants).

Another tool allows automatic generation of events for the Ants to find. This creates simulated attacks in the Hive based on different distribution patters of event type, location, and clustering.

Other tools include the injection of crafted Ants for testing, killing all Ants in the Hive, and querying the status of the nodes.

As an alternative to the command line tools, we created an HTML browser-based user interface. This is hosted on the Queen node and provides an alternative way to control the system and monitor its behavior.

As with all the other software components, details of these tools can be found in the Hive Mind User Guide [HMDoc].

Security and Resistance to Attack

Authentication

In order to prevent the introduction of malicious Ants, Ant Task code and parameters must be authenticated. When an Ant is received by a Hive node, the Ant's *Task id* is checked against a list of known tasks. If the Task id is unknown the Node will request the associated code. Depending on configuration, this can be either directly from the Queen, or from the Hive Node corresponding to the Ant's prior location.[6] In either case, if the requested Task id is unknown, the Queen (or Node) sends a reply indicating this, the Ant is deemed invalid, killed and (optionally) an informational message sent. Otherwise, the Task code is sent to the requestor, validated, added to the cache of Ant Tasks, and the Ant processed normally. If the Task code fails validation, it is dropped, the Ant killed and (optionally) an alert sent.

Task code is validated by use of digital signatures. When new Ant Task code is distributed to the Hive, either at system initiation or upon creation of new Ant Tasks, the Queen uses its private key to sign a hash of the Task code. This is bound to the code. The Hive Nodes use this plus the public key of the Queen (also distributed upon initialization) to validate the Task code.

Because Ants of the same Task id may have different parameters (e.g., thresholds, target names) it is important for them to also be validated. For parameterized Ants, the parameter will similarly be signed and validated.

Typically, Ant messages are not signed by the Hive Nodes forwarding them. Unless the Hive nodes were to generate or modify the Ant task code or significantly self-modify their parameters, it is not necessary for Ants to be signed by the Nodes. This eliminates the need for each node to have a private key and greatly simplifies key management, computational overhead, and overall system complexity.

However, if desired, when Hive Nodes are initiated, they could generate and share session keys with their immediate neighbor and the Queen. This would allow the Node process to sign each Ant they send and to provide credentials validating log messages and alerts. At the cost of additional computational overhead, this would greatly protect malicious Ant messages from being processed by the system. However, if the device on which the Node process was running were compromised, it would still be possible to inject malicious Ants, log and alert messages. They would not be able to place malicious code in the system. This is discussed further in the section on attack resilience.

[6] If the prior Node were to have performed the Ant Task, it must have a valid copy of the code. This creates a head/body propagation method, and is how the system implements "mobile code."

Attack Resilience

Attacks on the Ant-system can be either by injecting messages onto the network, by maliciously modifying the behavior of a Hive Node process, or by generating false alerts or log messages.

The monitored Nodes are assumed to be of susceptible to compromise and malicious activity originating from them a factor to be considered in the resiliency of the Hive Mind. The system design affords no special privileges to Node processes. Attacks that inject messages from Node processes are no different from injection attacks sourcing the messages from other locations. Without loss of generality, we will consider the injection point to be from a compromised Node process and treat these attacks as a special case of Node process compromise. In the following discussion unless specified we assume the Node process in not compromised.

Ant injection

Regardless of the source, Node processes will reject without performing the message's stated function messages that are not properly signed (see section on Authentication). Depending on the configured threshold, the Node process may also generate an alert reporting receipt of an invalid message. Attacks that naïvely inject invalid messages are effectively no different that brute force attacks that flood the network with random traffic targeted toward the Node processes on the monitored hosts. The impact to the Node processes is the cost of validating the messages.

Hive Node processes are typically configured to reject messages from sources other than their assigned neighbors or the Queen. This reduces the validation overhead from invalid messages, but can be somewhat diminished if UDP is used as the transport protocol and the message source address is spoofed. This does not allow bogus messages to be accepted, just that they must be validated.[7]

Because Ant Task processes require valid signed Ant Task functions and their associated parameters, only two types of injection attacks are of interest: replay attacks where a valid signed message is collected and retransmitted unaltered at a later time, and attacks where the modifiable (unsigned) fields of an Ant-message are maliciously changed and forwarded from a compromised Hive Node.

Replay Attacks

Control-messages are intended for a single use by a specific target. Because these messages include a target Node id and are time-stamped, Node processes can determine if a message is a replay, reject it, and generate an alert.

However, Ant-messages are passed multiple times across many different Nodes. While time-stamping Ant-messages could limit the time frame in which they could be replayed, this cannot prevent it. A malicious Node process could retransmit them, or because they are only partially

[7] Note: restricting accepting messages to only assigned neighbors or Queens can increase fragility in that messages from the Queen must come from the originally configured IP address (or an alternative), and can somewhat slow dynamically reconfiguring neighbor relations.

signed, inject modified versions. However, because Node processes create new Ant instances (i.e., generate new Ant-messages) from the known, signed sensor functions in their cache, the notion of replay of Ant-messages is moot.

As stated earlier, Node processes will only accept messages from its neighbors and Queen. To be accepted, replayed messages must come from a Node process's assigned neighbors. Except in the case where UDP is used and the attacker knows a Node process's neighbors, unless the attacker is sending the messages from a compromised neighbor or Queen, these replayed messages would be rejected and dropped without further validation, eliminating most opportunities for replay attacks. Those requiring a compromised, malicious Node process are discussed below in that context.

Parameter Manipulation

Ant Task parameters are by default signed by the Queen. At issue are the parameters of the Ants that change as the Ant moves through the Hive. Examples include age, type, direction, and state.

A commonly suggested idea is to require Hive Nodes to digitally sign the Ant messages they send to their neighbors. This would provide an additional layer of protection, but will not absolutely protect against a Node creating corrupted Ants. If the attacker had full privileged control of a Node, the attacker could use the Node's key to sign Ant messages.

The inherent resiliency and peer-to-peer nature of the Hive Mind allows parameter manipulation attacks to be readily mitigated. For example, Attacks that attempt to cause flooding or clearing Ants from a region will be mitigated by the systems Ant density control mechanism. The likelihood that an Ant will die is directly proportional to Ant density; new Ant creation is inversely proportional.

The peer-to-peer nature of the Hive Mind allows Nodes to monitor the behavior of their neighbors. If a Node receives an Ant that has parameters exceeding configured values (e.g. a lifespan counter larger than the configured initial value) can alert the Queen and black-list the compromised Node, instantly and effectively removing it from the system. In some cases that malicious Node could alter parameters to be within their proper range, but be logically or statistically invalid. Changing the direction vector such that the Ant could not have come from the neighboring node, is one example. Another example would be when an excessive number of Ants received from the malicious Node were excessively of a particular mode (e.g. all are in dropping mode). Similarly strategies to monitor the behavior of neighboring nodes for signs of compromise can be developed. Subtle changes to the Ants parameters that are not detected by neighboring nodes should be possible, however because this would only affect the Ants that one Node would process, or the Ants that the one Node could inject, the effect on the overall system would be negligible.

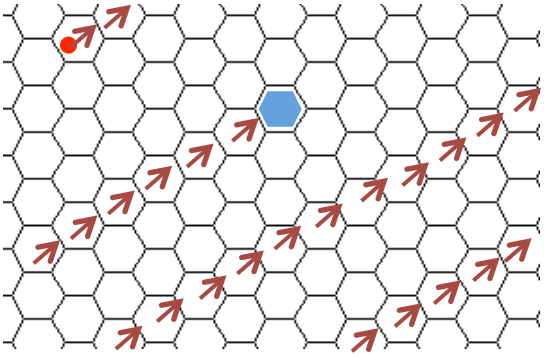
An interesting example of how severe the effects of parameter manipulation could be and why peer monitoring is important involves controlling the Ant state of those being sent from the compromised node. Consider if all Ants leaving the Node were changed to 'dropping', the direction vectors changed to lead out in random directions, and the lifespan set to be large or immortal.

Marker paths leading back to the compromised Node would envelope the entire Hive. This would in effect cast a net to continually draw all Ants into the compromised node, disrupting the entire Hive. Because we assume that the compromised node could sign the malicious Ant messages, authentication will not mitigate this attack; nor would the Hive's population control measures. Increasing the rate of Ant creation would create more Ants to immediately start following a marker trail. Killing Ants as they amass in the vicinity of the compromised Node would mitigate denial of service effects, but also disrupt monitoring. However, by enabling Hive Nodes to monitor for invalid parameters or other inappropriate behavior and to blacklist any misbehaving Nodes, the attack could be easily mitigated (see figure 9).

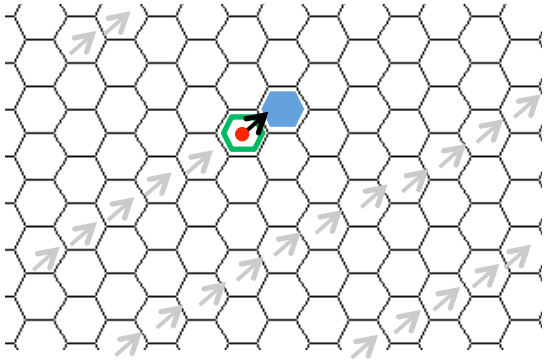
Other security concerns

Multiple compromised nodes could work together to have a greater effect on the Hive. In the simplest case a particular attack could be arithmetically magnified. More interesting is when malicious Ant actions are coordinated. For example, expanding on the above attack where a malicious Node sends out dropping Ants to direct as many Ants as possible to the compromised Node when, if the malicious nodes were to generate a route which would cause captured Ants to proceed round and round, without the situation being detected. As in the other attack, because a low Ant density situation would cause additional Ants to be created, this attack would ultimately not be likely to have a great impact.

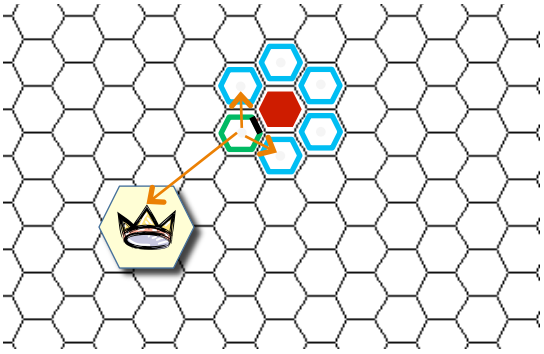
Because log messages from the Ants are sent to the Queen server using syslog, an unauthenticated UDP based protocol, injecting log messages is also a feasible attack. Because these messages are not alerts, nor are they used for operations, only for telemetry for studying the Hive behavior, such an attack would not have a security impact on the system. Where available, UDP-based syslog could also be replaced with syslog variants using TCP and/or digital signatures.



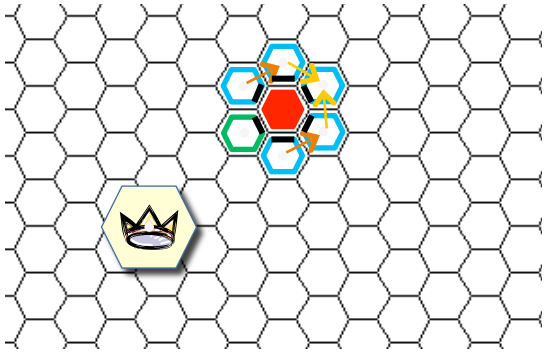
(a) Modified Ant sent by malicious Node, M, to create an excessively long path, directing any Ant intersecting the path back to M



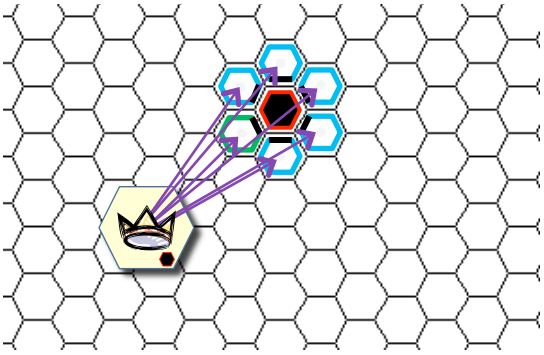
(b) Neighboring Node, P, receives the Ant and determines it and M are malicious.



