Aqua-Sim-NG Developer Tutorial

Table of Contents

This tutorial describes a detailed workflow to design, implement and test a sample Medium Access Control (MAC) protocol for underwater sensor networks in *Aqua-Sim-NG* plugin for *ns-3.29* network simulator.

In the first section, a step-by-step guide is provided to configure and install the latest version of Aqua-Sim-NG with ns-3.29 source code.

The second section shows a basic workflow for configuring and running sample simulation scripts, existing in the standard Aqua-Sim-NG repository. A detailed structure of the simulation scripts is provided.

The third section focuses on a new MAC protocol development from scratch. The internal structure of Aqua-Sim-NG simulator is provided, describing basic programming interfaces between different modules of a network simulation stack. This includes a description of MAC protocol interfaces between two adjacent layers: network (routing) layer from the top, and the PHY (channel) layer at the bottom, with the new designed MAC module in-between.

The fourth section focuses on writing simulation scripts with the new MAC module, building topologies for testing the MAC layer, and describing the built-in methods for gathering general network statistics for further collection and analysis.

The fifth section, presents an additional set of tools for gathering all simulation details, using *ns-3 trace files*. This includes building a visualized replay of network experiments using *NetAnim* and calculating some specific network parameters, such as *cumulative* and *instantaneous throughput*, using separate Python-based parsing scripts.

Section 1. Configuration and installation.

(Based on Ubuntu 18.04.2 LTS)

1.1 Basic host system requirements

For running ns-3 simulations with aqua-sim, a Linux-based operating system is needed with the following requirements:

- **recommended Linux distribution:** *Ubuntu 16.04 or highe*r, with *linux-kernel 4.4.0 or higher*.

- **gcc compiler version:** *5.4.0 or higher*

- for Ubuntu, run the following command to install all necessary (and beyond) requirements:

```
sudo apt-get install build-essential gcc g++ python python3 autoconf cvs bzr
unrar git bunzip2
```

1.2 Installation

The latest aqua-sim-ng repository is available here:
http://hudson.ccny.cuny.edu/download/aquasim-ng.tgz

To download and unpack the latest source code of aqua-sim-ng with ns-3.29 using the command line, do the following:

```
cd ~/
mkdir workspace
cd workspace
wget http://hudson.ccny.cuny.edu/download/aquasim-ng.tgz
tar xvf aquasim-ng.tgz
```

To install ns-3.29 with aqua-sim, go to ns-3.29 working directory:
(ns-3 working directory **is always the directory with waf binary file inside!!!)**

```
cd ~/workspace/aquasim-ng/ns-3.29
./waf clean
cp ./src/applications/model/onoff-application.h ./build/ns3
```

Configure ns-3. Run the following command:

```
./waf configure --build-profile=debug --enable-examples --disable-python
```

The command above will configure ns-3.29 with *debug build* profile, which allows more detailed output while running custom network protocols (that's what we need). During the process, you may see some warning messages, such as some options were "not enabled". You can install these options at a later time when you need them. Some notes on the configure command:

- *enable-examples* flag allows running additional simulation script from the examples folder of any compiled simulation module, including aqua-sim-ng.
- *disable-python* flag disables additional python-binding to run selected simulation scripts in Python programming language.

After configuring ns-3, build the simulator codebase, using the following command:

```
./waf build
```

This will take approximately **15 minutes**, depending on the available CPU resources.

1.3 Validating

After a successful build, you are ready to run simulation scripts and develop custom protocols.

You can also run the test script to verify the installation:

```
./test.py
```

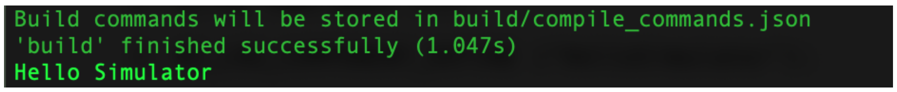
Section 2. Sample simulation scripts.

2.1 Running "hello-world" in ns-3

To run a sample simulation script, run the following command:

```
./waf --run hello-simulator
```

You should see the following output:



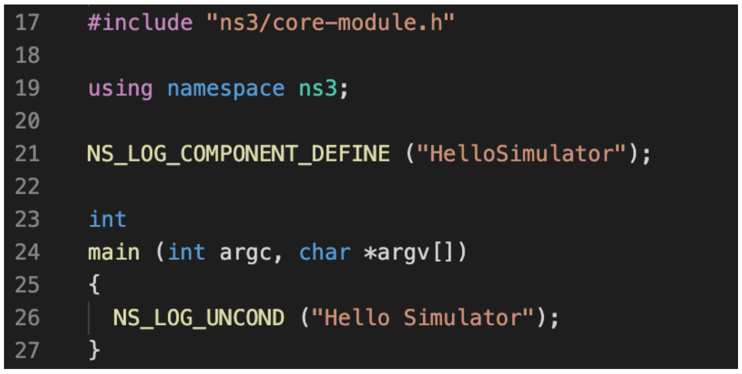
```
Build commands will be stored in build/compile_commands.json
'build' finished successfully (1.047s)
Hello Simulator
```

2.1.1 Inside "hello-simulator" script

The "hello-simulator" is the part of ns-3 "tutorial" module located in:

```
NSHOME=~/workspace
$NSHOME/aquasim-ng/ns-3.29/examples/tutorial/hello-simulator.cc
```

Let's take a look on a structure of the "hello-simulator.cc" script, line-by-line:



```
#include "ns3/core-module.h"

using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("HelloSimulator");

int
main (int argc, char *argv[])
{
  NS_LOG_UNCOND ("Hello Simulator");
}
```

```
Line 1: #include "ns3/core-module.h"

Line 2: using namespace ns3;

Line 3: NS_LOG_COMPONENT_DEFINE ("HelloSimulator");

Line 4: int
        main (int argc, char *argv[])
        {
Line 5:     NS_LOG_UNCOND ("Hello Simulator");
        }
```

The script presents a simple C++ program with main function, which returns a "Hello Simulator" string to standard output, using a built-in ns-3 logging method `NS_LOG_UNCOND`.

Line 1 imports the core ns-3 module, responsible for running the simulations and logging all the simulation events.

Line 2 declares the ns-3 namespace, including all the functions, methods and global objects provided by ns-3.

Line 3 defines the name "HelloSimulator" of the logging component, needed for the debugging purposes (currently not used).

Line 4 declares the main() function.

Line 5 returns a custom string ("Hello Simulator") to the standard output, using the default ns-3 method for that - `NS_LOG_UNCOND`

"hello-simulator.cc" scripts is a **very basic structure** of **ANY** ns-3 simulation program. All the following examples are based on this structure. Additional ns-3 modules (custom and standard) can be added for running a given simulation scenario – i.e. routing protocol, MAC protocol, the application layer, mobility model, etc.

2.2 wscript configuration files (script-name bindings)

You might have noticed, that in order to execute the simulation scenario described in "hello-simulator.cc" script, one must run the following command:

```
./waf  --run hello-simulator
```

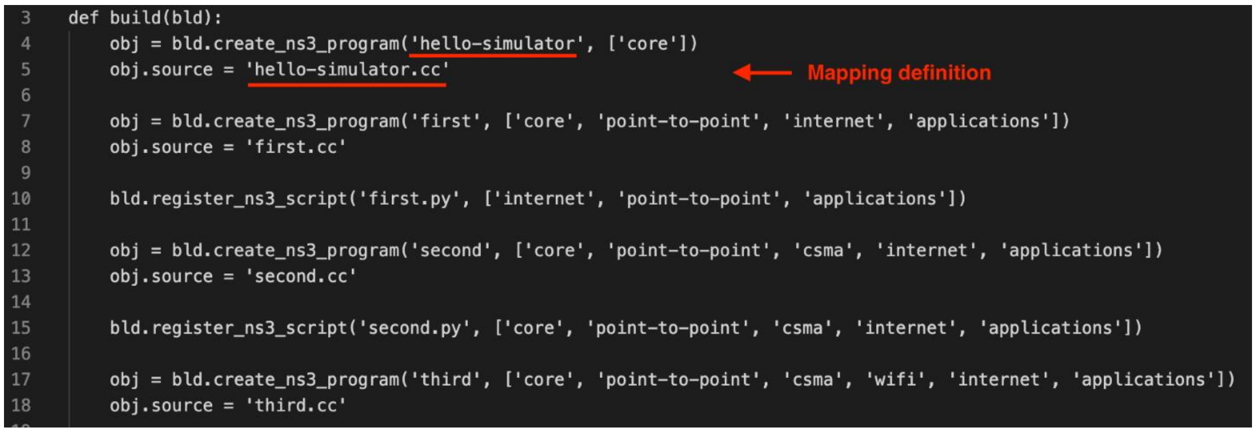
This looks a little bit counter-intuitive, as one might expect to see a path to the binary file of "hello-simulator.cc" program here, and then executing it.

Instead, ns-3 uses ***waf*** tool for building and executing C++ scripts. In the example above, ***waf*** tool was told to run (option --run) a simulation script named "hello-simulator".

How can ***waf*** tool know which simulation script to execute?

For that, ***waf*** uses ***wscript*** configuration files, which contain a map between the simulation scripts (in .cc format) and their "execution" names, entered into the command line.

For example, "hello-simulator" corresponds to **../examples/tutorial/hello-simulator.cc** script, in the ***wscript*** configuration file. See the corresponding line in ***../examples/tutorial/wscript*** configuration file:



```
def build(bld):
    obj = bld.create_ns3_program('hello-simulator', ['core'])
    obj.source = 'hello-simulator.cc'

    obj = bld.create_ns3_program('first', ['core', 'point-to-point', 'internet', 'applications'])
    obj.source = 'first.cc'

    bld.register_ns3_script('first.py', ['internet', 'point-to-point', 'applications'])

    obj = bld.create_ns3_program('second', ['core', 'point-to-point', 'csma', 'internet', 'applications'])
    obj.source = 'second.cc'

    bld.register_ns3_script('second.py', ['core', 'point-to-point', 'csma', 'internet', 'applications'])

    obj = bld.create_ns3_program('third', ['core', 'point-to-point', 'csma', 'wifi', 'internet', 'applications'])
    obj.source = 'third.cc'
```

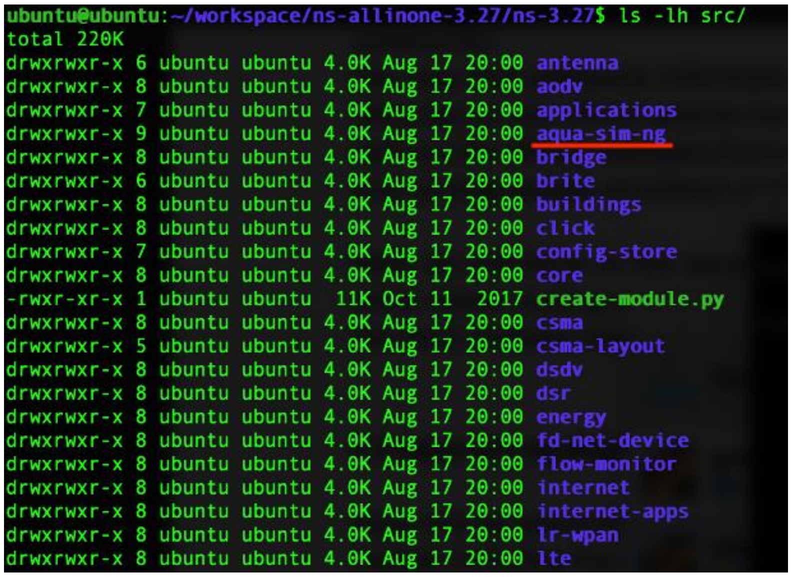
Note: NS-3 simulator is a very powerful simulation tool, used by many researchers in the computer networks field. It is free and open source, with large and responsive community.
You may find additional information about ns-3 with detailed and comprehensive tutorial and extra tips on the official website here: [1-2]. It is highly encouraged to have a look on the tutorial and run the official examples/tutorial scripts to get familiar with the basic concepts of building and running sample simulation scripts.

Next, we will focus on the ***aqua-sim-ng*** plugin for ns-3. We will run standard examples of some classic routing and MAC protocols for underwater sensor networks and get familiar with the main modules required for running the simulations.

2.3 Aqua-sim-ng extension for ns-3

Aqua-sim-ng presents a dedicated module which embeds into ns-3 simulator, as any other module including the standard ones, included in ns-3 by default. The examples of the standard modules are: *aodv, lte, wifi, network, internet,* etc. Additional information about the standard ns-3 modules can be found in the ns-3 "model library" here [3].

All the modules, including the standard and the custom ones, such as aqua-sim-ng, are located in ***src/*** folder in the ns-3 working directory:



```
ubuntu@ubuntu:~/workspace/ns-allinone-3.27/ns-3.27$ ls -lh src/
total 220K
drwxrwxr-x 6 ubuntu ubuntu 4.0K Aug 17 20:00 antenna
drwxrwxr-x 8 ubuntu ubuntu 4.0K Aug 17 20:00 aodv
drwxrwxr-x 7 ubuntu ubuntu 4.0K Aug 17 20:00 applications
drwxrwxr-x 9 ubuntu ubuntu 4.0K Aug 17 20:00 aqua-sim-ng
drwxrwxr-x 8 ubuntu ubuntu 4.0K Aug 17 20:00 bridge
drwxrwxr-x 6 ubuntu ubuntu 4.0K Aug 17 20:00 brite
drwxrwxr-x 8 ubuntu ubuntu 4.0K Aug 17 20:00 buildings
drwxrwxr-x 8 ubuntu ubuntu 4.0K Aug 17 20:00 click
drwxrwxr-x 7 ubuntu ubuntu 4.0K Aug 17 20:00 config-store
drwxrwxr-x 8 ubuntu ubuntu 4.0K Aug 17 20:00 core
-rwxr-xr-x 1 ubuntu ubuntu  11K Oct 11  2017 create-module.py
drwxrwxr-x 8 ubuntu ubuntu 4.0K Aug 17 20:00 csma
drwxrwxr-x 5 ubuntu ubuntu 4.0K Aug 17 20:00 csma-layout
drwxrwxr-x 8 ubuntu ubuntu 4.0K Aug 17 20:00 dsdv
drwxrwxr-x 8 ubuntu ubuntu 4.0K Aug 17 20:00 dsr
drwxrwxr-x 8 ubuntu ubuntu 4.0K Aug 17 20:00 energy
drwxrwxr-x 8 ubuntu ubuntu 4.0K Aug 17 20:00 fd-net-device
drwxrwxr-x 8 ubuntu ubuntu 4.0K Aug 17 20:00 flow-monitor
drwxrwxr-x 8 ubuntu ubuntu 4.0K Aug 17 20:00 internet
drwxrwxr-x 8 ubuntu ubuntu 4.0K Aug 17 20:00 internet-apps
drwxrwxr-x 8 ubuntu ubuntu 4.0K Aug 17 20:00 lr-wpan
drwxrwxr-x 8 ubuntu ubuntu 4.0K Aug 17 20:00 lte
```

Aqua-sim-ng module has a similar subfolder structure like any other ns-3 module. Let's have a look at it:

```
ls -l aqua-sim-ng
```

This command lists the folders in Aqua-sim-ng. Below explains the contents of the folders.

