

# ns-3 with CMake

Gabriel Ferreira

University of Brasília

September 30, 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

## Create a new module

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

```
wscript 1.46 KB

1 # -*- Mode: python; py-indent-offset: 4; indent-tabs-mode: nil; coding: utf-8; -*-
2
3 def build(bld):
4
5     module = bld.create_ns3_module('antenna', ['core'])
6
7     module.source = [
8         'model/angles.cc',
9         'model/antenna-model.cc',
10        'model/isotropic-antenna-model.cc',
11        'model/cosine-antenna-model.cc',
12        'model/parabolic-antenna-model.cc',
13        'model/three-gpp-antenna-model.cc',
14        'model/phased-array-model.cc',
15        'model/uniform-planar-array.cc',
16    ]
17
18     module.test = bld.create_ns3_module_test_library('antenna')
19     module.test.source = [
20         'test/test-angles.cc',
21         'test/test-degrees-radians.cc',
22         'test/test-isotropic-antenna.cc',
23         'test/test-cosine-antenna.cc',
24         'test/test-parabolic-antenna.cc',
25         'test/test-uniform-planar-array.cc',
26     ]
27
28     # Tests encapsulating example programs should be listed here
29     if (old.env['ENABLE_EXAMPLES']):
30         module_test.source.extend([
31             '# test/antenna-examples-test-suite.cc',
32         ])
33
34     headers = bld(features='ns3Header')
35     headers.module = 'antenna'
36     headers.source = [
37         'model/angles.h',
38         'model/antenna-model.h',
39         'model/isotropic-antenna-model.h',
40         'model/cosine-antenna-model.h',
41         'model/parabolic-antenna-model.h',
42         'model/three-gpp-antenna-model.h',
43         'model/phased-array-model.h',
44         'model/uniform-planar-array.h',
45     ]
```

```
CMakeLists.txt 101 lines

1 set(name antenna)
2
3 set(source_files
4     model/angles.cc
5     model/antenna-model.cc
6     model/isotropic-antenna-model.cc
7     model/cosine-antenna-model.cc
8     model/parabolic-antenna-model.cc
9     model/three-gpp-antenna-model.cc
10    model/phased-array-model.cc
11    model/uniform-planar-array.cc
12    )
13
14 set(header_files
15     model/angles.h
16     model/antenna-model.h
17     model/isotropic-antenna-model.h
18     model/cosine-antenna-model.h
19     model/parabolic-antenna-model.h
20     model/phased-array-model.h
21     model/three-gpp-antenna-model.h
22     model/uniform-planar-array.h
23    )
24
25 # Link to dependencies
26 set(libraries_to_link ${libraries})
27
28 set(test_sources test/test-angles.cc test/test-degrees-radians.cc test/test-isotropic-antenna.cc
29          test/test-cosine-antenna.cc test/test-parabolic-antenna.cc test/test-uniform-planar-array.cc
30        )
31
32 build_libr($name) ${source_files} ${header_files} ${libraries_to_link} ${test_sources})
```

# 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 NS3 IDE interface. The top part displays the code for `main.cc`, which includes the configuration of an application container and its start/stop logic. The bottom part shows the `Variables` view, which is currently displaying the `Memory View`. It lists several pointers and their corresponding memory addresses, such as `m_pNet`, `m_pDevice`, `m_pApplication`, `m_pScheduler`, `m_pForwarder`, `m_pDeviceList`, `m_pDeviceContainer`, and `m_stack`.

```
serverApps.Stop (Seconds (10.0)); serverApps = ns3::ApplicationContainer();

UpdPchClientHelper echoClient (interfaces.GetAddress (1));
echoClient.SetAttribute ("MaxPackets", Uinteger32 (1));
echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
echoClient.SetAttribute ("PowerSize", Uinteger32 (1000));

ApplicationContainer clientApps = echoClient.Install (nodes.Get (0));
clientApps = ns3::ApplicationContainer (echoClient);
clientApps.Start (Seconds (2.0));
clientApps.Stop (Seconds (10.0));
clientApps = ns3::ApplicationContainer ();

Simulator::iRun ();
Simulator::iDestroy ();
return 0;
}

NS3 IDE Project Explorer:
```

- create-build.sh
- ctrl
- doc
- examples
- examples-for-run.py
- glue
- error-model
- gavl
- matrix-topology
- net
- ns3
- ns3-ns2
- ns3-ns3
- ns3-simulator
- ns3-socket
- ns3-stats
- ns3-tp
- ns3-traffic-control
- tutorial
  - Makefile
  - examples-for-run.py
  - glue
  - main.cc
  - net.cc
  - empty
  - fourcc

Debug: first

Variables

- Thread-1
  - `m_pNet` (ns3::Ptr<ns3::NetDeviceContainer>)
  - `m_pDevice` (ns3::Object)
  - `m_pApplication` (ns3::Object)
  - `m_pScheduler` (ns3::Object)
  - `m_pForwarder` (ns3::Object)
  - `m_pDeviceList` (ns3::Vector<ns3::Ptr<ns3::NetDevice>>)
  - `m_pDeviceContainer` (ns3::InternetStackHelper)
  - `m_stack` (ns3::InternetStackHelper)



# Current workflow

## Debugging via the IDE: Code::Blocks

The screenshot shows the Code::Blocks IDE interface during a debugging session. The main window displays the source code of a C++ file named `basic_ipng.cc`. The code implements various network helper functions, including `InternetStackHelper`, `Ipv4AddressHelper`, `Ipv4InterfaceContainer`, and `Ipv4GlobalRoutingHelper`.

