Test Design

Table of Contents

Introduction

This document defines testing procedures for comparing BATMAN and 80211s in a wireless MESH network. In the actual field tests, these procedures were not were not implemented precisely as described here. The specific as-conducted testing procedures are described in the companion document *Execution.pdf*.

The testing criteria are latency, throughput, overhead, routing-performance, and convergence.

- Latency is the time for a 'small packet' to travel from one node to another. A 'small packet' is packet that can be sent between adjacent nodes in a single transmission.
- Throughput is the maximum amount of data that can be sent from one node to another over a time interval.
- Overhead is the throughput degradation that occurs due to adding more nodes to the MESH.
- Routing-performance is the time for a MESH to re-route after a node in its current route fails to transmit.
- Convergence is time for a newly introduced node to be able to be seen by the MESH.

Conventions

Here are conventions this document follows to enhance readability.

- When referring to a particular node in a test, the node will be referred by the number on the node on the diagram for the test, and the number will be underlined. For example, a node might be referred to as 4.

- When executing the procedure of a script, arguments to scripts or utilities that are placed in brackets ('<' and '>') are meant to be replaced with the appropriate value. For example, `<network>` is meant to be replaced with a value indicating the network being tested[1].

- In order for environment variables set in the scripts to affect the environment of the shell in which they are executed, the scripts need to be run with the source command. This command is omitted for simplicity.

[1]See the *Arguments* section of *Scripts*.

Equipment

The following table lists the items which will be needed to run the tests. For each item, the table gives the quantity of the item and the purpose the item will serve.

Item	Quantity	Purpose
fully-charged, pre-configured[1] laptops	15	To serve as nodes.
laptop with screen capture program	1	To run Mesh Monitor and serve as node.
numbered wireless cards	16	The internal wireless cards of most laptops do not support an 80211s network.

[1]Refers to the platform configuration requirements which give the required OS and tools that must be installed on each laptop. Also a recent pull of the git repository must be performed.

Scripts

Arguments

Here are the definitions of the arguments:

- `<network>` can be one of the following values: `batman|olsrd|80211s.`
- `<run#>` is an integer representing the current run number.
- `<node#>` is an integer between 1 and 16 and corresponds to the number of a node in the diagram of the test currently being run.
- `<number_list>` is a whitespace-delimited list of integers.

Environment Variables

The following environment variables are initialized or changed by some of the scripts.

- `node1, node2, node3, ... , node16` -- each of these variable holds the ip address of its corresponding node.[1]
- `me` -- holds the node number of the node on which it is accessed.[1]
- `network` -- holds one of the following values: “batman”, “olsrd”, “80211s”.[2]
- `test` -- holds the name of the current test
- `run` -- holds the current run number

[1]Initialized by `assign`.
[2]Set by `enable`.

Conventions

- When a script description states that the results will be “written to file”, the path of this file is given by `results/$network/$test/$run`. While this script is running the data that is written to file is also written to standard output.

Definitions

- `timesync-server` broadcasts a packet every second containing a time and sets the system clock to the value sent in the packet. This packet is to be received by `timesync-client`.
- `timesync-client` listens for a packet sent by `timesync-server` and, upon receipt of this packet, sets the system clock to the value given in the packet.
- `assign <node#>` sets environment variables *node1*, *node2*, *node3*,..., *node16*. The ip addresses assigned is computed from a table that is the same across all nodes. Sets environment variable *me* with the value of the given argument.
- `enable <network>` brings up the given network type on the node and sets the environment variable *network*.
- `disable` brings down `$network` on the node. If `$network` is not set, brings down the wireless interface.
- `throughput-server` runs the iperf server.
- `throughput-client <node#>` runs the iperf client to send data to the node with the given `<node#>`. This script runs for 5 seconds. The results of the test are written to file.
- `latency <node#>` runs ping to send IMCP packets to the node with the given `<node#>`. This script runs for 5 seconds. The output of the test are written to file.
- `remote-disable <number_list>` broadcasts a disable-request packet containing the numbers in the given list. The disable-request is to be received by `disable-listener`.
- `disable-listener` listens for disable-requests and, upon receipt of such a packet, checks if `$me` is in list of numbers in the disable-request. If so, it disables its network interface.
- `udp-sender <node#>` sends UDP packets to the node with the given node number. The UDP packets are to be received by `udp-listener`.
- `udp-listener <test>` listens for packets sent by `udp-sender`. The time at which each packet is received from `udp-sender` will be written to file.