- **doc:**
Contains module documentation in .rst format.

- **documentation:**
Contains more detailed user-oriented documentation, usually in .pdf format.

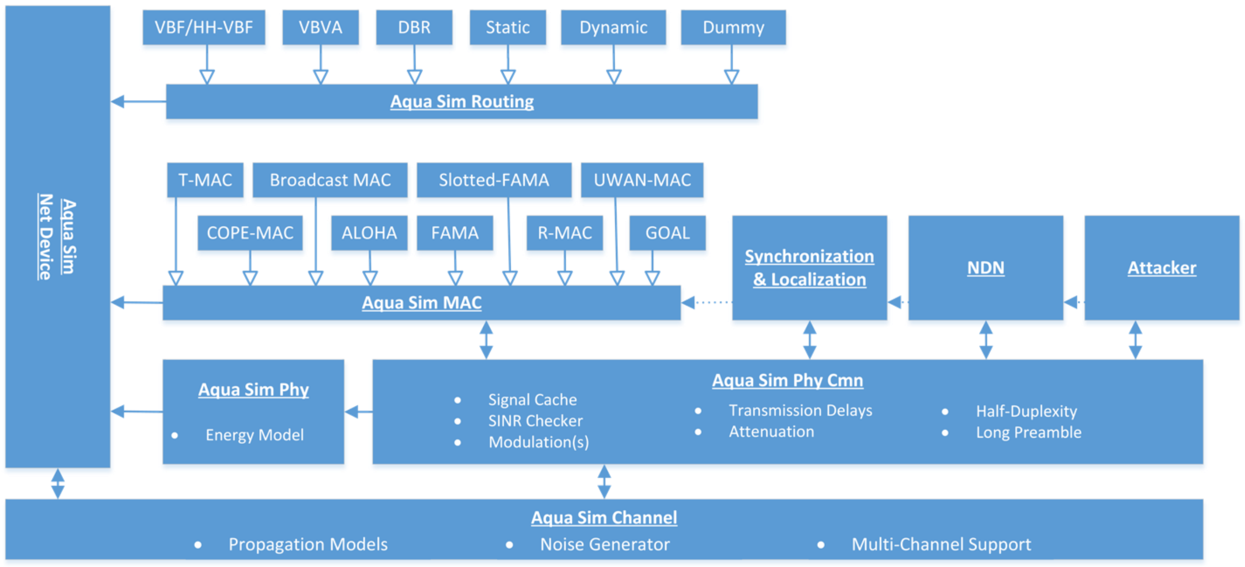
- **examples:**
Contains example script for running the simulations with aqua-sim network and MAC protocols.

- **helper:**
Contains helper modules for compiling and running the modules in a single simulation scripts.

- **model:**

Main folder which contains all the communication modules, related to aqua-sim-ng. Such as: MAC protocols (ALOHA, Broadcast MAC, etc.), routing protocols (VBF, DBR, Static, etc.), physical channel models, etc.

All available aqua-sim-ng libraries are shown in the diagram below:



- **test:**
Contains additional simulation scripts for testing specific modules in the model.

- ***wscript*** **configuration file:**
Contains mapping between the actual simulation examples and their execution names needed for ***waf*** builder tool to run the simulations.

2.4 Aqua-sim-ng simulation script

As for the simulation script example, a simple broadcast MAC protocol is taken.

The simulation script for Broadcast MAC already exists in the ***examples/*** subfolder under ***src/aqua-sim-ng/*** module, so in order to execute the script, run the following command:

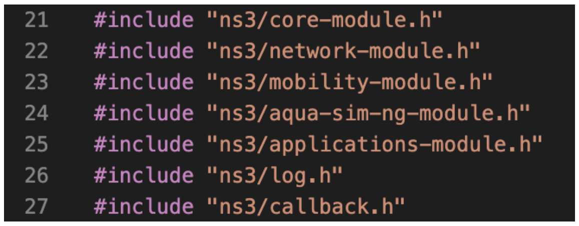
```
./waf --run broadcastMAC_example
```

The command above builds and executes a C++ script, located here: **aqua-sim-ng/examples/broadcastMAC_example.cc** (as can be checked in the ***wscript*** configuration file)

broadcastMAC_example.cc describes a simple simulation scenario with a total of 4 nodes (3 senders and 1 sink), placed in a line, with 100 meters in-between each other (see lines 97 and 108 in the source code below). All 3 nodes generate a constant on-off packet traffic, defined in the ***on-off-application*** module towards a single sink-node. **BroadcastMac** module is used on a MAC layer, with **DummyRouting** module on the routing layer, which is specifically used for testing MAC-layer protocols. Default simulation time is defined by ***simStop*** variable, set to 100 seconds (see line 45).

Let's have a look at some more contents of the script, highlighting the main parts of the simulation scenario created there, from top to bottom:

Lines 21-27:



```
21   #include "ns3/core-module.h"
22   #include "ns3/network-module.h"
23   #include "ns3/mobility-module.h"
24   #include "ns3/aqua-sim-ng-module.h"
25   #include "ns3/applications-module.h"
26   #include "ns3/log.h"
27   #include "ns3/callback.h"
```

Import all the ns-3 modules, needed for constructing given simulation scenario:

- **ns3/core-module.h:**
Needed for running the simulations (scheduling and parsing discrete simulation events), and for debugging purposes.

- **ns3/network-module.h:**
This module is responsible for creating the network nodes/devices and allocating necessary network interfaces in-between the layers.

- **ns3/mobility-module.h:**
This module is needed for allocating the network nodes into a specific topology – either static or dynamic, according to the specified mobility model (e.g. random-waypoint mobility model).

- **ns3/aqua-sim-ng-module.h:**
This is the main module required for running underwater protocol stack. It configures and installs the given network protocol stack into the allocated network nodes.

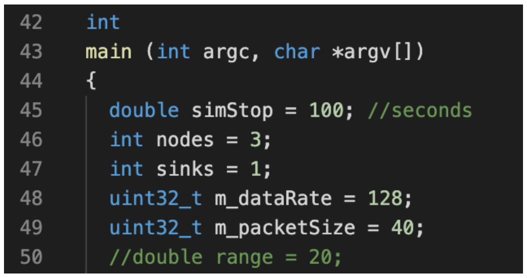
- **ns3/applications-module.h:**
This module is needed for configuring and generating packet traffic into the configured network. This includes defining application packet rate (packets/sec), traffic profile and overall duration.

- **ns3/log.h:**
Needed for logging simulation events.

- **ns3/callback.h:**

Needed for collecting additional simulation information using ns-3 trace-files (described in the Section 5).

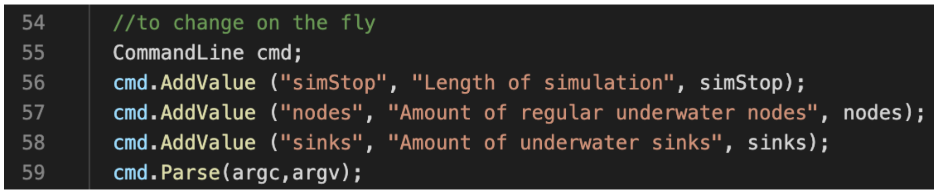
Lines 42-50:



```
42  int
43  main (int argc, char *argv[])
44  {
45    double simStop = 100; //seconds
46    int nodes = 3;
47    int sinks = 1;
48    uint32_t m_dataRate = 128;
49    uint32_t m_packetSize = 40;
50    //double range = 20;
```

- define main() function to run simulations
- define some local variables needed for initial configuration:
    - **simStop:** simulation duration
    - **nodes:** number of sender-nodes in a network
    - **sinks:** number of sink-nodes (receive-nodes) in a network
    - **m_dataRate:** application data rate, bps
    - **m_packetSize:** packet size of every generated packet, Bytes

Lines 54-59:



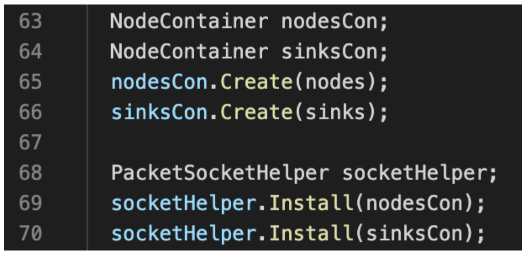
```
54  //to change on the fly
55  CommandLine cmd;
56  cmd.AddValue ("simStop", "Length of simulation", simStop);
57  cmd.AddValue ("nodes", "Amount of regular underwater nodes", nodes);
58  cmd.AddValue ("sinks", "Amount of underwater sinks", sinks);
59  cmd.Parse(argc,argv);
```

Add some "changeable" configuration parameters, which can be modified "on-the-fly", as a dedicated argument to waf command.

For example, to run this script with the modified parameters, one can run the following command, explicitly specifying the parameters being changed:

```
./waf --run "broadcastMAC_example --simStop=1000 --nodes=5 --sinks=2"
```

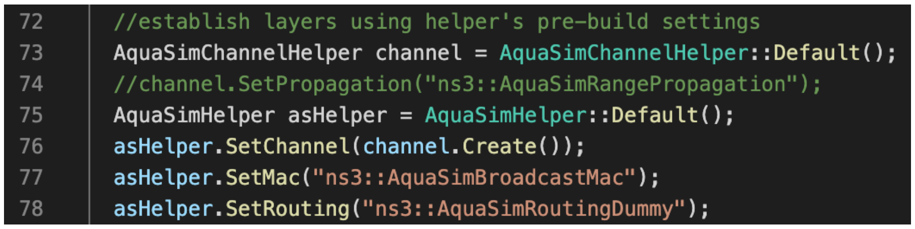
Lines 63-70:



```
63  NodeContainer nodesCon;
64  NodeContainer sinksCon;
65  nodesCon.Create(nodes);
66  sinksCon.Create(sinks);
67
68  PacketSocketHelper socketHelper;
69  socketHelper.Install(nodesCon);
70  socketHelper.Install(sinksCon);
```

Create nodes/sinks entities (empty so far) and the corresponding network socket interfaces (for sending/receiving generated packets).

Lines 72-78:



```
72  //establish layers using helper's pre-build settings
73  AquaSimChannelHelper channel = AquaSimChannelHelper::Default();
74  //channel.SetPropagation("ns3::AquaSimRangePropagation");
75  AquaSimHelper asHelper = AquaSimHelper::Default();
76  asHelper.SetChannel(channel.Create());
77  asHelper.SetMac("ns3::AquaSimBroadcastMac");
78  asHelper.SetRouting("ns3::AquaSimRoutingDummy");
```

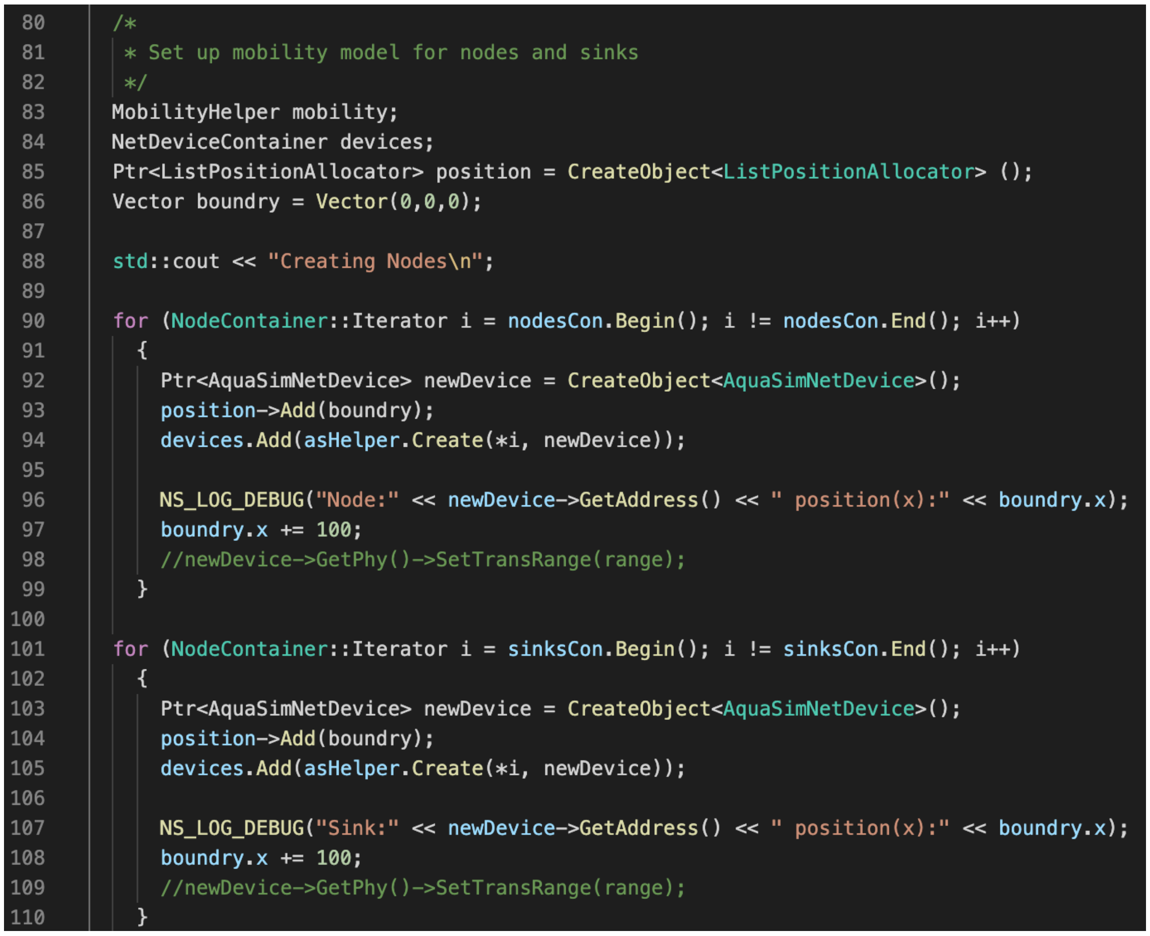
Line 73: Create **AquaSimChannelHelper** class. This class is used for creating and configuring a lower-layer acoustic PHY-model.

Initialize the first 3 layers of a communication stack – physical layer (channel model), MAC layer and Routing (network) layer, using the main helper class called **AquaSimHelper**:

Line 75: Create **AquaSimHelper** class. This is the main helper class in aqua-sim, which sets and configures the routing- and MAC layer-models, as well as channel-helper class to a node's ***NetDevice***.

Line 76: ***SetChannel()*** – setting PHY layer
Line 77: ***SetMac()*** – establishing **BroadcastMac** model on the MAC layer
Line 78: ***SetRouting()*** – setting a simple routing protocol ***AquaSimRoutingDummy*** designed specifically for testing MAC layer models.

