UC Davis HiveMind User Guide -- v0.4

(15 June 2013)

Introduction to HiveMind

The HiveMind project implements a lightweight, decentralized, autonomous, event monitoring system inspired by the behavior of ants and other social insects. HiveMind implements an "altruistic search" algorithm derived from ant foraging behavior to influence what monitoring sensors are run, where and when. This is an alternative to methods that in essence run "all sensors, at all times, in all places". The goal of the HiveMind is to reduce the monitoring system's demand on overall system resources: CPU, memory, network bandwidth, power consumption, and to let the monitoring behavior be an emergent property of the system rather than specified with expert system rules or other heuristics determining what, when, and where to monitor. A simplified explanation of the system follows.

Foraging *Ants* wander through the collection of monitored devices. Ants, implemented as messages, are received by a small node-manager process that mediates the Ant's sensing activity, and forwards the Ant to its next destination. If the Ant finds something of interest, it stops foraging and, much like real ants, wanders off leaving a marker trail pointing back toward were the *Activity of Interest (AoI)* was found. Other foraging Ants that encounter the marker trail, follow it, looking for their own Activity of Interest, along the way. If the trail runs out and nothing is found, the Ant keeps foraging, however if the Ant found something of interest, like the original Ant, it too will begin leaving its own marker trail also leading back to where the AoI was found. This behavior causes Ants to be directed to increase monitoring of the original device, and its neighbors. By assigning neighbor relations based on similarity, the Ants will increase monitoring of hosts with a higher likelihood of having the same issues. In many cases, the activities detected are in themselves not significant, but when a number are found together, they can provide evidence of proscribed events. When such a combination is found, an alert is sent.

The current focus has been to implement it on GENI-related testbeds (Emulab, DeterLab, ProtoGENI) and to study its performance. Because of this many options or parameter requirements are specific to these systems. However, with the proper configuration and little or no modification of the code, HiveMind should run in any network environment.

Installation

Prerequisites

1) A collection of hosts to monitor (the *Hive*). These can be any collection of hosts, an Emulab experiment, a ProtoGENI slice, etc.

2) A host that has network connectivity to the monitored Hive nodes (the *Queen*). Typically an extra host designated for HiveMind command and control functions. You must have administrative privileges (e.g. root or sudo capability) on this host to start and configure the HiveMind processes.
3) Python v2.6 or higher installed on all the above hosts.
4) A text file listing of the Hive hosts to be monitored.
    a. ProtoGENI users: This can be extracted from the manifest.
    b. Emulab/DeterLab/etc.: this can be extracted from the …
    c. Others: manually create the list in the format of:
    <iP address or hostname sufficiently qualified to be reachable from the Queen>,
    <optional name label>
    e.g.:
    pc-23.emulab.net, csa-master
    node-111.csa_test_15.GENIHiveMind.isi.deterlab.net, csa-master
    10.1.1.51, csa-slave-16
    192.168.1.22

→ Note: the current implementation has only been tested on Ubuntu systems using Python 2.6 and 2.7.

Installation and Monitoring Walk Through

Steps:

1) Copy the HiveMind archive (HiveMind.tgz) to a convenient directory on the "Queen" host. Typically this will be in the users directory on NFS systems. For others, the default configuration assumes it is /opt. Change to this directory, then un-tar the file:

```
tar zxf HiveMind.tgz
```

2) Change ownership of the files to the desired user:

```
chown –R <username> HiveMind
```

3) Change directory into the HiveMind "bin" directory. E.g. /opt/HiveMind/bin, or ~/HiveMind/bin on NFS systems.

4) Initialize HiveMind processes on the Queen. Open a shell on the Queen and change directory to the installed 'HiveMind' directory. From the command line enter the command:

```
./start_HiveMind.py
```

This configures remote syslog for HiveMind data and starts the web server for the web interface.

5) Install the node manager software on each Hive host. This may be accomplished by several alternative methods depending on how the Hive hosts are implemented:
    - For NFS systems (e.g. Emulab or DeterLab), the software should already be accessible. Skip this step.
    - Manually copy the node manager software onto each Hive host. This must be the same directory across all monitored hosts. (default= /home)
    - If ssh is enabled for the user with keys onto each of the Hive hosts, and the user has sudo provileges, the software will be automatically installed when attempting to start the node manager process.
    - → Note: depending on the privileges of the target directory, the user may need super-user (root, Administrator) privileges on the Hive hosts.