(c) P blacklists M and sends alerts to its neighbors bordering M and the Queen.



(d) Neighboring Nodes blacklist M and forward alerts to their neighbors bordering M.



(e) The Queen adds M to the blacklist and sends the neighbors of M new neighbor assignments to remove M from the hive.

Blacklisting

When a Node process determines that it is receiving invalid messages from one if its neighbors, the Node process can temporarily blacklist the misbehaving neighbor, inform the blacklisted Node's neighbors that it has been blacklisted, and generate an alert to the Queen. The Queen can send the Nodes neighbor reconfiguration messages allowing them to link around the compromised Node, effectively removing it from the system.

Figure 9: Illustration of Ant validation and blacklisting of malicious Nodes

Performance and Efficiency

An efficient security event monitoring system uses a minimum of system resources in its operation. In relation to the Hive Mind, this has two aspects to consider: (1) the number of Ant Task executions and Ant movements (or other communications), and (2) the resource requirements of the Ant Task executions and movements. In the first case, for efficiency, we seek to execute Ant Tasks functions only where, when and for what they are needed. We consider other Ant Task executions or Ant movements to be wasted resources[8] and avoided. In this context, *efficiency* is a measure of resource use relative to the number of Tasks successfully completed. An efficient system minimizes the number of Ant movements and Ant Task executions as well as both their resource requirements. Without loss of generality we exclude the resource requirements of Ant Tasks and movements from the efficiency analysis related to Ant foraging behavior. Although a relationship between the two cases exists, we analyze efficiency of how Ant resources are directed separately. Later we discuss how Ant Task functions and can be structured for improved efficiency.

Apart from efficiency we considered *performance* which we define as the time for an Ant to successfully find and complete its Task, or more generally the time for a collection of Ants to successfully find and complete a collection of Tasks. Because actual time is a function of the particular environment and software implementation, we use the number of "ticks", that is the number of Ant movement cycles that occur. This allows us to look at the performance of the method, not a particular implementation.

In addition to traditional Ants, Wasps and Callers we developed a variety of alternative Ant foraging behaviors to use in evaluating the Hive Mind system. These methods were used in tests in our simulated environment and evaluated analytically. These include:

- Ants: traditional foraging
- hAnts: Ants that follow a predefined space-filling curve over the Hive.
- sAnts: Ants that follow a predefined linear traverse of the Hive.
- hBees: Ants that use traditional foraging until they successfully complete a Task, then switch to a Hilbert traversal to look for more Tasks in the same region.
- Bees: Ants that use traditional foraging until they successfully complete a Task, then switch to a method that attempts to look for more Tasks in the same region by sharply increasing the Ant's drift and wander rates.
- Wasps: Ants that use traditional foraging, but do not leave linear marker trails. Instead they generate a cloud of markers in one or more rings of neighbors around the Wasp.
- Callers: Non-mobile Ants that self-monitor, and upon a successful Task, increase the rate of self-monitoring and inform their immediate neighbors of what Task was performed. These will in turn check for the this Task.
- rAnts: traditional foraging, but no communication, essentially a random search.
- ATAPAT: All things, all places, all times; analytic variant that runs all sensors on each Node, every tick.
- ATAPST: ATAPAT variant that runs all sensors on each Node, but only after a waiting period.

[8] It can be argued that these are not wasted resources, but insurance that Tasks will be identified. If so, then efficiency includes minimizing the premium costs.

Analysis

Analytic estimates of expected behavior based on statistical analysis matches what we have observed empirically from simulations and live tests run on collections of hosts. These results are reviewed in the following sections.

Performance

We consider performance to be a function of the time taken to find all outstanding Tasks in a test. Given a particular test scenario, the measure of performance, P, is equal to the ratio of the number of ticks from start, t_0, until completion, t_F, with $t_F > t_0$, to the number of Tasks in the test, where T is the non-empty set of Tasks, T.

$$P = (t_F - t_0)/|T|$$

Thus as the value of P decreases the higher the measured performance will be.

As defined, Performance does not consider the amount of resources (i.e. Ants) used. Increasing the number of active Ants during a test will in most situations decrease the time to service all the Tasks, and therefore improve performance. However, this does not convey the performance of the configured method separately from the level of resources assigned to the problem and can misrepresent the speed of the method.

Let A_t be the set of Ants active in the Hive at time t, and a = sum(t_0,t_F):$|A_t|$ / (t_F-t_0), be the effective number of Ants during the test. We define the Performance Factor to be:

$$PF = a \cdot P$$

The expected number of ticks to service all Tasks can be estimated for different foraging and communication methods. Performance can be bracketed between less complex instances. Consider a set of tests where random sets of target Tasks are created on Hive notes at t_0.

At one extreme, on each tick, all Hive nodes execute all Task Functions. Assuming no Task occurs more than once on a particular node, all Tasks will be serviced on the first tick, resulting in

P_{ATAPAT} = 1 / |T|.

Performance increases with the number of Tasks to be serviced. However, with n = number of Hive nodes, k = number of distinct Task functions defined, then:

PF_{ATAPAT} = (n·k)/|T|.

PF still increases with the number of Tasks to service, but is reduced by the size of the Hive and number of distinct Tasks the system processes.

Because all Tasks will be found on the next Tick, neither the number of tasks set, nor the number of trials affect performance. In these cases the prime factor in efficiency is the number of Tasks to expect over a period of time. Without loss of generality we can consider the behavior of a single Node, and because all Tasks set on the Node are found during the same Tick, we can ignore specifics of the number of Tasks set, and consider the expected number of Ticks between Tasks. Clearly if all

Task types appeared each Tick on each Node, this method would be very efficient. There would be no wasted actions to find and service the Tasks. However, when Tasks occur more sparsely, efficiency suffers.

Let p(t) be the probability at any Tick of a Task being set, and assuming the events are independent, E(t) = 1/p(t) is the expected time between Tasks occurring on a Node. If each Node checks for k tasks, then each Node will be expected to perform k*E(t) checks per Task. The number of checks which do not service a Task, increases as the probability of a Task occurring decreases. Increasing the clock speed, or increasing the number of Task types will also increase the number of wasted checks. In practice, if the Tasks represent security events, the probability at any moment of one occurring is extremely small. This results in very low efficiency.

The ATATSP variant, which waits a number of cycles between checks will have improved efficiency – the rate of Task checking is reduced – but will result in a commensurate decrease in performance. If w is the number of cycles to wait between checks, and we assume that Tasks may occur at any time uniformly distributed between checks, then the expected time a Task must wait to be serviced will be w/2. Thus performance will be

$$P = \sum_{\text{all trials}} \frac{w(t_F - t_0)}{2} = \frac{w}{2}$$

For standard foraging Ants, but without marker trails, servicing Tasks is effectively equivalent to a random search of the Hive. On each tick, $|A_t|$ nodes are checked. This is repeated until all Tasks are services. Assume $|A_t|$ is constant for all t. The estimated value of $(t_F\text{-}t_0)$ is then an analog to the Ticket Collector Problem in statistics. This is a harmonic series which can be estimated by

$$(t_F - t_0)\ (t_F\text{-}t_0) = |T| \cdot (0.577 + \ln |T|) \,/\, |A_t|.$$

This gives

$$P_{NCA} = |T| \cdot (0.577 + \ln |T|) \,/\, |A_t|,$$
$$PF_{NCA} = (0.577 + \ln |T|) \,/\, |A_t|.$$

Similarly to the previous cases, performance increases with an increase in the number of Ants, but is dominated by the number of Tasks to service.

Coverage

A related metric is how long a collection of Ants takes to visit all nodes in the Hive. This characterizes worst-case expectation of how long a Task may wait to be serviced and with what variability.

ATAPAT: Trivially, all nodes are checked in exactly on Tick.
TC = 1, (Var = 0)

Scan: each Node is visited without re-visiting a prior node per cycle. A scan takes exactly Hive-size Ticks. The longest time any Task must wait will be the size of the Hive.
TC = Hive-size, (Var=0)

hAnts: because a Hilbert traversal is an ordered scan without repeating Node visits during a cycle, this is equivalent to a simple scan.

Ants (random scan): Foraging Ants traverse the Hive using a directed random walk, that is, the Ant is given a general direction to travel, but it may with varying probability move to the side, or alter its given direction. At one extreme the Ant follows a straight path and may loop forever without visiting all nodes. At another, the Ant repeatedly visits the same set of nodes in its initial region. Setting the movement parameters to perform between these behaviors, results in complete coverage approximating that of randomly selecting Hive Nodes to check. Assuming random Node selection, the time to visit all Nodes is equivalent to a "ticket collector" problem. Letting h equal the size of the Hive, this gives an expected time to visit all nodes to be:

TC = sum(i=1..h):1/h;

Using a harmonic series approximation gives TC = h * (0.577 +ln(h))

Varying the number of Ants shows that while the number of ticks decreases as the number of Ants increases, the total number of Ant movements remains nearly constant. Plotting a trend line suggests that this is slowly increasing. As the number of Ants increases the number of collisions increase; more nodes are rechecked. For example, with 1024 nodes, and 1024 Ants place on separate nodes, due to collisions where multiple Ants move onto the same node, the time to visit all nodes is about 7 ticks.

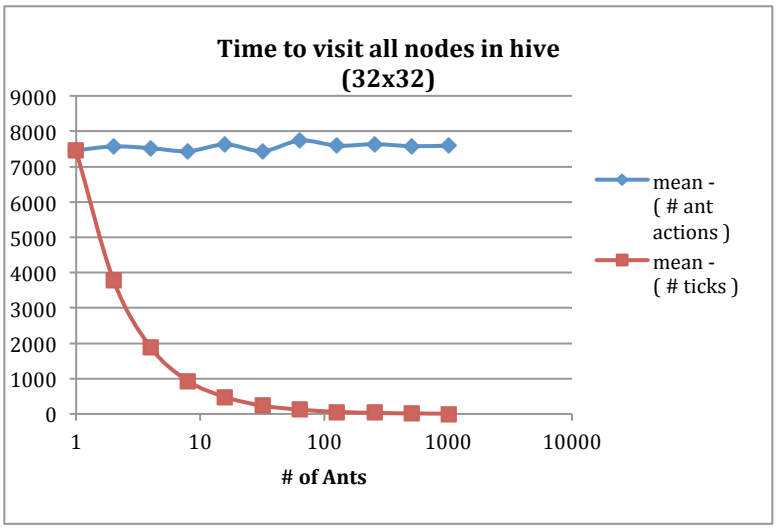


Figure 10: Time to visit all nodes in Hive decreases as number of Ants is increased, but total number of Ant actions slightly increases.

Callers : Equivalent to random scan.

Increasing the number of Ants decreases the number of Ticks to visit all Nodes in the Hive, but not directly as a multiple of the number of Ants. As a result of Ant collisions, the actual number of Ticks will be typically higher. As an extreme case, if a scan using two Ants was started with both Ants starting on the same Node and following the same path, rather than taking ½ the time, because both Ants check the same Nodes the number of Ticks would be unchanged.

This is also true of normal foraging Ants. Any Tick where two or more Ants visit the same Nodes will increase the time to visit all the Nodes. To illustrate this consider the case where all nodes in the Hive have an Ant on them. On the next Tick, all the Ants will move, but because the movement is random, many will move on the same Nodes. In fact, it will take about 8 Ticks for all Nodes to be visited. This can be seen by empirically, or analytically.

The time to visit all Nodes is still a ticket collector problem, but the number of boxes opened (Nodes checked) is decreased to accommodate the number of collisions. For example, rather than check 1024 Nodes, we check 1024 minus the number of collisions[9]. This decreases the likelihood that on any Tick previously unvisited Nodes are visited and extends the expected time to complete coverage.