Lines 80-110:



```
/*
 * Set up mobility model for nodes and sinks
 */
MobilityHelper mobility;
NetDeviceContainer devices;
Ptr<ListPositionAllocator> position = CreateObject<ListPositionAllocator> ();
Vector boundry = Vector(0,0,0);

std::cout << "Creating Nodes\n";

for (NodeContainer::Iterator i = nodesCon.Begin(); i != nodesCon.End(); i++)
  {
    Ptr<AquaSimNetDevice> newDevice = CreateObject<AquaSimNetDevice>();
    position->Add(boundry);
    devices.Add(asHelper.Create(*i, newDevice));

    NS_LOG_DEBUG("Node:" << newDevice->GetAddress() << " position(x):" << boundry.x);
    boundry.x += 100;
    //newDevice->GetPhy()->SetTransRange(range);
  }

for (NodeContainer::Iterator i = sinksCon.Begin(); i != sinksCon.End(); i++)
  {
    Ptr<AquaSimNetDevice> newDevice = CreateObject<AquaSimNetDevice>();
    position->Add(boundry);
    devices.Add(asHelper.Create(*i, newDevice));

    NS_LOG_DEBUG("Sink:" << newDevice->GetAddress() << " position(x):" << boundry.x);
    boundry.x += 100;
    //newDevice->GetPhy()->SetTransRange(range);
  }
```

Allocate nodes/sinks onto a 3D-space using the allocation Vector named `boundary`. For every node, update the allocation vector and add it as a node position into a ***position allocator***. Iterate though every node/sink, using the node containers, initialized earlier.

A **position allocator** is used for defining positions of the Nodes in 3D space, following the given position patterns, depending on a type of a position allocator, such as: lists, grids, disks, random boxes, etc. Some examples of different position allocators, available in NS-3:

- **ListPositionAllocator**:
  Store the nodes' positions as a vector with (x, y, z) coordinates in a list.

- **GridPositionAllocator**:
  Configure dimension parameters of a grid (i.e., x-step, y-step, height and length), and place the nodes at the corners of every cell in the grid.

- **RandomRectanglePositionAllocator**:
  Place nodes randomly inside a given rectangle, according to a specified probability distribution.

- **UniformDiscPositionAllocator**:
  Place nodes uniformly randomly inside a disk with specified dimensions – (x ,y)-center of the disk and its radius.

Note:
There are 3 main components of any NS-3 script, used for constructing a simulation scenario:

- **NodeContainer:**

NodeContainer is used to store a key NS-3 abstraction, called a **Node**. A **Node** represents a computer with network protocol stacks, attached to physical channel via a **NetDevice** – another key NS-3 abstraction, which can be interpreted as a *modem* or a *Network Interface Card (NIC)*, connecting a computer to a physical communication medium - a channel.
The **NodeContainer** provides a way to create, manage and access the **Node** object, which are created in the simulation script.
L**ines 63-64** above declare node containers for both sender-nodes and sink-nodes. The following **lines 65-66** create a given number of nodes, using ***Create()*** method of the **NodeContainer** class.

- **NetDeviceContainer:**

NetDeviceContainer is used to create and store **NetDeivce** objects for the created nodes. A **NetDevice** object is another basic NS-3 abstraction which can be imagined as a Network Interface Card (NIC), used for connecting a node to physical channel of the network. A **NetDevice** object "wires together" a model of a given physical layer (channel) to a given stack of L2-L5 protocols, defined and configured in **Node** objects.

- **Helper** classes:

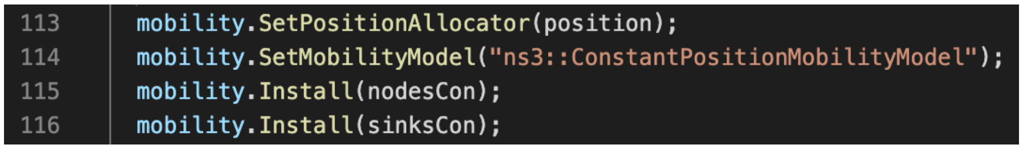
Various **Helper** classes are used in NS-3 to create and configure a given model of L1-L5 layer and attach it to the Nodes or NetDevices, using ***Install()*** method. For example, the **lines 123-127** in the source code below create and configure a helper instance of the **OnOffApplication** model, named **OnOffHelper**. **Line 129** then installs the configured application model to the **Node**-objects, stored in the given **NodeContainer**.

For more information about the **NodeContainers**, **NetDevices** and **Helper**, see a well-described, scripting example from official NS-3 tutorial, available here:
https://www.nsnam.org/docs/release/3.7/tutorial/tutorial_18.html

Line 83: Initialize a mobility model, used further down in the lines 113-116.

Line 84: Create a container class for storing the NetDevice objects. This container is used for iterating and setting the device's positions in the following for-loops.

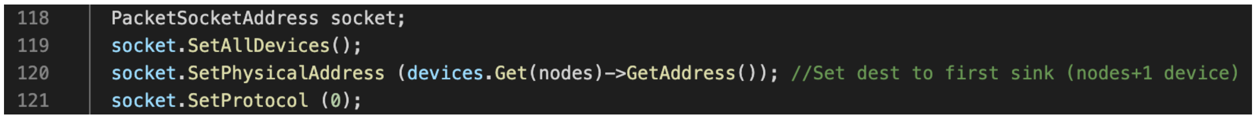
Lines 113-116:



```
113    mobility.SetPositionAllocator(position);
114    mobility.SetMobilityModel("ns3::ConstantPositionMobilityModel");
115    mobility.Install(nodesCon);
116    mobility.Install(sinksCon);
```

For every allocated node's positions, set a static mobility model, which fixes the nodes into the positions and does not allow them to move during the simulation (***ConstantPosition*** mobility model).

Lines 118-121:



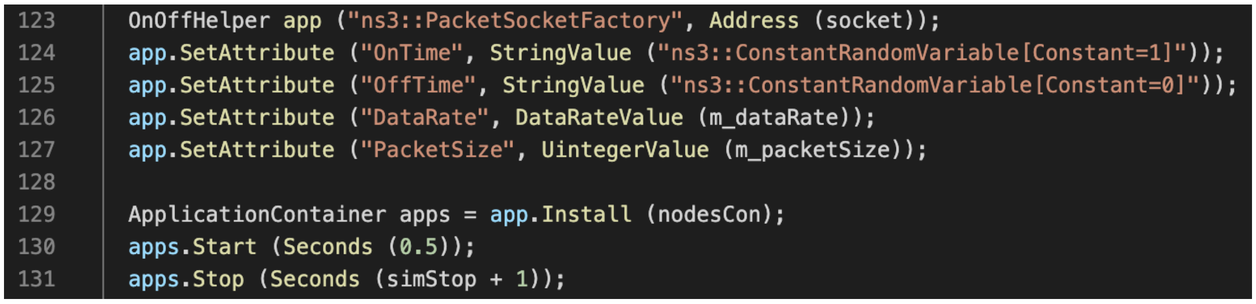
```
118    PacketSocketAddress socket;
119    socket.SetAllDevices();
120    socket.SetPhysicalAddress (devices.Get(nodes)->GetAddress()); //Set dest to first sink (nodes+1 device)
121    socket.SetProtocol (0);
```

Line 120: set physical address and create a receive-socket interface for a sink-node (receive-node), which has the ID number of 3 (equaled to the overall number of sender-nodes).

Line 121: Set the ID of the routing protocol. In all aqua-sim models, this parameter is set to 0.

Note: the code above allows initialization of a single sink only. If one needs to create multiple sinks (destination-nodes), one must iterate over all desired sinks, create application helper for them, and specify the destination socket-address. The example of such code can be found here: https://github.com/dugdmitry/aqua-sim-ng/blob/mac_routing_dev/examples/sfama_grid_test.cc#L174

Lines 123-131:



```
123    OnOffHelper app ("ns3::PacketSocketFactory", Address (socket));
124    app.SetAttribute ("OnTime", StringValue ("ns3::ConstantRandomVariable[Constant=1]"));
125    app.SetAttribute ("OffTime", StringValue ("ns3::ConstantRandomVariable[Constant=0]"));
126    app.SetAttribute ("DataRate", DataRateValue (m_dataRate));
127    app.SetAttribute ("PacketSize", UintegerValue (m_packetSize));
128
129    ApplicationContainer apps = app.Install (nodesCon);
130    apps.Start (Seconds (0.5));
131    apps.Stop (Seconds (simStop + 1));
```

Define an application layer, using ns-3-provided, ***on-off-application*** model, located in `src/applications/`

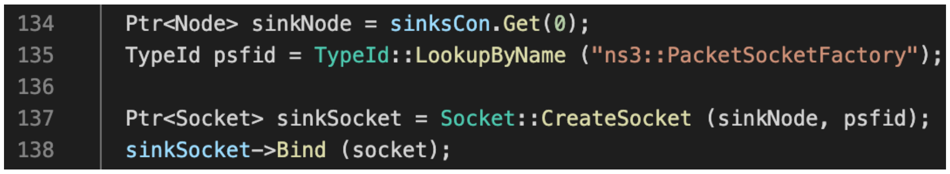
Some more information about the ns-3 on-off-application can be found in the official ns-3 library documentation here: https://www.nsnam.org/doxygen/classns3_1_1_on_off_application.html

Specify on-off application properties:

- **OnTime** – percentage of time a node is sending a packet
- **OffTime** – percentage of time a node is not sending a packet
- **DataRate** – number of packets a node sends per second
- **PacketSize** – size of every packet, in Bytes

Also, install the application to every node (excluding the sink), and specify when the application starts and stops throughout the simulation run.

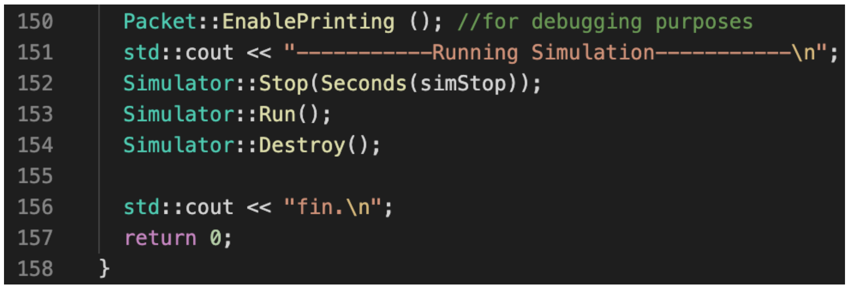
Lines 134-138:



```
134     Ptr<Node> sinkNode = sinksCon.Get(0);
135     TypeId psfid = TypeId::LookupByName ("ns3::PacketSocketFactory");
136
137     Ptr<Socket> sinkSocket = Socket::CreateSocket (sinkNode, psfid);
138     sinkSocket->Bind (socket);
```

Bind a receive-socket at the sink-node, so that the sink-node is able to receive the incoming packets from the sender-nodes. This procedure is necessary since the application and the corresponding sockets have been installed only on the sender-nodes upon this point.

Lines 150-158:

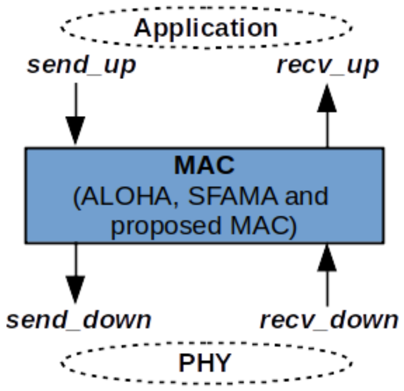


```
150     Packet::EnablePrinting (); //for debugging purposes
151     std::cout << "-----------Running Simulation-----------\n";
152     Simulator::Stop(Seconds(simStop));
153     Simulator::Run();
154     Simulator::Destroy();
155
156     std::cout << "fin.\n";
157     return 0;
158   }
```

Enable additional functions for debugging, configure the stop-time of the entire simulation (line 152), run the simulation (line 153) and schedule the simulation destruction (line 154) when the simulation stops. Return 0 in the main function to indicate a successful execution.

Section 3. MAC protocol design.

This section describes a development process of a new MAC protocol module for aqua-sim-ng. MAC protocols are located on layer 2 (L2) of OSI, which means that they are located in-between routing (L3) and channel (L1 - PHY) modules. Therefore, MAC module in aqua-sim-ng uses the following set of interfaces (methods) to communicate with the adjacent layers:



send_up():

This method is used by MAC layer to send a packet up to the stack (to the routing layer).

recv_up():

This method is used by MAC layer to receive a packet from the upper layer.

send_down():

Send a packet down to PHY layer.

recv_down():

Receive packet from PHY layer.

3.1 Pure ALOHA MAC protocol design

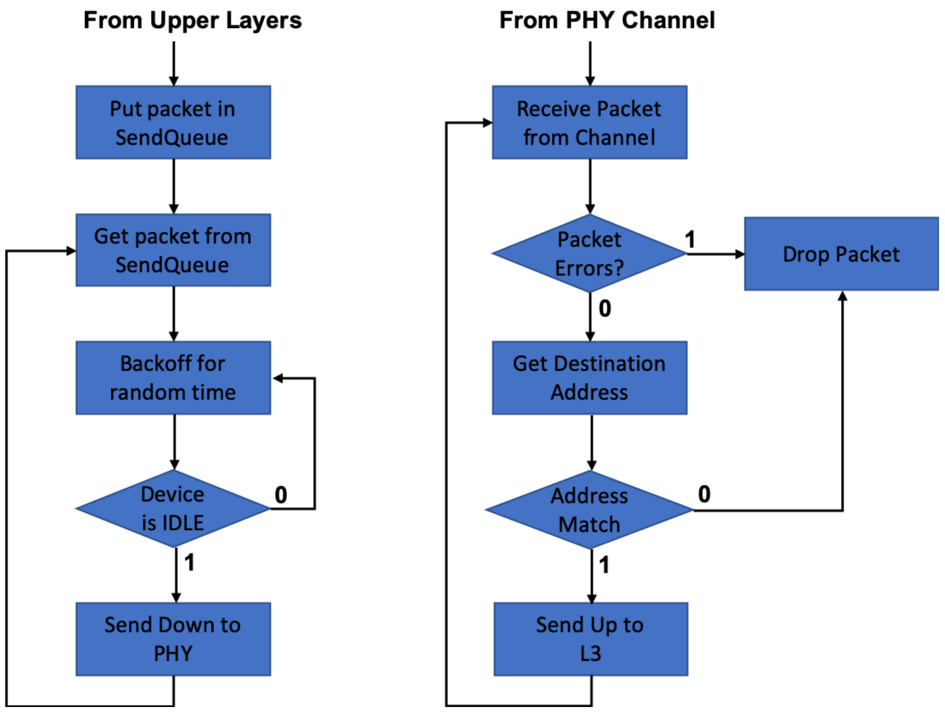
As for the example of a MAC protocol design, we will use a very simple MAC protocol used in old-generation wireless networks.