6) For this walk-through, we are assuming you are on an Emulab site using the included "HiveMind_test.ns" file.

7) Verify that all Hive hosts are reachable:

```
./node_admin.py --nodes nodelist.txt --ping
```

8) Start the node manager software. Several alternative methods:
    - Configure the Hive hosts to automatically start the node manager software when they boot (e.g. modify ProtoGENI rspec, or Emulab ns files to start the node manager on startup);

      .ns file example: (change file location to match install path)

```
set ns [new Simulator]
source tb_compat.tcl
set opt(NODE_COUNT) 3
set lanstr ""
set HiveMind [$ns node]
tb-set-node-os $HiveMind Ubuntu1004-STD
tb-set-node-startcmd $node($i) "sudo /users/hm/start_HiveMind.sh"
append lanstr "HiveMind "

for { set i 1 } {$i <= $opt(NODE_COUNT)} {incr i} {
    set node($i) [$ns node]
    tb-set-node-os $node($i) Ubuntu1004-STD
    append lanstr "$node($i)
}
set lan1 [$ns make-lan $lanstr 100Mb 0ms ]
$ns rtproto Static
$ns run
```

    - Manually start the HiveMind software on each node. From the command line:

```
./start_HiveMind.py
```

- If ssh has been enabled on the Hive hosts and the user has sudo privileges, use the node administrator program to start the software.

  ```
  ./node_admin.py --nodes nodelist.txt -s
  ```

  → Note: the node manager must run with super-user privileges for many sensor functions to work.

9) If needed allow firewall access between the Queen or other monitoring host to the Hive hosts. The default ports used are: tcp/50000 on each Hive host, tcp/50010 on the Queen. These can be modified by editing HiveMind.cfg/HiveMind.conf. (see configuration file details later in the document)

10) Use the node manager to validate that all node managers are available. From the command line on the Queen or other host able to connect to the Hive hosts. Change to the 'HiveMind' directory and enter the command:

    ```
    ./node_admin.py --nodelist.txt
    ```

    If all hosts are not up, recheck in a few minutes, or restart the processes.

11) If for some reason not all hosts are responding, wait and recheck, or restart the processes on those hosts.

12) Next assign neighbor relationships to the nodes. Enter the command:

    ```
    ./assign_neighbors.py --nodes=nodelist.txt
    ```

13) At this point the system should be running, though no Ants will be present. These can be added to the system either manually (see injecting ants, below), or created automatically by the system. For this tutorial, start 5 immortal ants using the command:

    ```
    ./start_5_ants.sh
    ```

    This starts 5 Ants, 1 for each of 5 sensor function, each with a different foraging direction. Examine the start_5_ants.sh file for more details.

14) To verify that ants are created and moving you can examine the raw log file activity, enter:

    ```
    sudo tail –f /var/log/HiveMind.log
    ```

    or use the monitoring script:

    ```
    sudo ./log_monitor.sh  3
    ```

Note the 3 option. This selects for several different type of activity including Ant movement. Details for the script are provided is a separate section. Also, the syslog system output is sometimes in bursts. Depending on how the system buffers activity, you may need to wait up to 30 seconds to see output.

15) Stop the log monitor using Control-C.

16) Open a separate window, cd to the HiveMind/bin directory and restart the log monitor, but now with the option 2 (show alerts, things found, and errors).

17) In the original window, set some targets to be found. Use the set_targets.py script.

```
./set_targets.py –t node-1 --type 2
```

This sets a target for sensor type 2 on node 1. In the log viewer window you should see a line indicating the something was found on node_1, type 2, by ant 1002.

18) Now set a type 1 target. This has an alert in its response.

```
./set_targets.py –t node-3 --type 1
```

This sets a target for sensor type 1 on node 3. In the log viewer window you should see a line indicating the something was found and an alert on node_3. You may also see an error message referring to email not being sent. The system can be configure to email alerts or other information to designated recipients, but only if the email has been properly configured. See the section on the HiveMind config file.

