1/31/17

Aqua-Sim Next Generation – Example Script Walk Through

Setup:

In this walk through we will be reviewing the *broadcastMAC_example* which can be found under aqua-sim-ng/examples directory by the source file name “broadcastMAC_example.cc”. This script depicts some of the basic components for setting up and running an underwater simulation on Aqua-Sim NG. Furthermore, we will discuss some of the helper scripts used and how they can benefit the user.

This example is constructed to mimic a basic string topology network with three regular nodes and a single sink. Each node is expected to create packets on its application layer and forward these generated packets towards the known sink device. Additionally, each node is expected to pass packets received from other nodes.

To begin, we first ensure that logging is defined within this example script:

```
 NS_LOG_COMPONENT_DEFINE(“ASBroadcastMac”);
```

By default Aqua-Sim NG will log all components within the script without the user having to define it as an argument. This is due to the following line:

```
LogComponentEnable ("ASBroadcastMac", LOG_LEVEL_INFO);
```

Next we include a collection of variables that can be hard-coded and changed dependent on the simulation constrains.

```
  double simStop = 100;
  int nodes = 3;
  int sinks = 1;
  uint32_t m_dataRate = 128;
  uint32_t m_packetSize = 40;
```

Important to note that dataRate is in Bps and packetSize is in Bytes. So for our application traffic generation the dataRate would be 1 Mib and our packetSize would be 320 bits.

We also have the option to change values directly through command line arguments using NS-3’s API:

```
  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);
```

To do this you use the keyword (first condition) when running an example script, which will overwrite the provided value to the specified variable (third condition).

Initializing Simulation:

The main purpose of this section is creating your simulated nodes and associated protocol stack on each. Additionally, we incorporate channel set up and attribute assignments in this section. To start we create two different containers, one for nodes and a separate for sinks:

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

To assist in creating these nodes and applying them to this container we use one of NS-3's provided helper classes, NodeContainer. By calling the create function for this class, we are provided with a specific container of the given size passed. Next we set up sockets for each of these containers. This is meant to assist packet handling:

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

Next we establish the channel and protocol stack which will be used in this simulation. Note, that this is a simplified example and can be expanded to support different protocol stacks on various nodes and multiple underwater channels. We first set up our channel as follows:

```
  AquaSimChannelHelper channel = AquaSimChannelHelper::Default();
  //channel.SetPropagation("ns3::AquaSimRangePropagation");
```

Since we are using the Aqua-Sim NG's channel helper this process is quite straight forward. The helper class will assign the default components to this newly created channel helper. This default setting includes a constant noise generator, defaulting to zero, and a simple propagation model. Instead we can overload the propagation model by applying the commented out line, which will set the channel's propagation model to range instead. Additionally, we can include attributes during this set up phase as well. An example of this is assigning the temperature and water salinty while assigning the range propagation to our channel model:

```
channel.SetPropagation("ns3::AquaSimRangePropagation", "Temperature", DoubleValue(20),
"Salinty", DoubleValue(30));
```

Next, we use the AquaSimHelper to assign all standard settings for our protocol stack and channel assignment. This is done in the following:

```
AquaSimHelper asHelper = AquaSimHelper::Default();
  asHelper.SetChannel(channel.Create());
  asHelper.SetMac("ns3::AquaSimBroadcastMac");
  asHelper.SetRouting("ns3::AquaSimRoutingDummy");
```

For this helper we use the default function. While the *Default()* function does not change the helper variable, it is meant to allow for easier assignments and changes for user creation beyond the standard constructor attributes. When creating the AquaSimHelper, we automatically assign standard features to our protocol stack, allowing for vanilla physical layer settings and avoiding any potential NULL pointers for upper protocol layers. These can be easily changed by either overloading these features within the *Default()* function of this helper or by setting new attributes similar to the propagation example previously mentioned. The process of calling *Create()* for the channel allows our Channel Helper to assign all previously defined attributes and return a populated AquaSimChannel pointer. For this example all nodes will be assigned to this single channel, and therefore we set this pointer through the AquaSimHelper *SetChannel()* function. For proof of concept and simplicity, we also set new MAC and Routing layer protocols. To accomplish this we use the uniquely defined component name which can be found under the source file of each protocol (e.g. NS_LOG_COMPONENT_DEFINE("AquaSimBroadcastMac");).

We also must set up the mobility model with a position list for all nodes to be added to. Alongside this, we also created a device container for preparation in creating and setting up all nodes in the next section.

```
MobilityHelper mobility;
NetDeviceContainer devices;
Ptr<ListPositionAllocator> position = CreateObject<ListPositionAllocator> ();
Vector boundry = Vector(0,0,0);
```

Node Creation and Settings:

To set up all nodes, we loop through both node and sink containers while creating assigning all components based on the previously assigned helper attributes.

```
  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);
```

By using *CreateObject* we are able to create a new device for the current node within the node container. We next add the current location (*boundary*) to our position list, which will serve as this nodes starting position in this simulation. Since we previously assigned all necessary attributes to the AquaSimHelper, we now will use this same helper to set up each node. This is done through the use of *Create(Ptr<Node>, Ptr<AquaSimNetDevice>)*, which assigns each protocol layer to the given device, followed by connecting each layer as necessary for Aqua-Sim NG to function, and updating the provided Node pointer. We store this newly created device in a device container for easier handling and access, if necessary. The use of *NS_LOG_DEBUG* is strictly to show that the current device was created properly during the beginning of the simulation run. Since this is a simplistic string topology, we only increment the x axis. We also provide an example of range assignment, which can be used if the user wishes to directly assign the maximum range of each node in their simulation.

Each node is meant to include some type of mobility model. This is supported through the extensive

NS-3 API. For our approach we assume all nodes' position is static, while a user could incorporate their own mobility model or adaptively changing model dependent on user required circumstances. We also must provide the mobility helper all starting positions for our nodes to assist in properly tracking each node's mobility.

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

Next we set up our socket for application layer usage. This is completed as follows:

```
  PacketSocketAddress socket;
  socket.SetAllDevices();
  socket.SetPhysicalAddress (devices.Get(nodes)->GetAddress());
  socket.SetProtocol (0);
```

We first apply this created socket to all devices using the *SetAllDevices()* function. To provide a destination for each packet created we decided to provide a physical address, given the sink's address. Since our addresses are assigned starting from 0 and we created our sink last, the address of our sink (since we are only using one in this example) would be equivalent to the variable *nodes*. This is because the *AquaSimAddress* class assigns addresses in incremental order.

Application Traffic Generator:

To assist in generating packets we use the OnOffHelper in this example. Based on the previously used created sockets, this application layer will generate traffic using the provided dataRate and packetSize set at the beginning of the example script.

```
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));
```

While this is a simple integration of one of NS-3's application protocols, it may be beneficial for users to create their own traffic generator dependent on what packets are expected to be originally created and received from upper layers in the protocol stack.

To install this OnOffHelper application to our nodes we do the following:

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

Here we pass the entire node container, along with a starting and stopping parameter for this traffic generator. It is important to note that this installation can be a single nodes if the simulation requires.

We must also bind this socket to our sink, in order for packets to be properly received. This is completed as follows:

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

Running Simulation:

The last step in this example script is to run your simulation. This is done through the Simulator API as follows:

```
Simulator::Stop(Seconds(simStop));
Simulator::Run();
Simulator::Destroy();
```

By using the provided *simStop* variable, we are able to set the simulation to run for a predetermined amount of time.

Contact Information:

Questions, comments, issues, or anything else can be directed to:

Robert Martin
Robert.Martin@engr.uconn.edu

or posted directly under aqua-sim-ng issues on GitHub.