An interesting implication is that while more Ants will decrease the expected time to visit all Nodes, or alternatively to service k Tasks, because the expected number of collisions rises with the number of Ants, the efficiency of the system decreases.

Cost

Let C be a vector representing the cost of a series of Ant actions in an experiment occurring between times t_0 and t_F, where t_0 is the tick the experiment begins and t_F is when the last Task is serviced. Each element C_i in C holds the sub-costs associated with the different types of resources used (e.g. CPU, communication bandwidth, memory, battery). The specific elements of C_i and their cost weighting will vary with the resource constraints of a specific implementation. Our evaluations focus on costs related to CPU and communications. This method can be extended to other types of costs.

Let C_t^P = processor cost, and C_t^C = communication cost of activity occurring at time t, and Q_t is a weighted linear combination of the associated cost elements. $Q_t = C_t * W$, where W is the vector of cost weights s.t. each element w_i in W >= 0 and sum(all elements of W) = 1.0. Then Q = sum(t_0->t_F): Q_t is then the weighted normalized cost of the experiment.

To distinguish the types of costs, let Q^C to be the cost of communications, and Q^P to the cost of servicing Tasks.

Initially we assume different cost types should be weighted equally, thus W = [0.5, 0.5], and $Q_t = (C_t^P + C_t^C) / 2$.

Each tick will have a number of communication costs and a number of processor costs. We assume all communication costs to be equal. As Ant movements are a message passed between Hive nodes, all Ant movements are a type of communication. We set all communication costs = 0.5. Similarly, we assume the cost of servicing all Tasks to have equal processor cost. For now, we set $C^P = 0.5$. Later we evaluate how design of the Task functions to split the cost between a "head" to determine if a Task may be present, and a "body" which makes a final determination and services the task as needed can improve efficiency and performance.

For normal Ants, each movement is followed by a Task check, thus the two can be considered a single cost = 1.0 per movement. Cost then can be characterize as the number of movements, and efficiency the number of movements required to service all the Task occurring during the experiment.

[9] A collision is when an Ant checks a Node already checked this Tick cycle.

Efficiency

Given the above characterization of cost, efficiency is the ratio of cost associated with actions that serviced Tasks to the total of all costs that occurred during the experiment, that is the sum of needed + wasted actions.

$$Z = Q_{NEEDED} / (Q_{NEEDED} + Q_{WASTED})$$

Again we first consider the boundary cases. At one extreme, when we know via an oracle where to execute Task functions, there is no waste. We have absolute efficiency.

$$Z_{ORACLE} = Q_{NEEDED} / Q_{NEEDED} = 1$$

For ATAPAT systems, no Ant movements or other communications in servicing Tasks occur, the $Q^C = Z^C = 0$. With ATAPAT all Task functions execute on each node on each tick, with *h* = number of Hive nodes, *k* = number of distinct Task functions defined, and *d* be the duration of the experiment, then:

$$Q^P_{ATAPAT\_TOTAL} = d \cdot h \cdot k \cdot (c^P + c^C)$$

Needed resources are a function of the number of Tasks that occur during the experiment.

$$Q^P_{ATAPAT\_NEEDED} = |T| \cdot (c^P + c^C).$$

Alternatively, rather than use a specific fixed integer value, we express the number probabilistically as the expected number of Tasks to service over the duration of an experiment of duration, *d*, ticks. Let p(t) be the probability of Task type t occurring on any particular node during on any tick. Let p = sum(p(t)) over all elements t of T. Then E(T) = d · h · p. Then the probabilistic cost efficiency estimate is

$$Z^P_{ATAPAT\_NEEDED} = E(T) = d \cdot h \cdot p$$

$$Z_{ATAPAT} = (Z^C_{ATAPAT\_NEEDED} + Z^P_{ATAPAT\_NEEDED}) / (Z^C_{ATAPAT\_TOTAL} + Z^P_{ATAPAT\_TOTAL})$$
$$Z_{ATAPAT} = (Z^P_{ATAPAT\_NEEDED}) / (Z^P_{ATAPAT\_TOTAL})$$
$$Z_{ATAPAT} = (d \cdot h \cdot p \cdot (c^P + c^C)) / (d \cdot h \cdot k \cdot (c^P + c^C))$$

$$Z_{ATAPAT} = p / k$$

For ATAPAT efficiency is increases with number of Tasks to be serviced and decreases as the number of Task types and size of the hive increases. Typically when monitoring for security events, p is very small, and the size of the hive is large, making this a very inefficient method. For comparison we define a reference set of parameters:

Size of the Hive:
n = 1000 nodes,
Number of distinct Task types:
k = 100.
Duration of experiment:
d = 1,000,000 (note: this is 10 ticks per second for about 28 hours)
Probability that any node will have a Task to service –

p = 0.0001 Tasks/node/tick, (note: given the above, this is about 1 Task per node per 15 minutes)

$E(T) = d \cdot h \cdot p =$ 100,000 Tasks

Putting this in perspective, we can expect 100 Tasks per node. Each node will execute 1,000,000 * 100 Task functions, or 10e6 Task functions / per Task found.

This gives a reference efficiency value of

Z_{ATAPAT_REF} = 10e-4 / 100 = 10e-6.

For ATAPST, where Task functions are only executed every w ticks (w > 1), efficiency is increased by increasing the expected number of Task discovered per check.

$Z_{ATAPST} = (Z^P_{ATAPST\_NEEDED}) / (Z^P_{ATAPST\_TOTAL})$
$Z_{ATAPST} = (d \cdot p \cdot h) / ((d/w) \cdot h \cdot k)$

$Z_{ATAPST} = (p \cdot w) / k$

With reference parameter w=10:

Z_{ATAPST_REF} = 10e-5 .

For normal foraging Ants, Ants move whether or not a Task is serviced, and for each movement there is a Task function execution. Thus $Q^C = Q^P$. With n = number of Ants,

$Q_{NCA\_NEEDED} = |T| \cdot (c^P + c^C)$
$Z_{NCA\_NEEDED} = p \cdot h \cdot (c^P + c^C)$
$Z_{NCA\_TOTAL} = d \cdot (c^P + c^C) \cdot n,$

then,

$Z_{NCA} = (p \cdot d \cdot h \cdot (c^P + c^C)) / (d \cdot n \cdot (c^P + c^C))$.

Defining the number of Ants as *a*, a fraction of the Hive size (e.g. 1 Ant / 10 nodes). Then $n = a \cdot h$ the efficiency formula excludes the size of the Hive.

$Z_{NCA} = (p \cdot d \cdot h \cdot (c^P + c^C)) / (d \cdot a \cdot h \cdot (c^P + c^C))$.
$Z_{NCA} = p / a$.

Efficiency is the amount of reward to the amount expended to reap it. Here p is the expected reward, and the amount of work is how cost of a worker times the number of workers. Efficiency can be improved by increasing the likelihood of reward, by reducing worker cost, or by reducing the number of worker, or some combination that results in a net gain

Expressing the number of Ants, n, as fraction of the Hive size, with the above reference parameter h = 1000, and a=1/10, and assuming the cost of a needed action is equal to the cost of one wasted:

$Z_{NCA\_REF} = 0.0001 / (0.10) = 1e\text{-}3$.

Efficiency is controlled by Ant density.
One observation is that efficiency is not related to the size of the Hive.

If we constrain the foraging path to a fixed sequential traversal of the Hive, we reduce the expected time to find Tasks, but do not improve efficiency. Efficiency is doing more with less.

Callers can be considered a variant of the ATAPST method, the difference being that the Task functions are executed at random intervals on an individual node basis. Also, the rate of execution is mediated by (a) recent successful Tasks, and (b) recent messages of successful Tasks from neighbors.

Callers, execute Task functions randomly based on a rate tied to the nodes "activation state". At higher activation states, Task functions are executed more frequently. The duration of the experiment, $d = d_L + d_H$, where d_L, d_H are the amount of time spent in low and high activation states. Let q_L, q_H be the probabilities of executing a Task function in each of the three activation states, with $0.0 < q_L < q_H < 1.0$.

$$Z_{NC\_CALLER\_NEEDED} = p \cdot h \cdot c^p$$

With static probabilities of executing a Task function:

$$Z_{NC\_CALLER\_TOTAL} = (q \cdot d) \cdot c^p \cdot h,$$

With variable probabilities of executing a Task function:

$$Z_{NC\_CALLER\_TOTAL} = ((q_L \cdot d) + (q_H \cdot d)) \cdot c^p \cdot h,$$
$$Z_{NC\_CALLER\_TOTAL} = (x_L + x_H) \cdot c^p \cdot h,$$

Where x_i are the expected number of Task function executions on a node over the period of the experiment.

The probability of transitioning from activation state AS(Low) to AS(High), p(L->H) = p(finding a task while in AS(low)). The time spent in AS(low) depends on how long the node can be expected to not find a Task, which is tied to the probability, p, of a Task occurring. We then have,

$$Z_{NC\_CALLER\_TOTAL} = ((1\text{-}p) \cdot x_L + p \cdot x_H) \cdot c^p \cdot h,$$

then,

$$Z_{NC\_CALLER} = (p \cdot d \cdot h \cdot c^p) / (((1\text{-}p) \cdot x_L + p \cdot x_H) \cdot c^p \cdot h) .$$
$$Z_{NC\_CALLER} = (p \cdot d) / (((1\text{-}p) \cdot q_L \cdot d) + (p \cdot q_H \cdot d)) .$$
$$Z_{NC\_CALLER} = 1 / (((1\text{-}p)/p \cdot q_L) + q_H) .$$
$$Z_{NC\_CALLER} = 1 / (((1/p - 1) \cdot q_L) + q_H) .$$
$$Z_{NC\_CALLER} = 1 / (q_L/p - q_L + q_H) .$$

??? is this right? Conclusions?

Evaluation

We evaluated the Hive Mind system using an implementation on the Deter testbed using a Hive of 400 Nodes. We also evaluated performance and efficiency using our Hive Mind simulator.

Because the Hive Mind model makes assumptions about how Tasks are distributed, our evaluations were repeated using different distribution patterns. These included:

- Single Task on single Node
- Single Task on multiple neighboring Nodes
- Multiple Tasks on single Node
- Multiple Tasks replicated across multiple neighboring Nodes
- Small groups of a single Task distributed across the Hive
- Single Task dispersed throughout Hive
- Random single tasks on Nodes throughout Hive

To eliminate additional sources of variability in these tests, rather than let the system dynamically control the number and type of Ants, a set number of Ants were started and held constant throughout the evaluation.

Time to find first Task

Starting in a quiet state where no Task to service or marker trails exist, we consider the time between the occurrence of an initial randomly placed Task (of potentially more) and the time an Ant services it. This initial delay occurs in the absence of any communication or Task patterns. This delay is a random event based on the particular foraging behavior of the Ant (e.g., a directed random walk, or fixed scan pattern). Ants foraging using a directed random walk must happen to visit the node with the Task. Letting n be the number of Hive nodes and k be number of Ants of that Task type, then as discussed earlier, the time to find the Task was estimated to be n/k, that is the Ant density. Overall, we expect that the number of Ant actions will be equal to the size of the Hive. Experimental data verify these estimates. The graphs in figure 9 show that the observed number of Ant actions to find the Task always clusters about the number of nodes in the Hive and number of ticks to find the Task clusters about the Ant density value.