The debugger sidebar on the left shows the current stack frame, which is `void InternetStackHelper::Initialize (ns3::NodeContainer & nodes)`. The call stack shows the function call chain starting from `main` through `ApplicationContainer` and `serverApp` to `echoServer`.

The bottom-left corner of the interface shows the current file path: `C:\Users\health\Desktop\NS3example\tutorial\inst.cc`.

The bottom status bar indicates the current configuration: `I/C++ Windows (C-L) - WINDOWS-1252 Line 54, Col 1, Pos 1618 Insert Read/Write default`.



# 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. The main editor area shows a C++ file named "point-to-point-helper.cc" with code related to NS3 PointToPointHelper. A green bar highlights the line "b->addDevice(devB);". The right sidebar contains the "Identity and Type" panel with details like Name: point-to-point-helper.cc, Type: C++ Source, and Location: Relative to Project. The "On Demand Resource Tags" and "Target Membership" sections are also visible. At the bottom, the "Auto" and "All Output" tabs are selected, and there are various navigation and search icons.

```
    209     "/$ns3:$pointToPointNetDevice/$queue/Dequeue";
210     Config::Connect (oss.str (), MakeBoundCallback
211     (&AsciiTraceHelper::DefaultDequeueSinkWithContext, stream));
212
213     oss.str ("");
214     oss << "$nodeId/" << "DeviceList/" << deviceid <<
215     "/$ns3:$pointToPointNetDevice/$queue/Drop";
216     Config::Connect (oss.str (), MakeBoundCallback
217     (&AsciiTraceHelper::DefaultDropSinkWithContext, stream));
218
219     oss.str ("");
220     oss << "$nodeId/" << "DeviceList/" << deviceid <<
221     "/$ns3:$pointToPointNetDevice/PhyRxDrop";
222     Config::Connect (oss.str (), MakeBoundCallback
223     (&AsciiTraceHelper::DefaultDropSinkWithContext, stream));
224
225 }
```

NS\_ASSERT (c.Get () == 2);
226 | return Install (c.Get (0), c.Get (1));
227 }

NetDeviceContainer
228 PointToPointHelper::Install (NodeContainer c)
229 {
230 NS\_ASSERT (c.Get () == 2);
231 return Install (c.Get (0), c.Get (1));
232 }

NetDeviceContainer
233 PointToPointHelper::Install (Ptr<Node> a, Ptr<Node> b)
234 {
235 NetDeviceContainer container;
236
237 Ptr<PointToPointNetDevice> devA = m\_deviceFactory.Create<PointToPointNetDevice> ();
238 devA->setAddress (Mac48Address::Allocate ());
239 a->addDevice (devA);
240
241 Ptr<QueuePacket> queueA = m\_queueFactory.Create<QueuePacket> ();
242 queueA->SetQueue (queueA);
243
244 Ptr<PointToPointNetDevice> devB = m\_deviceFactory.Create<PointToPointNetDevice> ();
245 devB->setAddress (Mac48Address::Allocate ());
246 b->addDevice (devB);
247 Ptr<QueuePacket> queueB = m\_queueFactory.Create<QueuePacket> ();
248 queueB->SetQueue (queueB);
249
250 // Aggregate NetDeviceQueueInterface objects
251 Ptr<NetDeviceQueueInterface> ndqIA = CreateObject<NetDeviceQueueInterface> ();
252 ndqIA->SetQueue (queueA);
253 devA->AggregateObject (ndqIA);
254
255 DevA->AggregateObject (ndqIA);
256
257 DevB->AggregateObject (ndqIB);
258 DevB->AggregateObject (ndqIB);
259 }

This is ns3::PointToPointHelper >0x7febf550 (11db)
 ↗ a (ns3::PointToPointNetDevice)
 ↗ b (ns3::PointToPointNetDevice)
 ↗ devA (ns3::PointToPointNetDevice)
 ↗ queueA (ns3::PointToPointNetDevice::Queue<ns3::Packet>)



# Where is the code and instructions?

The instructions are in the MR 460:

[https://gitlab.com/nsnam/ns-3-dev/-/merge\\_requests/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? Suggestions?

You can find me on Zulip or via the email  
[gabrielcarvfer@gmail.com](mailto:gabrielcarvfer@gmail.com)

Thank you for your time