ns-3 Network Simulator

Nikos Episkopos

nepiskopos@fogus.gr

INTRODUCTION

- Discrete-event network simulator
- Sufficiently realistic simulation models
- Can be used as a real-time network emulator
- Can interact with real network devices
- F.L.O.S.S. (licensed under the GNU GPLv2)
- Website
  - www.nsnam.org

WHY NS-3?

- ns-2 development stopped around 2010, so it is no longer developed nor maintained
- ns-3 is a new software development effort focused on improving upon the core architecture, software integration, models, and educational components of ns-2
- ns-3
  - models nodes with higher realism (like a real computer)
  - supports key interfaces such as sockets API and IP/device driver interface (Linux)
  - reuses kernel and application code
- ns-3 is NOT backwards compatible with ns-2

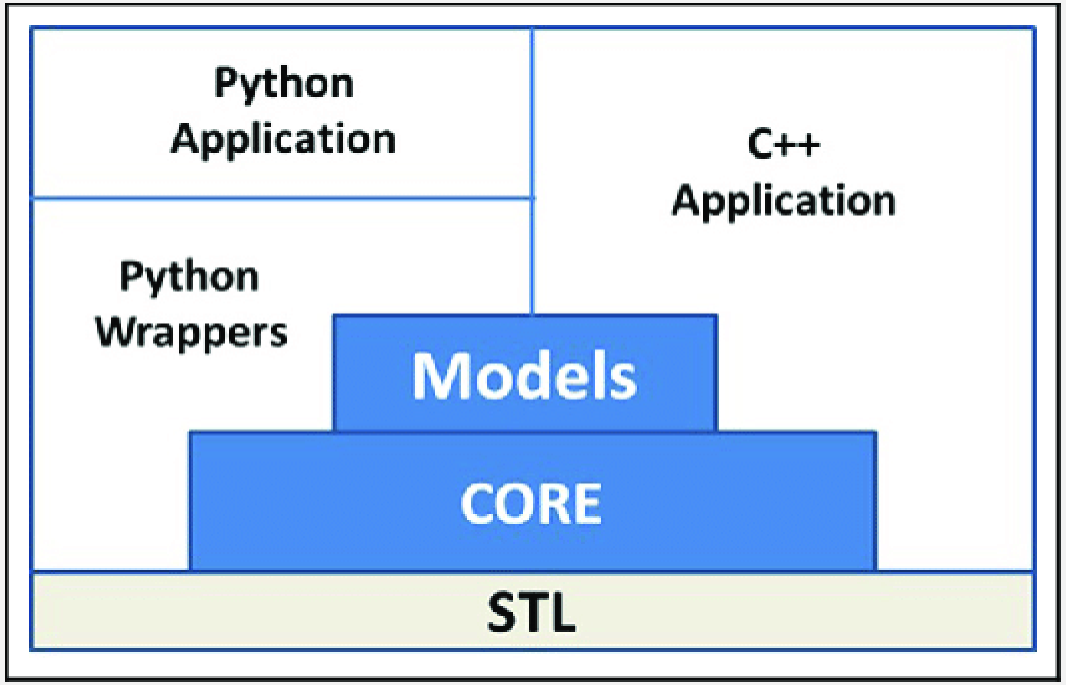
PLATFORMS

- As of version 3.33 (January 2021)
  - Arch Linux
  - Fedora 33
  - Ubuntu 20.04
  - Ubuntu 18.04
  - Ubuntu 16.04.6
  - Linux Mint 20 Ulyana
  - macOS 11.1 (Big Sur)
  - macOS 10.15.7 (Catalina)
  - macOS 10.13.6 (High Sierra)

TOOLKITS & IDE

- Supported programming languages
  - C++
  - Python
- Supported and currently tested toolkits:
  - GCC (g++): version 5.4.0 or later
  - Xcode: version 10.1 or later
  - Python: version 3.5.2 or later
- Waf
  - Python-based build automation tool
  - Used in building simulations
- No official IDE
  - Eclipse, NetBeans, QtCreator are being unofficially used by developers

NS-3 ARCHITECTURE



C++ FEATURES

- Both standard C++ and ns-3 specific headers are supported
  - #include <ns3/core-module.h>
  - #include <ns3/log.h>
  - #include <string>
  - #include <vector>
- Both standard C++ and ns-3 specific namespaces are supported
  - using namespace ns3;
  - ~~using namespace std;~~ **DO NOT USE**
- The use of explicit names is suggested instead of using the std namespace
  - std::string
  - std::cout

C++ SIMULATIONS

- Running an existing C++ simulation (from the examples)

```
ls examples/wireless/mixed-wired-wireless.cc
./waf --run mixed-wired-wireless
```

- Creating and running a new C++ simulation with a single script

```
mkdir scratch/mysim
touch scratch/mysim/file.cc
./waf --run mysim
```

- Creating and running a new C++ simulation with multiple scripts

```
mkdir scratch/mysim
touch scratch/mysim/file1.cc
touch scratch/mysim/file2.cc
./waf --run mysim
```

C++ SIMULATIONS

- Running an existing C++ simulation with a debugger
  - GDB

```
./waf --run mixed-wired-wireless --command-template="gdb %s"
```

  - Valgrind

```
./waf --run mixed-wired-wireless --command-template="valgrind %s"
```

- Running an existing C++ simulation with the PyViz visualizer

```
./waf --run mixed-wired-wireless --vis
```

PYTHON FEATURES

- Both standard Python and ns-3 specific modules are supported
  - import ns.core
  - import ns.mobility
  - import ns.wifi
  - import collections
  - import numpy

PYTHON SIMULATIONS

- We need to specify a complete path to the Python script file, and use the --pyrun command instead of --run

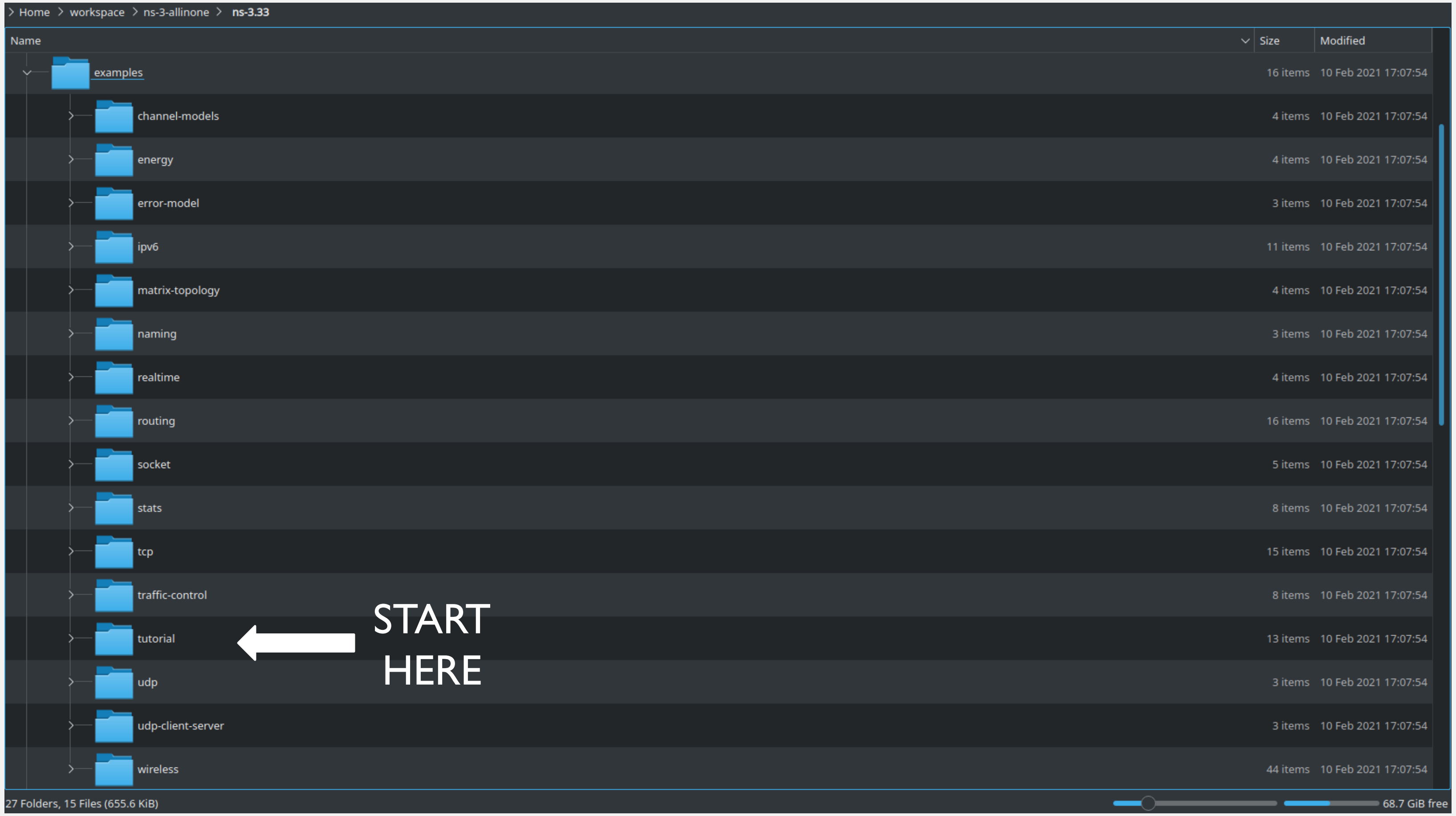
- Running an existing Python simulation (from the examples)

```
./waf --pyrun examples/wireless/mixed-wired-wireless.py
```