The expected time to find a Task for a single Ant which scans the Hive by sequentially visiting each

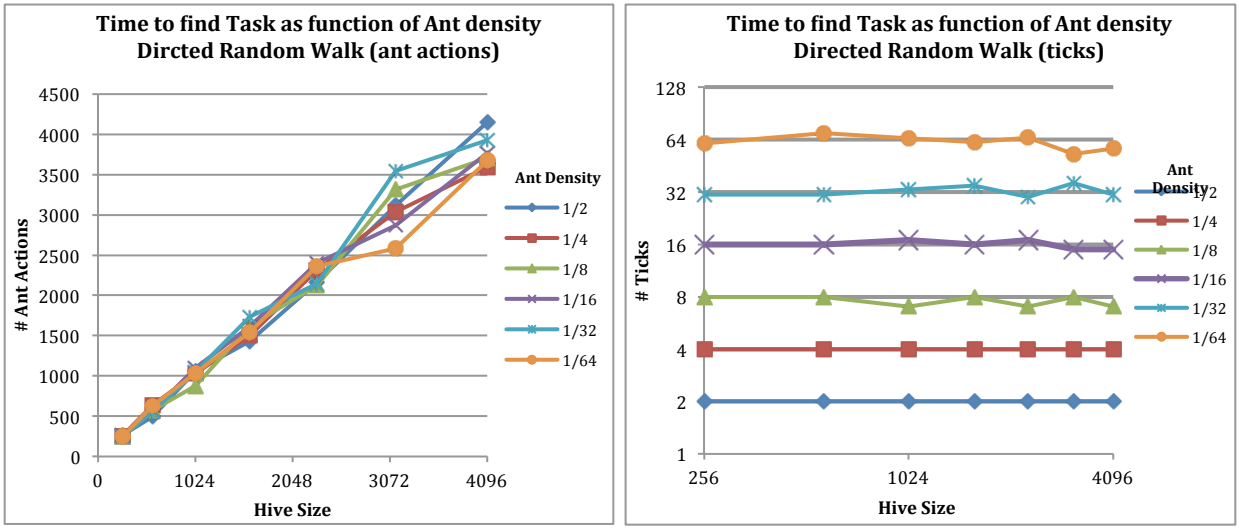


Figure 11: Time to find first target is function of Ant density, not Hive size.

Hive node is significantly better than the random foraging Ants.

On average the scanning Ant will find the Task after examining one half of the Hive nodes, and will have a maximum worse time equal to the size of the Hive. However, when multiple scanning Ants are used the results are no better than random foraging Ants. Because the scanning Ants are started at random positions, only the closest Ant immediately preceding the Task matters; all others will

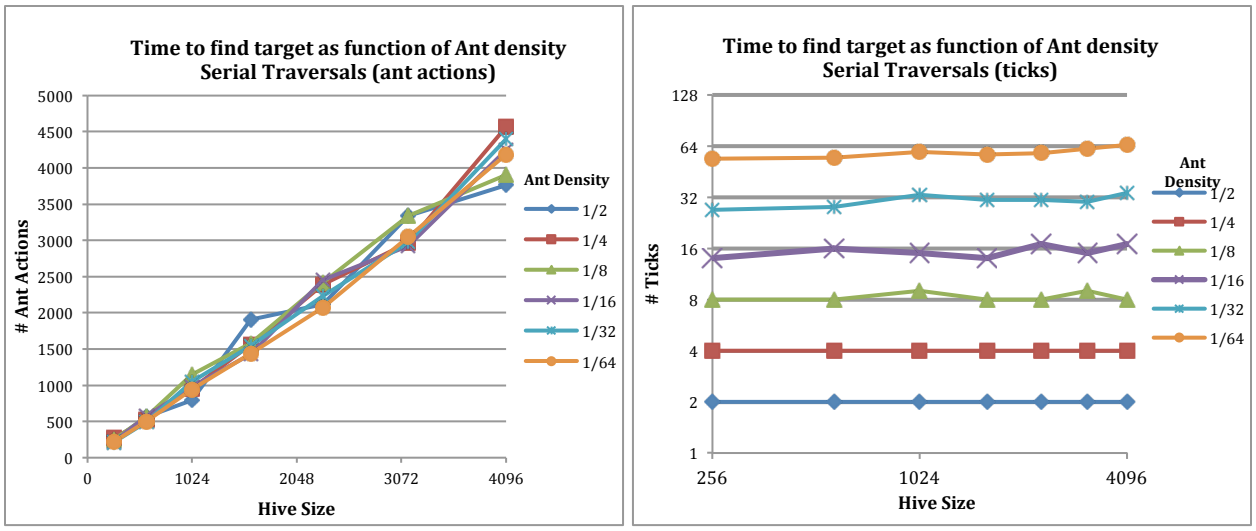


Figure 12: Scanning Ants – Time to find first target is function of Ant density, not Hive size.

not reach the Task first, and will heavily collide, that is, recheck nodes that have already been visited (figure 13). Given k Ants in a Hive of size n, the expected minimum distance to the Task is n/k. This holds for any fixed path foraging pattern. This is the same value calculated for directed random search Ants. No fixed path Ant pattern starting at random positions will provide better expected results that the directed random walk foraging Ants[10].

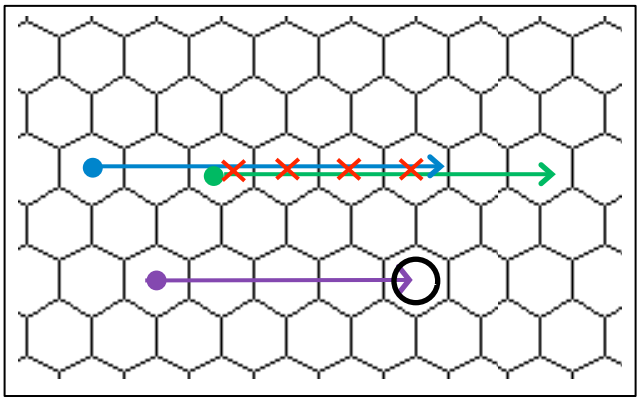


Figure 13: Scanning Ants – only the Ant preceding the Task will determine the latency. All others will recheck vacant nodes.

Wander and Drift

Ants using a directed random walk pattern will vary in the time to visit all nodes in the Hive. We have implemented random walks using two parameters, wander and drift, that affect how much and how the Ants paths will diverge from a straight line. The graphs in figure 14 show how wander and drift affect how quickly an Ant will cover the Hive. The width of the dot reflects the average amount of time observed. We see that too little or too much variation degrades performance. In

[10] If all k Ants could be started evenly at k node intervals of the Hive space, some Ant would be at most n/k nodes from the Task, expected distance, n/2k. Thus performance would be doubled and worst case reduced to n/k actions. However this takes global knowledge and control to accomplish.

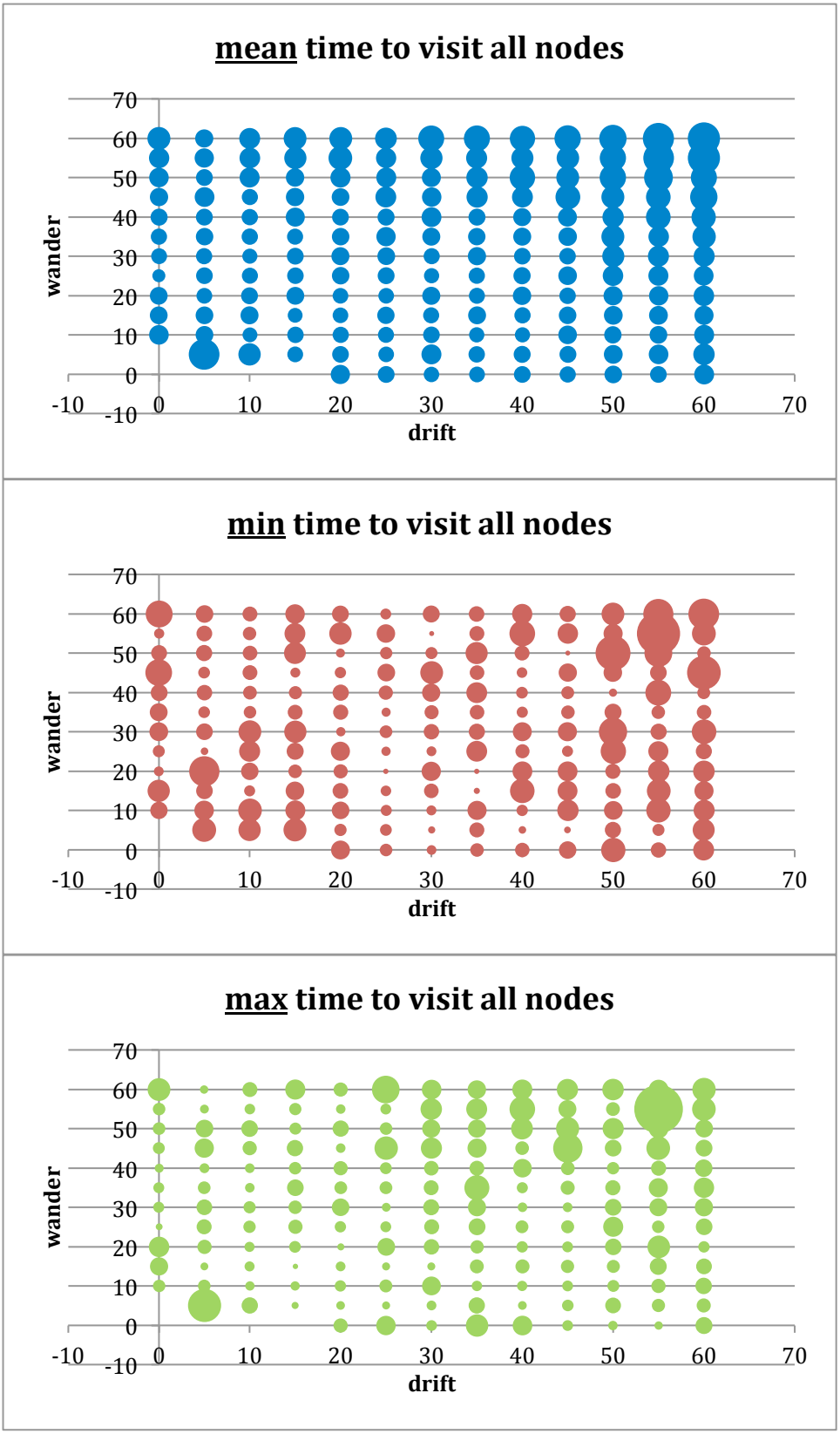


Figure 14: Coverage behavior for varying wander and drift parameters. Mean time to visit all nodes has wide range of acceptable values. Although, with high variance, this same range also produces lower min and max times.

some cases, the Ant follows a unvarying path and loops over a subset of the Hive, never visiting all nodes.

Conclusions on Performance and Efficiency

We designed the Hive Mind system to be adaptable to different memory constraints. As memory is available to the local Hive Mind process, more Ant Task function code will be cached. This can range from where all Ant Task functions in use are locally cached at each node, to where nothing is cached (i.e. fully mobile code). In contrast to Ants which can obtain Task function code from other Nodes or the Queen, Callers must cache all functions being used. The table below highlights which parameters will affect resource used across several conceptual cache sizes. Notice that as the cache size decreases, local memory resources decrease, but communication resource usage increases. In the full cache case, no code moves and requires no communication resources, however in the No Cache case, each tick requires the resources to move the Ant Task code between Nodes.

		Full Cache	**Partial Cache**	**No Cache (mobile code)**	**Caller**
Resource Type	CPU	# Ants; Linger	# Ants, Linger, slight increase when need to fetch, proportional to on cache size	# Ants; Linger; slight increase, must always fetch code	Delay between checks
	Mem	# Task types (sensor functions)	Proportional to cache size (< # Task types)	Size of single, largest Ant Task function	# Task types (sensor functions)
	Comm	# Ants; Linger; Ant message size	# Ants; Linger; number of cache misses; Ant message size, head vs. body	# Ants; Linger; Ant message size; size of body+head	# Tasks to service; neighborhood size

Table 1: How system parameters affect performance is related to cache sizing. -- As cache size decreases the degree to which code must move increases; with no cache we have fully mobile code. With an increase in mobility comes an increase in communication – to move the code and communicate that the code is needed.