Configuration

The tests will be run in an open field free from radio interference. Before any of the tests are run, the following must be to each node:

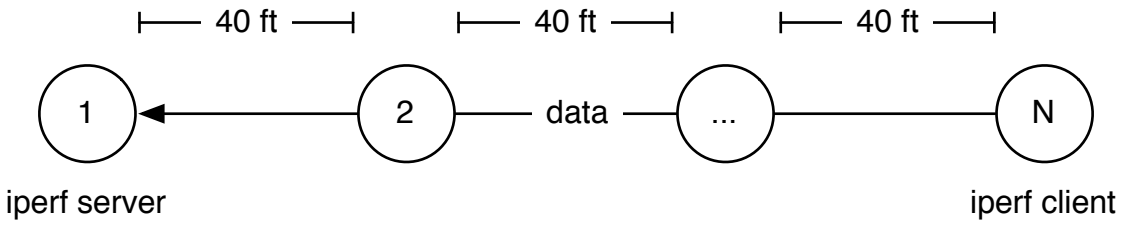
1. Power it on..
2. Insert a wireless card.
3. Open a terminal.
4. Execute `cd ~/MeshKit/testing`

5. Execute `sudo -i`
6. Execute `assign <node#>` where `<node#>` is the number on the wireless card.
7. Execute `test "$me"!='1' && timesync-client || timesync-server`
8. Execute `sleep 10`
9. Execute `disable`

Throughput Test

Plan

This test will be run 9 times. The nodes will be positioned in a line according to the diagram below. The first run will only enable the first two nodes. For each subsequent run, an additional node will be enabled, adding the next node in the line to the MESH. For each run, data will be sent from the newly added node to the other end of the line, and the data rate will be recorded.



Procedure

Before any of the runs, execute `test=throughput` on N.

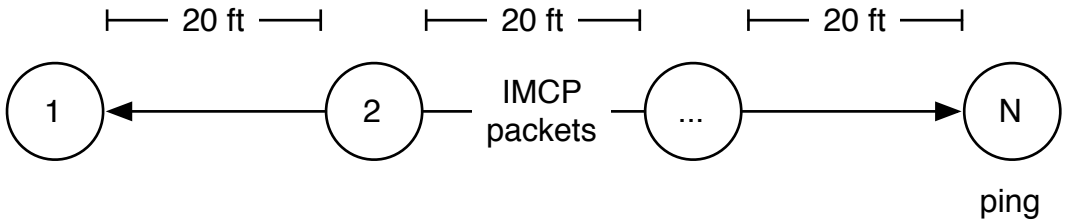
The following procedure will be executed for each run.

1. Execute `run=<run#>` on N.
2. Execute `enable <network>` on each node.
3. Execute `throughput-server&` on 1.
4. Execute `throughput-server_pid=$!` on 1.
5. Execute `throughput-client 1` on N.
6. Wait until step 5 finishes.
7. Execute `kill $throughput-server_pid` on 1.
8. Execute `disable` on each node.

Latency Test

Plan

This test will be run 9 times. The nodes will be positioned in a line according to the diagram below. The first run will only enable the first two nodes. For each subsequent run, an additional node will be enabled, adding the next node in the line to the MESH. For each run, data will be sent from the newly added node to the other end of the line and back, and the latency will be recorded as the round trip time.



Procedure

Before any of the runs, execute `test=latency` on N.

The following procedure will be executed for each run.