- Creating and running a new Python simulation with a single script

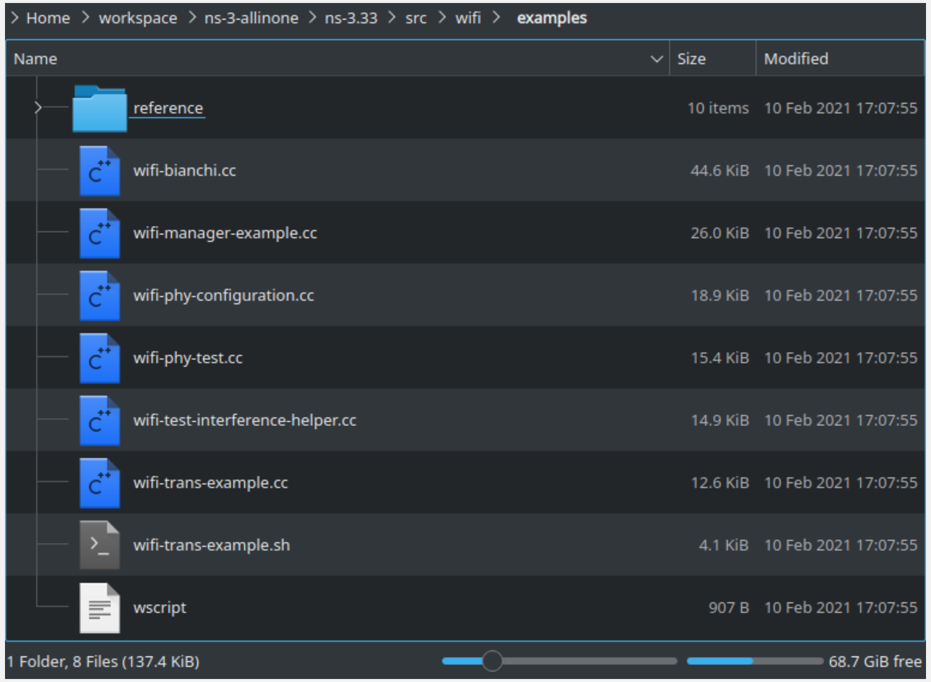
```
mkdir scratch/mysim
touch scratch/mysim/file.py
./waf --pyrun scratch/mysim/file.py
```

BUT… WHERE TO START?

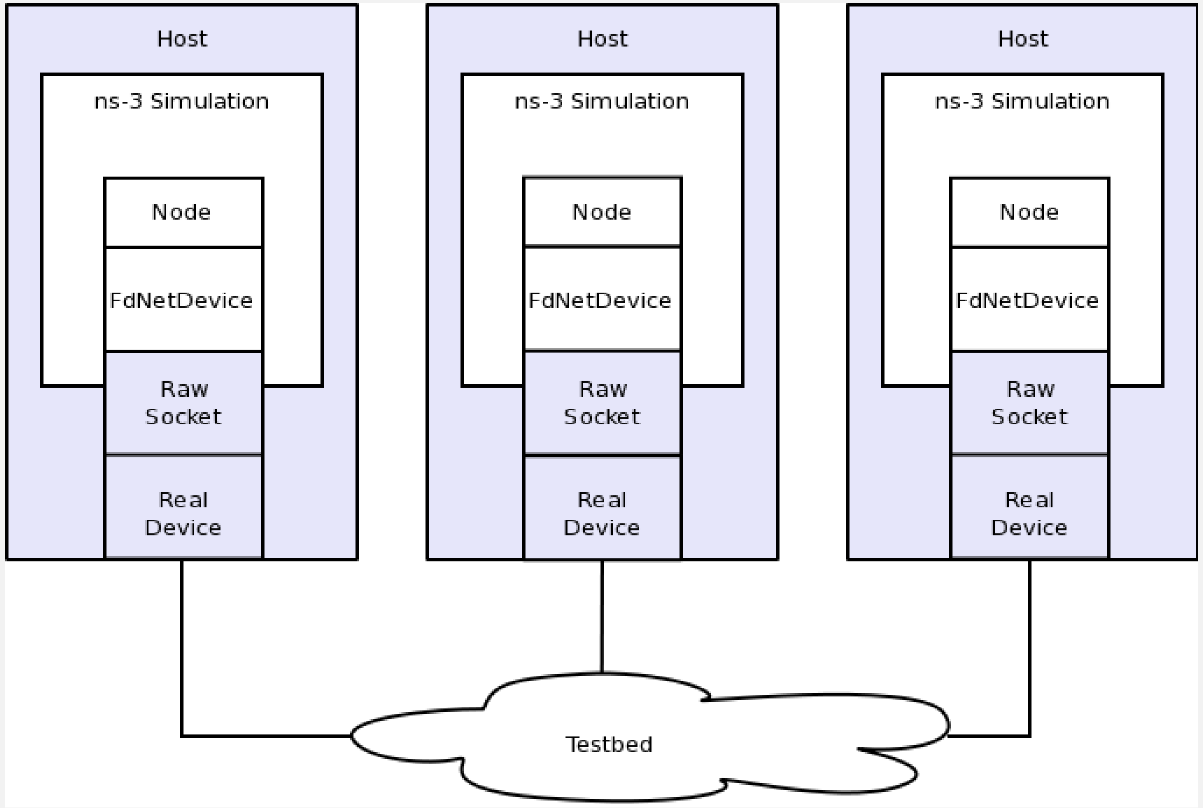


BUT… WHERE TO START?

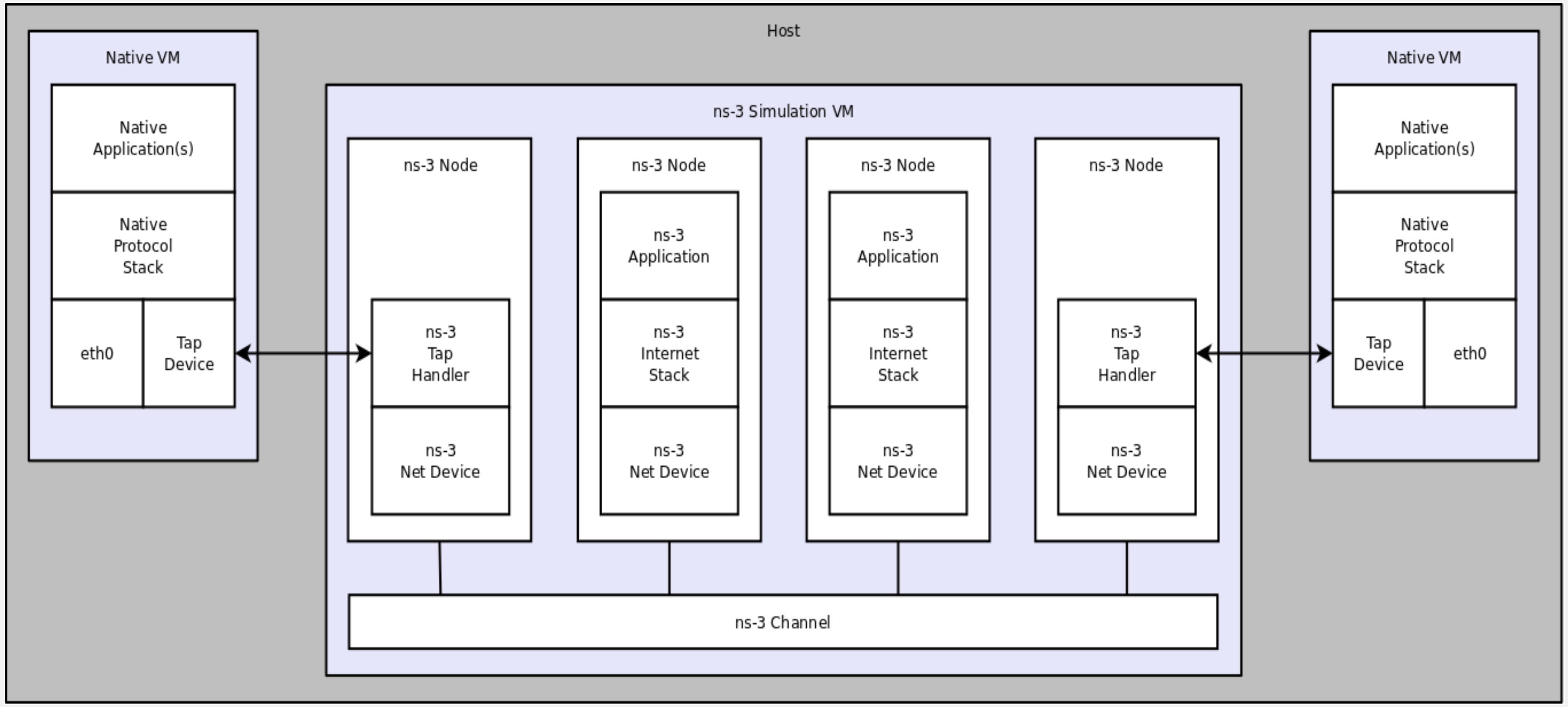
- Advanced examples exist for each module, in the respective directories



