ns-3 Waf build system

ns-3 Annual Meeting
June 2017
Software introduction

- Download the latest release
  - `tar xjf ns-allinone-3.26.tar.bz2`

- Clone the latest development code
  - `hg clone http://code.nsnam.org/ns-3-allinone`

Q. What is "hg clone"?
A. Mercurial (http://www.selenic.com) is our source code control tool.
Software building

- Two levels of ns-3 build

1) **bake** (a Python-based build system to control an ordered build of ns-3 and its libraries)

2) **waf**, a build system written in Python

3) **build.py** (a custom Python build script to control an ordered build of ns-3 and its libraries)  
   \(\text{\textless--- may eventually be deprecated}\)
ns-3 uses the 'waf' build system

• Waf is a Python-based framework for configuring, compiling and installing applications.
  – It is a replacement for other tools such as Autotools, Scons, CMake or Ant
  – http://code.google.com/p/waf/

• For those familiar with autotools:
  • configure ➔ ./waf configure
  • make ➔ ./waf build
waf configuration

• Key waf configuration examples
  ./waf configure
    --enable-examples
    --enable-tests
    --disable-python
    --enable-modules

• Whenever build scripts change, need to reconfigure

Demo: ./waf --help
  ./waf configure --enable-examples --enable-tests --enable-modules='core'

Look at: build/c4che/_cache.py
def build(bld):
    obj = bld.create_ns3_module('csma', ['network', 'applications'])
    obj.source = [
        'model/backoff.cc',
        'model/csma-net-device.cc',
        'model/csma-channel.cc',
        'helper/csma-helper.cc',
    ]
    headers = bld.new_task_gen(features=['ns3header'])
    headers.module = 'csma'
    headers.source = [
        'model/backoff.h',
        'model/csma-net-device.h',
        'model/csma-channel.h',
        'helper/csma-helper.h',
    ]

    if bld.env['ENABLE_EXAMPLES']:
        bld.add_subdirs('examples')

    bld.ns3_python_bindings()
waf build

- Once project is configured, can build via 
  ./waf build or ./waf
- waf will build in parallel on multiple cores
- waf displays modules built at end of build

Demo: ./waf build

Look at: build/ libraries and executables
Running programs

- ./waf shell provides a special shell for running programs
  - Sets key environment variables

  ./waf --run sample-simulator
  ./waf --pyrun src/core/examples/sample-simulator.py
Build variations

• Configuring a build type is done at waf configuration time
• debug build (default): all asserts and debugging code enabled
  ./waf -d debug configure
• optimized
  ./waf -d optimized configure
• static libraries
  ./waf --enable-static configure
Controlling the modular build

• One way to disable modules:
  – ./waf configure --enable-modules='a','b','c'

• The .ns3rc file (found in utils/ directory) can be used to control the modules built

• Precedence in controlling build
  1) command line arguments
  2) .ns3rc in ns-3 top level directory
  3) .ns3rc in user's home directory

Demo how .ns3rc works
Building without wscript

• The scratch/ directory can be used to build programs without wscripts

Demo how programs can be built without wscripts
Simulator core

- Simulation time
- Events
- Simulator and Scheduler
- Command line arguments
- Random variables

Execute a function (may generate additional events)