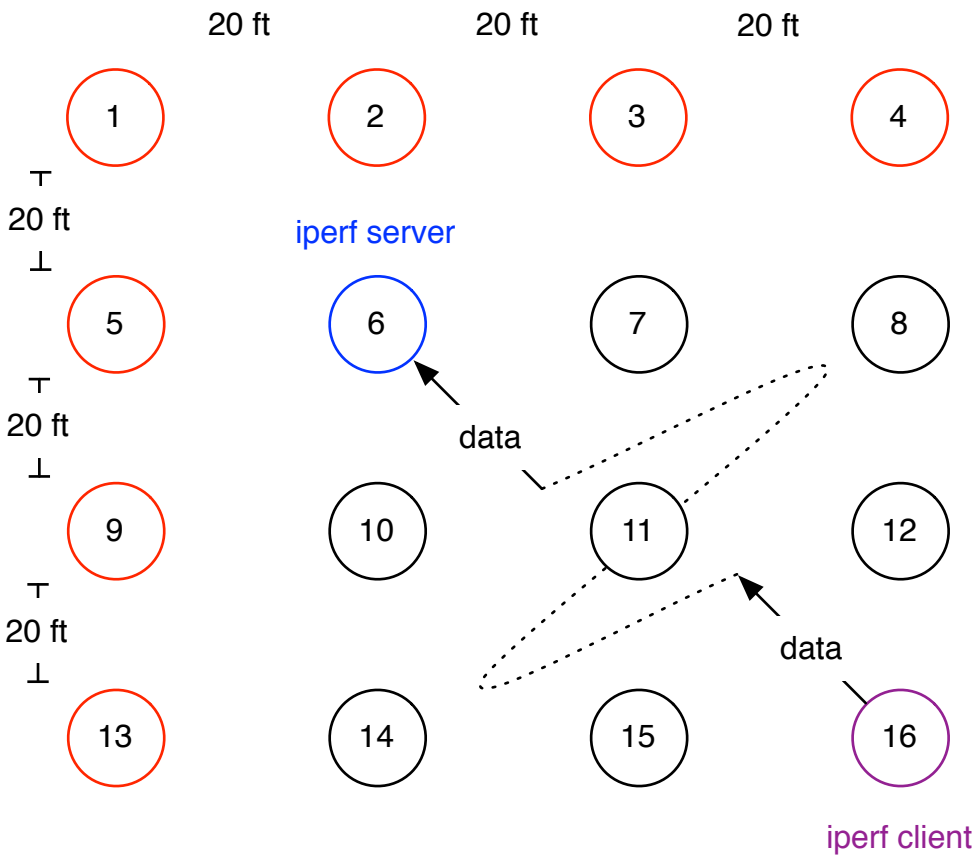
1. Execute `run=<run#>` on N.
2. Execute `enable <network>` on each node.
3. Execute `latency 1` on N.
4. Wait until step 3 finishes.
5. Execute `disable` on each node.

Overhead

Plan

This test will be run 5 times. The nodes will be positioned in a grid according to the diagram below. All nodes except the red ones will be enabled. Data will be sent from 16 to 6 along some route, and the throughput will be noted. Then the red nodes will be enabled creating more network overhead.

The absolute value of the change in the throughput will be recorded as delta. The overhead metric will be delta as a percentage of the original throughput and will be calculated by dividing the delta by original throughput. A low overhead metric is preferred.



Procedure

Before any of the runs, execute `test=overhead` on 16.

