SAFE: Simulation Automation Framework for Experiments

L. Felipe Perrone
Andrew W. Hallagan (BCSE ’11)
Bryan C. Ward (BCSE ’11)

Department of Computer Science
Bucknell University
General architecture

- Experiment Execution Manager
  - Design Points Generation
  - Simulation Client
    - Code Generation
      - ns-3 script
      - ns-3 output data
    - simulation data
  - model, design point
  - client machine
  - user interface
    - Experiment Description
    - Model Description
    - Data Extraction
    - Visualization
  - server machine
  - Database
Experienced user interface

- Status: in progress (Andrew Hallagan).
- Design language to describe experiments.
- Design language to describe model used in experiments.
- Validate documents written in languages above.
- Generate points in design of experiment space and translate them into executable ns-3 scripts.
Status: in progress (Bryan Ward).

- Control the execution of simulations for each design point.
- Implement MRIP functionality on a collection of networked hosts.
- Design database to store experiment description and output data.
- Provide hooks to terminate individual simulation runs when they are determined 'complete'.
- Provide mechanism to record samples of metrics and ignore (or not) data generated before steady-state.
Web browser interface to:

- Define experiments getting input from forms.
- Launch execution of experiments.
- Control experiments running on distributed hosts.
- Issue database queries to pull experimental data.
- Visualize graphed output data.
- Process output data for interoperation with external tools.
Steady-state detection

- Status: started, but little advance on implementation.
- C++ statistics class ported from SWAN and augmented.
- Takes the shape of an external program to process metric samples already extracted from database.
- Simple strategies to determine end of transient didn’t yield accurate results. Review of current literature indicates several possible algorithms.
Data collection

- Status: in progress (Felipe Perrone).
- Record ‘samples’ of variables (attributes and non-attributes) every time there’s a change in their value. Tag data with timestamp and an identifier of the source.
- Compute basic statistics on samples.
Concepts

- **Probe**: Mechanism for detecting changes to a variable.
- **Collector**: Contains and processes samples generated by a probe.
- **Aggregator**: Dispatches samples to appropriate output – stdout if simulation is not run from SAFE or the ‘Simulation Client’ otherwise. Encapsulates the protocol used to talk to the latter.
Probe/Collector/Aggregator

ns-3 simulation script

probe collector aggregator

Simulation Client
Data collection requirements - control

**Global disable:** No samples; negligible run time cost.

**Global enable:** All probes report samples during a window of simulation time specified by a start and an end value. Outside this window, no samples are reported.

**Local enabled:** Only individually selected probes report samples during a window of simulation time.
Data collection requirements - sample types

**Integer:** A standard integer data type (64 bit?)

**Double:** A standard double data type.

**Scalar:** The probe generates scalar data types.

**Non-scalar:** The probe generates a data type that can be seen as a collection of scalar values (e.g. a vector of values).
Data collection requirements - reporting

On change: A new sample of an enabled probed is generated when its value changes.

Format: Samples correspond to messages like
<timestamp,metric id,value>
Points to notice

- ns-3 defines the TracedValue template class - when a variable changes, a pre-determined function is called.

- Some ns-3 classes use TracedValue to define *trace sources*, which can be connected to *trace sinks* via `Config::Connect` (one identifies the source using a path to the right object and the callback to serve as sink).

- What we are calling *probe* is not exactly a trace source. The value monitored by a probe is not an attribute; it can be “just a variable” in the scope of some method (main use case).
Implementation (1)

- Probe abstract class:
  - static Ptr<Probe> CreateProbe(Ptr<Object> owner, TypeId tid));
  - virtual bool GetProbeStatus (void) const = 0;
  - virtual void SetProbeStatus (bool enabled) = 0;
  - boolean status is an attributed of Probe

- Derive classes: ProbeInt, ProbeDouble, ProbeIntVector, ProbeDoubleVector
Implementation (2)

- Collector class:
  - Passes samples along without any processing.
  - Passes along statistics on windows of samples.
  - Computes reductions on non-scalar samples.
Aggregator class:
- Singleton configured with the right destination for output of data.
- Implements the protocol to communicate data downstream.