The following sections introduce additional tools and viewers and discus those already mentioned in greater detail.

Interacting with nodes -- command line interface

The node administrator program is the primary means for the user to interact with the HiveMind system on the monitored hosts.

```
Usage: node_admin.py --nodes <nodelist> [options]
       node_admin.py –n <number nodes to find> [options]

Options:
  -h, --help       show this help message and exit
  --nodes=NODES    filename containing list of nodes to manage
  -s               restart non-responding node managers
  -q               send quit message to stop node managers
  -k               force kill of node manager processes
  --run            send command to allow ants to be automatically created
  --wait           send command to stop ants from being automatically created
  --kill           send command to kill all received ants
                   using the –n option; writes to file specified with --nodes
  -n NODE_COUNT    number of nodes, range 1 to this value to include in slice
  -b NAME_BASE     suffix for nodes found using –n option
  --map            save node map file; typically used to save list of node discovered
  --ping           ping hosts to see if they are alive
  --user=USERNAME  Override user name to use when logging into nodes.
  --opt=OPT        change adjustable value on all nodes. Format is
                   OPTION_NAME=VALUE
```

The “- -nodes” option specifies the file containing the list of nodes to monitor. Generally, this will be prepared ahead of time and used with all calls using the node administrator program. These names need to be specific enough for them to be accessed from where the node administrator script is running. For example, from the Queen host in an Emulab-based experiment, only the host name (e.g. node-1) need be specified.

Alternatively, the -n option allows the users to specify using the first n nodes following a naming convention (e.g. given then name base of “node” and a –n of 256, the program will use the hosts “node-1” through “node-256”. The default base is “node”, but this can be changed with the –b (“basename”) option.

The –n option may determine that some hosts are not reachable, either the host is not available, or the node manager process is not running. If this is the case, you will receive less than n hosts. If you need more nodes, we suggested that you attempt to restart node managers or increase the n (up to the size of the pool available) and retry.

Adding the –map option will create a node list based on the host that were available. This file can be used instead of using the –n option for all node administrator commands.

The –ping option simply attempts to see if the hosts are alive, but does not check the status of the node manager process. Simply running node_adm.py with no options (other than --nodes or -n) just checks the status of the node managers.

The default node manager configuration does not automatically start generating Ants. Until neighbors have been assigned, this is pointless. Use the –run option to start generating Ants. The –wait command will stop further creation. The existing Ants will continue to run until they eventually die. The –kill option causes all Ants to be quickly killed.

The -s option starts remote node managers. This uses the ssh command it will only function if ssh is allowed and the remote user has sudo privileges. Manager processes may not start if the listening port has not yet been released by the operating system. This is often the case when you stop the managers, then immediately restart them. Usually waiting one minute is sufficient. The -q quit command simple sends a message the remote node managers to quit. A more forceful –k kill command is also available to kill node manager processes. Like start, this also requires ssh and sudo privileges.

The --user option allows executing the -s (start node managers) or the -k (kill node managers) using a different username on the remote hosts than that the use is currently logged on with.

The -opt option allows changing configuration parameters of the running node managers. When fully implemented this will allow modification of how the system behaves without having to modify the configuration file and restarting the node managers.

Using the Activity Viewer

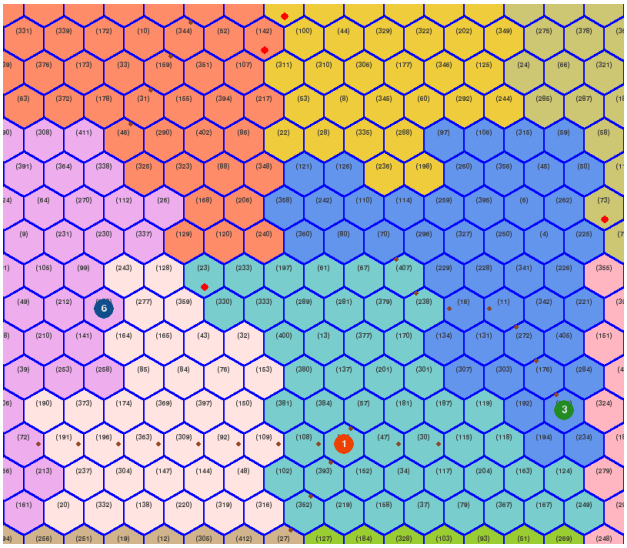
The Activity Viewer displays graphically the movements of the Ants, their state, markers, and detection targets. This is not a real-time viewer. At live speeds, viewer updates occur extremely fast resulting in what appears random. To be useful, we use collected activity traces. These traces are extracted from the logs generated.