It has the following logical workflow, from top (L3) to bottom (L1):

1) Routing layer sends a packet down to MAC layer
2) MAC layer receives a packet from the routing layer (***recv_up*** call) and puts it into a *MAC send queue* (FIFO queue)
3) MAC layer gets a packet from queue
4) Sleep for random time interval
5) If net_device is IDLE, send packet down to PHY layer (***send_down*** call)
6) Go to next packet (step 3)

From bottom (PHY) to the top (Routing):

1) MAC layer receives a packet from PHY layer (***recv_down*** call)
2) Check if destination address of the incoming packet matches with the node's address
3) If yes, MAC layer sends a packet up to the stack (***send_up*** call)



3.2 MAC module structure

Every module in ns-3 is written in C++, including this *pure ALOHA* example. Therefore, for our new module, we need to create 2 files:

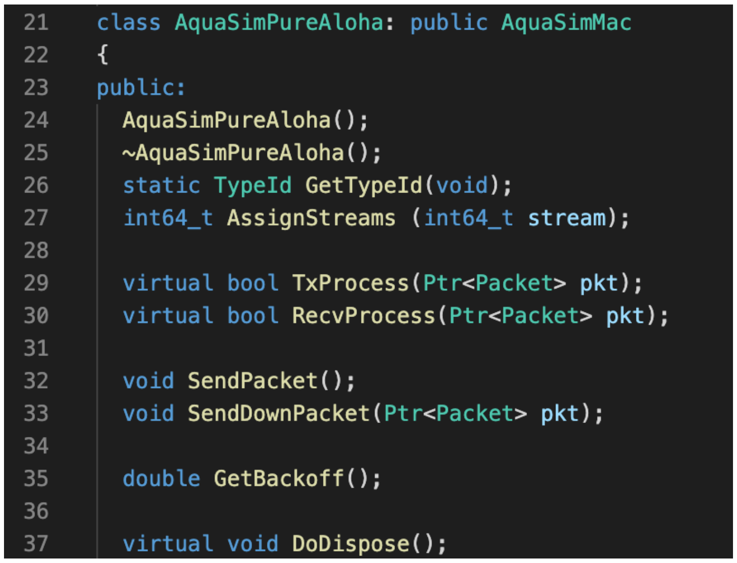
- *aqua-sim-mac-pure-aloha.h*
- *aqua-sim-mac-pure-aloha.cc*

The first *.h* file contains a declaration of classes, methods and variables, necessary for the MAC implementation.
The second *.cc* file contains an implementation of these classes, methods and variables. Both files should be located under ***model/*** subfolder in aqua-sim-ng.

3.2.1 Declaring aqua-sim-mac-pure-aloha.h source file

Complete source code can be found in **Appendix A**.
Here is the description of main components of the module

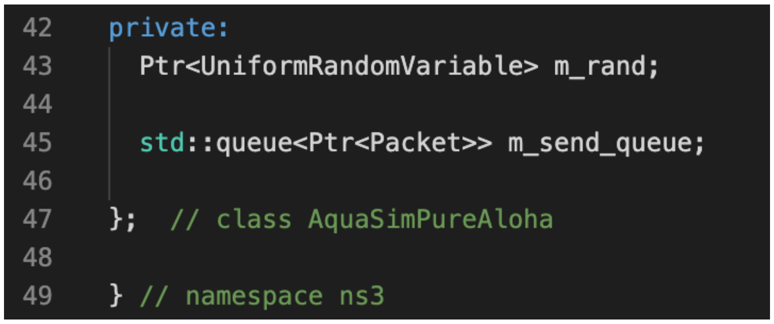
Line 21-37:



```
21  class AquaSimPureAloha: public AquaSimMac
22  {
23  public:
24    AquaSimPureAloha();
25    ~AquaSimPureAloha();
26    static TypeId GetTypeId(void);
27    int64_t AssignStreams (int64_t stream);
28
29    virtual bool TxProcess(Ptr<Packet> pkt);
30    virtual bool RecvProcess(Ptr<Packet> pkt);
31
32    void SendPacket();
33    void SendDownPacket(Ptr<Packet> pkt);
34
35    double GetBackoff();
36
37    virtual void DoDispose();
```

- Creating main constructor and destructor of **AquaSimPureAloha** class
- Inheriting ***TxProcess()*** and ***RecvProcess()*** methods from the parent **AquaSimMac** class for overriding
- Creating methods for:
    - Scheduling packet transmission event: ***SendPacket()***
    - Actual packet sending to PHY: ***SendDownPacket()***
    - Generating a backoff value before sending: ***GetBackoff()***
- Inheriting ***DoDispose()*** method – for clearing out the memory after simulation destruction.

Line 42-49:



```
42  private:
43    Ptr<UniformRandomVariable> m_rand;
44
45    std::queue<Ptr<Packet>> m_send_queue;
46
47  };  // class AquaSimPureAloha
48
49  } // namespace ns3
```

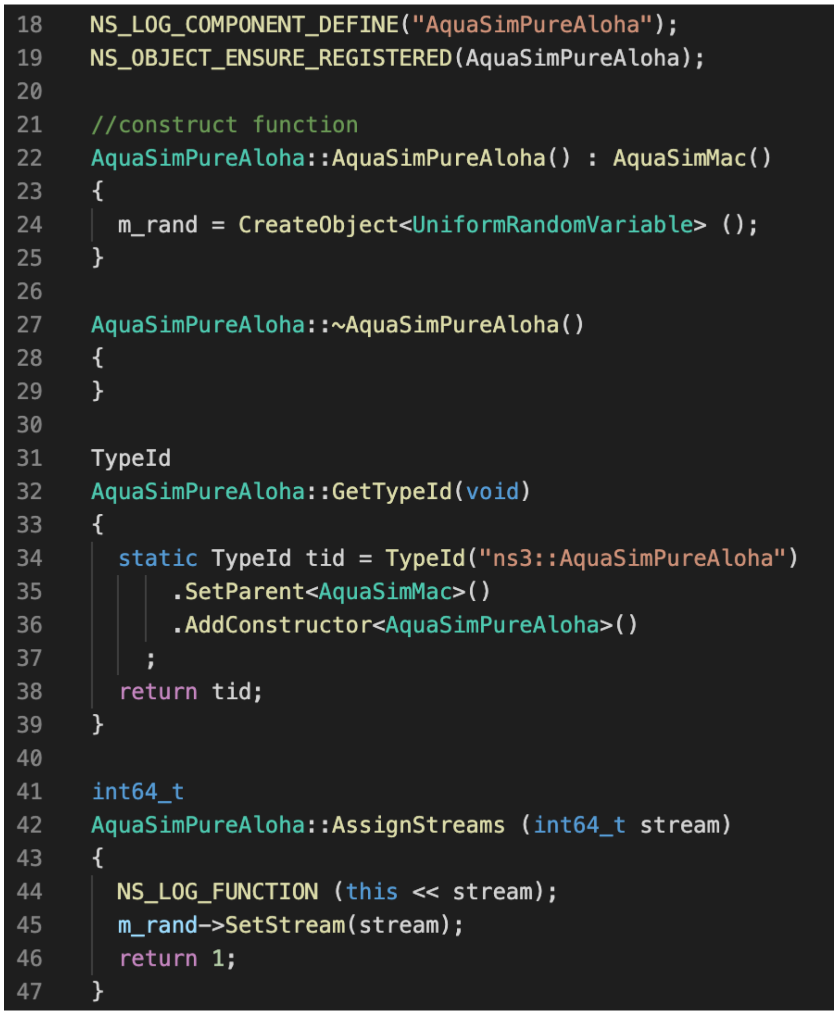
Create some private class variables, such as:

- **m_rand** – store random-number generator to generate backoff values
- **m_send_queue** – store the packets, coming from L3, in a send queue

3.2.2 Implementing aqua-sim-mac-pure-aloha.cc source file

Complete source code can be found in **Appendix B**.
Here is the description of main components of the module:

Line 18-47:



```
18  NS_LOG_COMPONENT_DEFINE("AquaSimPureAloha");
19  NS_OBJECT_ENSURE_REGISTERED(AquaSimPureAloha);
20
21  //construct function
22  AquaSimPureAloha::AquaSimPureAloha() : AquaSimMac()
23  {
24    m_rand = CreateObject<UniformRandomVariable> ();
25  }
26
27  AquaSimPureAloha::~AquaSimPureAloha()
28  {
29  }
30
31  TypeId
32  AquaSimPureAloha::GetTypeId(void)
33  {
34    static TypeId tid = TypeId("ns3::AquaSimPureAloha")
35        .SetParent<AquaSimMac>()
36        .AddConstructor<AquaSimPureAloha>()
37      ;
38    return tid;
39  }
40
41  int64_t
42  AquaSimPureAloha::AssignStreams (int64_t stream)
43  {
44    NS_LOG_FUNCTION (this << stream);
45    m_rand->SetStream(stream);
46    return 1;
47  }
```

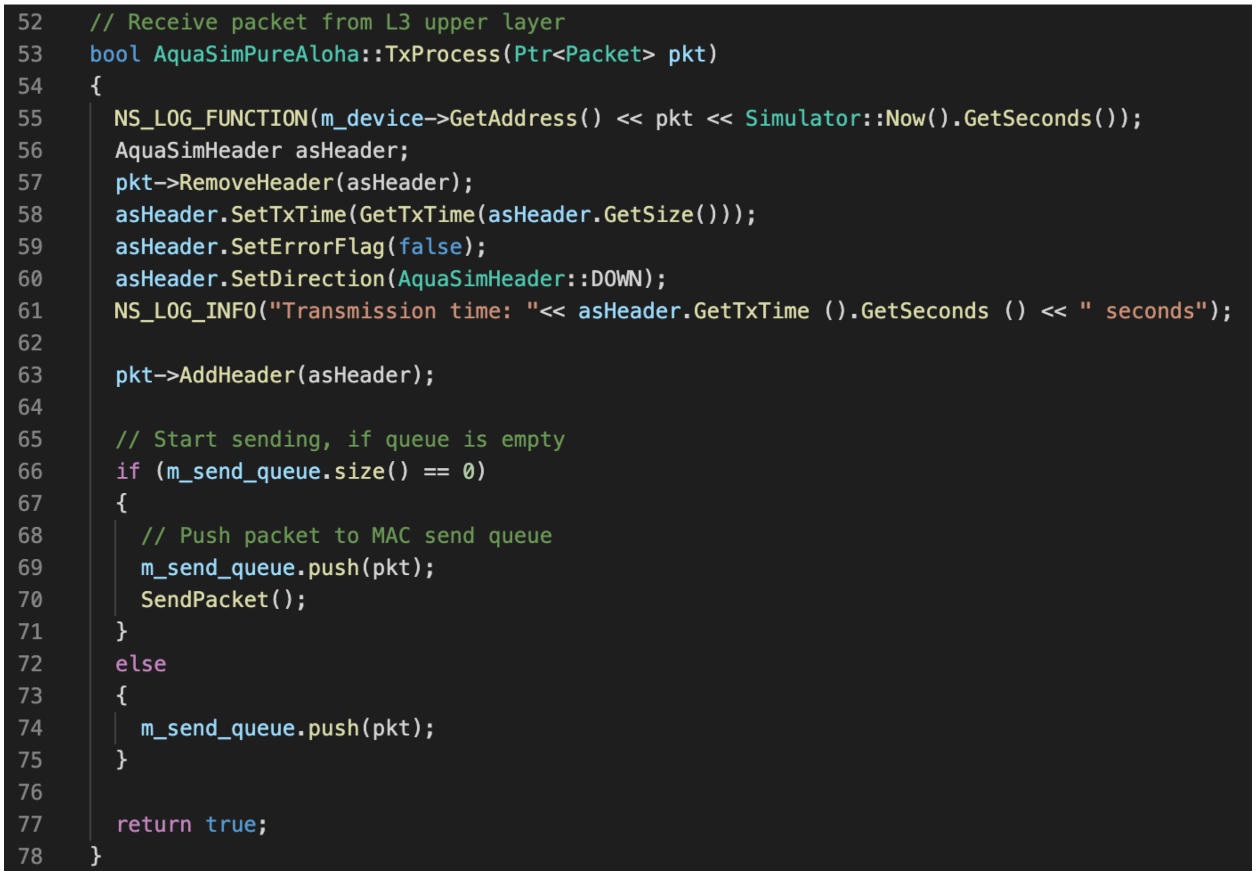
- Defining Logging components (lines 18-19)
- Declaring the constructor, with `m_rand` variable initialization (lines 22-25)
- Declaring the destructor (lines 27-29)
- Assigning **TypeId** attributes (lines 31-39)
- Assigning and setting a **random variable stream**, used to derive random values from ns-3 pseudo-random generator class (lines 41-47). See a detailed info on ns-3 random variables here: https://www.nsnam.org/docs/manual/html/random-variables.html

All NS-3 classes are inherited from the base-class, called **ns3::Object**. Consequently, every child class of Object class also inherits and overrides a metadata class called **TypeId**, which is used to store a unique information about a given class, such as:

- Unique string identifier, i.e. **ns3::AquaSimPureAloha** (**line 34** above)
- Information about the base class of the given class: i.e., **AquaSimMac** (**line 35**)
- Set of available constructors to execute (**line 36**)

A detailed information about the **ns3::Object** base class, smart-pointers and how they build a foundation of any NS-3 module, can be found here:
https://www.nsnam.org/docs/release/3.9/manual/manual_31.html

Lines 52-78:



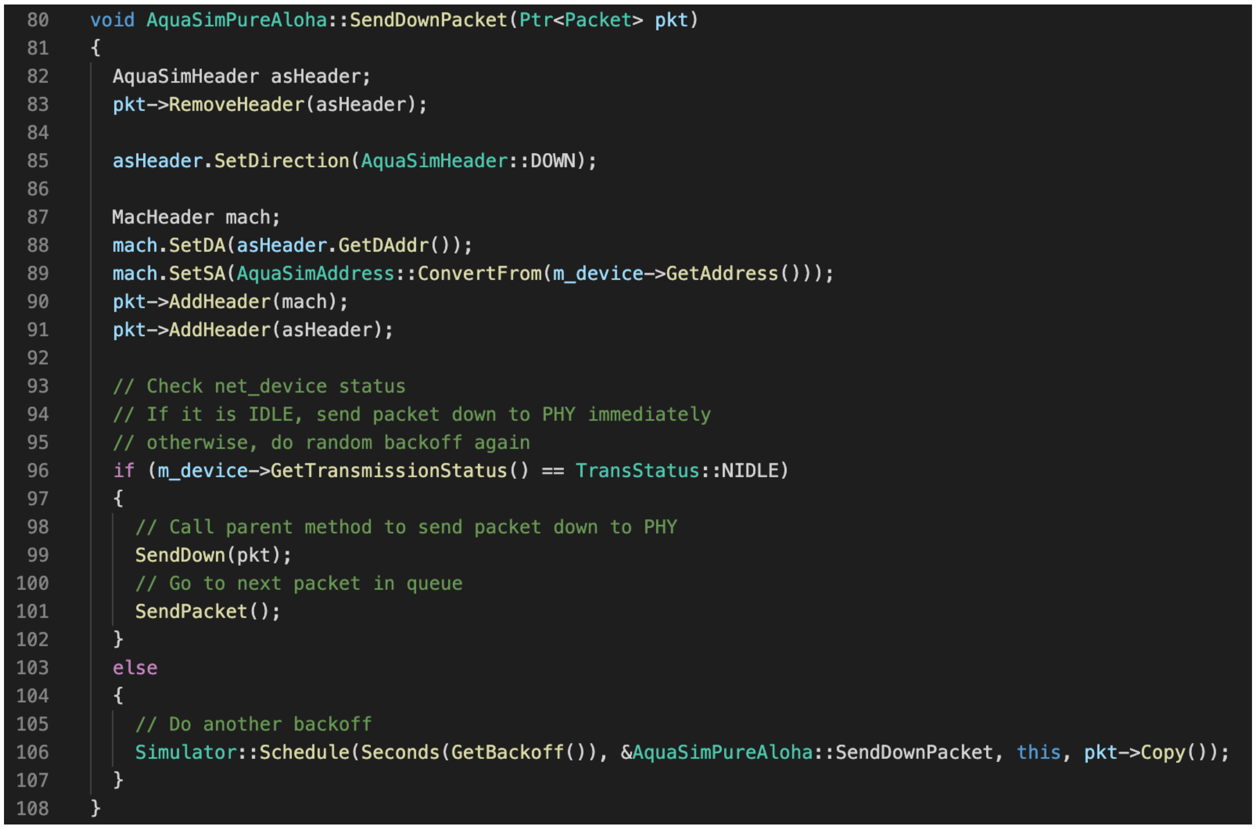
```
52  // Receive packet from L3 upper layer
53  bool AquaSimPureAloha::TxProcess(Ptr<Packet> pkt)
54  {
55    NS_LOG_FUNCTION(m_device->GetAddress() << pkt << Simulator::Now().GetSeconds());
56    AquaSimHeader asHeader;
57    pkt->RemoveHeader(asHeader);
58    asHeader.SetTxTime(GetTxTime(asHeader.GetSize()));
59    asHeader.SetErrorFlag(false);
60    asHeader.SetDirection(AquaSimHeader::DOWN);
61    NS_LOG_INFO("Transmission time: "<< asHeader.GetTxTime ().GetSeconds () << " seconds");
62
63    pkt->AddHeader(asHeader);
64
65    // Start sending, if queue is empty
66    if (m_send_queue.size() == 0)
67    {
68      // Push packet to MAC send queue
69      m_send_queue.push(pkt);
70      SendPacket();
71    }
72    else
73    {
74      m_send_queue.push(pkt);
75    }
76
77    return true;
78  }
```

This is the first method for processing the packets coming from the upper layers. It checks packet address fields and puts an incoming packet into a ***send queue***.