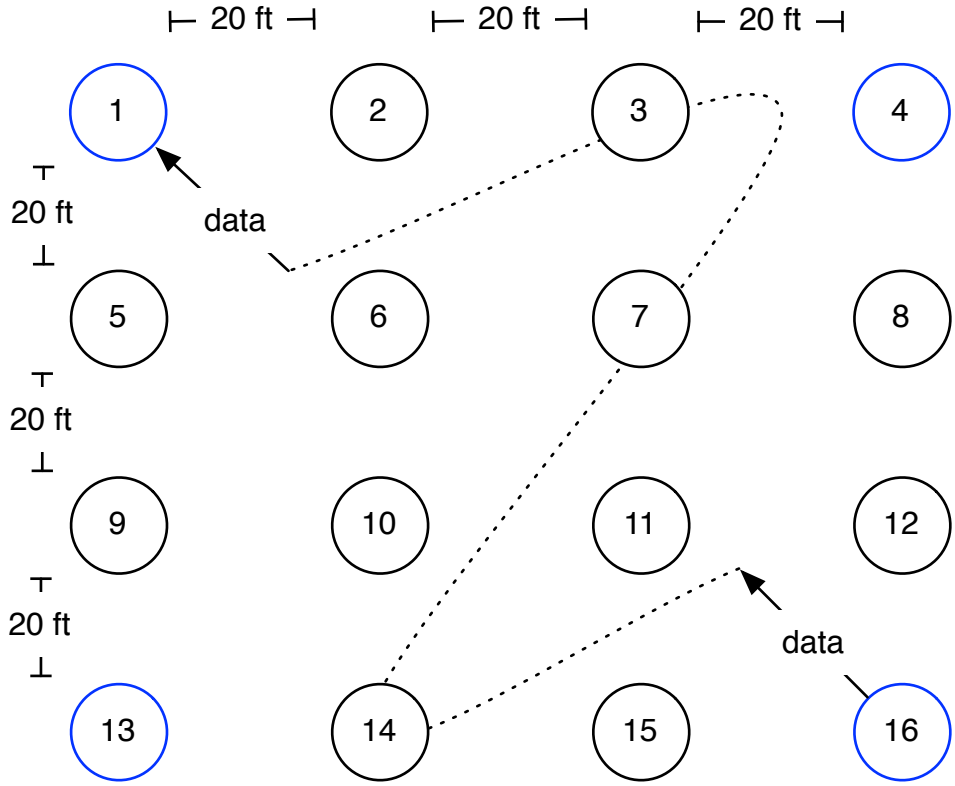
The following procedure will be executed for each run.

1. Execute `run=<run#>` on 16.
2. Execute `enable <network>` on each black, purple and blue node (6, 7, 8, 10, 12, 14, 15, 16).
3. Execute `throughput-server&` on 6.
4. Execute `throughput-server_pid=$!` on 6.
5. Execute `throughput-client 6` on the 16.
6. Wait until step 5 finishes.
7. Execute `enable <network>` on each red node (1, 2, 3, 4, 5, 9, 13).
8. Repeat steps 5 and 6.
9. Execute `disable` on each node.
10. Execute `kill throughput_server_pid` on 6.

Routing-performance Test A

Plan

This test will be run 5 times. The nodes will be positioned in a grid according to the diagram below. 16 will rapidly send UDP packets to 1. 1 will record the time at which it receives each UDP packet. traceroute will be run to ensure that the route does not use 13 or 4. Simultaneously all the black nodes will be disabled. When looking at the times at which 1 received each UDP packet, there should be a gap at some point. This gap is the routing-performance time.



Procedure

Before any of the runs, execute `test=routing-performanceA` on 1.

The following procedure will be executed for each run.

1. Execute `run=<run#>` on 1.
2. Execute `enable <network>` on each node.
3. Execute `disable-listener&` on each node.
4. Execute `disable-listener_pid=$!` on each node.
5. Execute `udp-listener&` on 1.
6. Execute `udp-listener_pid=$!` on 1.

7. Execute `udp-sender 1&` on 16.
8. Execute `udp-sender_pid=$!` on 16.
9. Execute `traceroute $node1` on 16. Inspect the output to ensure that the route does not include 13 or 4.
10. Execute `remote-disable 2 3 5 6 7 8 9 10 11 12 13 15` on 16.
11. Wait 5 seconds.
12. Execute `kill $disable-listener_pid` on each node.
13. Execute `kill $disable_listener_pid` on each node.
14. Execute `kill $udp-sender_pid` on 16.
15. Execute `kill udp-listener_pid` on 1.
16. Execute `disable` on each 1, 4, 13, and 16.