Running the Viewer

`HiveMind_viewer.py`

```
Usage: HiveMind_viewer.py [options]

Options:
  -h, --help      show this help message and exit
  -s TFILE        filename of trace to display
  -m NMFILENAME   filename of node_map file.
  -l              loop replay input file when at end
  -w              wait for keypress when input exhausted
  -d DELAY        number of milliseconds to pause between node updates
  -i SKIP_LINES   start delay after skipping this many lines
  -q              suppress most output
  -E              show enclaves by color
```

The viewer will display a mesh background; each cell represents one of the monitored devices. Its identifier will be displayed near the top of the cell. If used, the color of the cell represents the enclave the device belongs to. Targets appear as a small red dot along the bottom of the cell. Currently only 5 positions are used, the position referring from left to right to sensor functions 1-5. Sensor functions of larger numbers are mapped to 1-5 using the modulo function.



The Ants are represented as a circle in the center of the cell. The number in the circle is that of the Ant's sensor function. The color of the circle refers to the Ant's state: blue = foraging, green = dropping, red = following, yellow = changed sensor function. A centered black dot indicates that the Ant at that position died.

Small brown dots near the edge of the cells indicate a dropped marker; the side indicating the direction back to the discovered activity.

When the viewer is running the following keys can be used to control the display:

P	pause the viewer	T	skip to next detection target set

<	display slower	ESC	quit
>	display faster		

The following will display a previously collected trace. From the HiveMind directory enter:

```
./HiveMind_viewer.py -d 20 -E -q -s demos/trace-demo-1
```

Included in the demos directory are several traces that demonstrate different aspects of the system.

Extracting Traces

After the test or tests have been completed, the activity that occurred is in /var/log/HiveMind.log on the Queen host. This information will be used to create traces that can be sent to the viewer for visualization and study. To prepare trace files to view your own tests:

Stop all node managers (optional, this will prevent additional data being written to the log file)

```
./node_admin.py –nodes nodelist.txt -q
```

Repeat as needed until it returns no active nodes. *Note: you can restart the tests if the account can ssh to the Hive nodes and run a process using sudo. Otherwise, once stopped they will have to be manually restarted.*

Save the log file to a working directory with a unique file name. For example, on the Queen host:

```
sudo cp /var/log/HiveMind.log logs/HiveMind.test-1.log
```

Strip out all the unneeded lines relative to a trace and make a compressed format for visualization.

```
./clean_log.pl < logs/HiveMind.test-1.log > logs/HiveMind.test-1.log.clean

./make_traces.pl < logs/HiveMind.test-1.log.clean
```

This last command places the trace file in a "traces" directory as "trace_xxx". You will need to rename or copy this file for archiving. The make_trace program also creates (optionally) traces of each ant individually. These will go into an "ant traces" directory.

These final traces can be displayed using the HiveMind viewer as discussed above.

Adding new sensors

Sensor functions or simply “sensors” refer to the code executed to check for some state or event occurring on or observed from a host. For example, an unexpected account or service, an excessive number of process failures, network connections from an unapproved source, or changes in security settings. Several simple generic sensors are included for testing that serve as examples and place holders for the desired sensors. These are:

Bad data, bad user, bad host, bad file, bad target, bad process, bad port

These are suitable for testing and basic evaluation. However to modify the system to look for activities of particular interest, new sensors can be added by editing the sensors library.