CAPABILITIES

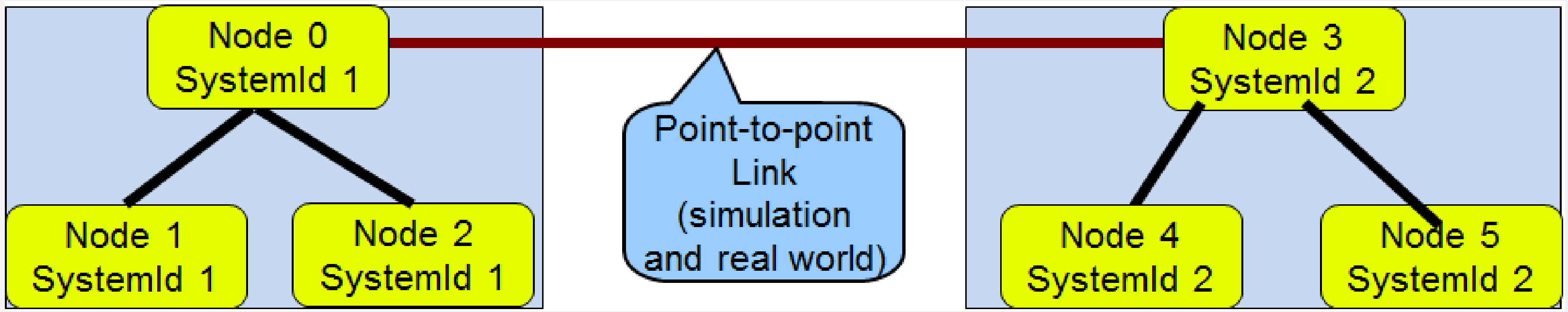


CAPABILITIES



CAPABILITIES

- Distributed Simulation with MPI
  - MPI: Message Passing Interface
    - Library (and protocol) for distributed applications
  - MPI simulator (as of ns-3.8)
    - Nodes in the simulation assigned different System Ids
    - Nodes with different System Ids run on different cluster machines
    - Nodes on different machines may communication using p2p links only



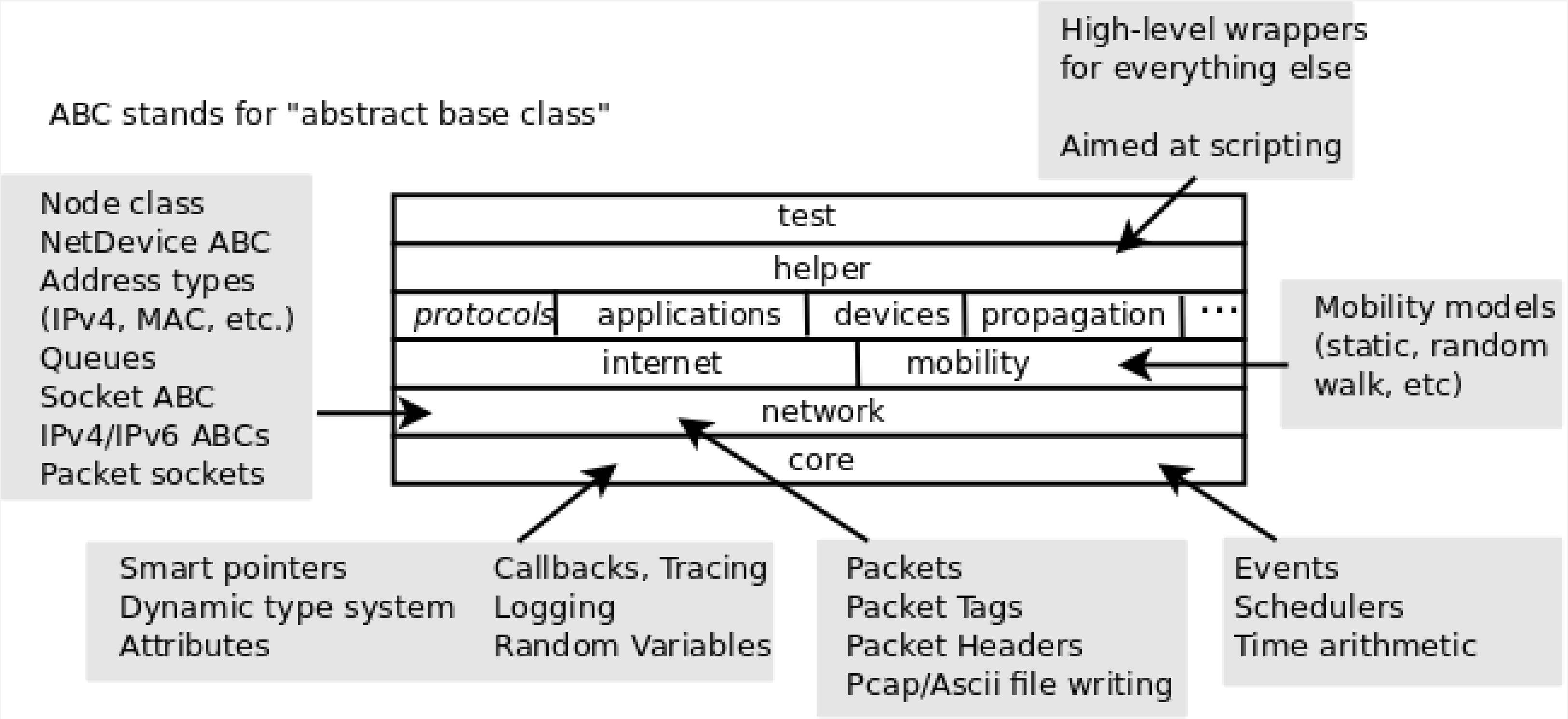
IMPORTANT MODULES

- Core
  - time management, scheduler, event listener, configuration, etc.
- IPv4Address
  - IPv4 subnet, addresses assignment and configuration, etc.
- Applications
  - Set, start and stop applications, use sockets, etc.
- Mobility
  - Sets, tracks and maintains the position and speed of objects, etc.
- Buildings
  - Places buildings, places nodes within buildings, helps in measuring the signal loss through walls, etc.

IMPORTANT MODULES

- LTE
  - eNB, EPC, UE, CSG, X2, S11, S1, RRC, RLC, etc.
- WiFi
  - AP, SSID, PHY, MAC, ARF, etc.
- Wireless Signal Propagation
  - Friis, Jakes, Nakagami, Buildings, LogDistance, etc.
- Traffic
  - Traffic modeling and queueing
- And many more…

IMPORTANT MODULES



SIMULATION

- Simulation time moves discretely from event to event
- 1 time frame = 1 ms
- Functions can be scheduled to occur at specific simulation times
- The Scheduler orders the event execution
- Simulation finishes either at a specified time point or when events end
- Events can be scheduled to be executed at a specific time or with context
- Events can be scheduled to be repeatedly executed at a specific interval

SIMULATION

- Set explicit simulation stop time

```
Simulator::Stop (Seconds (100));
```

- Start the simulation (should be called after simulation stop time is set)

```
Simulator::Run ();
```

- The final step, after simulation is complete, is to destroy it

```
Simulator::Destroy ();
```

SIMULATION

- Schedule a function to occur at the 1st second of the simulation

```
Simulator::Schedule (Seconds (1), &Function, &param1, &param2, …);
```

- Schedule a function to occur every 10 seconds repeatedly

```
void Function {
	…
	Simulator::Schedule (Seconds (10), &Function, &param1, &param2, …);
}

int main (int argc, char *argv[]) {
	…
	Simulator::Schedule (Seconds (10), &Function, &param1, &param2, …);
```

LOGGING

- ns-3 has built-in logging facility to stderr
- Logging is mainly intended for debugging but can be abused to provide tracing
- Supports multiple log levels like syslog
- Can trace functions and call arguments
- Can be driven from the shell

LOGGING

- Add logging to our code

```
using namespace ns3;
NS_LOG_COMPONENT_DEFINE (“Example");
int main (int argc, char *argv[]) {
…
```

```
if (address == iaddr.GetBroadcast ()) {
    NS_LOG_LOGIC ("For me (interface broadcast address)");
    return true;
}
```

LOGGING

- Enable log output
  - Method 1: setting the NS_LOG environment variable from the shell
    - Enable logging for everything

```
NS_LOG="*" ./waf --run MySim
```

    - Enable logging for individual components

```
NS_LOG="Ipv4L3Protocol" ./waf --run MySim
```

  - Method 2: use explicit logging statements in our code

```
LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);
```

TRACING

- Generate behavior output for further study
- Using Trace Helpers

```
AsciiTraceHelper ascii;
Ptr<OutputStreamWrapper> stream = ascii.CreateFileStream ("stream.tr");
wifiPhy.EnableAsciiAll (stream);
```

- Using PCAP

```
pointToPoint.EnablePcapAll ("first");
pointToPoint.EnablePcap ("first", p2pNodes.Get (0)->GetId (), 0);
csma.EnablePcap ("second", csmaDevices.Get (0), true);
```