Ant Task Types

A fundamental assumption of the Hive Mind is that when one Ant finds a problem (i.e. system out of spec, does not match its baseline, or has out-of-date data), other similar problems are likely to occur on the target Host. This being the case, after an Ant identified a problem, waiting for other Ants currently foraging for a random assortment of problems to eventually arrive lowered performance and yielded an overall lower efficiency. This was due to (a) wasted resources checking Nodes away from the target, and (b) the overhead in executing a variety of unrelated Tasks. If instead, every time an Ant Task was successful it triggered additional, more extensive actions, efficiency is increased. The Ant performs a quick fast check ("is anything here") and only if true will it bother to proceed further ("what is it", "is it something I care about", "is it incorrect" "what is incorrect about

it", "did the problem spread further", etc.) and execute a bundle of other actions, which could in turn execute still others. For example, if an Ant determined that unexpectedly a new user was added, we would also want to see if the account was a privileged (e.g., root) account, if any processes were started, files created, etc.

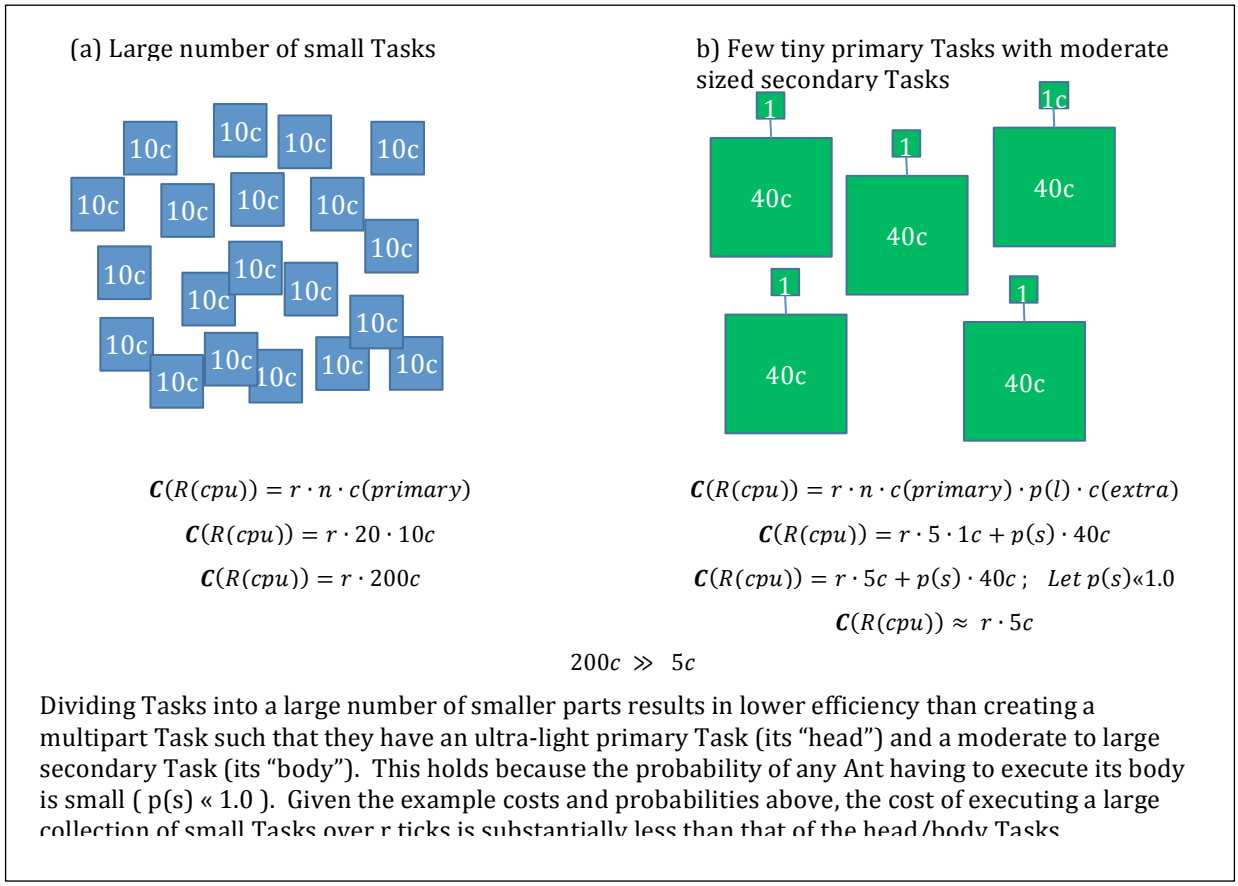


Figure 15: Construction of

Upon analysis, we concluded that simply dividing a detection task into a number of less complex mini-tasks is not necessarily an efficient solution. Done this way, the system will move and execute a large number of unneeded Task functions. A better solution is to design Ants Tasks such that they have a very small "head" which in the "smallest impact" way determines if there was any evidence that deeper investigation is warranted, much like a triage function. This acts as a filter and prevents wasted execution. If after executing the head function, nothing of interest is identified, the Ant moves on, otherwise successive layers of more costly code are executed. Because Ants move and execute their task function every tick, reducing the cost of the "head" will have a significant improvement in efficiency. By causing the Ant to "dig deeper" when it finds something rather than wait for other Ants to eventually arrive and execute their, possibly unrelated, Task functions, we can realize a significant improvement in performance.

In the above analysis, as the number of Ant-types decreases, for a given Ant-density, performance increases. To maintain an equivalent Ant density, if we lower the number of Ant types, the number of Ants increases as does the replicated number of Ants of a particular Task. Conversely, as the number of Ant task types increases, the number of Ants searching for a particular task must drop to maintain the same Ant density. This indicates that Ant density must increase directly with the

number of Task types (i.e., sensor functions). This will have a direct impact on both node (memory, CPU), network (bandwidth usage) and power (processing, transmission) resource use. By designing the system to use significantly fewer Ant Task types, additional improvements in efficiency can be achieved.

This mimics what we see in nature. There are not hundreds or thousands of types of ants in a single nest—only typically 3 (worker, soldier, queen), only to a limited extent can further specialization be argued. Similarly, in a single ecosystem, there are not large numbers of types of ants, even cross species, to justify large numbers of different Ant Task types. Natural ants can individually perform many different tasks, selecting among them based on their prior action and what is immediately needed. Different ant types are only morphological. Some ant Tasks require different physical capabilities (e.g., queens lay eggs, soldiers are better at defending the nest). This supports our conjecture that a wide variety of Ant Task types is not an efficient solution.

In addition to Task function execution costs, the cost of communication, primarily of Ant movement, is most significant. If mobile code is not required, particularly when the Task functions can be cached on the Nodes, mobile Ants may not be needed, and that better performance and efficiency can be gained by using the non-mobile "Callers" instead. This is particularly the case when communication bandwidth is constrained, or the cost to transmit has a strong negative impact on the system (e.g. the energy used to transmit radio signals drains battery power more significantly than receiving messages).

Foraging Behavior

Provided that the Ant has sufficient randomness in its movements, the basic Ant foraging method provides good coverage of the space and does find all its Tasks. However, there is a high degree of variance in these times. Multiple Ants of the same Task type increase performance and decreases variability, but uses more resources and decreases efficiency. This also was seen to vary with different Task distribution patterns. The alternative foraging strategies suggest that the simple Ant foraging method may not be the best choice in a cyber-environment. For example, the scanning method had very high performance and efficiency compared to other simple foraging methods. However, the randomness in the basic Ant foraging pattern is a simple way to support communication and coordination of Ant resources. This may be challenging using the other methods. Additional study is needed.

The Caller method showed to be a very lightweight method in terms of communication bandwidth, remaining silent unless problems were found. Recall the only time a Caller used the network was when a Task was successfully completed and its neighbors were informed. Contrast this to the Ants which generate 1 Task execution plus 1 movement for every Ant for every tick. Because execution times of Callers functions are random, like Ants, they exhibit a high degree of variance in the time to detect a problem, however, unlike the Ants they performed very well on detecting problems that occur in neighborhoods. If fully mobile code is not required, and

The stigmergic communication analog used in the Hive Mind Ants or the simple method that Callers use to communicate their own activity to their neighbors exploits assumptions about problem (i.e. Task) distribution in the Hive. This builds on a combination of three assumptions: (1) Tasks that occur on one node are more likely to occur on its neighbors than elsewhere, (2) there is increased likelihood that there will be multiple, different Task types will occur on a single Node, (3) a set of Tasks types may be widespread across the entire Hive. Examples include, respectively, a common vulnerability on workstations managed by the same person, a server where a number of system changes resulted from being compromised, a worm that is active across the organization.

Without these assumptions, the performance advantage of the Hive Mind (or any method that attempts to predict where to direct resources) is questionable. Consider the scenario where a variety of single random tasks are randomly distributed across the Hive. In this scenario, there is no statistical reason to direct resources to any particular Node or region of the Hive, or to increase the search for any particular Task type. A random search would perform as well, or better. In the case of Ants, the communication overhead of trail marking provides no value, and will direct Ants to regions that have already been cleared of Tasks.

Additional work is needed to understand the impact of the various configuration parameters on performance and efficiency. Given the large amount of flexibility in the Hive Mind, a method such as genetic algorithms which can optimize a system across a variety of parameters should be considered.

Criticality of neighbor relations

The concept of "neighbors" is key to the operation of the Hive Mind system. The ant-based paradigm capitalizes on how ants finding "items of interest" direct other ants to a region where more "items of interest" are likely to exist. This underlies the concept of "locality of resources," simply put, "where you find one you should find many." For example, where an ant finds some sugar or grain, other ants are likely to find more sugar or grain. Communication in ants exploits this natural feature. Ant pheromone trails lead other ants to the area. Other insects such as tarantula hawks or a predatory wasp do not target prey that has a statistical correlation with other prey. Communication in this case would provide negative benefit, directing other wasps away from places where the prey may be to areas where the prey had already been removed. In general animal communication is optimized to the niche the animal occupies. In order for the Hive Mind system to operate efficiently, locality of resources must occur. The Hive Mind system also operates under the assumption that Nodes that are similar will tend to have similar problems to detect or correct. For example, systems running Windows XP SP 3 will tend to have similar properties to each other, and somewhat distinct from those running Windows 7, or Windows Server 2003, Mac OS X, or Linux. Systems running Java will have similar problems, even more so, systems running the same version of Java. Systems where a particular Internet browser is used will have problems differing from those using other browsers. Systems connected to the Internet would have similar problems as compared to systems running in a network "DMZ," a protected back-end server cluster, or an isolated network. Systems operated only by trained system administrators would have problems differing from those operated by programmers, clerical workers, students, or family computers.

An ongoing challenge has been to efficiently generate an effective organization of Hive Nodes such that a Node's neighbors are similar to it as compared to other Nodes in the Hive. Theoretically, the Nodes could occupy a location in *n*-dimensional space and the Ants could operate in that space. This would require the Queen process to determine regions of space that each Node controls, and based on the relation of these to determine which Nodes should be considered which Nodes' neighbors. Unfortunately, for large numbers Nodes defined over many attributes, this appears to be combinatorially infeasible. Instead we arrange the Nodes into a 2-dimensional hexagonal grid such

that adjoining Nodes tend to be like each other. This provides the Ants a convenient landscape to explore[11].

The initial method was to use a space-filling curve such a Hilbert curve (see Figure 16a) to map the Hive Nodes onto an artificial landscape for the Ants to operate. In this application, the Hilbert curve maps a linear ordering of data objects onto the nodes in a 2-dimensional grid. This results in a placement such that objects close to each other in the linear ordering will tend to be close to each other in the grid. The neighbors of a node are assigned to be those nodes bordering it in the grid. We also evaluated other space-filling curves such as the Z-order curve (see Figure 16b), but although computationally faster, with limited analysis not determined to be of higher utility. Because we don't have an actual 2-dimensional relation between Nodes, this must be artificially induced. As discussed above, this is based on the Nodes' attributes.