Each sensor should have 4 functions defined using the following naming and return convention:

1) <sensor_name>: function to detect the desired event, activity, etc.
2) <sensor_name>_r: function specifying the response if any to be taken. This can include active response to remediate the problem, trigger additional actions, or generate alerts. For testing, this is used to undo the test conditions set as they are found.
3) <sensor_name>_t: function specifying how to create the conditions that the sensor function can detect. Primarily used for testing.
4) <sensor_name>_u: function specifying how to unset the conditions set with the _t function. This is different from response is that it only does the undo and may be linked to the specifics of the _t function rather than a generic event of this type.

Sensor function groups are stored in the sensors.py file.

Creating attack targets

Creating an attack target is simply creating the conditions that an Ant sensor function is coded to detect. This can be done manually, or by taking advantage of the "set_target" methods specified as part of the sensors functions (see sensors.py).

The *set_targets.py* script allows you to generate one or more attacks across a variety of sizes and patterns.

```
Usage: set_targets.py [options]

Options:
  -h, --help            show this help message and exit
  -l                    loop creating targets until killed
  -i INTERVAL           number of seconds to wait between setting targets
  -r                    remove target before creating next
  -f NODE_MAP_FILESPEC  filespec to read node map from (default =
                        logs/node_map
  -t TARGET_NAME        specified target for the attack
  --multiple            set targets on multiple hosts. Density control number
                        of target hosts
  --group               place attack targets in a group of nearby nodes.
  --density=GROUP_DENSITY
                        approximately how many targets to create in the group
  --spread=GROUP_SPREAD
                        approximate diameter to put group of targets in
  --type=TARGET_TYPE    set a specific attack type as the target
  --onetype             make grouped attack types all one type
  --onezone             make grouped attacks only target one zone per group
```

Examples:

Generate a group of random events on host "node-2"

```
./set_targets.py -t node-2
```

Generate a group of random events on multiple random hosts

```
./set_targets.py --multiple
```

Generate a single event to trigger sensor function (task) 4

```
./set_targets.py -t node-3 --type 4
```

Generate task type 4 events on a group of nearby nodes nearby host "node-12"

```
./set_targets.py --group --type 4 –t node-12
```

Continuously generate multiple random events on random hosts every 10 seconds

```
./set_targets.py --multiple –l –I 10
```

Note: the bundled version requires modifying the target_specs.py file for each attack type to use. See comments in the file.

Injecting Ants

Ants are implemented as messages sent between hosts. To add an Ant to the system we need only send a properly formatted and valid Ant message to the node manager port on a host. Ant messages have the following format:

ant:<ant_id>, <ant_type>, <age>, <heading>, <state>, <task>, <parms>, <focus>, <flt_len>, <flt_cnt>, <charge>, <came_from>, <found_at>

Where

ant_id: a unique integer identifier *(unsigned int)*
ant_class: defines the ant behavior class: 1: scout (normal), 2: worker, 3: soldier, 4: messenger, 5: wasp *(unsigned int)*
age: the remaining number of movements between hosts before its likelihood of dying significantly increases. *(unsigned int)*
heading: the general direction the Ant is traveling in. Scale is compass degree (e.g. 0-359) where 0 is 'up'. *(unsigned int)*
state: the state of the ant – 0: idle, 1: foraging, 2: dropping: 3: following *(unsigned int)*
task: the id number of the sensor task to execute. The valid range is determined by those sensor functions defined. Demo configuration uses range 1 to 6. *(unsigned int)*
parms: Additional parameters for sensor functions that require
focus: defines how likely it is for an Ant to change task. This is a float in range 0.0 – 1.0 (least likely to highly likely.
flt_len: For Levy flights, length of the current flight
flt_cnt: For levy flights, tracks the n-th flight taken
charge: remaining amount of marker pheromone the ant is carrying. Decremented each hop. When this value reaches 0 the Ant stops dropping and switches to Foraging.
came_from: Holds the prior node the Ant has come from, used to support trail marking
found_at: Holds reference to the host the Ant sensor found its issue.

Ants can be injected either from the command line or via the web gui.

Command Line

```
echo <ant-message> | nc <target-host> <node-manager-port>
```

e.g., to inject an ant with id 1001,

```
echo "ant:1001,2,0,030,1,3,'',0,0,0,0,'','\'" | nc node-71 50000
```

Ants can only have a task number for a defined sensor function. These depend on the sensors installed. Any Ant with an unknown task is destroyed by the node manager.

