# Characterization Testing with Statistics

From TinyOS Wiki

Example of a single statistics plot graphing a single value over time. We can see how editing code affected the performance of the CC2420 radio until the issue was improved.

Statistics allow TUnit to *characterize* performance, giving us a better insight into what's happening above and beyond simple threshold and assertion testing. This is very important for embedded systems, where hardware speed and interaction can be critical.

## Contents

- 1 Statistics Introduction
  - 1.1 Interface
  - 1.2 Configuration Usage
  - 1.3 Module Usage
  - 1.4 Storage and Reports
- 2 See Also
- 3 Case Studies
- 4 Next

## Statistics Introduction

### Interface

Statistics, like assertions, must be issued only while a test is running, before calling done(). The Statistics interface (http://tinyos.cvs.sourceforge.net/*checkout*/tinyos/tinyos-2.x-contrib/tunit/tos/interfaces/Statistics.nc?content-type=text%2Fplain) looks like this:

```
interface Statistics {

  /**
   * Log some statistics. TUnit takes care of the split-phase stuff so
   * your application doesn't have to. If you call log() properly, you can
   * safely assume your statistics will get sent to the computer before
   * the test is allowed to stop.
   */
  command error_t log(char *units, uint32_t value);

}
```

### Configuration Usage

You create a new Statistics logger in your test's configuration file. Below is a truncated example, and the full version of this throughput test can be found in tinyos-2.x-contrib/tunit/tests/tinyos-2.x/tos/chips/cc2420/TestTxThroughputNoCca/ (http://tinyos.cvs.sourceforge.net/tinyos/tinyos-2.x-contrib/tunit/tests/tinyos-2.x/tos/chips/cc2420/TestTxThroughputNoCca/) .

```
configuration TestTunitC {
}

implementation {
  components TestThroughputP,
      new TestCaseC() as TestThroughputC,
      new StatisticsC() as ThroughputStatsC;

  ...

  TestThroughputP.Statistics -> ThroughputStatsC;

  ...

}
```

We can create as many StatisticsC instances as we want and give them unique names. The name we give each StatisticsC instance is the name used at the top of the Statistics graph. Each StatisticsC instance will plot exactly one integer value over time, stored on the computer. Notice the name of the graph on the right is the same as name we gave the StatisticsC component.

## Module Usage

Call the Statistics interface a maximum of one time for each Test Suite to plot a single value over time. TUnit will ensure that each Statistics message gets across to the computer before running the next Test Case or exiting the Test Suite. Again, the full test case is in the repository (http://tinyos.cvs.sourceforge.net/tinyos/tinyos-2.x-contrib/tunit/tests/tinyos-2.x/tos/chips/cc2420/TestTxThroughputNoCca/) .

```
module TestTunitP {
  uses {
    interface TestCase as TestThroughput;
    interface Statistics;
    interface Timer<TMilli>;
    ...
  }
}

implementation

  ...

  /***************** Timer Events ****************/
  event void Timer.fired() {
    call RunState.toIdle();
    call Statistics.log("[packets/sec]", (uint32_t) ((float) sent / (float) 60));
    assertResultIsAbove("Throughput is too low", LOWER_BOUNDS, sent);
    call TestThroughput.done();
  }

}
```

We see that the log command provides a unit for the statistics in [packets/sec], and gives the single-value statistics to be logged in *integer* form. In this case, we find out how many packets were sent over the course of 60 seconds. The Timer we show takes 60 seconds to fire, during which time the mote is sending packets as fast as possible. When the Timer fires, dividing the number of packets sent by 60 gives us packets per second.

The next line defines a threshold of failure. Our throughput must be above some *LOWER_BOUNDS* or the test fails.

Finally, after assertions and statistics were logged, the test is done().

```
TUnit Tip
Keep the unit description succinct. A better way to write the units [packets/sec] would have
been [pkts/sec]. We use [ and ]'s around our unit descriptions for clarity.
```

This particular Test Case, having a Timer that takes 60 seconds to fire, would normally cause TUnit to timeout. The suite.properties file for this test (http://tinyos.cvs.sourceforge.net/tinyos/tinyos-2.x-contrib/tunit/tests/tinyos-2.x/tos/chips/cc2420/TestTxThroughputNoCca/suite.properties?view=markup) explicitly increases the timeout value to 2 minutes to allow the test to run to completion.

If we wanted to graph a second value over time, like Packets per Minute, we'd have to create a *new StatisticsC() as PacketsPerMinuteC;* in our configuration file, wire to the test module, and do something like:

```
call PacketsPerSecond.log("[pkts/sec]", (uint32_t) ((float) sent / (float) 60));
call PacketsPerMinute.log("[pkts/min]", sent);
```

### Storage and Reports

All statistics are stored in your local TUnit results/stats directory in both CSV (comma-delimited) raw data files and .png image files. For example, the http://www.lavalampmotemasters.com website stores the progress of testing over time in reports/stats/TotalProgress.csv (http://www.lavalampmotemasters.com/reports/stats/TotalProgress.csv) , which is translated to the image TotalProgress.png (http://www.lavalampmotemasters.com/reports/stats/TotalProgress.png) with the help of TUnit and JFreeChart.

When running Ant to create HTML reports, TUnit post-processing (executed from the build.xml (http://tinyos.cvs.sourceforge.net/*checkout*/tinyos/tinyos-2.x-contrib/tunit/build_example.xml?content-type=text%2Fplain) file with Ant) automatically edits the HTML reports to include the .png images for each test.

## See Also

- TUnit Philosophy
- TUnit's internal statistics test (http://tinyos.cvs.sourceforge.net/tinyos/tinyos-2.x-contrib/tunit/tests/tinyos-2.x/tos/lib/tunit/TestStatistics/)
- TestThroughputNoCca example (http://tinyos.cvs.sourceforge.net/tinyos/tinyos-2.x-contrib/tunit/tests/tinyos-2.x/tos/chips/cc2420/TestTxThroughputNoCca/)
- Statistics library implementation in TUnit (http://tinyos.cvs.sourceforge.net/tinyos/tinyos-2.x-contrib/tunit/tos/lib/tunitstats/)

## Case Studies

- TestTxThroughputNoCca (http://tinyos.cvs.sourceforge.net/tinyos/tinyos-2.x-contrib/tunit/tests/tinyos-2.x/tos/chips/cc2420/TestTxThroughputNoCca/)
  - Collecting transmitter throughput statistics.

- TestMac (http://tinyos.cvs.sourceforge.net/tinyos/tinyos-2.x-contrib/tunit/tests/tinyos-2.x/tos/chips/general_radio_tests/TestMac/)
  - Multiple statistics logging

## Next

- Multi-Node Unit Testing