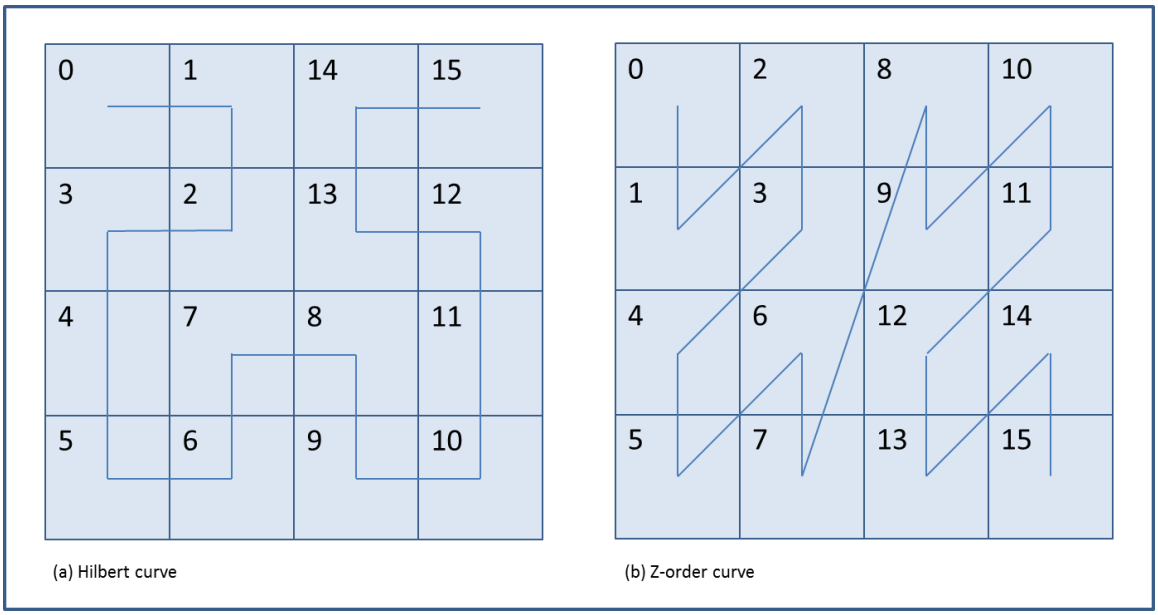


Figure 16: Space-filling Curves

Our initial tests associated the monitored hosts by Aggregator ID, generated a linear ordering of the nodes based on simple clustering of the attribute value, and then applied a Hilbert curve to map the monitored hosts to the grid of Hive Nodes. This method worked well for single attributes, and visualizes well (see Figure 16b). However, applying this to hosts or other data objects based on multiple parameters was problematic. A smooth, low error linear arrangement of objects across multiple parameters such that a Node's neighbors are those most similar to it is challenging.

Auction-based algorithm

In this method, the Hive is initialized and each Node is linked to its adjacent neighbors. Each entity (e.g., a monitored host) is then uniquely mapped to a random Node position in the Hive. Each Node determines its similarity to each of its assigned neighbors. A series of cycles follow.

[11] A 2-dimensional space is not strictly required. Mathematically the Ants could operate in any dimensional space. However, visualizing the Ants' behavior is most approachable when displayed 2-dimensionally.

A Node is selected at "random." Its least similar neighbor is selected and offered to the other Nodes in the Hive in an "auction" for one of their neighbors. Each Node receiving the offer determines its similarity to the auctioned Node; if the auctioned node would result in a greater overall similarity to its neighbors, it replies with a list of those of its neighbors it would be willing to trade to the auctioning Node. After all Nodes have completed bidding, the auctioning Node informs the winner (if any) which Node it would like and the two swap Neighbor links. This repeats until no bids are received for any auctioned nodes. A second phase can follow where Nodes auction their second least similar Neighbor. The auctioning terminates when no further bids are accepted, or a cycle is identified. For very large Hives, rather than offer the auctioned Node to all Nodes, a sampling can be used.

In practice this method appears to converge quickly, and had good overall low inter-Node differences. However because it is a stochastic method local minima may possibly lead to poor solutions. If the overall inter-Node differences were determined to exceed a threshold, the process could be rerun. Another drawback is that the planar relation of the Nodes is not preserved. Neighbors are cross-linked back and forth across the Hive. This is not a problem for the Ants, but visualizing the system becomes useless. An alternative method where the Nodes don't swap Neighbors, but swap locations in the Hive (position in the grid) has been considered. This would preserve the planar structure, but its convergence properties have not yet been studied.

Space-filling induction

A promising method for assigning Neighbors uses a space-filling curve to induce a linear similarity ordering on the hosts, and then with a different space-filling function, mapping the hosts to Nodes in the 2-dimensional Hive. For example, with 2-dimensional data, create a matrix of bins dividing up the range of each parameter into fixed number of discrete values, (e.g., integer 0 – 63) and assigning each data object to a bin, then by tracing the cell order determined by the space-filling curve, we can induce a linear ordering on the data. The Z-order curve is well suited for this task in higher dimensions. By taking this ordering and using another space-filling curve to map the linear ordered list of objects to a 2-dimensional space, we can induce an artificial planar space for the Hive Mind Ants to operate.

Because space-filling curves have points at which there are relatively large jumps in similarity of neighbors, methods that can generate a smoother distribution are still needed. Also, methods that can allow dynamic reassignment of neighbors (such as when host configurations change, e.g., software or OS version have changed) need to be investigated. All the methods listed require an external process to assign neighbors. A method for hosts to join the Hive, be assigned Nodes, and have the Nodes self-arrange to maximize inter-Neighbor similarity is both needed and truer to the autonomous, decentralized concept.

Hive size and Ant density

The size of a Hive (in number of Nodes) must be sufficiently large to allow emergent behavior. The number is affected by a number of factors, such as the Ant density, number of different Ant types,

actual sensor functions used, length duration of marker trails, etc. No hard minimum value has been determined, however, we are using 64 nodes as a guideline.

Unlike real ants, we see no effective upper limit to the size of the Hive. Because the Ants are not bound to a nest, but are nomadic, there is no restriction on how far they may wander. More important than size is the Ant density in the Hive, that is, the average number of Ants observed in a region over some period of time. The effectiveness of the system tends to decrease inversely to Ant density. Lower Ant densities result in longer periods before Ants detect problems, and decreased likelihood that an Ant will encounter a marker trail which slows the rate at which Ants are directed into a region. Performance is also decreased at the opposite extreme, when the Ant density is too high. As time to detect an event decreases as the Ant density increases, but only up to a point, past which detection time again decreases. This occurs because the Hive Nodes spend an increased amount of time processing Ant Tasks when nothing exists to detect. This particularly the case when there are also a large variety of Ant types. So many Ants are queued up to run that those needed must wait. High Ant density has an additional negative impact when there are many events to detect among many Nodes. This results in an explosion of intersecting Ant Trails, which leads to a great deal of confusion and useless redirection of Ants, further impacting local queues.

A configurable parameter controls the desired Ant density. The Hive Nodes regulate Ant density by increasing the likelihood that Ants will die (be removed from Hive) the when local density is above a threshold and increasing the rate at which new Ants are created when below the threshold. Density is averaged over a period of time to prevent chaotic fluctuations in Ant birth/death.

The optimal value for these parameters has not been determined is in practice must be based on factors other than detection speed (such as system resources used).

Applications of the Hive Mind Method

The Hive Mind project has focused on security event monitoring tasks, however we have identified a number of different applications where the Hive Mind system can be applied. Because it is fundamentally a method for directing and coordinating the movement of "objects" and "capabilities" the method is suited to a broad variety of applications where lightweight and/or tunable, autonomous control is desired. Fundamentally, the Hive Mind is useful for monitoring and change detection in situations where resource use needs to and can be balanced against detection time. As such, on average, the Hive Mind will find things more slowly but using dramatically fewer resources than "heavyweight" monitoring systems. That tradeoff is tunable so that one can turn it all the way up to be as resource-intensive and high-performance as a heavyweight system or turn it all the way down to have negligible impact on surrounding systems.

It is also important to note that using the Hive Mind is not mutually exclusive to using a more "heavyweight" monitoring system. There are times when a heavyweight system (e.g., Bro [Pax99] or Snort [Roe99]) is likely both appropriate and useful, such as the intrusion detection system running at a network gateway. Moreover, the Hive Mind can also even be specifically linked to a heavyweight system. For example, a particular "sensor" for the Hive Mind might also be nothing more than a means for intelligently triggering certain Nessus [Der00] or Nmap [Lyo09] probes or for turning on and off anti-virus or malware detection rules in a heavyweight system.

The Hive Mind framework makes the most sense to use in environments where network bandwidth and/or processing power is limited, such as in control system environments, but it also clearly has broader application given that few organizations want to use more bandwidth and processing power than they have to in order to perform security monitoring and/or situational awareness monitoring tasks.

The Hive Mind dynamically adjusts behavior to meet demands by changing the number or location of Ants, and changing Ant types to handle different types and distribution of tasks:

- When a variety of tasks are needed on a particular node, Ants are directed to increase activity on that node.
- When a particular activity is occurring across a number of neighboring hosts, Ants will be directed into that region to focus resources there.
- When a particular type of activity is occurring widely across the system, the number of Ants working to support that task will increase throughout the system.

Because the Hive Mind system uses a variety of Ant-classes, it is suited to a wide variety of detection and state management tasks:

- Detecting situations that are in violation of a stated policy
- Detecting situations where local configuration has changed from its baseline.
- Updating local state to current global values.

Even in degenerate cases where the Hive Mind methods are not optimal, it will still be able to accomplish its task. For example, when target activities are completely uncorrelated across the system, the communication overhead to direct Ants serves no value, and a sequential or random search would be as or more effective. However, the Ant system will still detect the target activities.

The focus of the Hive Mind is on efficiency. The Hive Mind accomplishes this efficiency by taking advantage of (1) light-weight actions directed where and when they are needed, (2) a model that is scalable to large numbers of fully decentralized nodes, and (3) coordination by emergent behavior and minimal by a centralized master process. Because this method uses stochastic methods and avoids brute-force actions, the time to completion will vary, and though not likely, in extreme circumstances may never complete. The method is well suited to environments where resource use (CPU, memory, bandwidth, battery power, etc...) is highly constrained, and can be easily configured to conserve particular resources over others. Also, dynamic situations where a good-enough solution is acceptable can be accommodated.

There are also key places where the Hive Mind will not perform well, most notably in situations where detection time is critical or where resource use isn't the key factor.

Some examples include:

- Detection of Advanced Persistent Threats,
    - The Hive Mind is well suited for scenarios such as monitoring for advanced persistent threats, where by definition, there is time to perform the detection and heavyweight monitors cannot necessarily be run at 100%, 24 hours a day, 7 days per week to do the detection. Suppose that Ants deployed on a banking system and create baselines of the monitored servers' configurations. At some time in the future, the monitored system is compromised. During the compromise, an unapproved user is added to several of the devices, rogue processes are installed, and ports are opened to allow remote connections to an unapproved service. When an Ant tasked to validate some aspect of the configuration (e.g., processes) finds a discrepancy, it will in effect alert other Ants to investigate this server and those similar to it. Ants tasked to find the other issues directed to the compromised servers similarly alert more Ants to more thoroughly investigate them and their peers. Some Ants that, based on their task, did not find a problem on a compromised server will be recruited to join the search for the same problems found on these servers on others in the system. In this scenario, the Ants are configured to respond to automatically remove the unapproved user, process and associated open port, and to report their discovery and action to security personnel.
- Vulnerability detection
    - Lightweight, tunable, vulnerability detection, network change detection, and situational awareness are particularly strong uses for the Hive Mind, e.g., in comparison to Nessus. As an alternative to full, heavyweight vulnerability scanning at particular times, suppose that the Ants are deployed to look for evidence of a vulnerability using very light weight checks. If a check concludes that no vulnerability exists, no more resources are used and the Ant moves to the next

computer. On the other hand, it the Ant suspects a problem, it initiates a set or related vulnerability checks to determine if a problem exists, triggering deeper checks for related vulnerabilities only when evidence warrants it. Random checks are never performed, as this is wasteful. Upon finding an actual vulnerability, considering that the host may have other vulnerabilities, the Ant continues looking for problems on other hosts, but more importantly, leaves a marker trail directing other Ants to the vulnerable host to perform further checks. Because similarly configured devices may also have the same vulnerability, Ants directed to this host check its neighbors along the way.