Monitoring Activity

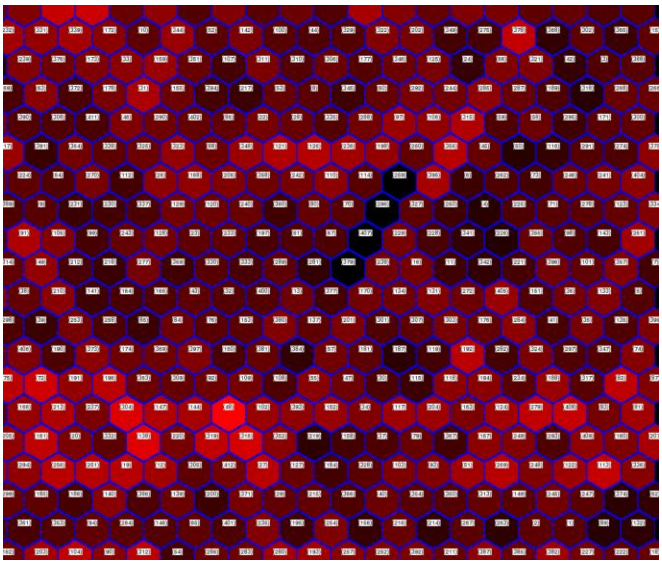
This tool filters log activity for review. Run from the command line. Usage:

```
log_monitor.py       show only "alerts"

log_monitor.py 0     show only errors

log_monitor.py 1     show errors and "alerts"

log_monitor.py 2     show errors, "alerts" and "things found"

log_monitor.py 3     show errors, "alerts", "things found" and "ant
                     movements"

log_monitor.py 4     show everything
```

Heat Map Visualizer

This tool visually shows the relative frequency of how often ants visit each node. Darker colors indicate less often visited, lighter colors, more often. The figure below shows the distribution based on the provided demo trace after about 15,000 ant-movements. Notice the 4 diagonal black cells. This indicates nodes that were not responding, which is the case here. They were stopped as part of the test.

This tool can be used to verify that Ant movements are sufficiently random to ensure statistically even coverage and to study how various parameters and the presence of discovered targets affects Ant distribution.



```
Usage: heatmap_viewer.py [options]

Options:
  -h, --help      show this help message and exit
  -s TFILE        filename of trace to display
  -m NMFILENAME   filename of node_map file.
  -l              loop replay input file when at end
  -w              wait for keypress when input exhausted
  -d DELAY        number of milliseconds to pause between node updates
  -i SKIP_LINES   start delay after skipping this many lines
  -q              suppress most output
  -E              show enclaves by color
  -b              batch mode, no animation, just show final map
```

-s specifies the trace to visualize. This can use traces composed of single or multiple Ants.

-m specifies the node map describing the nodes present, their neighbor relation, and assigned enclave. Generally this will be the default node map file saved when running the "assign_neighbors.py" script.

-l restarts the trace file when exhausted in a continuous loop.

-w specifies to wait for key press before closing display screen. Note: the prompt is in the shell window.

-d how many Ant-movements to wait between screen updates. If greater than 1, updates all nodes based on current statistics. If equal to 1, updates on every Ant-movement. This update is only based on the stats of the current moved Ant.

-I Allows skipping into the file. Used when many lines at beginning are idle time (e.g. no ant movements), or to skip a startup period where the statistics may be skewed.