TRACING

- Using the Config Subsystem to Connect to Trace Sources
- Different built-in trace sources exist for different components of the protocol stack and/or the device components

```
void CwndTracer (uint32_t oldval, uint32_t newval) {…}
…

Config::ConnectWithoutContext (
"/NodeList/0/$ns3::TcpL4Protocol/SocketList/0/CongestionWindow",
  MakeCallback (&CwndTracer));
```

- All trace sources
  - www.nsnam.org/doxygen/_trace_source_list.html

TRACING

- Let's trace the RSRP and SINR of all LTE UE devices

```
void reportFunction (Ptr<OutputStreamWrapper> stream, std::string context, uint16_t cellId,
                     uint16_t rnti, double rsrp, double sinr, uint8_t componentCarrierId) {
    *stream->GetStream()     << Simulator::Now().GetMilliSeconds() << "," << context << ",“
                             << cellId << "," << rnti << "," << rsrp << "," << sinr << std::endl;
}

int main (int argc, char *argv[]) {
    …
    AsciiTraceHelper asciiTraceHelper;
    Ptr<OutputStreamWrapper> stream = asciiTraceHelper.CreateFileStream ("traces.csv");
    Config::Connect ("/NodeList/*/DeviceList/*/$ns3::LteUeNetDevice/ComponentCarrierMapUe/
    */LteUePhy/ReportCurrentCellRsrpSinr", MakeBoundCallback (&reportFunction, stream));
```

TRACING

- Result

	A	B	C	D	E	F	G
1	200	2	1	0	2.83644E-10	597974	
2	200	0	13	0	8.66357E-10	1826440	
3	200	1	17	0	1.12498E-10	237167	
4	201	2	1	0	2.83601E-10	597883	
5	201	0	13	0	8.66366E-10	1826460	
6	201	1	17	0	1.12515E-10	237202	
7	202	2	1	0	2.83558E-10	597792	
8	202	0	13	0	8.66376E-10	1826480	
9	202	1	17	0	1.12531E-10	237236	
10	203	2	1	0	2.83514E-10	597701	
11	203	0	13	0	8.66385E-10	1826500	
12	203	1	17	0	1.12547E-10	237270	
13	204	2	1	0	2.83471E-10	597610	
14	204	0	13	0	8.66395E-10	1826520	
15	204	1	17	0	1.12563E-10	237304	
16	205	2	1	0	2.83428E-10	597519	
17	205	0	13	0	8.66405E-10	1826540	
18	205	1	17	0	1.12579E-10	237338	
19	206	2	1	0	2.83385E-10	597428	
20	206	0	13	0	8.66414E-10	1826560	

ATTRIBUTES

- ns-3 provides a more convenient API than pure C++ for the attributes
  - All attributes are exported into a string-based namespace (ns3)
  - Supports regular expressions
  - Allows individual instances of attributes to be manipulated
  - Attributes can be referenced by name without specifying a path through the object namespaces
  - A config class allows users to manipulate the attributes

ATTRIBUTES

- Configure full buffer traffic
  - Using config class for global attribute manipulation for all UDP clients

```
Config::SetDefault ("ns3::UdpClient::MaxPackets", UintegerValue (1000000));
Config::SetDefault ("ns3::UdpClient:: PacketSize", UintegerValue (10 * 1024));
```

  - Using config class for global attribute manipulation for a UDP client

```
Config::Set ("ns3::UdpClient::MaxPackets", UintegerValue (1000000));
Config::Set ("ns3::UdpClient:: PacketSize", UintegerValue (10 * 1024));
```

  - Using attribute manipulation for specific instance

```
dlUdpClientHelper.SetAttribute ("MaxPackets", UintegerValue (1000000));
dlUdpClientHelper.SetAttribute ("PacketSize", UintegerValue (10 * 1024));
```

ATTRIBUTES

- Set and get attribute from a specific object

```
Config::SetAttribute ("/NodeList/0/DeviceList/3/Phy/TxGain", DoubleValue (1.0));
double val;
nodePtr->GetAttribute ("/NodeList/0/DeviceList/3/Phy/TxGain", &val);
```

- Retrieving objects without specifying a path through the object namespaces

```
nodePtr -> GetDevice(3) -> GetObject <LteUeNetDevice> () -> GetImsi ();
```

COMMAND-LINE ATTRIBUTES

- Parse command line arguments to specify values or override default values

```
int main (int argc, char *argv[]) {
CommandLine cmd (__FILE__);
cmd.Parse (argc, argv);
```

- Then execute the simulation overriding the default argument values

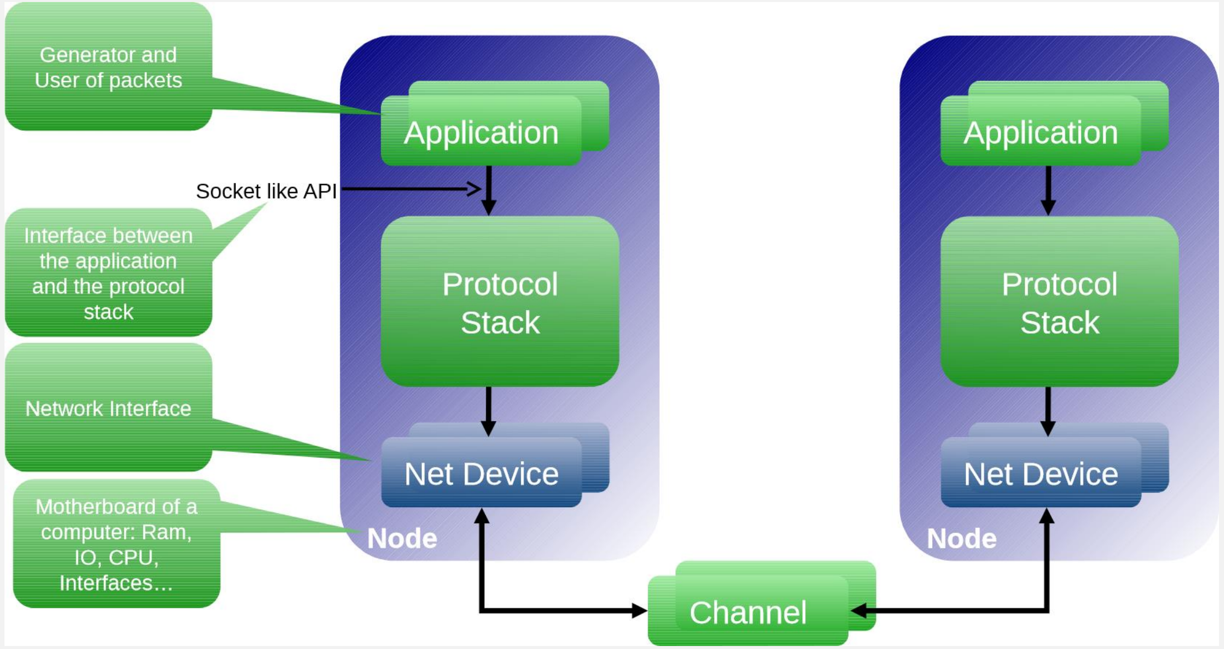
```
./waf --run "MySim --nBlocks=10"
```

SIMULATION ARCHITECTURE

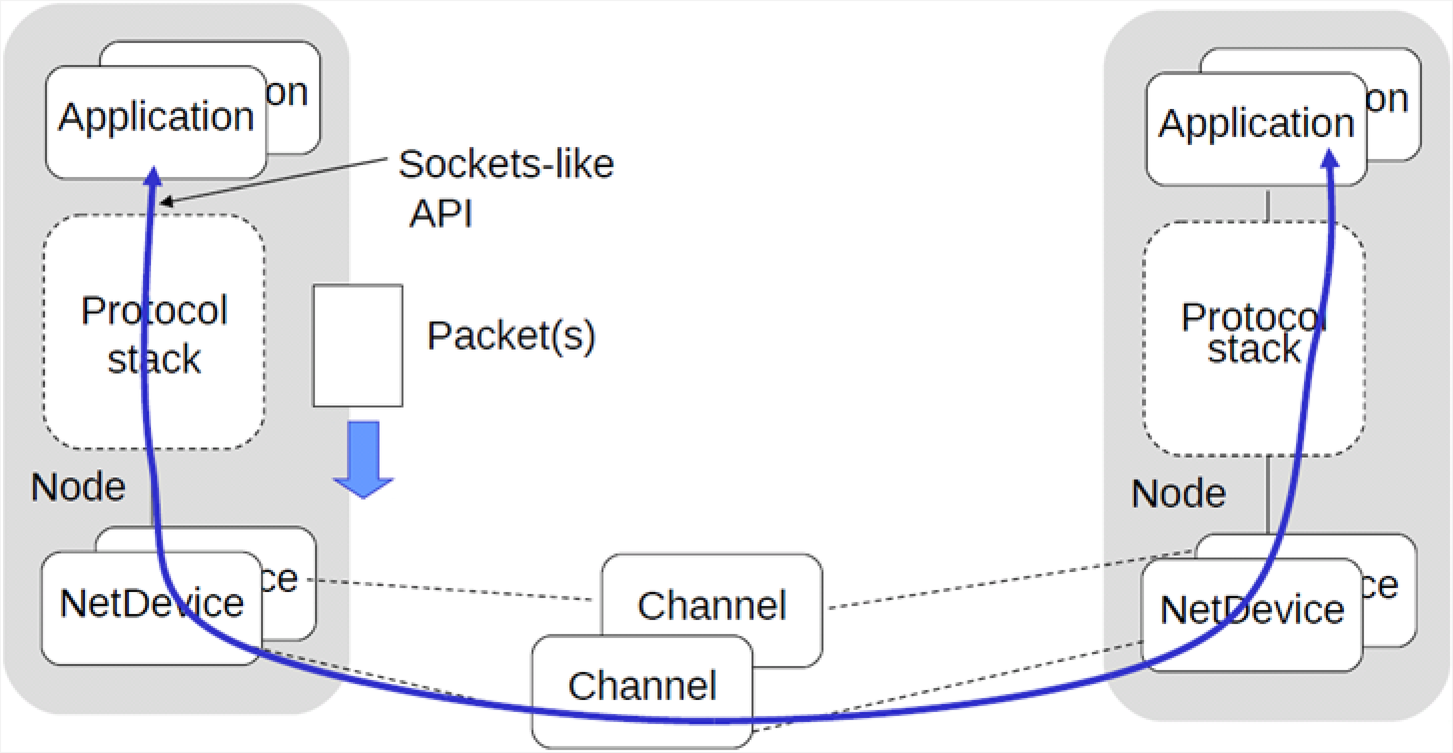
- Basic steps to create a simulation
  1. Create the network topology
  2. Create the data demand on the network
  3. Execute the simulation
  4. Analyze the results