- Log a reception time of a packet (line 55)
- Remove packet header (line 57), update the header fields (lines 58-60)
- Log the transmission time of the packet, based on its size (line 61)
- Add the updated header back to the packet (line 63)

- Put packet in the queue (lines 66-75):
    - If queue is empty --> put packet in queue, call the transmission method
    - If queue is not empty --> just put packet in the queue and return

Lines 80-108:

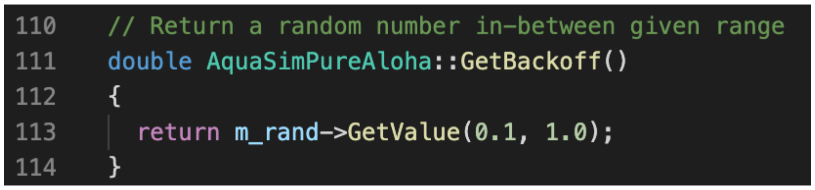


```
80   void AquaSimPureAloha::SendDownPacket(Ptr<Packet> pkt)
81   {
82     AquaSimHeader asHeader;
83     pkt->RemoveHeader(asHeader);
84
85     asHeader.SetDirection(AquaSimHeader::DOWN);
86
87     MacHeader mach;
88     mach.SetDA(asHeader.GetDAddr());
89     mach.SetSA(AquaSimAddress::ConvertFrom(m_device->GetAddress()));
90     pkt->AddHeader(mach);
91     pkt->AddHeader(asHeader);
92
93     // Check net_device status
94     // If it is IDLE, send packet down to PHY immediately
95     // otherwise, do random backoff again
96     if (m_device->GetTransmissionStatus() == TransStatus::NIDLE)
97     {
98       // Call parent method to send packet down to PHY
99       SendDown(pkt);
100      // Go to next packet in queue
101      SendPacket();
102    }
103    else
104    {
105      // Do another backoff
106      Simulator::Schedule(Seconds(GetBackoff()), &AquaSimPureAloha::SendDownPacket, this, pkt->Copy());
107    }
108  }
```

This is the method for performing the actual packet transmission – sending it down to PHY (channel) layer.

- Update some packet header fields again (line 82-85)
- Create a MAC header and update its fields (line 87-91):
    - Set destination address (DA) to MAC header
    - Set source address (SA)
    - Add packet and MAC headers back to initial packet
- Check the net_device status. If it is busy --> do additional random backoff. If it is IDLE --> send packet immediately (line 96-107):
    - Check net_device transmission status (line 96)
    - Send down packet immediately, calling the parent's class (line 99)
    - Go to next packet in the queue (line 101)
    - Schedule this method call once again, after additional random backoff (line 106)

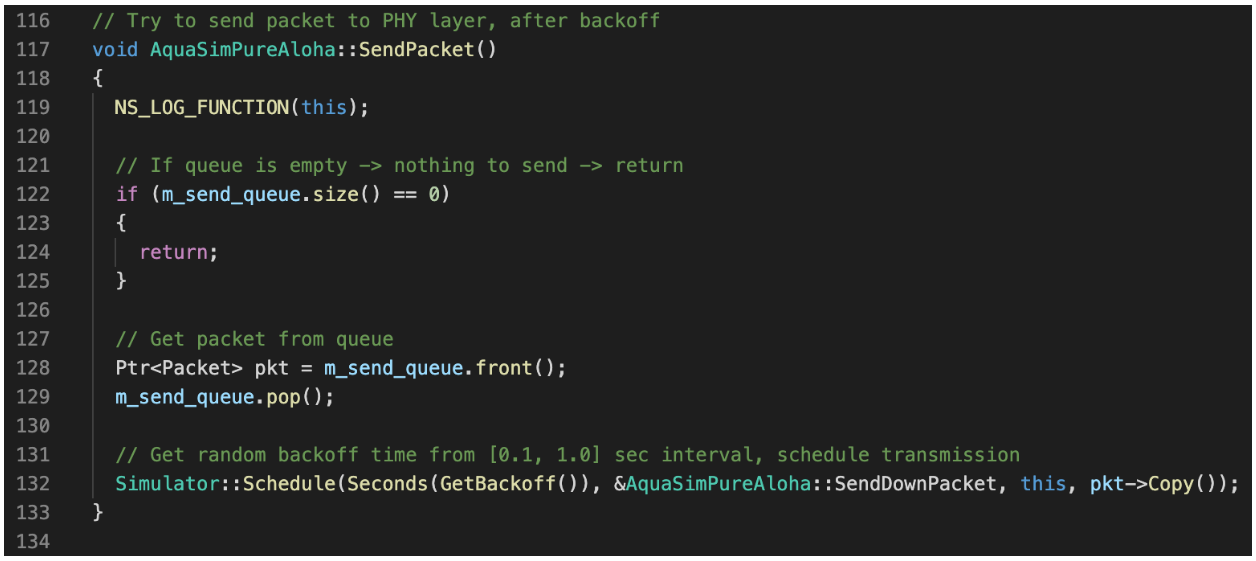
Lines 110-114:



```
110   // Return a random number in-between given range
111   double AquaSimPureAloha::GetBackoff()
112   {
113     return m_rand->GetValue(0.1, 1.0);
114   }
```

Use the **m_rand** variable object to get a random value in-between [0.1, 1.0] range.

Lines 116-134:

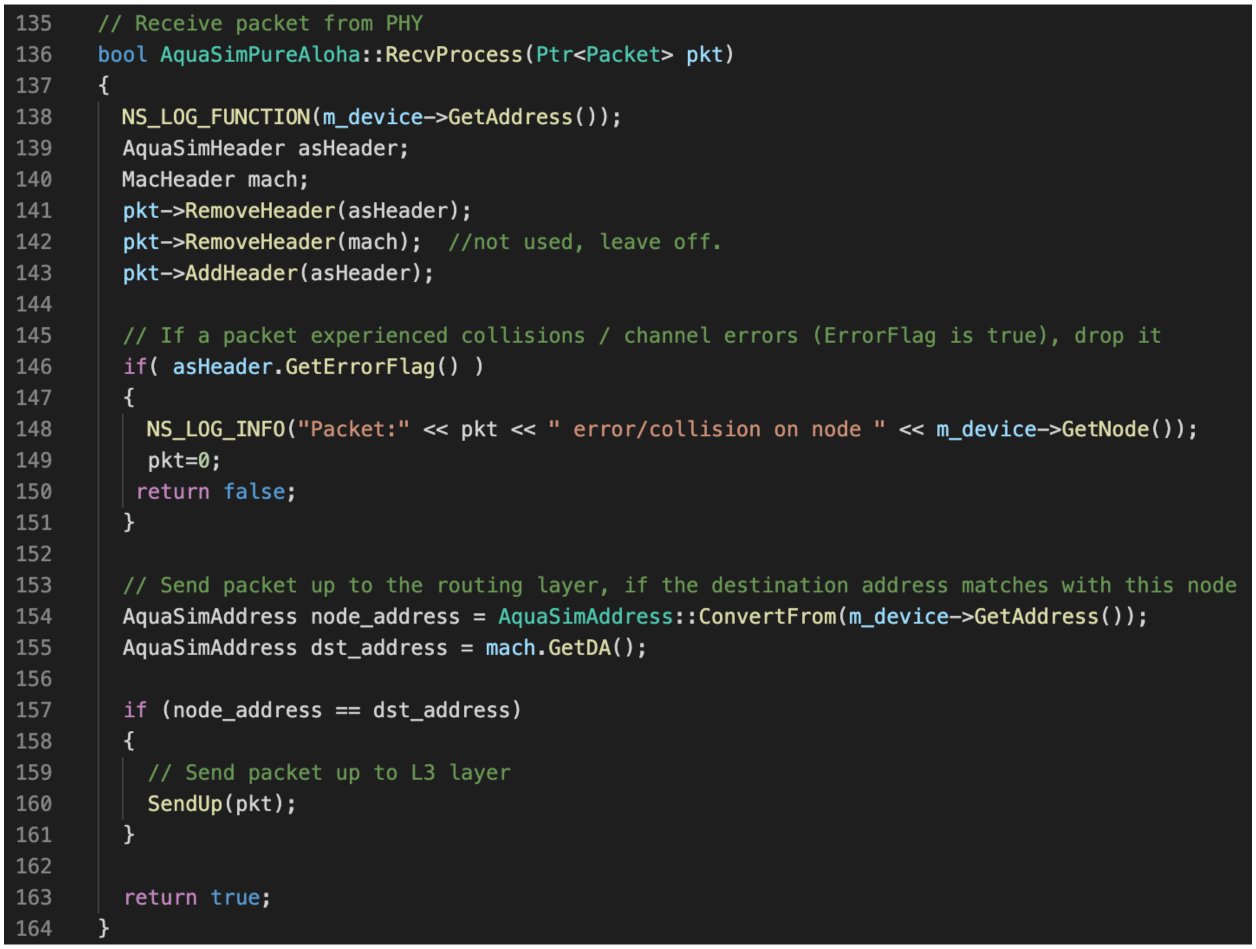


```
116   // Try to send packet to PHY layer, after backoff
117   void AquaSimPureAloha::SendPacket()
118   {
119     NS_LOG_FUNCTION(this);
120
121     // If queue is empty -> nothing to send -> return
122     if (m_send_queue.size() == 0)
123     {
124       return;
125     }
126
127     // Get packet from queue
128     Ptr<Packet> pkt = m_send_queue.front();
129     m_send_queue.pop();
130
131     // Get random backoff time from [0.1, 1.0] sec interval, schedule transmission
132     Simulator::Schedule(Seconds(GetBackoff()), &AquaSimPureAloha::SendDownPacket, this, pkt->Copy());
133   }
134
```

This method gets a packet from send FIFO queue and schedules the actual packet transmission, after selecting a random backoff value.

- Log this method call (line 119)
- Return, if the send queue is empty (line 122-125)
- If queue is not empty, get a packet from it (line 128-129)
- Schedule the actual transmission after ***backoff value*** seconds (line 132)

Lines 135-164:

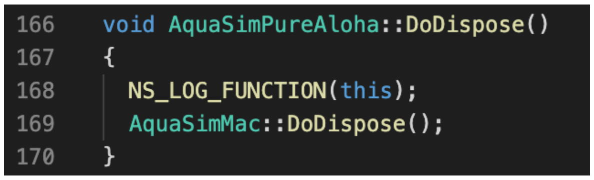


```
135  // Receive packet from PHY
136  bool AquaSimPureAloha::RecvProcess(Ptr<Packet> pkt)
137  {
138    NS_LOG_FUNCTION(m_device->GetAddress());
139    AquaSimHeader asHeader;
140    MacHeader mach;
141    pkt->RemoveHeader(asHeader);
142    pkt->RemoveHeader(mach);  //not used, leave off.
143    pkt->AddHeader(asHeader);
144
145    // If a packet experienced collisions / channel errors (ErrorFlag is true), drop it
146    if( asHeader.GetErrorFlag() )
147    {
148      NS_LOG_INFO("Packet:" << pkt << " error/collision on node " << m_device->GetNode());
149      pkt=0;
150     return false;
151    }
152
153    // Send packet up to the routing layer, if the destination address matches with this node
154    AquaSimAddress node_address = AquaSimAddress::ConvertFrom(m_device->GetAddress());
155    AquaSimAddress dst_address = mach.GetDA();
156
157    if (node_address == dst_address)
158    {
159      // Send packet up to L3 layer
160      SendUp(pkt);
161    }
162
163    return true;
164  }
```

This method receives the packets, coming from **PHY layer**.

- Log address of the node receiving a packet (line 138)
- Read the header fields in both packet and MAC headers (line 139-143)
- If the packet is received with errors (check the error flag in the header) --> drop it and return (line 146-151)
- If the packet is received without errors, check the destination address of the packet. If the destination address matches with the node address --> send packet up to the stack (line 154-161)

Line 166-170:

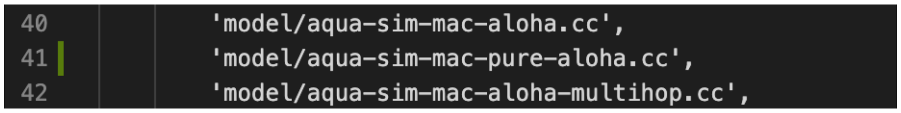


```
166  void AquaSimPureAloha::DoDispose()
167  {
168    NS_LOG_FUNCTION(this);
169    AquaSimMac::DoDispose();
170  }
```

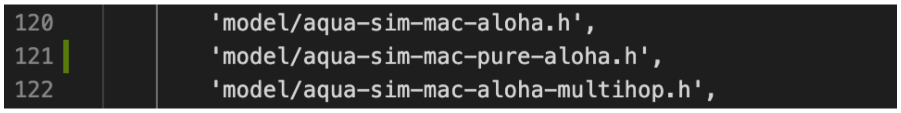
Call the parent's disposal method, after the simulation is completed.