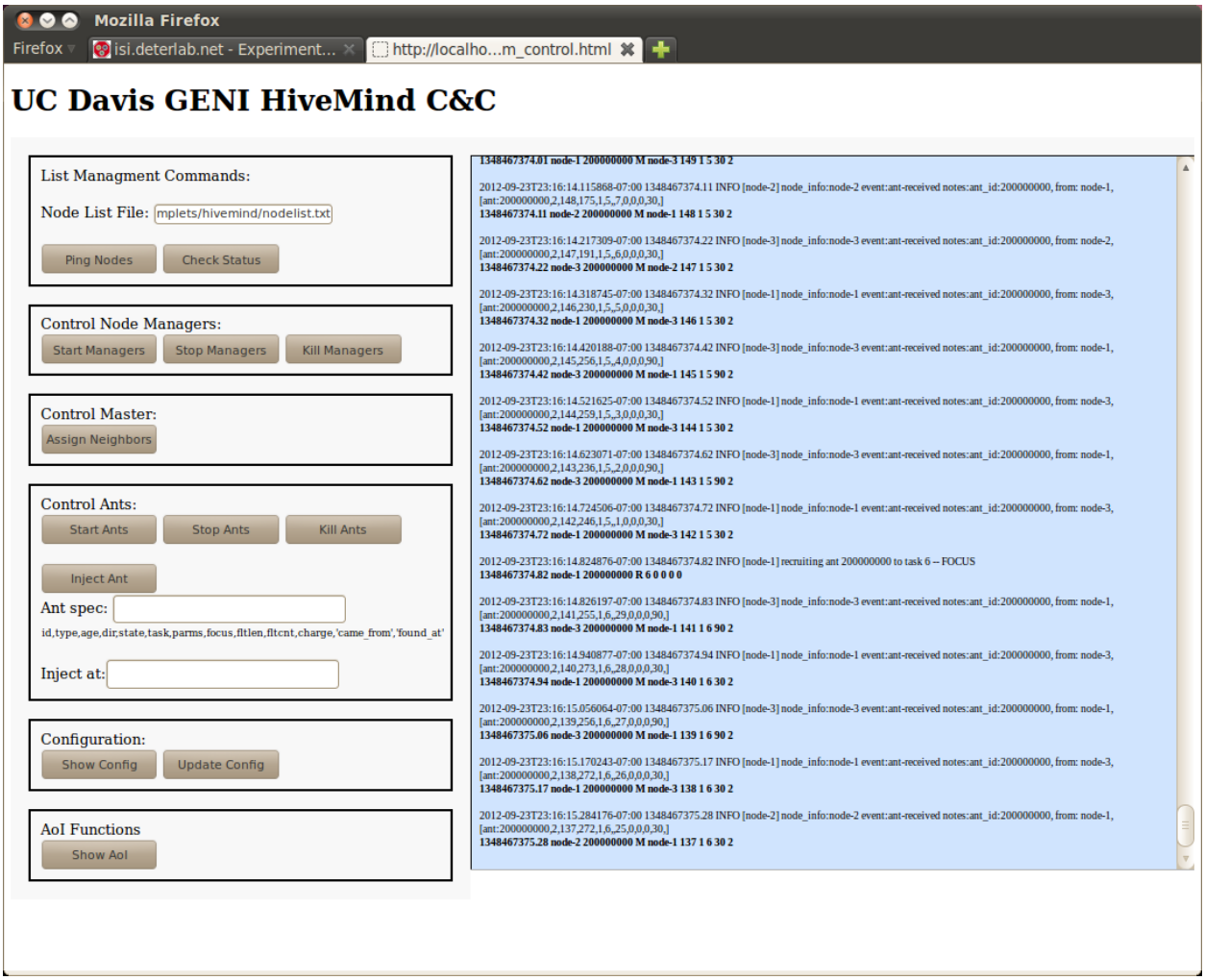
-q quiet mode. Suppress most output.

-E show Enclaves by color [currently disabled, will refer to cell outline when enabled].

-b batch mode. Suppress all animation. Just display heat map after all data has been processed.

Using the web interface

As an alternative to the command line interface a web browser based GUI is available on the Queen node: http://<queen-host> The GUI provides equivalents for most all the command line options, plus a status monitoring page.