- See “Building Topologies” for more information
  - www.nsnam.org/docs/tutorial/html/building-topologies.html

SIMULATION ARCHITECTURE

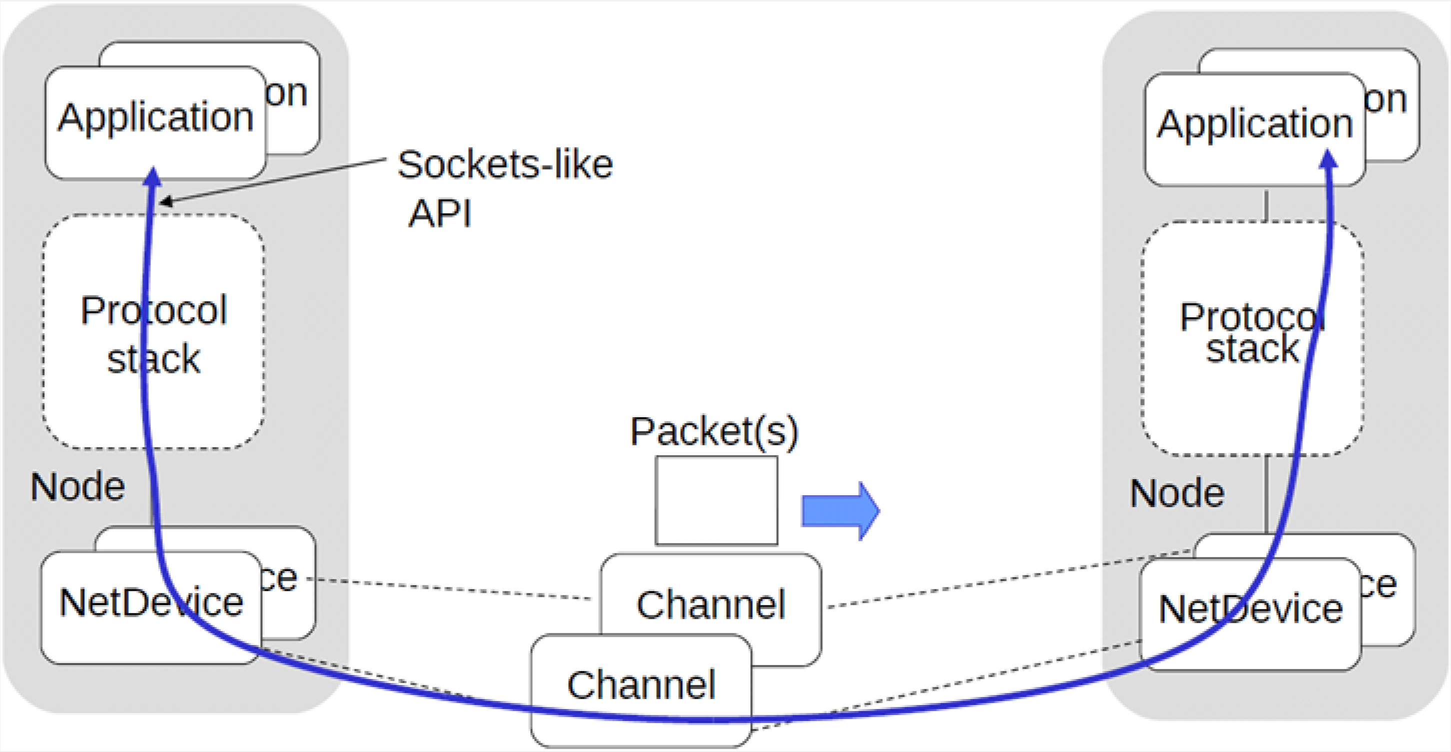


SIMULATION ARCHITECTURE

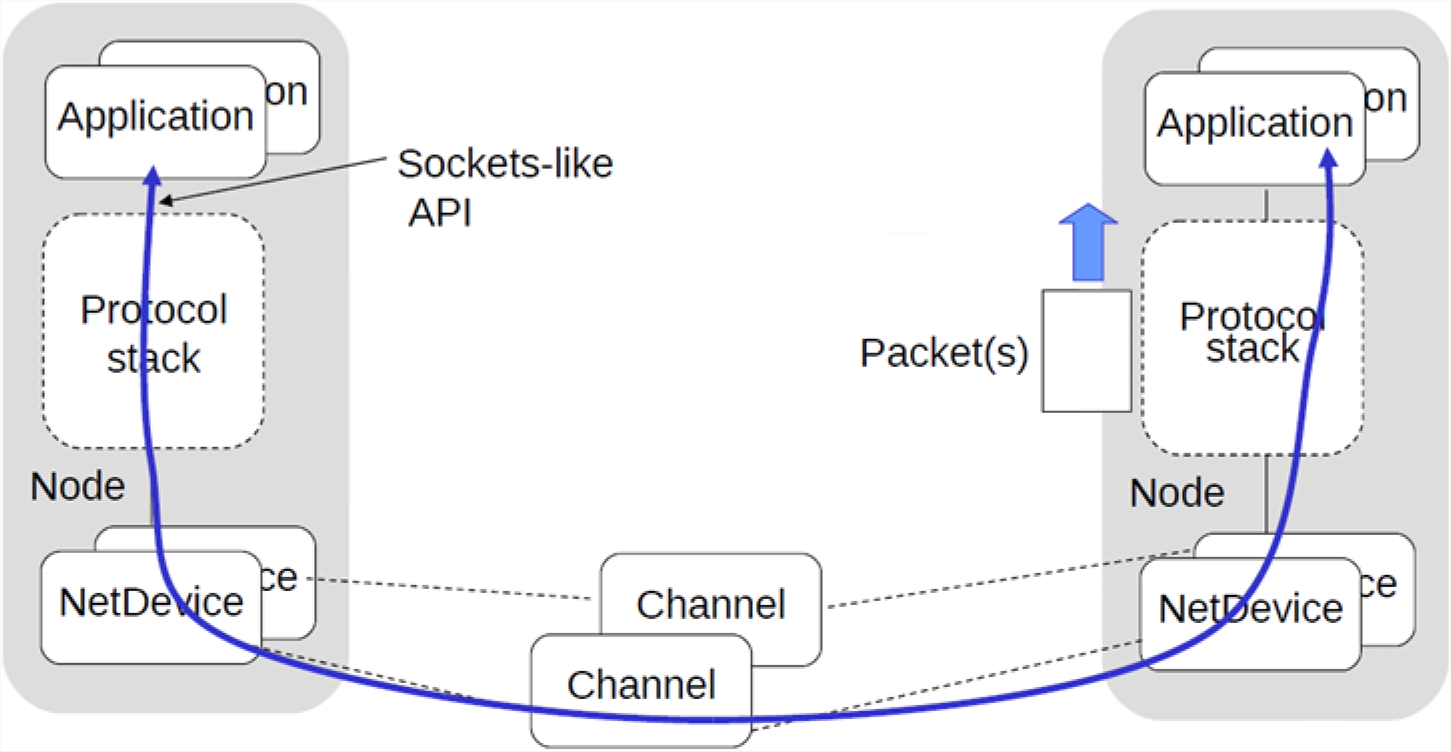


FOGUS
INNOVATIONS & SERVICES

SIMULATION ARCHITECTURE



SIMULATION ARCHITECTURE

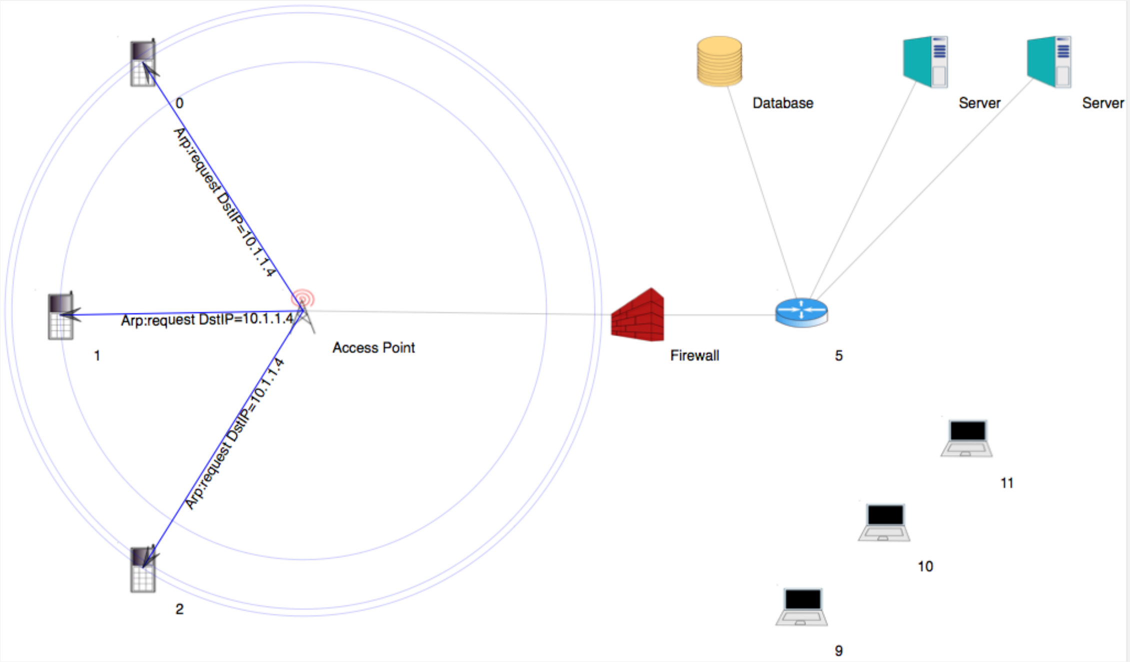


NETWORKING

- ns-3 supports the following major types of networks
  - Bus networks
  - WiFi networks
    - 802.11 DCF (infrastructure & adhoc)
    - 802.11a, 802.11b, 802.11g, 802.11n (2.4 & 5 GHz bands), 802.11ac and 802.11ax (2.4 & 5 GHz bands) physical layers
  - LTE networks
    - Subset of LTE features is supported
    - Some features are not supported yet (e.g. PLMN IDs)

NETWORKING

- Multiple entities are provided for every network type



BUS NETWORKS

- Device
  - A generic device for a Point to Point Network
  - Parameters & objects: queue, data rate, interframe transmission gap, etc.
- Channel
  - A very simple point to point channel
  - Like full duplex RS-232 or RS-422 with null modem and no handshaking
  - There is a state (IDLE, TRANSMITTING) associated with each wire
  - No multi-drop capability (up to two point-to-point net devices are connected)
  - Two "wires" in the channel, one for each end device

BUS NETWORKS

- Let’s create a simple bus network between 2 nodes

```
NodeContainer nodes;
nodes.Create (2);

PointToPointHelper pointToPoint;
pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));

NetDeviceContainer devices = pointToPoint.Install (nodes);
```

- See “Point to Point” for more information
  - www.nsnam.org/docs/models/html/point-to-point.html

WIFI NETWORKS

- Device
  - WiFi capable device with channel, PHY, MAC and remote station information
- Remote Station
  - Holds per-remote-station state
  - Keeps track of association status in an infrastructure network
  - Performs the selection of TX parameters on a per-packet basis
- SSID
  - SSID of WiFi AP device
- MAC
  - MAC model for different types of WiFi devices (ad-hoc, AP, non-AP)
- PHY
  - Physical layer, holds attributes like Frequency (MHz), Channel width & number, TX & RX gains, Transmission power, SNR, Antennas, and more

WIFI NETWORKS

- Let's create a basic WiFi access point

```
NodeContainer wifiApNode;
wifiApNode.Create (1);      // Create 1 access point node object
// Create a channel helper and phy helper, and then create the channel
YansWifiChannelHelper channel = YansWifiChannelHelper::Default ();
YansWifiPhyHelper phy = YansWifiPhyHelper::Default ();
phy.SetChannel (channel.Create ());
// Create a WifiMacHelper, which is reused across STA and AP configurations
WifiMacHelper mac;
// Create a WifiHelper, which will use the above helpers to create Wifi devices.
WifiHelper wifi;
wifi.SetStandard (WIFI_STANDARD_80211n_5GHZ);
// Declare NetDeviceContainers to hold the container returned by the helper
NetDeviceContainer wifiApDevice;
// Perform the installation
mac.SetType ("ns3::ApWifiMac");
wifiApDevice = wifi.Install (phy, mac, wifiApNode);
```

WIFI NETWORKS

- Let’s connect 10 station nodes to the AP

```
NodeContainer wifiStaNode;
wifiStaNode.Create (10);   // Create 10 station node objects

// Perform the installation
mac.SetType ("ns3::StaWifiMac");
NetDeviceContainer wifiStaDevices = wifi.Install (phy, mac, wifiStaNodes);
```

- See “WiFi user” for more information
  - www.nsnam.org/docs/models/html/wifi-user.html