3.2.3 Additional declarations for waf builder

For the waf builder to see the new **.h** and **.cc** modules, their names should be added into waf configuration file – ***wscript***, located in the main ***aqua-sim-ng/*** main folder:



```
        'model/aqua-sim-mac-aloha.cc',
        'model/aqua-sim-mac-pure-aloha.cc',
        'model/aqua-sim-mac-aloha-multihop.cc',
```



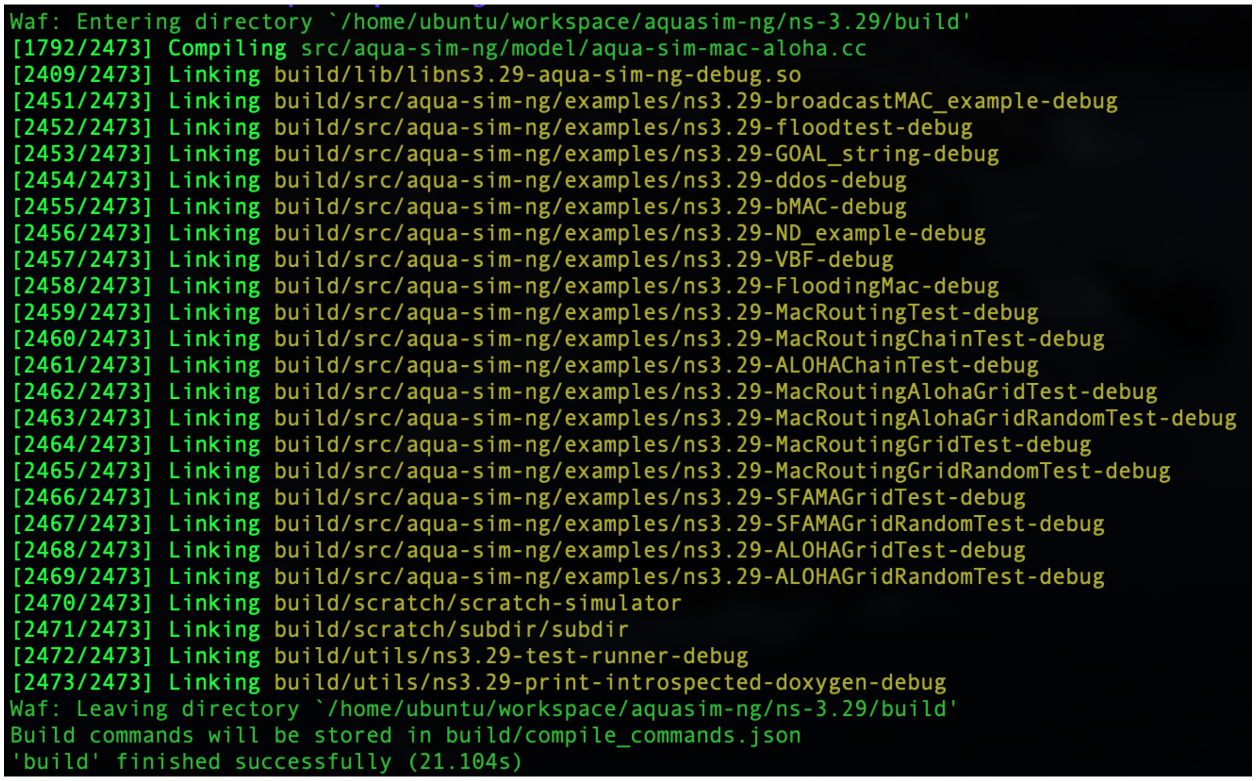
```
        'model/aqua-sim-mac-aloha.h',
        'model/aqua-sim-mac-pure-aloha.h',
        'model/aqua-sim-mac-aloha-multihop.h',
```

3.2.3 Compile the source code

To compile the new source code, run the following command:

```
./waf build
```

If the compilation is successful, you should see the following output:



```
Waf: Entering directory `/home/ubuntu/workspace/aquasim-ng/ns-3.29/build'
[1792/2473] Compiling src/aqua-sim-ng/model/aqua-sim-mac-aloha.cc
[2409/2473] Linking build/lib/libns3.29-aqua-sim-ng-debug.so
[2451/2473] Linking build/src/aqua-sim-ng/examples/ns3.29-broadcastMAC_example-debug
[2452/2473] Linking build/src/aqua-sim-ng/examples/ns3.29-floodtest-debug
[2453/2473] Linking build/src/aqua-sim-ng/examples/ns3.29-GOAL_string-debug
[2454/2473] Linking build/src/aqua-sim-ng/examples/ns3.29-ddos-debug
[2455/2473] Linking build/src/aqua-sim-ng/examples/ns3.29-bMAC-debug
[2456/2473] Linking build/src/aqua-sim-ng/examples/ns3.29-ND_example-debug
[2457/2473] Linking build/src/aqua-sim-ng/examples/ns3.29-VBF-debug
[2458/2473] Linking build/src/aqua-sim-ng/examples/ns3.29-FloodingMac-debug
[2459/2473] Linking build/src/aqua-sim-ng/examples/ns3.29-MacRoutingTest-debug
[2460/2473] Linking build/src/aqua-sim-ng/examples/ns3.29-MacRoutingChainTest-debug
[2461/2473] Linking build/src/aqua-sim-ng/examples/ns3.29-ALOHAChainTest-debug
[2462/2473] Linking build/src/aqua-sim-ng/examples/ns3.29-MacRoutingAlohaGridTest-debug
[2463/2473] Linking build/src/aqua-sim-ng/examples/ns3.29-MacRoutingAlohaGridRandomTest-debug
[2464/2473] Linking build/src/aqua-sim-ng/examples/ns3.29-MacRoutingGridTest-debug
[2465/2473] Linking build/src/aqua-sim-ng/examples/ns3.29-MacRoutingGridRandomTest-debug
[2466/2473] Linking build/src/aqua-sim-ng/examples/ns3.29-SFAMAGridTest-debug
[2467/2473] Linking build/src/aqua-sim-ng/examples/ns3.29-SFAMAGridRandomTest-debug
[2468/2473] Linking build/src/aqua-sim-ng/examples/ns3.29-ALOHAGridTest-debug
[2469/2473] Linking build/src/aqua-sim-ng/examples/ns3.29-ALOHAGridRandomTest-debug
[2470/2473] Linking build/scratch/scratch-simulator
[2471/2473] Linking build/scratch/subdir/subdir
[2472/2473] Linking build/utils/ns3.29-test-runner-debug
[2473/2473] Linking build/utils/ns3.29-print-introspected-doxygen-debug
Waf: Leaving directory `/home/ubuntu/workspace/aquasim-ng/ns-3.29/build'
Build commands will be stored in build/compile_commands.json
'build' finished successfully (21.104s)
```

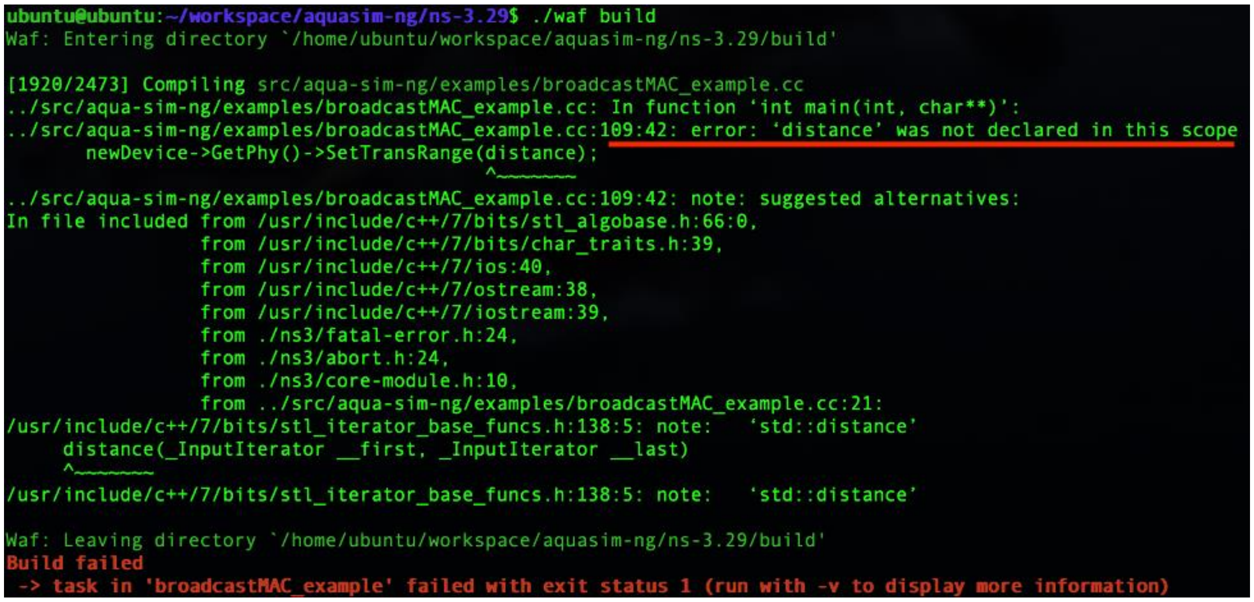
3.2.4 Debugging

There are multiple types of bugs/errors which might occur during the writing of NS-3 scripts or NS-3 modules in C++. In general, these types of errors can be categorized as:

1) Compilation/build errors:

These types of errors occur when a C++ compiler cannot build a source code due to either syntax or type errors, specific to C++ programming language. If such an error occurs, you would see an error output after running `./waf build` command.

Here is the example error-output, occurred due to a syntax error:



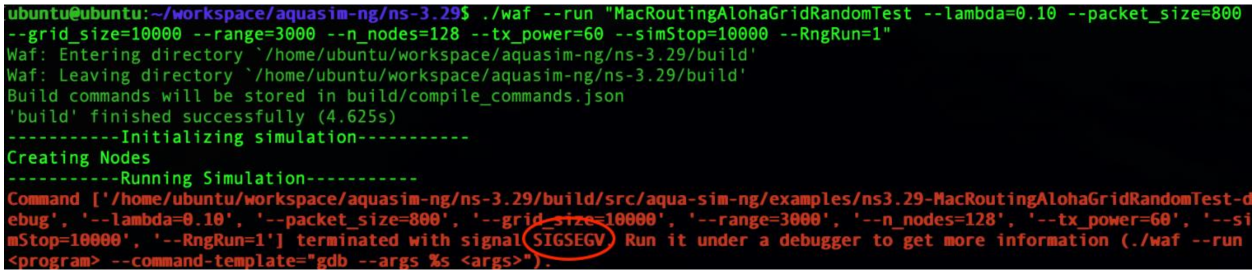
```
ubuntu@ubuntu:~/workspace/aquasim-ng/ns-3.29$ ./waf build
Waf: Entering directory `/home/ubuntu/workspace/aquasim-ng/ns-3.29/build'

[1920/2473] Compiling src/aqua-sim-ng/examples/broadcastMAC_example.cc
../src/aqua-sim-ng/examples/broadcastMAC_example.cc: In function 'int main(int, char**)':
../src/aqua-sim-ng/examples/broadcastMAC_example.cc:109:42: error: 'distance' was not declared in this scope
       newDevice->GetPhy()->SetTransRange(distance);
                                          ^~~~~~~~
../src/aqua-sim-ng/examples/broadcastMAC_example.cc:109:42: note: suggested alternatives:
In file included from /usr/include/c++/7/bits/stl_algobase.h:66:0,
                 from /usr/include/c++/7/bits/char_traits.h:39,
                 from /usr/include/c++/7/ios:40,
                 from /usr/include/c++/7/ostream:38,
                 from /usr/include/c++/7/iostream:39,
                 from ./ns3/fatal-error.h:24,
                 from ./ns3/abort.h:24,
                 from ./ns3/core-module.h:10,
                 from ../src/aqua-sim-ng/examples/broadcastMAC_example.cc:21:
/usr/include/c++/7/bits/stl_iterator_base_funcs.h:138:5: note:   'std::distance'
     distance(_InputIterator __first, _InputIterator __last)
     ^~~~~~~~
/usr/include/c++/7/bits/stl_iterator_base_funcs.h:138:5: note:   'std::distance'

Waf: Leaving directory `/home/ubuntu/workspace/aquasim-ng/ns-3.29/build'
Build failed
 -> task in 'broadcastMAC_example' failed with exit status 1 (run with -v to display more information)
```

In the error-output above, you can see a description of an error and its location in the code. Here, a `distance` variable was tried to be used before its actual declaration in the code – **line 109, position 42**.

2) Runtime errors:

These errors usually occur when a specific variable, pointer or a class object has been "lost" in the code and, therefore, is pointing to a wrong location in a memory. This also applies to some C++ data structures, such as **std::vector** or **std::map**, when a program tries to reach an element outside the current boundaries of that structure.

A sample error-output below throws out an **SIGSEGV** error during the runtime, because a program tried to access an "empty" pointer (a pointer, which was declared in *.h* source file, but was never initialized/created in the actual *.cc* declaration):



```
ubuntu@ubuntu:~/workspace/aquasim-ng/ns-3.29$ ./waf --run "MacRoutingAlohaGridRandomTest --lambda=0.10 --packet_size=800
--grid_size=10000 --range=3000 --n_nodes=128 --tx_power=60 --simStop=10000 --RngRun=1"
Waf: Entering directory `/home/ubuntu/workspace/aquasim-ng/ns-3.29/build'
Waf: Leaving directory `/home/ubuntu/workspace/aquasim-ng/ns-3.29/build'
Build commands will be stored in build/compile_commands.json
'build' finished successfully (4.625s)
-----------Initializing simulation-----------
Creating Nodes
-----------Running Simulation-----------
Command ['/home/ubuntu/workspace/aquasim-ng/ns-3.29/build/src/aqua-sim-ng/examples/ns3.29-MacRoutingAlohaGridRandomTest-d
ebug', '--lambda=0.10', '--packet_size=800', '--grid_size=10000', '--range=3000', '--n_nodes=128', '--tx_power=60', '--si
mStop=10000', '--RngRun=1'] terminated with signal SIGSEGV. Run it under a debugger to get more information (./waf --run
<program> --command-template="gdb --args %s <args>").
```

As can be noticed, there is no specific "compilation error", which could indicate to an exact reason of such an error. Therefore, in order to debug such type of errors, the developers usually rely on a few tools:

- **Using custom "breakpoints", inserted in the source code:**
  To locate a place in the code, where a runtime error occurs, a developer might insert some special lines in the code, which would output some additional useful information (i.e., a value of some variable or the contents of a data structure) during the runtime. When a runtime error occurs, one can take a look at the debug output at trace the error back to the source.