- Detecting widespread events
    - The Hive Mind also performs well in events that are "wide spread" and might be noticed by multiple Ants. Suppose the Ants deployed on a computing system include those to determine if hard drives are nearing their capacity. Drives that exceed this predefined threshold are noted by the foraging Ants and marker trails are created directing other Ants to assess this device (and its neighbors), but the situation is not directly reported. As other Ants find more devices with hard drives near capacity, they too leave marker trails for other Ants to assess the devices they identified. By observing a number of marker trails for this problem a separate type of Ant determines that this is a widespread problem and reports it to administrative personnel.
- Detecting unexpected changes from a baseline
    - High assurance systems undergo strict configuration management. The software and processes on these systems are highly controlled. Except for those stored data and processes that change dynamically (e.g. log files, database records, ephemeral server processes) little changes. New programs are not installed, applications and operating systems are not patched or upgraded, new services are not activated, new users are not added. Only at periodic, controlled jobs are changes made, and the systems are evaluated before and after the change to ensure that the changes are understood, approved, and documented. Any unexpected change could be indicative of a problem needing attention.
    - The Hive Mind Ants can have Tasks that create baselines for the monitored attributes. When at a future time something does not match or is not within an expected range, system personnel can be notified. At times of configuration change, the Ants can be induced to generate new baselines.
- General anomaly detection
    - An extension baseline monitoring is generalized anomaly detection. The Ants monitor potential baseline attributes to determine which attributes are invariant and which range or behavior is commonly seen in others. When the baseline behavior for an attribute has been determined, future deviations can be reported as anomalies. Because anomalies are a frequent source of false positives and wasted effort by security personnel attempting to determine if a security problem exists, individual anomalies would not be reported, but only when a number of anomalies on a Node, or an anomaly is appearing more widely throughout the Hive.

- Detecting policy, state and configuration violations
    - Many aspects of a system are relatively static and can be stated as a "policy," that is, some observable state that should be within a specified range. Some examples include:
        - number of kilobytes sent (must be less than some threshold),
        - minimum amount of free space on disk drives that must be maintained (value must be greater than some threshold),
        - temperature of system components (must be less than some threshold)
        - if a particular service must be running ("up" must be true),
        - if specific access rights must be maintained on certain files or directories (rights must match configuration),
        - number of times black-listed websites were accessed (must be less than some threshold), or
        - if certain files must not be present on the system ("found" must be false)

      These can report levels of violation, such as when the amount of free disk space is low, very low, or critical.
- Distributing updates of local information (e.g., security patches),
    - Related to vulnerability detection is the notion of actually patching such vulnerabilities. Suppose that an Ant carrying the code for a patch to Adobe Acrobat for Windows is created (and signed) by the "Queen" and is pushed out into the "Hive." The Ant carrying the patch passes between nodes until it finds a Windows host that has not yet been updated. The patch is presented and after validation accepted and applied. The Ant then continues to look for other devices needing the patch, directing other patching Ants toward the device in case it is missing others. Additionally, because the patch was accepted, the Ant will with increased likelihood be cloned, just as those who fail to have their patches accepted are more likely to die. Over a period of time, the patches will be distributed across the system.

Other broader uses of the Hive Mind system can be provided. Many are not directly related to security, and there may be many unrealized applications. For example, a system that maintains the best location or value across a number of parameters, e.g., source of best price, best route, allowed users, source for distributed, redundant data storage, etc., are all potential applications.

Conclusions

Fundamentally a method of resource coordination

Upon reflection, it is apparent that the Hive Mind method is, above all, a method of resource coordination. This is accomplished without centralized support through the emergent swarming behavior.

At issue is what "resource" we are coordinating. In the natural world, ants are both the resource (the capability to take an action) and the means by which resources (items of interest) are moved. Communication directs additional resources (Ants) to an area where an abundance of resources (items) are available. This is unlike most other forms of animal communication where in contrast, information moves to direct the movement of the animals.

Consider wasps directing other wasps to an attack target, wolves calling other wolves to prey, or crows calling to mob raptors. Information is moved, by sound or scent, to cause active resources (wolves, crows, and wasps) to move to where they are needed.

To stay true to the Ant analogy, we must determine which resources are moved, and which items are acted upon. Unfortunately, this does not make efficient use of the environment in which the Ants operate, that is, a network of computers.

Ant Types

We have investigated three types of "Ant," true Ants, Wasps, and Callers. Wasps differ from Ants in that, rather than leave a linear marker trail leading back to the problem site, *Wasps* emit a cloud of marker to the surrounding neighborhoods. Any Ants (true Ants or Wasps) in the region will be aggressively directed to the problem site and its neighborhood. A radically different type of Ant, a *Caller*, is not mobile, but self-monitors and informs neighbors when problems arise. All of these types of Ants have different performance characteristics, resource use, and applications.

A true-Ant's foraging pattern is a directed random walk. Each Ant is assigned a direction vector indicating the general compass direction in which they move. This is not a strict direction, but more a suggestion. As Ants progress across the Hive they will make random deviations, but will generally progress in a given direction. This creates foraging patterns that are "directed random walks." This allows the Ants to cover the entire Hive and not get stuck in loops.

As Ants forage, they trigger execution of Ant Task functions. Looked at from a macro level (without the effect of trail following), we see that Ant Task functions are in effect randomly triggered. When the Ant Task functions are cached on the Nodes (rather than when moved via mobile code), an equivalent behavior can be created without any Ants, that is, by having each Hive Node at random intervals select a random Ant Task function and execute it. This eliminates the overhead of Ant movement for foraging.

If a Hive Node executes a Task function, it results in an Ant changing to a marking mode and leaving a trail leading back to this Hive Node. If instead the Node increases the rate at which it selected random Ant Task functions to execute, it sends messages to its neighbors communicating the

problem found. These nodes respond by executing their own Ant Task functions for the problem, and in turn if they find a problem, they also increase their detection rate and inform their neighbors. This quickly coordinates detection without the overhead of the Ant foraging. We refer to this as the "caller model," roughly based on dogs calling when they are excited by something.

For simplicity, assume a synchronous monitoring system, where each Ant moves with each clock tick. In the Ant foraging method, each Ant in the Hive, executes one movement and one function execution each clock tick, regardless if the action results in detection of a problem or not. Assuming the probability of a problem is near zero, with *n* Ants operating over k ticks, we have *n* x *k* Ant movements and *n* x *k* Ant Task executions. In the caller model, assuming that the rate of random function execution is set to result in n executions over *k* ticks, the Task execution cost is the same, but the communication cost is zero. Only if a problem occurs are messages passed.

When a problem is found, a Caller will send 6 messages (assuming a 6-way neighbor relation), and induce 6 function executions. If configured to pass messages to additional neighbors, this number will increase. This may be desired if the event is very severe and quick discovery is needed. Thus, upon discovery the caller will issue 6 messages and cause 6 executions. Although via recruitment, some Ants will change their Task to that of the Ant who was successful at the Node, the Ant model will not trigger additional executions or Ant movements. From this we see that Ants have overall higher communication requirements, but highly stable overall resource use characteristics.

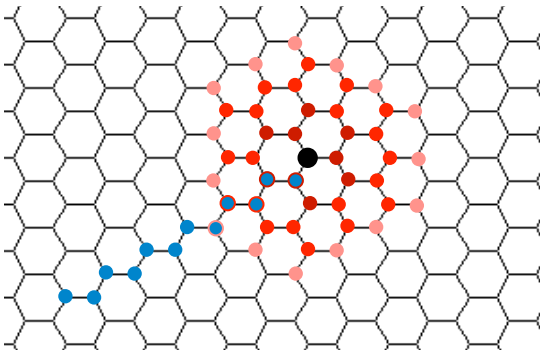


Figure 17 – activity pattern for "Callers"

Wasps have an intermediate behavior, but because they are mobile, they suffer the resource cost of movement. Because they cause a burst of communication to other Nodes in the neighborhood, successful detection causes an increase in messages passed. Typically this will be to 2–3 levels of neighbor away, causing 6–36 messages to be sent. Wasps, coupled with true Ants, are very effective at drawing a large number of Ants into a region. This creates an aggressive response, but can cause temporary instability by creating great differences in Ant density in the Hive. Empirical results suggest a 10–15% Wasp population is a good balance to increase response speed, yet not negatively affect system behavior.

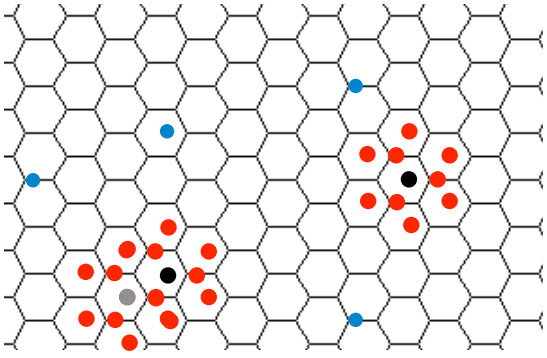


Figure 18: activity patterns for "wasp" type Ants

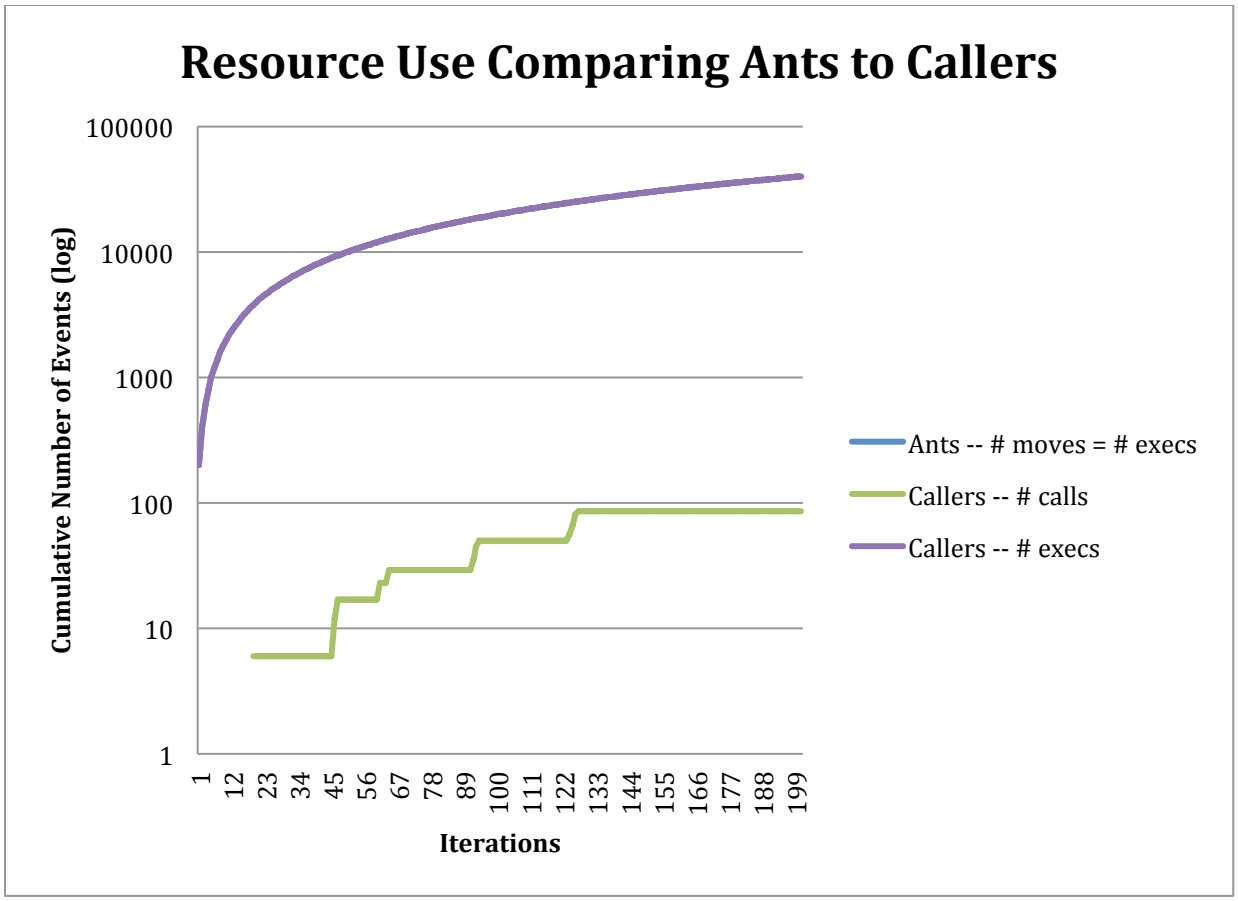


Figure 19: This graph shows the resource use relations between Callers and Ants in terms of number of Task function executions (execs) and the number of messages sent ('moves' for Ants, 'calls' for Callers) over a monitoring period of 200 iterations. The lines for the number of Ant moves, Ant executions, and Caller executions are visually overlapping. Both Ant lines are the same. In this example, the final line for the number of Caller executions is actually 86 higher. Although not plotted, the lines for Wasps overlays the Ant's execs line, but due to the communication overhead of spreading their marker cloud, is slightly higher in the number of moves. Wasps are Ants, only with a different marker laying behavior.