LTE NETWORKS

- EPC
  - MME (Mobility Management Entity)
  - PDN (Packet Data Network)
  - PGW (Packet Data Network Gateway Entity)
  - SGW (Serving Gateway Entity)
  - S11 (S11 Service Access Point & Messages)
  - S1 (S1 Service Access Point & Messages)
- UE devices
  - PHY (models the UE-side of the physical layer)
  - CSG ID (Closed Subscriber Group ID)
  - Radio bearer manager (manages information possessed by the eNB RRC for a UE)

LTE NETWORKS

- eNB devices
  - X2 (interconnecting interface between two eNBs)
  - PHY (models the eNB-side of the physical layer)
  - RRC (LTE Radio Resource Control entity at the eNB)
  - RLC (LTE Radio Link Control entity at the eNB)
  - Component carrier & carrier aggregation
  - CSG ID (Closed Subscriber Group ID)
  - MIMO (Tx Diversity, Spatial Multiplexity Open/Closed Loop, Multi-User)
  - Multiple PLMN IDs are <u>not</u> supported (no MNOs separation)

LTE NETWORKS

- Let's create an LTE eNB

```
// Create an LTE helper
Ptr<LteHelper> lteHelper = CreateObject<LteHelper> ();

NodeContainer enbNodes;
enbNodes.Create (1);        // Create 1 eNB node object

MobilityHelper mobility;    // Create a constant mobility model
mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
mobility.Install (enbNodes);

// Install an LTE protocol stack on the eNB
NetDeviceContainer enbDevs = lteHelper->InstallEnbDevice (enbNodes);
```

LTE NETWORKS

- Let's create 2 LTE Ues and attach them to the eNB

```
NodeContainer ueNodes;
ueNodes.Create (2);                    // Create 2 UE node objects

mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
mobility.Install (ueNodes);            // The mobility object from before

// Install an LTE protocol stack on the UEs
NetDeviceContainer ueDevs = lteHelper->InstallUeDevice (ueNodes);

// Attach the UEs to the eNB
lteHelper->Attach (ueDevs, enbDevs.Get (0));
```

LTE NETWORKS

- Let's activate a data radio bearer between each UE and the eNB it is attached to:

```
enum EpsBearer::Qci q = EpsBearer::GBR_CONV_VOICE;
EpsBearer bearer (q);
lteHelper->ActivateDataRadioBearer (ueDevs, bearer);
```

LTE NETWORKS

- LTE model parameters

```
default ns3::LteHelper::Scheduler "ns3::PfFfMacScheduler"
default ns3::LteHelper::PathlossModel "ns3::FriisSpectrumPropagationLossModel"
default ns3::LteEnbNetDevice::UlBandwidth "25"
default ns3::LteEnbNetDevice::DlBandwidth "25"
default ns3::LteEnbNetDevice::DlEarfcn "100"
default ns3::LteEnbNetDevice::UlEarfcn "18100"
default ns3::LteUePhy::TxPower "10"
default ns3::LteEnbPhy::TxPower "30"
```

- See "LTE user" for more information
  - www.nsnam.org/docs/models/html/lte-user.html

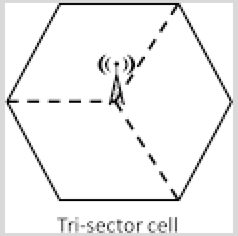
NETWORK LAYOUT & BUILDINGS

- BS & eNBs
  - Constant position in Cartesian (x,y,z) coordinates
  - eNB types (macrocell / femtocell) are defined by the transmission power and the antenna
- UEs
  - Constant or changing position in Cartesian (x,y,z) coordinates
  - UEs can be scheduled to change position according to the mobility model applied
  - A node can have multiple devices (e.g. one for LTE and one for WiFi)
- Buildings
  - 3D solid rectangular defined by 6 coordinates (xMin, xMax, yMin, yMax, zMin, zMax)
  - Can be considered as a grid of rooms (apartments) with constant dimensions per room
  - Not all mobility models are building-aware (devices may "walk through" buildings without explicitly rerouting them)
  - Buildings can not be visualized using the visualizer