Advance the virtual time to the next event (function)
Simulator example

```cpp
#include <iostream>
#include "ns3/simulator.h"
#include "ns3/nstime.h"
#include "ns3/command-line.h"
#include "ns3/double.h"
#include "ns3/random-variable-stream.h"

using namespace ns3;

int main (int argc, char *argv[])
{
  CommandLine cmd;
  cmd.Parse (argc, argv);

  MyModel model;
  Ptr<UniformRandomVariable> v = CreateObject<UniformRandomVariable> ();
  v->SetAttribute ("Min", DoubleValue (10));
  v->SetAttribute ("Max", DoubleValue (20));

  Simulator::Schedule (Seconds (10.0), &ExampleFunction, &model);
  Simulator::Schedule (Seconds (v->GetValue ()), &RandomFunction);

  EventId id = Simulator::Schedule (Seconds (30.0), &CancelledEvent);
  Simulator::Cancel (id);

  Simulator::Run ();

  Simulator::Destroy ();
}
```
Simulator example (in Python)

```python
# Python version of sample-simulator.cc
import ns.core

def main(dummy_argv):
    model = MyModel()
    v = ns.core.UniformRandomVariable()
    v.SetAttribute("Min", ns.core.DoubleValue(10))
    v.SetAttribute("Max", ns.core.DoubleValue(20))

    ns.core.Simulator.Schedule(ns.core.Seconds(10.0), ExampleFunction, model)
    ns.core.Simulator.Schedule(ns.core.Seconds(v.GetValue()), RandomFunction, model)
    id = ns.core.Simulator.Schedule(ns.core.Seconds(30.0), CancelledEvent)
    ns.core.Simulator.Cancel(id)

    ns.core.Simulator.Run()
    ns.core.Simulator.Destroy()

if __name__ == '__main__':
    import sys
    main(sys.argv)
```
Simulation program flow

1. Handle program inputs
2. Configure topology
3. Run simulation
4. Process outputs
Command-line arguments

• Add CommandLine to your program if you want command-line argument parsing

```c
int main (int argc, char *argv[]) {
    CommandLine cmd;
    cmd.Parse (argc, argv);
}
```

• Passing --PrintHelp to programs will display command line options, if CommandLine is enabled

  ```bash
  ./waf --run "sample-simulator --PrintHelp"
  ```

-PrintHelp: Print this help message.
-PrintGroups: Print the list of groups.
-PrintTypeIds: Print all TypeIds.
-PrintGroup=[group]: Print all TypeIds of group.
-PrintAttributes=[typeid]: Print all attributes of typeid.
-PrintGlobals: Print the list of globals.
Time in ns-3

- Time is stored as a large integer in ns-3
  - Minimize floating point discrepancies across platforms
- Special Time classes are provided to manipulate time (such as standard operators)
- Default time resolution is nanoseconds, but can be set to other resolutions
  - Note: Changing resolution is not well used/tested
- Time objects can be set by floating-point values and can export floating-point values
  
  ```
  double timeDouble = t.GetSeconds();
  ```
  
  - Best practice is to avoid floating point conversions where possible
Events in ns-3

• Events are just function calls that execute at a simulated time
  – i.e. callbacks
  – this is another difference compared to other simulators, which often use special "event handlers" in each model

• Events have IDs to allow them to be cancelled or to test their status
Simulator and Schedulers

- The Simulator class holds a scheduler, and provides the API to schedule events, start, stop, and cleanup memory.
- Several scheduler data structures (calendar, heap, list, map) are possible.
- "RealTime" simulation implementation aligns the simulation time to wall-clock time.
  - Two policies (hard and soft limit) available when the simulation and real time diverge.
Random Variables

- Currently implemented distributions
  - Uniform: values uniformly distributed in an interval
  - Constant: value is always the same (not really random)
  - Sequential: return a sequential list of predefined values
  - Exponential: exponential distribution (poisson process)
  - Normal (gaussian), Log-Normal, Pareto, Weibull, triangular

```python
# Demonstrate use of ns-3 as a random number generator integrated with
# plotting tools; adapted from Gustavo Carneiro's ns-3 tutorial
import numpy as np
import matplotlib.pyplot as plt
import ns.core

# mu, var = 100, 225
rng = ns.core.NormalVariable(100.0, 225.0)
x = [rng.GetValue() for t in range(100000)]

# the histogram of the data
n, bins, patches = plt.hist(x, 50, normed=1, facecolor='g', alpha=0.75)

plt.title('ns-3 histogram')
plt.text(60, .025, r'$\mu=100, \sigma=15$')
plt.axis([40, 160, 0, 0.03])
plt.grid(True)
plt.show()
```
Random variables and independent replications

• Many simulation uses involve running a number of *independent replications* of the same scenario

• In ns-3, this is typically performed by incrementing the simulation *run number* — *not by changing seeds*