TUnit Philosophy

From TinyOS Wiki

TUnit is intended to be easy to use so people will want to use it. Admittedly, setting up TUnit may not be the easiest thing in the world. But once it's running and understood, it turns out to be a vital tool for writing quality, robust code.

Contents

Testing AND Characterization

Whereas other unit testing frameworks only test for pass/fail conditions, TUnit takes this a step further by allowing you to not only test performance and behavior, but also *characterize* performance and behavior. It does this through the use of Statistics. Single or multiple statistics may be characterized by your node, showing you over time how the performance of your system has been altered. So not only can you say "I want this test to fail if my performance drops below X%", but you can also see how the performance is being affected by the changes you make in your code over time.

TUnit will also track the ROM and RAM sizes of your test over time, adding extra information to your debugging process.

Example of TUnit Statistics being used to characterize code and hardware performance over time.

Testing Lots of Hardware

TUnit also differs from other unit testing platforms in that the code it tests is not expected to only run on a single processor. While JUnit easily runs tests on only a single machine, TUnit may run tests on several nodes at once. These nodes are always placed within clear RF range of each other to mitigate the effects of the physical environment on the code you are attempting to test. And there haven't been many test cases in the past where more than two nodes are required. A test with more than two nodes is normally more of a performance or characterization test, answering questions like "How well does my radio stack share the channel with multiple transmitters?" as opposed to "Does my code work?"

Any node taking part in a test can make assertions, end the test, and inject conditions, only the Driving Node is responsible for kicking things off. This prevents synchronization problems that may crop up during characterization testing.

Unit and Integration Testing

We explicitly avoid using TUnit to unit test full networks. Any test where you have an actual full blown network setup is no longer a unit test, it's more of a functional or system test. Instead of testing at the network level, TUnit is better aimed at testing the actual code responsible for forming the network. Code should always be written so it can be tested - and writing testable code implies that code is modular and inherently designed with architecture in mind. Collection and Dissemination can be tested by tricking nodes into believing there is are lots of nodes nearby (controlled by the test itself), and making sure the code behaves correctly under specified conditions.

As opposed to full integration testing or functional testing, the goal of unit testing is to break your code down into small pieces and isolate them from the rest of the system. Even the lowest levels of your platform's radio stacks may be tricked into believing they are actually talking to a radio when they aren't, allowing you to truly see how the modules behave by forming your own conditions. For example, when applying unit tests to the lowest levels of the standard TinyOS 2.x CC2420 receive branch, we've discovered several bugs that not one of the vast number of CC2420 users has known about or brought up. These bugs occur so intermittently there would be no way to discover they exist without unit testing.

In fact, taking a radio stack as a further example, we can isolate each layer and individual module of the stack to test functionality on an individual level. After verifying each layer and interface behaves as it should, we can then use TUnit to perform automated integration testing between multiple modules and radio hardware. Finally, we can do some amount of integration testing to verify multiple nodes can communicate using a full radio stack and the actual radio hardware. We can also use TUnit to characterize the throughput, acknowledgment success rate, reception rate, and more of the radio stack.

See Also

- TUnit
- Setting up TUnit

Next

- TUnit Test Flow

Retrieved from "http://tinyos.stanford.edu/tinyos-wiki/index.php?title=TUnit_Philosophy&oldid=556"

- This page was last modified on 15 January 2008, at 08:05.
- This page has been accessed 8,134 times.