NETWORK LAYOUT & BUILDINGS

- LTE 3-sector macro-cell hex grid layer

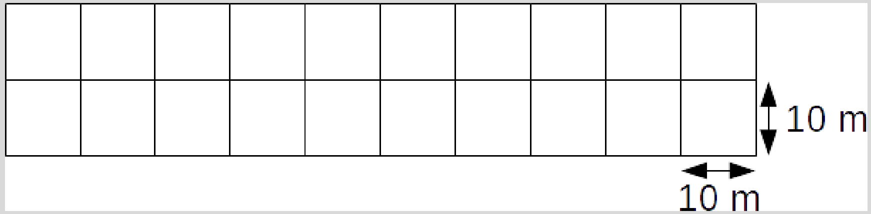
```
Ptr <LteHelper> lteHelper = CreateObject<LteHelper> ();
NodeContainer macroEnbNodes;
macroEnbNodes.Create (numberMacroEnbs);
Ptr<LteHexGridEnbTopologyHelper> lteHexGridEnbTopologyHelper = CreateObject<LteHexGridEnbTopologyHelper> ();
lteHexGridEnbTopologyHelper->SetLteHelper (lteHelper);
lteHexGridEnbTopologyHelper->SetAttribute ("InterSiteDistance", DoubleValue (500)); //500 meters
lteHexGridEnbTopologyHelper->SetAttribute ("MinX", DoubleValue (0)); //First macrocell in (x,y,z)=(0,0,30)
lteHexGridEnbTopologyHelper->SetAttribute ("GridWidth", UintegerValue (1));
Config::SetDefault ("ns3::LteEnbPhy::TxPower", DoubleValue (46)); //46 dBm
lteHelper->SetEnbAntennaModelType ("ns3::ParabolicAntennaModel");
lteHelper->SetEnbAntennaModelAttribute ("Beamwidth", DoubleValue (70));
lteHelper->SetEnbAntennaModelAttribute ("MaxAttenuation", DoubleValue (20));
NetDeviceContainer macroEnbDevs = lteHexGridEnbTopologyHelper->SetPositionAndInstallEnbDevice (macroEnbNodes);
```



NETWORK LAYOUT & BUILDINGS

- Building creation inspired by dual stripe scenario

```
Ptr<GridBuildingAllocator>  gridBuildingAllocator = CreateObject<GridBuildingAllocator> ();
gridBuildingAllocator->SetAttribute ("GridWidth", UintegerValue (1));
gridBuildingAllocator->SetAttribute ("LengthX", DoubleValue (10*nApartmentsX));
gridBuildingAllocator->SetAttribute ("LengthY", DoubleValue (10*nApartmentsY));
gridBuildingAllocator->SetAttribute ("Height", DoubleValue (3*nFloors));
gridBuildingAllocator->SetBuildingAttribute ("NRoomsX", UintegerValue (nApartmentsX));
gridBuildingAllocator->SetBuildingAttribute ("NRoomsY", UintegerValue (nApartmentsY));
gridBuildingAllocator->SetBuildingAttribute ("NFloors", UintegerValue (nFloors));
gridBuildingAllocator->SetAttribute ("MinX", DoubleValue (0));
gridBuildingAllocator->SetAttribute ("MinY", DoubleValue (0));
BuildingContainer buildings = gridBuildingAllocator->Create (1);
Ptr<Building> building = buildings.Get(0);
building->SetBuildingType (Building::Residential);
building->SetExtWallsType (Building::ConcreteWithoutWindows );
```



MOBILITY

- The mobility support in ns-3 includes:
  - A set of mobility models which are used to track and maintain the current cartesian position and speed of an object.
  - A "course change notifier" trace source which can be used to register listeners to the course changes of a mobility model
  - A number of helper classes which are used to place nodes and setup mobility models (including parsers for some mobility definition formats).
- The base class for coordinates, velocities and accelerations is ns3::Vector
- There are also some additional related structures used to support mobility models.
  - Rectangle
  - Box
  - Waypoint

MOBILITY

- ConstantPosition
- ConstantVelocity
- ConstantAcceleration
- GaussMarkov
- Hierarchical
- RandomDirection2D
- RandomWalk2D
- RandomWaypoint
- SteadyStateRandomWaypoint
- Waypoint

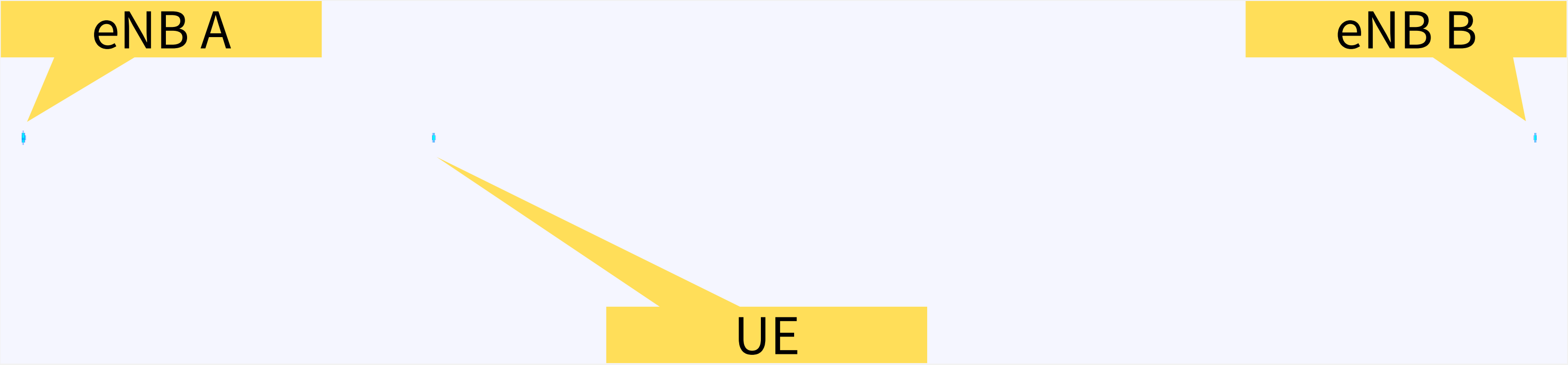
MOBILITY

- Let’s create a constant velocity model

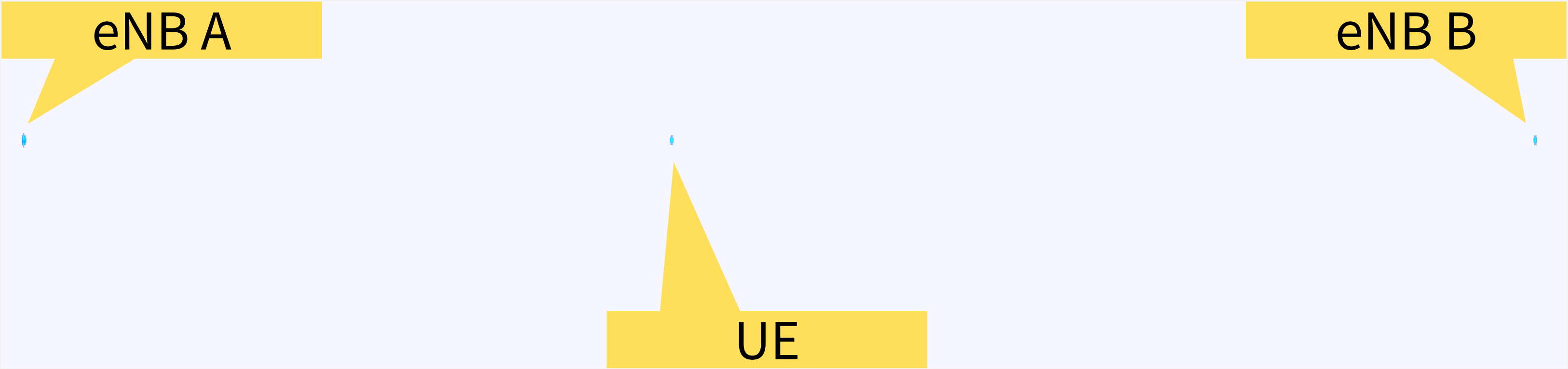
```
MobilityHelper mobility;
mobility.SetMobilityModel ("ns3::ConstantVelocityMobilityModel");
mobility.Install (ueNodes);
// Set initial position at (0,0,0)
ueNodes.Get(0) -> GetObject<MobilityModel> () -> SetPosition (Vector (0, 0, 0));
// Set constant velocity at 1 m/s at both X and Y axis
ueNodes.Get (0) -> GetObject<ConstantVelocityMobilityModel> () -> SetVelocity (Vector (1, 1, 0));
```