Experiments detailing performance of these methods are forthcoming.

By only sending messages when needed, rather than continuously, Callers minimize communication overhead, causing little impact on the network or other communication channel. Because of this, the Caller model is well suited to situations where network bandwidth, message timing or the cost of transmitting a message is high. This is an important consideration in control systems commonly enforce strict constraints on communications. The timings of multiple polling cycles from master to slave devices are often required to be deterministic and operated at near capacity. This leaves little remaining bandwidth for additional communication. When battery powered radio communications are used, the number of transmissions can be a significant factor. Because the cost to transmit a radio message is significantly higher than the cost to receive a message, the number of transmissions must be limited to prevent premature loss of power.

The purpose of Ant trails is to direct Ants to the host (and neighborhood) where other Ants found problems. Recall that the assumption was that Nodes similar to the Node where a problem is found

are more likely to have similar problems. This implies that we want to check the Node and neighbors. This is accomplished by, respectively, increasing the rate of Task function execution and communicating with neighbors. This directs detection resources where it they are most likely needed. In contrast, Ant trails induce Ants to inspect Nodes at a distance from the neighborhood where it is needed.

Future Work

The Hive Mind system is filled with emergent complexity affected by many configuration and design choices. Exactly how to determine optimal values remains an open research question. The interaction between parameters makes studying the effect of different values in isolation not as definitive as is needed to make concrete configuration recommendations, as does managing performance to meet a particular set of resource constraints. Some of the more interesting open challenges are discussed here.

Mapping data object to Hive Nodes

As discussed earlier, constructing an artificial landscape that maintains locational similarity is a key requirement for the Ant-based system. An efficient and accurate method for assigning neighbor relations is needed. The method discussed earlier that uses Hilbert [HR31] and Z-order [Mor66] curves[12] to map multidimensional data objects into a hyperspace, then using an *n*-space-filling curve applies a linear ordering on the data, which is then mapped into a two dimensional space using a different space-filling curve looks very promising. Evaluating the efficacy of this technique is a high priority. If adequate, it will eliminate a significant impediment to a number of key applications of the Hive Mind.

However, this is a static method, whereas attributes of the monitored devices are likely to be dynamic. As the monitored devices configuration changes, the correctness and effectiveness of the mapping will diminish. While the neighbor assignment process could be run periodically, a method that dynamically and autonomously assigns and reassigns neighbors would improve performance and overall resilience.

Determination of optimal Ant Density values

Earlier we discussed how Ant Density affects system performance. A better understanding of how this should be optimally configured is needed. Moreover, Ant density should not be a directly configurable parameter, but should be automatically adjusted by the Hive Mind. This would simplify operation and allow the system to dynamically adjust itself to meet changing conditions.

Determination of optimal configuration values

The Hive Mind software has a large number of configurable parameters. The default values are those that have appeared to perform will under general situations. Having firm knowledge of the best values for different situations will be important to maximize performance; optimal values are not known. With, for example, Ant Density the optimal choice of marker trail length and the duration for which it lasts have a great impact on system performance. Short trails attract fewer Ants and select for Ants already in the neighborhood. Long trails attract more Ants, select for Ants operating at more remote areas of the Hive, and, because they can cause movement of a large

[12] Hilbert curves [HR31] are commonly used to help build spatial database indices. When searching records geographically near a location, the Hilbert arrangement can help determine the most likely places to examine. Z-order curves were integrated into Oracle's database system in 1995 [GG98].

number of Ants, long trails will express a greater overall impact on Hive behavior. Ant Density is a related factor. At low Ant densities, longer trails are needed to be effective to redirect Ants. At high Ant Densities, shorter trails may be acceptable. If long trails are needed to direct Ants not already in the neighborhood, a lower Ant Density might need to be enforced to prevent instability. Some Ant Tasks may have higher security significance and a faster, more aggressive response may be desired. Varying trail length and duration based on the Task may also be a performance improvement.

Allowing the system to self-evaluate and determine how path length and duration should be set would be a further improvement. Genetic and other competitive algorithms should be considered.

Performance analysis

In addition to the above challenges related to performance, a focused study of the actual detection time and system resource usage should be performed across a variety of configurations, operating environments, and attack scenarios.

A formal analysis and empirical evaluation of the resilience of system to attacks is needed. The visualization and monitoring tools already developed would readily support such study.

Acknowledgements

This research was funded by the National Science Foundation and the GENI Project Office under NSF Grant Number CNS-0940805 and GENI Project Number 1792 (http://groups.geni.net/geni/wiki/HiveMind). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of any of the sponsors of this work.

The researchers also gratefully acknowledge the extremely valuable contributions of the two past co-principal investigators, Dr. Deborah Frincke and Dr. Carrie Gates, as well as UC Davis Professor Matt Bishop, a senior investigator on the project, and the UC Davis computer science students who have contributed to this project, including Vinod Balachandran, Mina Doroud, Jonathan Ganz, Vishak Muthukumar, and Teng Wang. Finally, the researchers wish to thank the staff of the DeterLab, Emulab, and ProtoGENI testbeds for their extremely helpful assistance with conducting the Hive Mind experiments. The QEMU virtualization on DeterLab, in particular, was invaluable for testing the Hive Mind at a meaningful scale.

References

[BBK+06] Terry Benzel, Robert Braden, Dongho Kim, Clifford Neuman, Anthony Joseph, Keith Sklower, Ron Ostrenga, and Stephen Schwab. Experience with DETER: A Testbed for Security Research. *Proceedings of the 2nd International ICST Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TridentCom)*, 2006.

[BDT99] Eric Bonabeau, Marco Dorigo, and Guy Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems (Santa Fe Institute Studies in the Sciences of Complexity Proceedings)*. Oxford University Press, USA, 1999.

[Bel05] Fabrice Bellard. QEMU, a Fast and Portable Dynamic Translator. In *Proceedings of the Annual USENIX Technical Conference*, 2005.

[Bis09] Matt Bishop et al. GENI and Security Workshop Final Report, February 2009.

[Der00] Renaud Deraison. *The Nessus Attack Scripting Language Reference Guide*. Tenable Network Security, Inc, 2000.

[DS04] Marco Dorigo and Thomas Stützle. *Ant Colony Optimization*. The MIT Press, 2004.

[ESK01] Russell C. Eberhart, Yuhui Shi, and James Kennedy. *Swarm Intelligence (The Morgan Kaufmann Series in Evolutionary Computation)*. Morgan Kaufmann, 2001.

[Flux] Flux Research Group, University of Utah, ProtoGENI, http://www.protogeni.net.

[FPP86] J. Doyne Farmer, Norman H. Packard, and Alan S. Perelson. The Immune System, Adaptation, and Machine Learning. *Physica D: Nonlinear Phenomena*, 22(1–3):187–204, 1986.

[GG98] Volker Gaede and Oliver Günther. Multidimensional Access Methods. *ACM Computing Surveys*, 30(2):170–231, June 1998.

[HDL+90] L. Todd Heberlein, Gihan V. Dias, Karl N. Levitt, Biswanath Mukherjee, Jeff Wood, and David Wolber. A Network Security Monitor. *Proceedings of the IEEE Symposium on Research in Security and Privacy*, 1990.

[HMDoc] University of California, Davis, Department of Computer Science. Hive Mind Documentation. 2013. https://github.com/UCDavisHiveMind/HiveMind/tree/master/HiveMind/documentation

[HR31] Heinz Hopf and Willi Rinow. Über den begriff der vollständigen differentialgeometrischen fläche. *Commentarii Mathematici Helvetici*, 3(1):209–225, 1931.

[KFL94] Calvin Ko, George Fink, and Karl Levitt. Automated Detection of Vulnerabilities in Privileged Programs by Execution Monitoring. *Proceedings of the 10th Annual Computer Security Applications Conference (ACSAC)*, 1994.

[Lon01] Chris Lonvick. The BSD syslog Protocol. RFC 3164. August 2001.

[Lyo09] Gordon Fyodor Lyon. *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning*. January 1, 2009.

[Man83] Benoit B Mandelbrot. *The Fractal Geometry of Nature*. Times Books, 1983.

[Mor66] Guy M Morton. A Computer Oriented Geodetic Data Base and a New Technique in File Sequencing. International Business Machines Company, 1966.

[Pax99] Vern Paxson. Bro: A System for Detecting Network Intruders in Real-Time. *Computer Networks*, 31(23):2435–2463, 1999.

[PNBA06] H. Van Dyke Parunak, Paul Nielsen, Sven Brueckner, and Rafael Alonso. Hybrid Multi-Agent Systems: Integrating Swarming and BDI Agents. *Proceedings of the 4th International Workshop on Engineering Self-Organising Systems (ESOA)*, 2006.

[Roe99] Martin Roesch. Snort - Lightweight Intrusion Detection for Networks. *Proceedings of the 13th Large Installation System Administration Conference (LISA)*, 1999.

[SHF97] Anil Somayaji, Steven Hofmeyr, and Stephanie Forrest. Principles of a Computer Immune System. *Proceedings of the New Security Paradigms Workshop (NSPW)*, 1997.

[Smi02] Frank Smieja. The Pandemonium System of Reflective Agents. *IEEE Transactions on Neural Networks*, 7(1):97–106, 2002.

[Tem11] Steven J. Templeton. Security Aspects of Cyber-Physical Device Safety in Assistive Environments. *Proceedings of the 4th International Conference on PErvsive Technologies Related to Assistive Environments (PETRAE)*, 2011.

[Tur48] Alan M. Turing. Intelligent Machinery. Technical report, National Physical Laboratory, 1948.

[VVKK04] Fredrik Valeur, Giovanni Vigna, Christopher Kruegel, and Richard A. Kemmerer. A Comprehensive Approach to Intrusion Detection Alert Correlation. *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 1(3):146–169, 2004.

[WLS+02] Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb, and Abhijeet Joglekar. An Integrated Experimental Environment for Distributed Systems and Networks. *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI)*, 2002.

[ZHR+07] Jingmin Zhou, Mark Heckman, Brennan Reynolds, Adam Carlson, and Matt Bishop. Modelling Network Intrusion Detection Alerts for Correlation. *ACM Transactions on Information and System Security (TISSEC)*, 10(1), 2007.