  These breakpoints can also stop a program execution, if some condition inside a program's state is incorrect. See more info about **Assertions** in NS-3 documentation here: https://www.nsnam.org/doxygen/group__assert.html

- **Use GDB debugger:**
  This is a more "advanced" tool, compared to the previous method, which allows a step-by-step execution of a program, examining every internal state of a program in-between the steps (GDB ***breakpoints***). The GDB debugger uses its own command-syntax. Some additional information on how to use GDB in NS-3 debugging can be found here:
  - https://www.nsnam.org/doxygen/group__assert.html
  - https://www.nsnam.org/wiki/HOWTO_use_gdb_to_debug_program_errors

3) Logical errors in NS-3 scripts or modules:

These types of errors occur when an output/result from an NS-3 program is different from the expected outcome. This might happen, for example, when there are no received packets during the entire simulation run, when a custom MAC protocol is used. Or, if a simulation script allocates the nodes in such a way, which they are too far from each other (outside the maximum transmission range), etc.

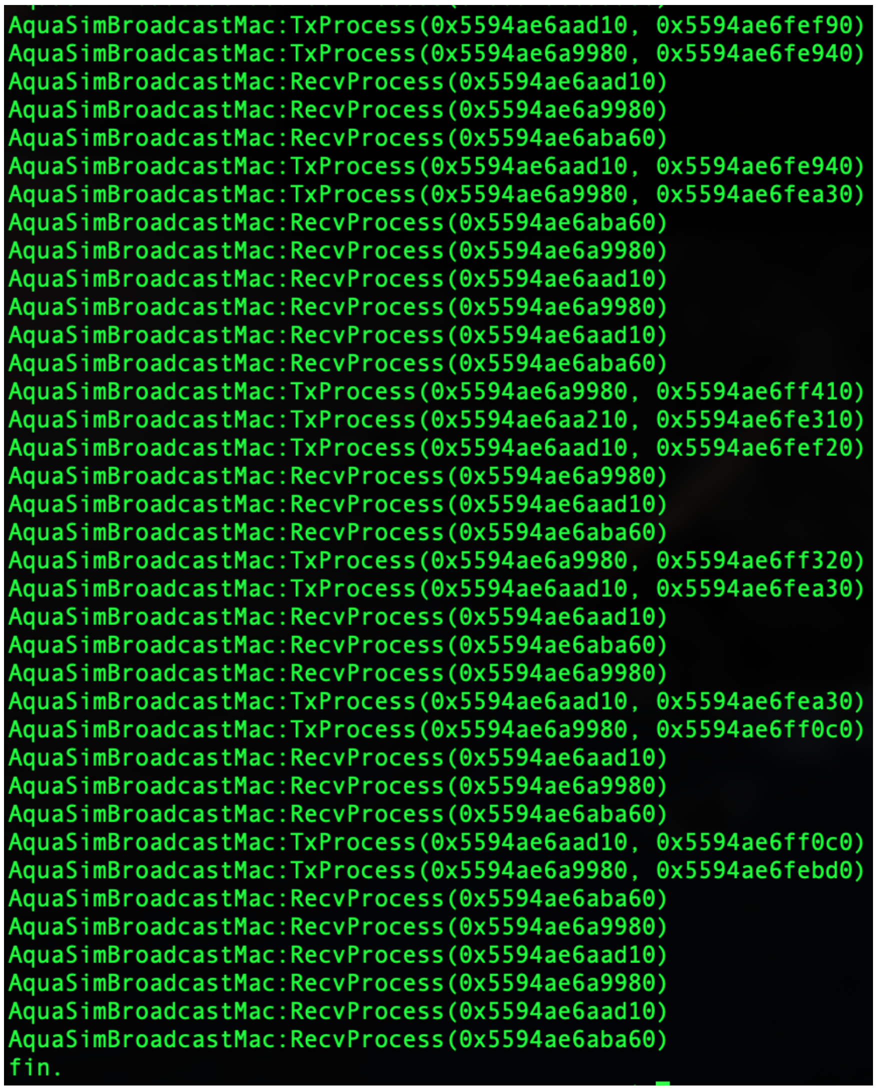
Such types of errors/misbehaviors are more difficult to debug, since a possible error can occur on any layer of a program, depending on its structure.

In such a case, the first thing which should be done is ***to narrow down a scope of the problem*** by trying to replace certain modules in the simulation program to the similar ones. For example, if there are no packets received when a **BroadcastMac** module is used, but the packets start to be received when a MAC layer is changed from **BroadcastMac** to **PureAloha** module, the problem is clearly in the **BroadcastMac** module.

Secondly, when a particular module where a problem occurs is identified (**BroadcastMac** module in this example), we can enable a debug-output for that particular module as the program is running. For example, in order to run a sample **broadcastMac** scenario with the additional debug information enabled, the following flags should be added to a corresponding command:

```
./waf  --run broadcastMAC_example NS_LOG=AquaSimBroadcastMac=level_all
```

The command above specifies an additional debug-output of **AquaSimBroadcastMac** module to a terminal. A part of the debug-output is shown below:



```
AquaSimBroadcastMac:TxProcess(0x5594ae6aad10, 0x5594ae6fef90)
AquaSimBroadcastMac:TxProcess(0x5594ae6a9980, 0x5594ae6fe940)
AquaSimBroadcastMac:RecvProcess(0x5594ae6aad10)
AquaSimBroadcastMac:RecvProcess(0x5594ae6a9980)
AquaSimBroadcastMac:RecvProcess(0x5594ae6aba60)
AquaSimBroadcastMac:TxProcess(0x5594ae6aad10, 0x5594ae6fe940)
AquaSimBroadcastMac:TxProcess(0x5594ae6a9980, 0x5594ae6fea30)
AquaSimBroadcastMac:RecvProcess(0x5594ae6aba60)
AquaSimBroadcastMac:RecvProcess(0x5594ae6a9980)
AquaSimBroadcastMac:RecvProcess(0x5594ae6aad10)
AquaSimBroadcastMac:RecvProcess(0x5594ae6a9980)
AquaSimBroadcastMac:RecvProcess(0x5594ae6aad10)
AquaSimBroadcastMac:RecvProcess(0x5594ae6aba60)
AquaSimBroadcastMac:TxProcess(0x5594ae6a9980, 0x5594ae6ff410)
AquaSimBroadcastMac:TxProcess(0x5594ae6aa210, 0x5594ae6fe310)
AquaSimBroadcastMac:TxProcess(0x5594ae6aad10, 0x5594ae6fef20)
AquaSimBroadcastMac:RecvProcess(0x5594ae6a9980)
AquaSimBroadcastMac:RecvProcess(0x5594ae6aad10)
AquaSimBroadcastMac:RecvProcess(0x5594ae6aba60)
AquaSimBroadcastMac:TxProcess(0x5594ae6a9980, 0x5594ae6ff320)
AquaSimBroadcastMac:TxProcess(0x5594ae6aad10, 0x5594ae6fea30)
AquaSimBroadcastMac:RecvProcess(0x5594ae6aad10)
AquaSimBroadcastMac:RecvProcess(0x5594ae6aba60)
AquaSimBroadcastMac:RecvProcess(0x5594ae6a9980)
AquaSimBroadcastMac:TxProcess(0x5594ae6aad10, 0x5594ae6fea30)
AquaSimBroadcastMac:TxProcess(0x5594ae6a9980, 0x5594ae6ff0c0)
AquaSimBroadcastMac:RecvProcess(0x5594ae6aad10)
AquaSimBroadcastMac:RecvProcess(0x5594ae6a9980)
AquaSimBroadcastMac:RecvProcess(0x5594ae6aba60)
AquaSimBroadcastMac:TxProcess(0x5594ae6aad10, 0x5594ae6ff0c0)
AquaSimBroadcastMac:TxProcess(0x5594ae6a9980, 0x5594ae6febd0)
AquaSimBroadcastMac:RecvProcess(0x5594ae6aba60)
AquaSimBroadcastMac:RecvProcess(0x5594ae6a9980)
AquaSimBroadcastMac:RecvProcess(0x5594ae6aad10)
AquaSimBroadcastMac:RecvProcess(0x5594ae6a9980)
AquaSimBroadcastMac:RecvProcess(0x5594ae6aad10)
AquaSimBroadcastMac:RecvProcess(0x5594ae6aba60)
fin.
```

From the output above, a user can see Tx/Rx events, happened on the MAC layer during the simulation. This might lead to a source of the problem a user is looking for. A user can specify

virtually any NS-3 module in the debug-output by calling its named, specified in its **TypeId** declaration.

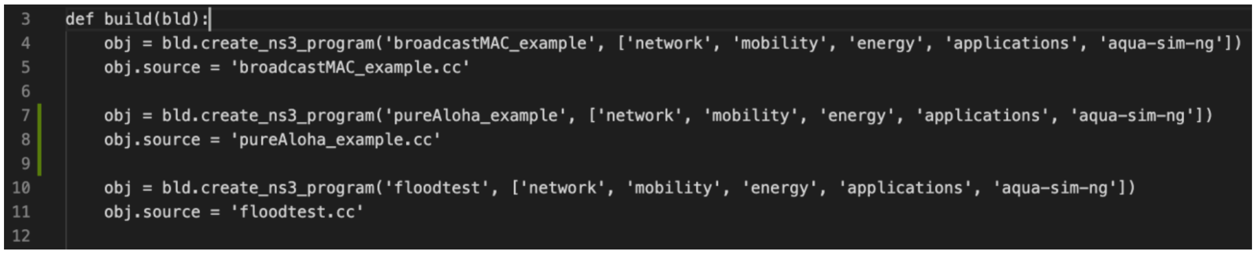
3.3 Creating a simulation script

Now let's create a simulation script with our developed pure ALOHA MAC module. For that, let's use the simulation script for broadcast MAC (broadcastMAC_example) from the **section 2.4**.

1) Duplicate the *broadcastMAC_example.cc* script and rename it to *pureALOHA_example.cc*, and change the MAC protocol definition inside the script on **line 77**:

- Replace `asHelper.SetMac("ns3::AquaSimBroadcastMac")` with:
- `asHelper.SetMac("ns3::AquaSimPureAlohaMac")`

2) Add the script definition into wscript configuration file under ***aqua-sim-ng/examples/*** folder:

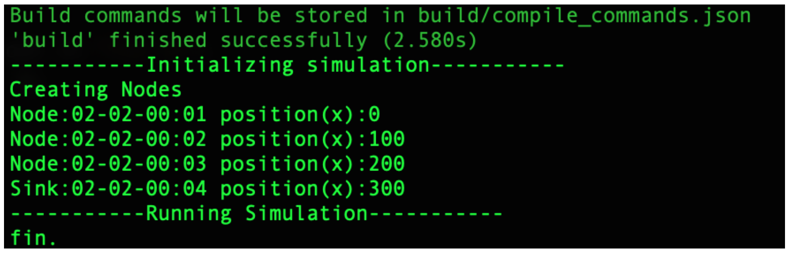


```
def build(bld):
    obj = bld.create_ns3_program('broadcastMAC_example', ['network', 'mobility', 'energy', 'applications', 'aqua-sim-ng'])
    obj.source = 'broadcastMAC_example.cc'

    obj = bld.create_ns3_program('pureAloha_example', ['network', 'mobility', 'energy', 'applications', 'aqua-sim-ng'])
    obj.source = 'pureAloha_example.cc'

    obj = bld.create_ns3_program('floodtest', ['network', 'mobility', 'energy', 'applications', 'aqua-sim-ng'])
    obj.source = 'floodtest.cc'
```

3) Run the script, using the same ***./waf*** command we used for ***broadcastMAC***:

```
./waf --run pureAloha_example
```

The script should compile all the new modules and run the simulation:



```
Build commands will be stored in build/compile_commands.json
'build' finished successfully (2.580s)
-----------Initializing simulation-----------
Creating Nodes
Node:02-02-00:01 position(x):0
Node:02-02-00:02 position(x):100
Node:02-02-00:03 position(x):200
Sink:02-02-00:04 position(x):300
-----------Running Simulation-----------
fin.
```

Complete source code of **pureAlohe_example.cc** is given in **Appendix C**.

Section 4. Tests and performance evaluation.

To tests the developed protocol, we can use the following approach:

- try different network topologies:
  - complete graph with N nodes
  - star topology
  - line topology, etc.
  - dynamic topologies (when nodes move during the simulation, according to the mobility models)

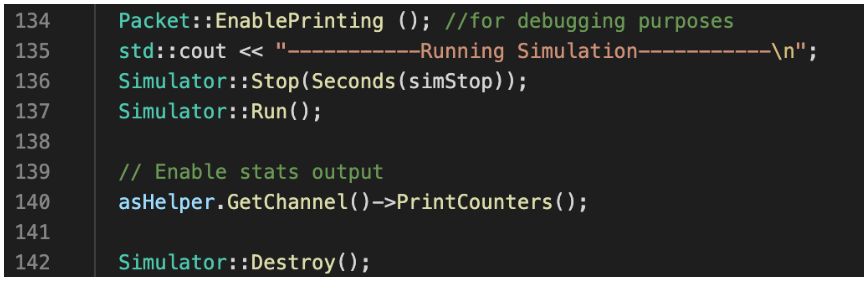
- experiment with the network traffic:
  - change the packet rate on the nodes
  - vary a number of sources (the nodes which generate traffic) and destinations
  - change total simulation/application duration

- measure network performance parameters:
  - Network throughput
  - Packet Delivery Ratio (PDR)
  - End-to-end packet delay
  - Energy consumption per packet

4.1 Channel methods

Aqua-sim-ng provides a standard method for gathering and presenting some generic statistics from the simulation runs. This method is called ***PrintCounters()*** and can be inserted into the simulation script using the following code:

```
asHelper.GetChannel()->PrintCounters();
```
See the last lines of **pureAloha_example.cc** (line 140):



```
134    Packet::EnablePrinting (); //for debugging purposes
135    std::cout << "-----------Running Simulation-----------\n";
136    Simulator::Stop(Seconds(simStop));
137    Simulator::Run();
138
139    // Enable stats output
140    asHelper.GetChannel()->PrintCounters();
141
142    Simulator::Destroy();
```

PrintCounters() method outputs the following metrics:

- Sent Pkts(Source_NetDevice->Stack):

Total amount of sent packets

- SendUp Pkts(Sink_RoutingLayer):

Total amount of successfully received packets

- Recv Pkts(@PhyLayer):

Number of packets, which reached the receivers (including the collided ones).

- TotalEnergyConsumption:

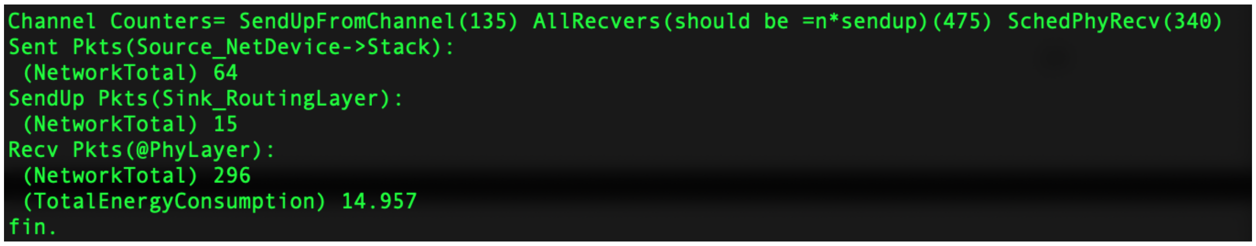
Total energy spent for transmitting/receiving packets throughout the simulation, in Joules.

