SWANS++ User's Guide

by David Choffnes

Last updated: 04/09/07

I. Introduction

By now, you should have read the SWANS++ project Web page, downloaded the code and used the installation instructions to install SWANS++. The purpose of this guide is to explain some of the more important features of SWANS++ and how to use them in your simulations. This is **not** intended to be a complete description of SWANS++, or anything close. For implementation details, read the source, which is generally commented well. If you have questions, you can always post them on the forum at our SourceForge project page.

II. Configuration

SWANS++ currently supports a large number configuration parameters that can be set at runtime. For the most part, runtime configurations are provided for the most commonly changed elements and to leave the rest as part of the compilation unit.

If you've used JiST/SWANS before, you know that simulations are started (conventionally) by running a "driver" file, i.e., a file located in the jist.swans.driver package. Each of these files allows one to run a simulation instance by initializing simulated objects, "gluing" them all together and starting the simulation. We found that writing a separate driver file for each simulation configuration was quite tedious and redundant, so SWANS++ now includes a GenericDriver class that attempts to merge all common simulation scenarios into one file that is easily configurable in terms of environment settings.

The command line proved to be too simple of an interface for controlling these settings. We currently use XML files to configure runtime parameters for simulations, as demonstrated in the GenericDriver class. Specifically, we use XML serialization and deserialization of the JistExperiment class to configure experiments. The JistExperiment class is expected to be used as a singleton object and thus is accessible staticly from any class in the simulation environment. This makes it particularly convenient to access simulation configuration data.

Currently, configuration data for all configurable classes is stored in JistExperiment. I've wrestled with providing subclasses or per-class configuration files, but I think that the single configuration object is currently the easiest way to go. My experience with embedding configuration objects inside JistExperiment is that XML deserialization fails. If someone gets this to work, please contact the lead developer, David Choffnes.

Each configurable setting has a default value specified in JistExperiment. If you want to use the default value for a setting, simply omit that settings from the XML file that you use to run a simulation instance. See the included XML files for examples of setting different levels of simulation detail. Finally, note that the JistExperiment class contains descriptions of each simulation parameter. For the sake of brevity, I will not repeat them here.

III. Visualization (Ceratias)

Perhaps the most anticipated feature of SWANS++ is the include runtime visualization and steering tool, **Ceratias**. As such, most of you will want to visualize your simulations as soon as you can get it up an running. Before you dive in, take a few moments to read about the design and implementation of this feature.

First of all, I created Ceratias not because I'm a visualization expert nor because I like creating fancy UIs. I created it because I just plain needed it. Let's face it: text-based logging of simulation events occurring over hundreds (if not more) nodes is way too tedious to process. On the other hand, it's very easy to check complicated, perhaps cross-node, conditions simply by

looking at how nodes are behaving. Born of necessity, Ceratias is not the be-all or end-all of visualization tools for simulation. It will not fix your code for you, and it will not visualize *anything* unless you explicitly code for it. What Ceratias will do is allow an experienced author of simulation code to see what was previously obfuscated by endless lines of log output. If used properly, it will help the developer to find flaws in mobililty models and protocol code. It also provides a convenient and appealing way to showcase your work to others by *showing* them instead of simply telling them about an idea.

Now that I've (hopefully) lowered your expectations sufficiently, let me tell what what Ceratias *does* do. Ceratias is a tool for generically visualizing and steering an ongoing simulation. Let's take these features one at a time.

- **Ceratias is generic**: It provides basic abstractions for simulation visualization, such as icons for nodes, coloring of those nodes, animated transmission circles, display of text-based output and marking of the field (e.g., drawing a persistent circle on the field). With these abstractions, I have been able to sufficiently instrument mobility models and routing protocol code so as to efficiently debug and validate their operation.
  Because Ceratias is generic, it does not visualize anything for which it has not been *instrumented*. In the initial version of SWANS++, I have instrumented the GPSR, DSR (ns-2 port) routing protocols. Additionally, I have provided visualization for most (if not all) of the mobility models.
- **Visualization**: The visualization window enables access to view the ongoing simulation. While it is running, you can change the zoom level in addition to toggling certain visualization features. Running Ceratias in parallel with your simulation will reduce performance compared to running it "naked." However, Ceratias allows you to hide various components of the tool to improve performance. For example, by hiding the simulation field, you can recover *nearly all* of the performance overhead from running the tool. For example, if you want to speed things up, you hide the simulation field until the simulation time at which you want to take a look at what's going on.
  In future versions of the tool, we will provide the ability to add custom tabs and visualization abstractions.
- **Steering**: In short, simulation steering is the ability to change simulation settings at runtime. Ceratias provides a convenient interface to access the steering functionality added by SWANS++. Perhaps the most simple feature is to pause and resume the simulation via Ceratias. This feature actually blocks processing of the simulation event queue and incurs zero CPU overhead. Other features include starting and stopping of individual nodes. This, for example, can simulate a car crash. I've included these basic steering capabilities for now; more advanced ones are currently being tested and will become public in the future.
- **Runtime Interaction**: Ceratias shows you the state of the system as it runs. It does not use or create trace files, though nothing precludes it from doing so. The important point to remember is that when using Ceratias, what you see is what you get. For instance, as soon as a routing-protocol problem manifests itself, you can instantly diagnose it and debug it, without waiting for the simulation to finish (as done with trace-based visualizations).

Ceratias is under constant development. If you are interested in contributing to this effort, please contact the lead developer (David Choffnes) before writing any new code--If your code does not fit my model for visualization, I won't accept it.

Finally, to enable visualization, you must set the "useVisualizer" property of the JistExperiment object to "true".

IV. Street Mobility (STRAW)

If using **STRAW** for your mobility model, please see the STRAW documentation.