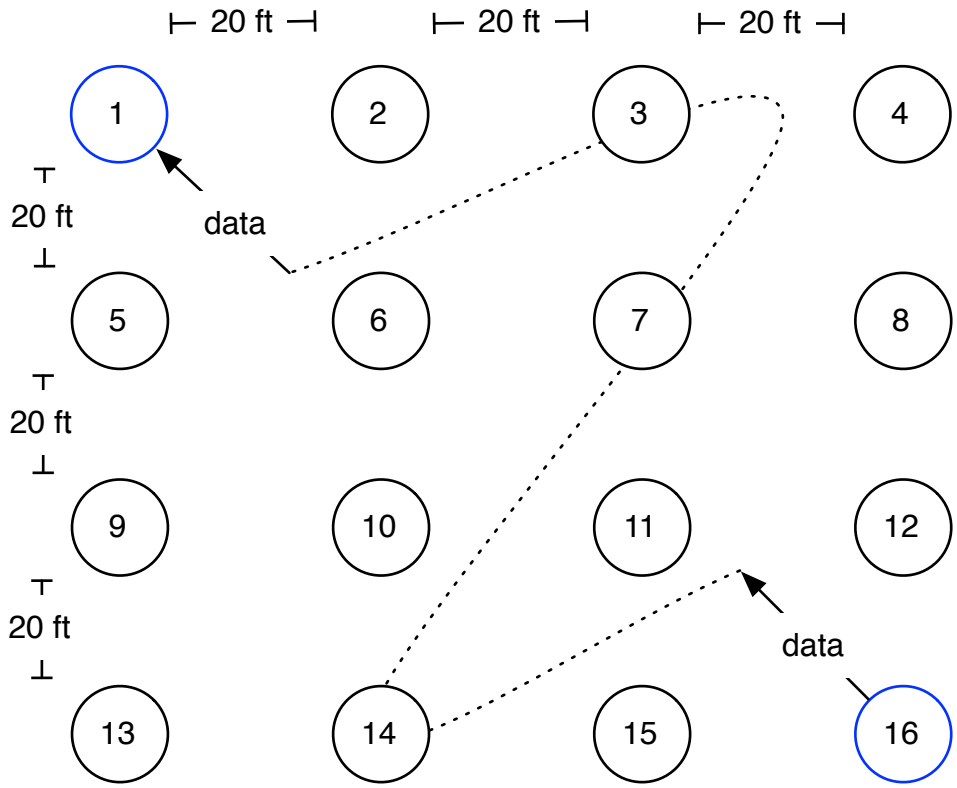
Routing-performance Test B

Plan

The nodes will be positioned in a grid according to the diagram below. 16 will rapidly send UDP packets to 1. 1 will record the time at which it receives each UDP packet. traceroute will be run to see which nodes are on the route. A random node in the route will be disabled. When looking at the times 1 received each UDP packet, there should be a gap at some point. This gap is the routing-performance time.

Note: The same nodes cannot be disabled when testing BATMAN against 80211s because the routes will be different, which may lead to an incorrect comparison if there are too few runs.

Layout



Procedure

Before any of the runs, execute

1. Execute `test=routing-performanceB` on 1.
2. Execute `run=<run#>` on 1.
3. Execute `enable <network>` on each node.
4. Execute `disable-listener&` on each node.
5. Execute `disable-listener_pid=$!` on each node.
6. Execute `udp-listener&` on 1.
7. Execute `udp-listener_pid=$!` on 1.
8. Execute `udp-sender 1&` on 16.
9. Execute `udp-sender_pid=$!` on 16.

The following procedure will be executed 10 times.

10. Execute `traceroute $node1` on 16. Randomly select a node `nodeX` in the route.
11. Execute `remote-disable $nodeX` on 16.
12. Wait 6 seconds.

