

ns-3 with CMake

Gabriel Ferreira

University of Brasília

September 29, 2021

My goals with CMake

IDE support

1. Easier debugging and profiling
2. Auto-completion
3. Macro expansion (showing what the actual code looks like)
4. Visually show errors, warnings, unused includes, etc

Unsupported/incomplete features

1. Bake integration
2. NSC examples
3. Planetlab examples
4. Python bindings

The code is there, but apiscan and generation fail for some modules

What is missing?

1. Implement remaining features
I can do that!
2. Stylistic and workflow related refactoring
I need maintainers feedback
3. Update ns-3 official docs with instructions

Current workflow

1. Create a new module
2. Configure CMake
3. Call the generated build system to build (e.g. ninja)
4. Call binaries directly or via the IDE
5. Debug via the IDE

Current workflow

1. Create a new folder with a CMakeLists.txt file. Set library name, list of source files and add the macro at the end.



UnB

Current workflow

Configure CMake

1. This is required every time the CMake is changed
 - e.g. to include new source files or settings were changed
2. Previous options are preserved in the CMake cache
 - a.k.a. no need to pass all the options again when reconfiguring
3. Can be done either manually via terminal or GUI, or automatically via a config file (setting the flag values)
 - e.g. `cmake -DCMAKE_BUILD_TYPE=debug
-DNS3_EXAMPLES=ON -DNS3_TESTS=ON -G Ninja ..`



Current workflow

Call the generated build system to build

1. Call generated build system to build everything (or a specific target and its dependencies)
e.g. `ninja (target)`

Current workflow

Call binaries directly or via the IDE

1. Works nicely on platforms that support RPATH, which is enabled by default (e.g. Linux and MacOS)
2. Windows requires ns-3-dev/build in the PATH, which can be done automatically
3. Something like waf –run, –gdb, –valgrind, –command-template, –pyrun, –visualize would require a wrapper script

Current workflow

Debugging via the IDE: CLion

The screenshot shows the CLion IDE interface with the following details:

- Project View:** Shows a tree structure of files and folders. The current file is `main.cpp`, which contains C++ code for a network application.
- Code Editor:** Displays the source code for `main.cpp`. A blue arrow points to line 79, indicating a breakpoint has been set there.
- Breakpoints:** A list of active breakpoints is shown on the left side of the editor.
- Variables View:** Located at the bottom left, it shows the memory state of variables. A specific variable, `m_ip`, is expanded to show its internal structure.
- Memory View:** A detailed view of the memory location `0x0000000000000000`, showing pointers to nodes and their properties.
- Toolbars and Menus:** Standard CLion toolbars and menus are visible at the top and bottom of the interface.



Current workflow

The screenshot shows the Code::Blocks IDE with the GDB debugger attached to a process. The stack window displays the call stack, starting with the initialization of the InternetStackHelper. The registers window shows the CPU register values. The memory window shows a memory dump at address 0x10071100. The locals window shows the local variables for the current function. The breakpoints window shows a single breakpoint set at the start of the InternetStackHelper::Initialize function. The assembly window shows the assembly code for the current instruction. The command window shows the command used to break at the current location.



Current workflow

Debugging via the IDE: XCode

The screenshot shows the XCode IDE interface during a debugging session. The top menu bar indicates "Running first > My Mac". The left sidebar displays system monitoring for CPU, Memory, Energy Impact, Disk, and Network. A list of threads is shown, with Thread 1 (Queue: com.apple.xbsd) being the current target, marked with a blue dot.

The main editor area shows a portion of the source code for `point-to-point-helper.cc`. The code is annotated with several `NS_ASSERT` statements and includes calls to `MakeBoundCallback` and `DefaultDropSinkWithContext`.

A vertical green bar highlights the line `b->addDeviceAddress(dev8);`. A tooltip above this line reads "`b->addDeviceAddress(dev8);` Thread 1: step over".

The bottom status bar shows the current stack frame: `0x3: this = ns3::PointToPointHelper > 0x7feff6f50`. A memory dump tool window is open, showing memory at address `0x7feff6f50` with the value `111db`.

The right panel contains the "Identity and Type" inspector, showing the file is a C++ source file relative to the project. It also lists "On Demand Resource Tags" and "Target Membership" (including ALL_BUILD and ZERO_CHECK).

The bottom right corner features the University of Brasília logo (UnB) and the text "UnB".

```
    200     "/$ns3:$pointToPointNetDevice/$queue/Dequeue";
201     Config::Connect (osss.str (), MakeBoundCallback
202     (&AsciiTraceHelper::DefaultDequeueSinkWithContext, stream));
203
204     osss.str ("");
205     osss << "$nodeId/" << "DeviceList/" << deviceid <<
206     "/$ns3:$pointToPointNetDevice/$queue/Drop";
207     Config::Connect (osss.str (), MakeBoundCallback
208     (&AsciiTraceHelper::DefaultDropSinkWithContext, stream));
209
210     osss.str ("");
211     osss << "$nodeId/" << "DeviceList/" << deviceid <<
212     "/$ns3:$pointToPointNetDevice/PhyRxDrop";
213     Config::Connect (osss.str (), MakeBoundCallback
214     (&AsciiTraceHelper::DefaultDropSinkWithContext, stream));
215
216     osss.str ("");
217     osss << "$nodeId/" << "DeviceList/" << deviceid <<
218     "/$ns3:$pointToPointNetDevice/PhyTxDrop";
219     Config::Connect (osss.str (), MakeBoundCallback
220     (&AsciiTraceHelper::DefaultDropSinkWithContext, stream));
221
222     NetDeviceContainer
223     PointToPointHelper::Install (NodeContainer c)
224     {
225       NS_ASSERT (c.Get (0) == 2);
226       | return Install (c.Get (0), c.Get (1));
227     }
228
229     NetDeviceContainer
230     PointToPointHelper::Install (Ptr<Node> a, Ptr<Node> b)
231     {
232       NetDeviceContainer container;
233
234       Ptr<PointToPointNetDevice> devA = m_deviceFactory.Create<PointToPointNetDevice> ();
235       devA->setAddress (Mac48Address::Allocate ());
236       a->addDeviceAddress (devA);
237       Ptr<QueuePacket> queueA = m_queueFactory.Create<QueuePacket> ();
238       Ptr<PointToPointNetDevice> devB = m_deviceFactory.Create<PointToPointNetDevice> ();
239       devB->setAddress (Mac48Address::Allocate ());
240       b->addDeviceAddress (devB);
241       Ptr<QueuePacket> > queueB = m_queueFactory.Create<QueuePacket> ();
242       devB->setQueue (queueB);
243       // Aggregate NetDeviceQueueInterface objects
244       Ptr<NetDeviceQueueInterface> ndqIA = CreateObject<NetDeviceQueueInterface> ();
245       ndqIA->SetInterface (0) -> ConnectObjectTraces (queueA);
246       devA->AggregateObject (ndqIA);
247       devA->AggregateObject (queueA);
248       DevA->AggregateObject (queueB);
249       DevB->AggregateObject (queueB);
250     }
```



UnB

Where is the code and instructions?

The instructions are in the MR 460:

https://gitlab.com/nsnam/ns-3-dev/-/merge_requests/460

The sources are in the branch for the MR 460:

<https://gitlab.com/Gabrielcarvfer/ns-3-dev/-/tree/buildsystem-cmake>

Questions? Concerns?

You can find me on Zulip or via the email
gabrielcarvfer@gmail.com

Thank you for your time