Multi-Node Unit Testing

From TinyOS Wiki

Multi-Node unit tests require some thinking up front about how to setup the test. Unlike isolation tests and single-node test suites, there is typically only a single test in each multi-node test suite. A multi-node test implies the radio is in use and communicating to a nearby node.

Multi-node testing consists of a single Driving Node and several Supporting Nodes.

Multiple nodes in a TUnit test bed are intentionally placed close together to reduce the effects of RF issues. After all, we're testing software, not physical RF performance.

Contents

Test Basic Radio Communication

When using the radio, we need to turn it on before the test begins and turn it back off when the test is complete. To do this, access the *SetUpOneTime* and *TearDownOneTime* TestControl interfaces to turn the radio on and off.

```
TUnit Tip
Be polite to the next test suite that runs!
Always turn off your radio when your current test is complete with the
TearDownOneTime interface.
```

Every node defined in your tunit.xml Test Run will execute SetUpOneTime before the test runs and TearDownOneTime when the test completes.

Below we create a test for a 2-node Test Run where the Driving Node sends a single message to the Supporting Node. The Supporting Node turns on its radio and waits around to receive a message, and when it does, the test is complete.

Configuration Example

MultiNodeTestC.nc example configuration file

```
 0|  configuration MultiNodeTestC {
 1|  }
 2|
 3|  implementation {
 4|
 5|    components new TestCaseC() as TestSendReceiveC;
 6|
 7|    components MultiNodeTestP,
 8|      new AMSenderC(0),
 9|      new AMReceiverC(0),
10|      ActiveMessageC;
11|
12|    MultiNodeTestP.SetUpOneTime -> TestSendReceiveC.SetUpOneTime;
13|    MultiNodeTestP.TearDownOneTime -> TestSendReceiveP.TearDownOneTime;
14|
15|    MultiNodeTestP.TestSendReceive -> TestSendReceiveC;
16|    MultiNodeTestP.AMSend -> AMSenderC;
17|    MultiNodeTestP.Receive -> AMReceiverC;
18|    MultiNodeTestP.SplitControl -> ActiveMessageC;
19|
20|  }
```

Module Example

MultiNodeTestP.nc example module file

```
 0|  #include "TestCase.h"
 1|
 2|  module MultiNodeTestP {
 3|    uses {
 4|      interface TestControl as SetUpOneTime;
 5|      interface TestControl as TearDownOneTime;
 6|
 7|      interface TestCase as TestSendReceive;
 8|      interface AMSend;
 9|      interface Receive;
10|      interface SplitControl;
11|    }
12|  }
13|
14|  implementation {
15|
16|    message_t myMsg;
17|
18|    /***************** TestControl Events ****************/
19|    /**
20|     * This event is run once by EVERY node before the test begins.
21|     * We turn on the radio, and when the radio turns on it's done().
22|     */
23|    event void SetUpOneTime.run() {
24|      call SplitControl.start();
25|    }
26|
27|    /**
28|     * This event is run once by EVERY node after the test completes.
29|     * We turn off the radio, and when the radio turns off it's done().
30|     */
31|    event void TearDownOneTime.run() {
32|      call SplitControl.stop();
33|    }
34|
35|    /***************** SplitControl Events ****************/
36|    event void SplitControl.startDone(error_t error) {
37|      call SetUpOneTime.done();
38|    }
39|
40|    event void SplitControl.stopDone(error_t error) {
41|      call TearDownOneTime.done();
42|    }
43|
44|    /***************** Tests ****************/
45|    /**
46|     * This test is executed only on the Driving Node, node 0.
47|     * No Supporting Node can start a test.
```

```
48|     *
49|     * Since there will only be a single Supporting Node in this
50|     * test, we know its address will be 1, so we send the message
51|     * to address 1.
52|     */
53|    event void TestSendReceive.run() {
54|      error_t error;
55|      error = call AMSend.send(1, &myMsg, 0);
56|
57|      assertEquals("Send failed", SUCCESS, error);
58|
59|      if(error) {
60|        call TestSendReceive.done();
61|      }
62|    }
63|
64|
65|    /***************** Receive Events ****************/
66|    /**
67|     * Since the Driving Node is the only node that
68|     * gets to run the test and send a message, the only
69|     * node that can get this Receive event is a Supporting Node.
70|     */
71|    event message_t *Receive.receive(message_t *msg, void *payload, uint8_t len) {
72|      assertSuccess();
73|      call TestSendReceive.done();
74|    }
75|
76|
77|    /***************** AMSend Events ****************/
78|    event void AMSend.sendDone(message_t *msg, error_t error) {
79|    }
80|
81|  }
```

Notice a few things about this multi-node unit test:

1. The Driving Node is the only node allowed to run() the actual test. 2. Before the test on the Driving Node is run, the Supporting Node has already run SetUpOneTime and turned on its radio. 3. The Supporting Node didn't start the test, but is allowed to make assertions and even end the test on lines 72 and 73. Supporting Nodes may also send Statistics to the computer if you'd like, as is the case with 4. The radio is turned on and turned off in the *SetUpOneTime* and *TearDownOneTime* events on both nodes.

This test would require a suite.properties file that looks like:

```
0|  @exactnodes 2
1|
```

because the test is very specific to a 2-node Test Run.

Other Multi-Node Test Parameters

TUnit compiles in a preprocessor variable you can access to find out the number of nodes being handled in your current Test Run. This variable is **TUNIT_TOTAL_NODES**. This allows you to create a test that requires a minimum of 2 nodes but is allowed to scale up to a bunch of nodes.

Sometimes you'll also need to explicitly know whether or not your node is a Driving Node. To do this, access the ActiveMessageAddress.amAddress() interface OR AMPacket.address(), and compare it against address 0. The Driving Node is always address 0, with Supporting Nodes at addresses 1, 2, 3, etc.

See Also

- TUnit

Case Studies

- TestRxThroughputWithNoCca (http://tinyos.cvs.sourceforge.net/tinyos/tinyos-2.x-contrib/tunit/tests/tinyos-2.x/tos/chips/cc2420/TestRxThroughputWithNoCca/)
  - Upon receiving the first message from the Driving Node, the Supporting Node receiver starts a timer and accumulates received messages. When the timer fires, it reports performance and verifies enough packets got through.

- TestSwAcks (http://tinyos.cvs.sourceforge.net/tinyos/tinyos-2.x-contrib/tunit/tests/tinyos-2.x/tos/chips/cc2420/TestSwAcks/)
  - An idle Supporting Node simply sends back an acknowledgment to the Driving Node.

- TestLplDefaultContinuousDelivery (http://tinyos.cvs.sourceforge.net/tinyos/tinyos-2.x-contrib/tunit/tests/tinyos-2.x/tos/chips/cc2420/TestLplDefaultContinuousDelivery/)
  - A Test Suite where 2 to 19 Supporting Nodes deliver LPL messages to a single listener, the Driving Node. At the end of a period of time, a dynamic threshold is calculated at +/- 10% and assertions are made against each of the nodes in the network to determine if the channel is being shared fairly amongst the multiple transmitters.