Node Manager Configuration Parameters

These determine the operational characteristics of the node and created Ants. The parameters, default settings and a description of their function follow.

```
[NodeMgr]

#- set the default host and port for the Advocate process
hmadv_addr = HiveMind
hmadv_port = 50010

#- set the default port for this process process
node_mgr_port = 50000

#- set the hostname for the "queen" and port
logger_node = HiveMind
logger_port = 514

#- set if we can use NFS
use_NFS = True

#
#- set the directory for where local log files are to be saved
#- --> for Hive nodes, generally
logdir =../logs
#logdir = .

#- set directory where HiveMind files are to be stored on Hive nodes
install_dir = /home/HiveMind/bin

#- set seed for random number generator (to make results repeatable sequence)
#- >> set to zero for random sequence between different runs
rgenseed = 1114582
#rgenseed = 1114581


#- settings for email alert notificaiton
#- -- receivers should be a single or comma separated list of email addresses
email_alerts = True
email_server = <email_server>
sender  = HiveMind@<domain>
receivers = <recipient_1>@<domain>,<recipient_2>@<domain>
#-
#- the expected life of an ant will be about the linger_time * default_ant_age
#- also, the expected number of nodes the ant will touch will be that of the
#-   default ant age.
#- -- note:
#-
DEFAULT_ANT_AGE = 200
```

```
#- determines how much crowding factor affects chance to die
CROWDING_BIAS = 0.0

#- allows selection between having the system maintain ant density, or ants are created manually
CREATE_ANTS = True

#- determines ratio of normal to swarming ants (0.0: all type 1 (normal), 1.0: all type 2 (swarming))
PROB_TYPE_2_ANT = 1.10

#- number of ants we want each node to see per observation period
#- -- this is number ants seen / size of obsrv. period in seconds
#DESIRED_ANT_DENSITY = 0.5
DESIRED_ANT_DENSITY = 0.8
OBSERVATION_PERIOD = 10.0

GRANULARITY = 5
NUM_BINS = 12
#AVERAGE_OVER_k_SECS = NUM_BINS*GRANULARITY
#ANT_COUNT_TARGET = (DESIRED_ANT_DENSITY * AVERAGE_OVER_k_SECS) / OBSERVATION_PERIOD

LINGER_TIME = 0.1
#LINGER_TIME = 0.001

death_chance = 0.3333333

#- when a sensor finds a target, it will take action to remove it
ACTIVE_RESPONSE = True

#- define the names for the directions
#- these must be in order clockwise (if names matter)
#- the number defines the number of links out of a cell
#- note: because of forcing to a 2-D plane, only 3,4,6,8 make sense (to me)(yet)
#-      but currently only 4 and 8 are valid
#-      but a hex only version is available
DIRECTIONS = ur r dr dl l ul
#DIRECTIONS = u r d l

#- maximum number of waiting connections on the socket
backlog = 1024

size = 1024

#- this controls the degree of random walk while foraging
forage_perturb = 40
follow_perturb = 3
drop_perturb = 10

#- this controls the likelihood an ant will follow an encountered path and will continue to follow it
follow_chance = 2.0

#- controls length of pheromone trail
```

```
max_pheromone_charge = 12

#- controls how long the pheromone will last
max_pheromone_strength = 4.0

#- swarm type percentage of max_pheromone_strength
type_2_strength_factor = 0.6

#- controls how ants can be recruited to an active task
recruitment_probability = 0.55
recruitment_duration = 5
recruiting_focus = 6

#- controls if and after how long unsuccessfully searching before ant will change to a different task
#- -- focus is number of steps before maybe changing
#- -- prob is how likely it is that the ant will change after focus is lost
#- -- a focus of zero mean never change task due to loss of focus
default_focus = 30
default_focus_change_prob = 0.50


#- Levy flights are a more realistic movement behavior for foraging ants.
#- After a period of unsuccessfull foraging, head out of the area and
#- try again, each failure results in a longer "flight" to the next
#- foraging area
USE_LEVY_FLIGHTS = False
LEVY_SCALE = 1.5
LEVY_WANDER = 0.1

#- track drop stats - as a simpler alternative to active agents to inform
#- neighbors of local activity, nodes may notice what is being dropped and
#- if they see a significant number of the same activity being dropped it
#- may choose to issue a special alert regarding widespread issues.
#- -- issue is that it only is responsive to activity signficant in itself
#-    to warant an ant alert.
TRACK_DROP_STATS = False
TDS_SIG_THRESH = 10

#- parameters controlling "bird" agents
USE_BIRDS = False
BIRD_RANGE = 10
BIRD_COUNT = 4
```