Afterward, run the following:

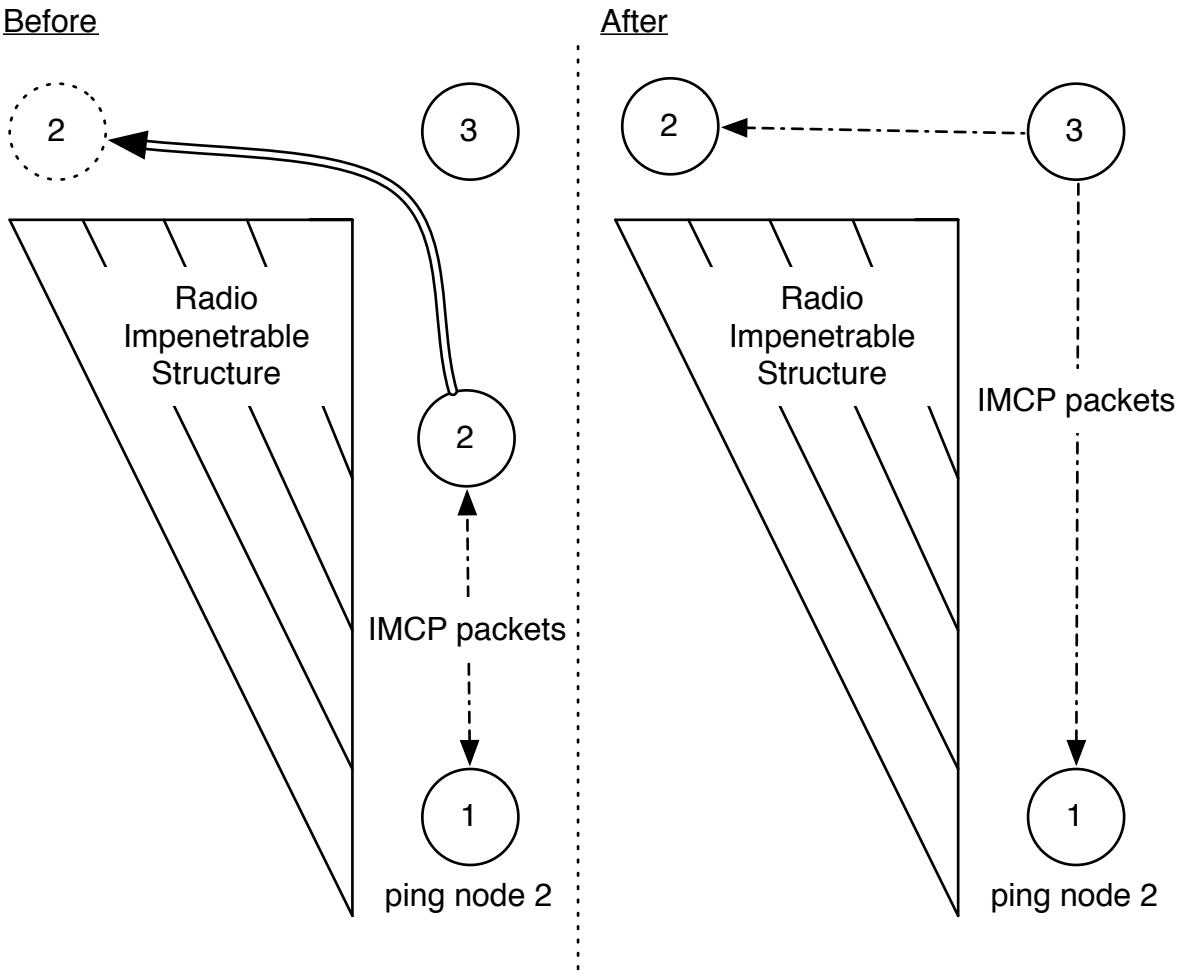
13. Execute `kill $disable-listener_pid` on each node.
14. Execute `kill $disable_listener_pid` on each node.
15. Execute `kill $udp-sender_pid` on 16.
16. Execute `kill udp-listener_pid` on 1.
17. Execute `disable` on each node.