4.1.1 Example

In the previous ***pureALOHA_example.cc*** script, after `Simulator::Run()` line, add the following:

```
asHelper.GetChannel()->PrintCounters();
```

After running the script, you should see the following output:



```
Channel Counters= SendUpFromChannel(135) AllRecvers(should be =n*sendup)(475) SchedPhyRecv(340)
Sent Pkts(Source_NetDevice->Stack):
 (NetworkTotal) 64
SendUp Pkts(Sink_RoutingLayer):
 (NetworkTotal) 15
Recv Pkts(@PhyLayer):
 (NetworkTotal) 296
 (TotalEnergyConsumption) 14.957
fin.
```

Section 5. Extra tools.

This section will cover some additional tools for NS-3 simulator, useful for deeper analysis of the simulation results.

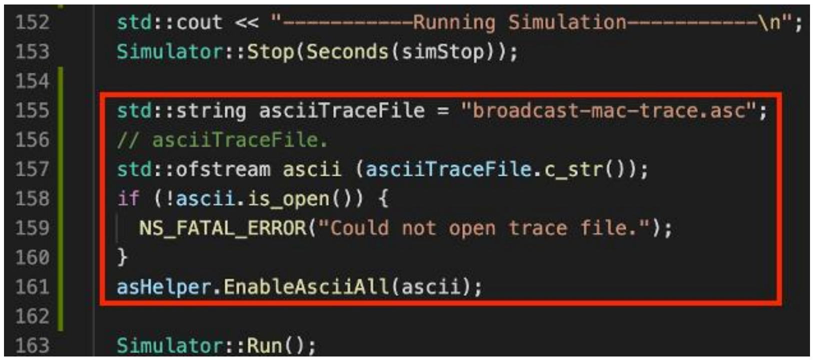
5.1 Trace files

NS-3 trace tool provides a very powerful way to obtain any information about any simulation event, happened during the simulation.

This tool is based on C++ callbacks, which are inserted into the desired parts of the code, logging all the events going through it. The logs are then saved in *.asc* format and can be post-processed (parsed) using custom-written parsers, using e.g. Python language.

5.1.1 Enabling trace files

To enable trace-files in a simulation script, one should insert the following lines at the end of a simulation program, right before **Simulator::Run();** line:



```
152   std::cout << "-----------Running Simulation-----------\n";
153   Simulator::Stop(Seconds(simStop));
154
155   std::string asciiTraceFile = "broadcast-mac-trace.asc";
156   // asciiTraceFile.
157   std::ofstream ascii (asciiTraceFile.c_str());
158   if (!ascii.is_open()) {
159     NS_FATAL_ERROR("Could not open trace file.");
160   }
161   asHelper.EnableAsciiAll(ascii);
162
163   Simulator::Run();
```

Line 155:	Specify name of a trace-file.
Line 156-160:	Create a trace file in the same directory. Overwrite any file with the same name.
Line 161:	Call **EnableAsciiAll()** method in aqua-sim-helper to allow executing Tx/Rx trace-callbacks on PHY-layer during a simulation run. This will write ***every*** Tx/Rx event, happened in the simulation, to an .asc file for further analysis.

5.2 Network visualization

In NS-3, the topological information from a given simulation scenario can be visualized using NetAnim tool [4]. This tool provides a separate **AnimationInterface()** class, which writes simulation events to a separate *.xml* file in a pre-defined format. Then, a separate ***./netanim*** tool is executed, animating the events stored in the *.xml* file.

References

1. https://www.nsnam.org/documentation/

2. https://www.nsnam.org/docs/release/3.30/tutorial/html/index.html

3. https://www.nsnam.org/docs/models/html/index.html

4. https://www.nsnam.org/wiki/NetAnim_3.105

Appendix A. Source code of aqua-sim-pure-aloha.h

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
 */

#ifndef AQUA_SIM_MAC_PURE_ALOHA_H
#define AQUA_SIM_MAC_PURE_ALOHA_H

#include "aqua-sim-mac.h"

#include "ns3/event-id.h"
#include "ns3/random-variable-stream.h"
#include "ns3/packet.h"


namespace ns3 {
/**
 * \ingroup aqua-sim-ng
 *
 * \brief Implementation of Pure ALOHA for underwater channel
 */
class AquaSimPureAloha: public AquaSimMac
{
public:
  AquaSimPureAloha();
  ~AquaSimPureAloha();
  static TypeId GetTypeId(void);
  int64_t AssignStreams (int64_t stream);

  virtual bool TxProcess(Ptr<Packet> pkt);
  virtual bool RecvProcess(Ptr<Packet> pkt);

  void SendPacket();
  void SendDownPacket(Ptr<Packet> pkt);

  double GetBackoff();

  virtual void DoDispose();


protected:

private:
  Ptr<UniformRandomVariable> m_rand;

  std::queue<Ptr<Packet>> m_send_queue;
};  // class AquaSimPureAloha

} // namespace ns3

#endif /* AQUA_SIM_MAC_PURE_ALOHA_H */
```

Appendix B. Source code of aqua-sim-pure-aloha.cc

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
 */


#include "aqua-sim-mac-pure-aloha.h"
#include "aqua-sim-header.h"
#include "aqua-sim-header-mac.h"

#include "ns3/packet.h"
#include "ns3/log.h"
#include "ns3/random-variable-stream.h"
#include "ns3/simulator.h"


namespace ns3{

NS_LOG_COMPONENT_DEFINE("AquaSimPureAloha");
NS_OBJECT_ENSURE_REGISTERED(AquaSimPureAloha);

//construct function
AquaSimPureAloha::AquaSimPureAloha() : AquaSimMac()
{
  m_rand = CreateObject<UniformRandomVariable> ();
}

AquaSimPureAloha::~AquaSimPureAloha()
{
}

TypeId
AquaSimPureAloha::GetTypeId(void)
{
  static TypeId tid = TypeId("ns3::AquaSimPureAloha")
      .SetParent<AquaSimMac>()
      .AddConstructor<AquaSimPureAloha>()
    ;
  return tid;
}

int64_t
AquaSimPureAloha::AssignStreams (int64_t stream)
{
  NS_LOG_FUNCTION (this << stream);
  m_rand->SetStream(stream);
  return 1;
}


/*========================== Send and Receive ===========================*/

// Receive packet from L3 upper layer
bool AquaSimPureAloha::TxProcess(Ptr<Packet> pkt)
{
  NS_LOG_FUNCTION(m_device->GetAddress() << pkt << Simulator::Now().GetSeconds());
  AquaSimHeader asHeader;
  pkt->RemoveHeader(asHeader);
  asHeader.SetTxTime(GetTxTime(asHeader.GetSize()));
  asHeader.SetErrorFlag(false);
  asHeader.SetDirection(AquaSimHeader::DOWN);
  NS_LOG_INFO("Transmission time: "<< asHeader.GetTxTime ().GetSeconds () << " seconds");

  pkt->AddHeader(asHeader);

  // Start sending, if queue is empty
  if (m_send_queue.size() == 0)
  {
    // Push packet to MAC send queue
    m_send_queue.push(pkt);
    SendPacket();
  }
  else
  {
    m_send_queue.push(pkt);
  }

  return true;
}

void AquaSimPureAloha::SendDownPacket(Ptr<Packet> pkt)
{
  AquaSimHeader asHeader;
  pkt->RemoveHeader(asHeader);

  asHeader.SetDirection(AquaSimHeader::DOWN);

  MacHeader mach;
  mach.SetDA(asHeader.GetDAddr());
  mach.SetSA(AquaSimAddress::ConvertFrom(m_device->GetAddress()));
  pkt->AddHeader(mach);
  pkt->AddHeader(asHeader);

  // Check net_device status
  // If it is IDLE, send packet down to PHY immediately
  // otherwise, do random backoff again
  if (m_device->GetTransmissionStatus() == TransStatus::NIDLE)
  {
    // Call parent method to send packet down to PHY
    SendDown(pkt);
    // Go to next packet in queue
    SendPacket();
  }
  else
  {
    // Do another backoff
    Simulator::Schedule(Seconds(GetBackoff()), &AquaSimPureAloha::SendDownPacket, this, pkt->Copy());
  }
}

// Return a random number in-between given range
double AquaSimPureAloha::GetBackoff()
{
  return m_rand->GetValue(0.1, 1.0);
}

// Try to send packet to PHY layer, after backoff
void AquaSimPureAloha::SendPacket()
{
  NS_LOG_FUNCTION(this);

  // If queue is empty -> nothing to send -> return
  if (m_send_queue.size() == 0)
  {
    return;
  }

  // Get packet from queue
  Ptr<Packet> pkt = m_send_queue.front();
  m_send_queue.pop();

  // Get random backoff time from [0.1, 1.0] sec interval, schedule transmission
  Simulator::Schedule(Seconds(GetBackoff()), &AquaSimPureAloha::SendDownPacket, this, pkt->Copy());
}

// Receive packet from PHY
bool AquaSimPureAloha::RecvProcess(Ptr<Packet> pkt)
{
  NS_LOG_FUNCTION(m_device->GetAddress());
  AquaSimHeader asHeader;
  MacHeader mach;
  pkt->RemoveHeader(asHeader);
  pkt->RemoveHeader(mach);  //not used, leave off.
  pkt->AddHeader(asHeader);

  // If a packet experienced collisions / channel errors (ErrorFlag is true), drop it
  if( asHeader.GetErrorFlag() )
  {
    NS_LOG_INFO("Packet:" << pkt << " error/collision on node " << m_device->GetNode());
    pkt=0;
   return false;
  }

  // Send packet up to the routing layer, if the destination address matches with this node
  AquaSimAddress node_address = AquaSimAddress::ConvertFrom(m_device->GetAddress());
  AquaSimAddress dst_address = mach.GetDA();

  if (node_address == dst_address)
  {
    // Send packet up to L3 layer
    SendUp(pkt);
  }

  return true;
}

void AquaSimPureAloha::DoDispose()
{
  NS_LOG_FUNCTION(this);
  AquaSimMac::DoDispose();
}

} // namespace ns3
```

Appendix C. Source code of pureAloha_example.cc

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
 */

#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/mobility-module.h"
#include "ns3/aqua-sim-ng-module.h"
#include "ns3/applications-module.h"
#include "ns3/log.h"
#include "ns3/callback.h"


/*
 * PureAlohaMAC
 *
 * String topology:
 * N ---->  N  -----> N -----> N* -----> S -----> S*
 *
 */

using namespace ns3;

NS_LOG_COMPONENT_DEFINE("PureAlohaMacTest");

int
main (int argc, char *argv[])
{
  double simStop = 100; //seconds
  int nodes = 3;
  int sinks = 1;
  uint32_t m_dataRate = 128; //bps
  uint32_t m_packetSize = 40; //bytes
  double range = 3000;  //meters

  LogComponentEnable ("PureAlohaMacTest", LOG_LEVEL_INFO);

  //to change on the fly
  CommandLine cmd;
  cmd.AddValue ("simStop", "Length of simulation", simStop);
  cmd.AddValue ("nodes", "Amount of regular underwater nodes", nodes);
  cmd.AddValue ("sinks", "Amount of underwater sinks", sinks);
  cmd.Parse(argc,argv);

  std::cout << "-----------Initializing simulation-----------\n";
```

```
NodeContainer nodesCon;
NodeContainer sinksCon;
nodesCon.Create(nodes);
sinksCon.Create(sinks);

PacketSocketHelper socketHelper;
socketHelper.Install(nodesCon);
socketHelper.Install(sinksCon);

//establish layers using helper's pre-build settings
AquaSimChannelHelper channel = AquaSimChannelHelper::Default();
channel.SetPropagation("ns3::AquaSimRangePropagation");
AquaSimHelper asHelper = AquaSimHelper::Default();
asHelper.SetChannel(channel.Create());
asHelper.SetMac("ns3::AquaSimPureAlohaMac");
asHelper.SetRouting("ns3::AquaSimRoutingDummy");

/*
 * Set up mobility model for nodes and sinks
 */
MobilityHelper mobility;
NetDeviceContainer devices;
Ptr<ListPositionAllocator> position = CreateObject<ListPositionAllocator> ();
Vector boundry = Vector(0,0,0);

std::cout << "Creating Nodes\n";

for (NodeContainer::Iterator i = nodesCon.Begin(); i != nodesCon.End(); i++)
  {
    Ptr<AquaSimNetDevice> newDevice = CreateObject<AquaSimNetDevice>();
    position->Add(boundry);
    devices.Add(asHelper.Create(*i, newDevice));

    NS_LOG_DEBUG("Node:" << newDevice->GetAddress() << " position(x):" << boundry.x);
    boundry.x += 100;
    newDevice->GetPhy()->SetTransRange(range);
  }

for (NodeContainer::Iterator i = sinksCon.Begin(); i != sinksCon.End(); i++)
  {
    Ptr<AquaSimNetDevice> newDevice = CreateObject<AquaSimNetDevice>();
    position->Add(boundry);
    devices.Add(asHelper.Create(*i, newDevice));

    NS_LOG_DEBUG("Sink:" << newDevice->GetAddress() << " position(x):" << boundry.x);
    boundry.x += 100;
    newDevice->GetPhy()->SetTransRange(range);
  }


 mobility.SetPositionAllocator(position);
 mobility.SetMobilityModel("ns3::ConstantPositionMobilityModel");
 mobility.Install(nodesCon);
 mobility.Install(sinksCon);

 PacketSocketAddress socket;
 socket.SetAllDevices();
 socket.SetPhysicalAddress (devices.Get(nodes)->GetAddress()); //Set dest to first sink (nodes+1 device)
 socket.SetProtocol (0);

 OnOffHelper app ("ns3::PacketSocketFactory", Address (socket));
 app.SetAttribute ("OnTime", StringValue ("ns3::ConstantRandomVariable[Constant=1]"));
 app.SetAttribute ("OffTime", StringValue ("ns3::ConstantRandomVariable[Constant=0]"));
 app.SetAttribute ("DataRate", DataRateValue (m_dataRate));
 app.SetAttribute ("PacketSize", UintegerValue (m_packetSize));

 ApplicationContainer apps = app.Install (nodesCon);
 apps.Start (Seconds (0.5));
 apps.Stop (Seconds (simStop + 1));


 Ptr<Node> sinkNode = sinksCon.Get(0);
 TypeId psfid = TypeId::LookupByName ("ns3::PacketSocketFactory");

 Ptr<Socket> sinkSocket = Socket::CreateSocket (sinkNode, psfid);
 sinkSocket->Bind (socket);

/*
 *  For channel trace driven simulation
 */
/*
 AquaSimTraceReader tReader;
 tReader.SetChannel(asHelper.GetChannel());
 if (tReader.ReadFile("channelTrace.txt")) NS_LOG_DEBUG("Trace Reader Success");
 else NS_LOG_DEBUG("Trace Reader Failure");
*/

 Packet::EnablePrinting (); //for debugging purposes
 std::cout << "-----------Running Simulation-----------\n";
 Simulator::Stop(Seconds(simStop));
 Simulator::Run();

 // Enable stats output
 asHelper.GetChannel()->PrintCounters();
```

```
  Simulator::Destroy();

  std::cout << "fin.\n";
  return 0;
}
```