
ns-3 Training

Debugging

**MNM Workshop
March 2015**

Writing and debugging your own examples

Writing and debugging new programs

- Choosing between Python and C++
- Reading existing code
- Understanding and controlling logging code
- Error conditions
- Running programs through a debugger

Python bindings

- ns-3 uses the 'pybindgen' tool to generate Python bindings for the underlying C++ libraries
- Existing bindings are typically found in the bindings/ directory of a module
- Some methods are not provided in Python (e.g. hooking trace sources)
- Generating new bindings requires a toolchain documented on the ns-3 web site

Debugging support

- Assertions: `NS_ASSERT (expression);`
 - Aborts the program if expression evaluates to false
 - Includes source file name and line number
- Unconditional Breakpoints: `NS_BREAKPOINT ();`
 - Forces an unconditional breakpoint, compiled in
- Debug Logging (not to be confused with tracing!)
 - Purpose
 - Used to trace code execution logic
 - For debugging, not to extract results!
 - Properties
 - `NS_LOG*` macros work with C++ IO streams
 - E.g.: `NS_LOG_UNCOND ("I have received " << p->GetSize () << " bytes");`
 - `NS_LOG` macros evaluate to nothing in optimized builds
 - When debugging is done, logging does not get in the way of execution performance

Debugging support (cont.)

- Logging levels:
 - NS_LOG_ERROR (...): serious error messages only
 - NS_LOG_WARN (...): warning messages
 - NS_LOG_DEBUG (...): rare ad-hoc debug messages
 - NS_LOG_INFO (...): informational messages (eg. banners)
 - NS_LOG_FUNCTION (...):function tracing
 - NS_LOG_PARAM (...): parameters to functions
 - NS_LOG_LOGIC (...): control flow tracing within functions
- Logging "components"
 - Logging messages organized by components
 - Usually one component is one .cc source file
 - NS_LOG_COMPONENT_DEFINE ("OlsrAgent");
- Displaying log messages. Two ways:
 - Programatically:
 - LogComponentEnable("OlsrAgent", LOG_LEVEL_ALL);
 - From the environment:
 - NS_LOG="OlsrAgent" ./my-program

Running C++ programs through gdb

- The gdb debugger can be used directly on binaries in the build directory
- An easier way is to use a waf shortcut

```
./waf --command-template="gdb %s" --run <program-name>
```

Running C++ programs through valgrind

- valgrind memcheck can be used directly on binaries in the build directory

- An easier way is to use a waf shortcut

```
./waf --command-template="valgrind %s" --run  
  <program-name>
```

- Note: disable GTK at configure time when running valgrind (to suppress spurious reports)
- `./waf configure --disable-gtk --enable-tests ...`

Testing

- Can you trust ns-3 simulations?
 - Can you trust *any* simulation?
 - Onus is on the simulation project to validate and document results
 - Onus is also on the researcher to verify results
- ns-3 strategies:
 - regression tests
 - Aim for ***event-based*** rather than ***trace-based***
 - unit tests for verification
 - validation of models on testbeds where possible
 - reuse of code

Test framework

- ns-3-dev is checked nightly on multiple platforms
 - Linux gcc-4.x, i386 and x86_64, OS X, FreeBSD clang, and Cygwin (occasionally)
- `./test.py` will run regression tests

Walk through test code, test terminology (suite, case), and examples of how tests are run

Improving performance

- Debug vs optimized builds
 - `./waf -d debug configure`
 - `./waf -d debug optimized`
- Build ns-3 with static libraries
 - `./waf --enable-static`
- Use different compilers (icc)
 - has been done in past, not regularly tested