Routing-performance Test C

Plan

1 will rapidly send UDP packets to 2. 2 will record the time at which it receives each UDP packet. 2 will be moved to an to a position at where 1 cannot see 2 directly, forcing 1 to re-route through 3. When looking at the times 2 received each UDP packet, there should be a gap at some point. This gap is the routing-performance time.

Layout



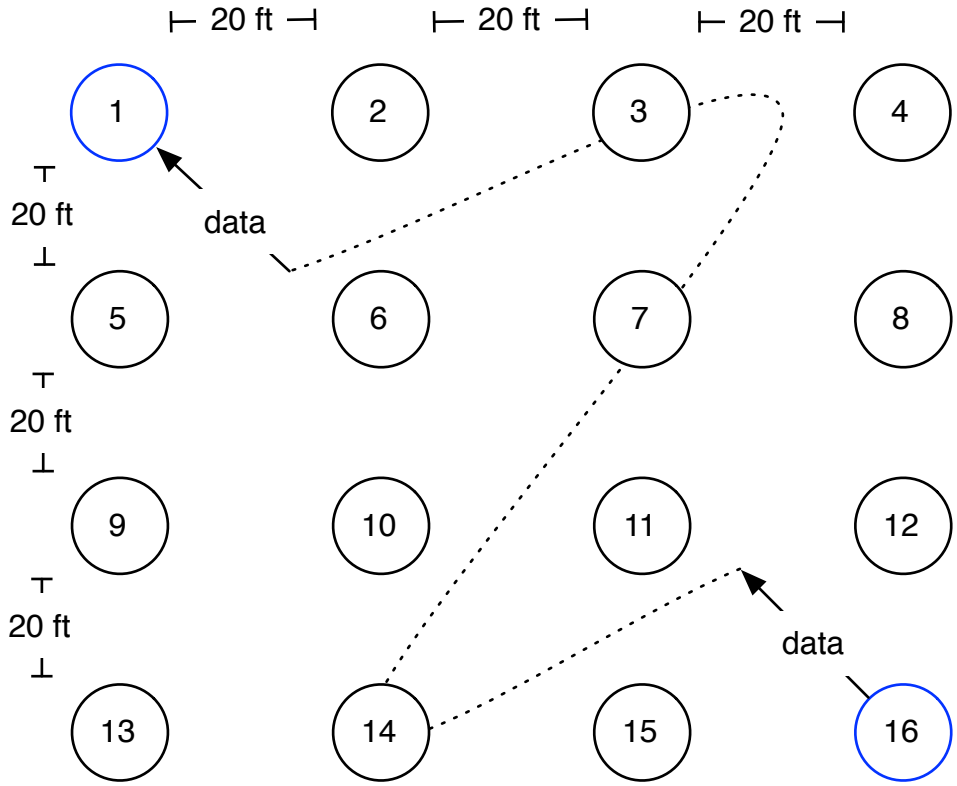
Monitor Test

Purpose

This test will measure how correctly the Mesh Monitor displays the mesh.

Plan

Data will be sent from 16 to 1. Traceroute will be running continuously; whenever it finishes, the time will be recorded and it will be restarted. A screen capture program will be run on 16 and the time recorded. The Mesh Monitor will be run on 16 and will be set to view the route between 16 and 1. One of the nodes in the route will be disabled and the time recorded--this step will be performed 10 times with at least 6 seconds between disabling each node. Later, the recorded data can be viewed to see how responsive the Mesh Monitor was.



Procedure

Repeat *Convergence Test B* but with

1. `heartbeat.py` running on each node
2. *Mesh Monitor* running on 16

3. the screen capture program running on 16 and `date > results/$network/$test/capture-time` executed as it is started
4. `test=monitor`
5. `traceroute $node1 > results/$network/$test/$run; date;` running repeatedly on 16.