MOBILITY

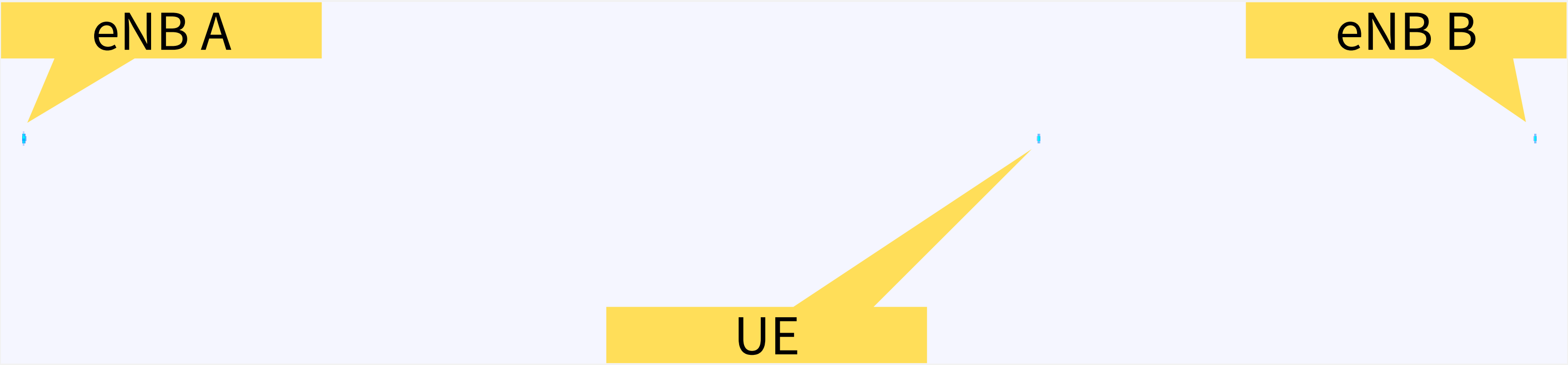
MOBILITY



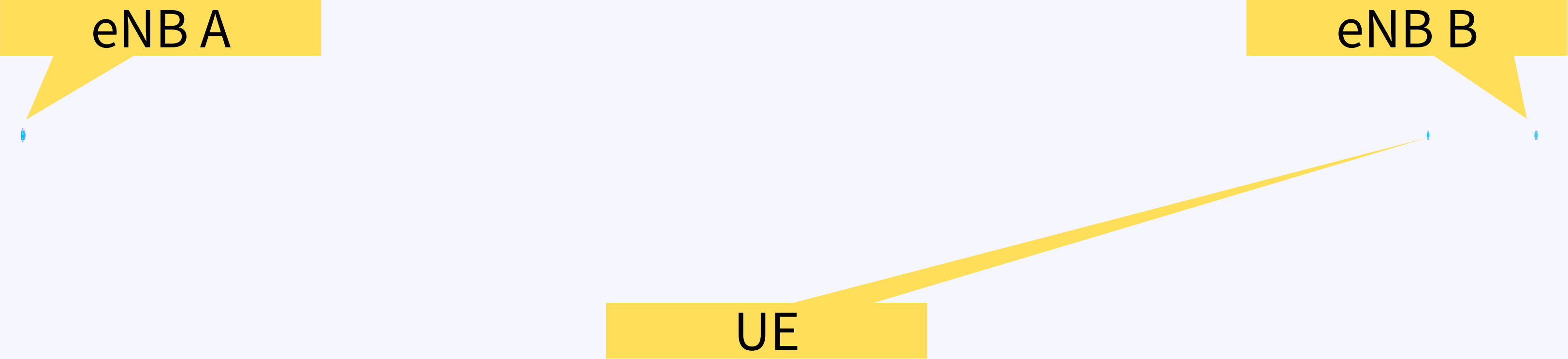
MOBILITY



MOBILITY



MOBILITY



DISTRIBUTIONS

- Uniform through UniformRandomVariable class
- Normal through NormalRandomVariable class
- Constant through ConstantRandomVariable class
- Exponential through ExponentialRandomVariable class
- Gamma through GammaRandomVariable class

- … and many more as documented at www.nsnam.org/docs/manual/html/random-variables.html

DISTRIBUTIONS

- Example of using variables generated from a Standard Normal Distribution

```
Ptr<NormalRandomVariable> var = CreateObject<NormalRandomVariable> ();
var>SetAttribute ("Mean", DoubleValue (0));
var->SetAttribute ("Variance", DoubleValue (1));
uint32_t intValue = var->GetInteger ();
double doubleValue = var->GetValue ();
```

- Example of using variables generated from a Standard Uniform Distribution

```
Ptr<UniformRandomVariable> var = CreateObject<UniformRandomVariable> ();
var>SetAttribute ("Min", DoubleValue (0));
var->SetAttribute ("Max", DoubleValue (1));
uint32_t intValue = var->GetInteger ();
double doubleValue = var->GetValue ();
```

HANDOVER

- LTE UE makes radio measurement in two stages: **idle** and **connected**
- In connected state mode with UE mobility, starting radio measurement and making a handover decision is done by defined condition which is called "Event"
- Handover algorithm receives measurement reports from an eNB RRC instance
- Tells the eNB RRC instance when to do a handover
- Only 3 handover algorithms are implemented
  - A3RsrpHandoverAlgorithm
  - *A2A4RsrqHandoverAlgorithm*
  - *NoOpHandoverAlgorithm*

HANDOVER

- A2A4 RSRQ handover algorithm (2 events)
  - A2: serving cell's RSRQ becomes worse than threshold
  - A4: a neighboring cell's RSRQ is higher than the serving cell's RSRQ by a certain offset

```
lteHelper->SetHandoverAlgorithmType ("ns3::A2A4RsrqHandoverAlgorithm");
lteHelper->SetHandoverAlgorithmAttribute ("ServingCellThreshold", UintegerValue (30));
lteHelper->SetHandoverAlgorithmAttribute ("NeighbourCellOffset", UintegerValue (1));
```

FOGUS
INNOVATIONS & SERVICES

HANDOVER

- A3 RSRQ handover algorithm (1 event)
  - A3: a neighbor cell's RSRP is better than the serving cell's RSRP

```
lteHelper->SetHandoverAlgorithmType ("ns3::A3RsrpHandoverAlgorithm");
lteHelper->SetHandoverAlgorithmAttribute ("Hysteresis", DoubleValue (3.0));
lteHelper->SetHandoverAlgorithmAttribute ("TimeToTrigger", TimeValue (MilliSeconds (256)));
```

- No Operation handover algorithm
  - No handover!!!


```
lteHelper->SetHandoverAlgorithmType ("ns3::NoOpHandoverAlgorithm");
```

RADIO PROPAGATION MODELS

- Models propagation loss and propagation delay
- Propagation loss models calculate the Rx signal power considering the Tx signal power and the mutual Rx and Tx antennas positions
- A propagation loss model can be "chained" to another one, making a list
- The final Rx power takes into account all the chained models
- One can use a slow fading and a fast fading model (for example), or model separately different fading effects

RADIO PROPAGATION MODELS

- FriisPropagationLossModel
- JakesPropagationLossModel
- LogDistancePropagationLossModel
- NakagamiPropagationLossModel
- OhBuildingsPropagationLossModel
- HybridBuildingsPropagationLossModel

- … and many more as documented at www.nsnam.org/docs/models/html/propagation.html

RADIO PROPAGATION MODELS

- Propagation in open areas

```
lteHelper->SetAttribute ("PathlossModel", StringValue
("ns3::FriisSpectrumPropagationLossModel"));
lteHelper->SetSchedulerType ("ns3::RrFfMacScheduler");
lteHelper->SetHandoverAlgorithmType ("ns3::A2A4RsrqHandoverAlgorithm");
lteHelper->SetHandoverAlgorithmAttribute ("ServingCellThreshold", UintegerValue (30));
lteHelper->SetHandoverAlgorithmAttribute ("NeighbourCellOffset", UintegerValue (1));
```

RADIO PROPAGATION MODELS

- Propagation in urban areas

```
Ptr <LteHelper> lteHelper = CreateObject<LteHelper> ();
lteHelper->SetAttribute ("PathlossModel", StringValue ("ns3::HybridBuildingsPropagationLossModel"));
lteHelper->SetPathlossModelAttribute ("ShadowSigmaExtWalls", DoubleValue (0));
lteHelper->SetPathlossModelAttribute ("ShadowSigmaOutdoor", DoubleValue (1));
lteHelper->SetPathlossModelAttribute ("ShadowSigmaIndoor", DoubleValue (1.5));
lteHelper->SetPathlossModelAttribute ("Los2NlosThr", DoubleValue (1e6));
lteHelper->SetSpectrumChannelType ("ns3::MultiModelSpectrumChannel");
```

TRAFFIC MODELS

- Manage and manipulate the transmission of packets
- A traffic model sits in between the network devices and any network protocol (e.g. IP)
- In charge of processing packets and performing actions on them
- Scheduling, dropping, marking, policing, etc.
- Both IN (Rx) and OUT (Tx) directions can be managed

TRAFFIC MODELS

- QueueDisc (Queue disciplines)
- RedQueueDisc (Random Early Detection queue discipline)
- FifoQueueDisc (FIFO queue)
- PfifoFastQueueDisc (3-band FIFO queue)
- PrioQueueDisc (Priority queue disc)
- MqQueueDisc (Multi-queue disc)

- ... and many more as documented at
  - www.nsnam.org/doxygen/group__traffic-control.html

TRAFFIC MODELS

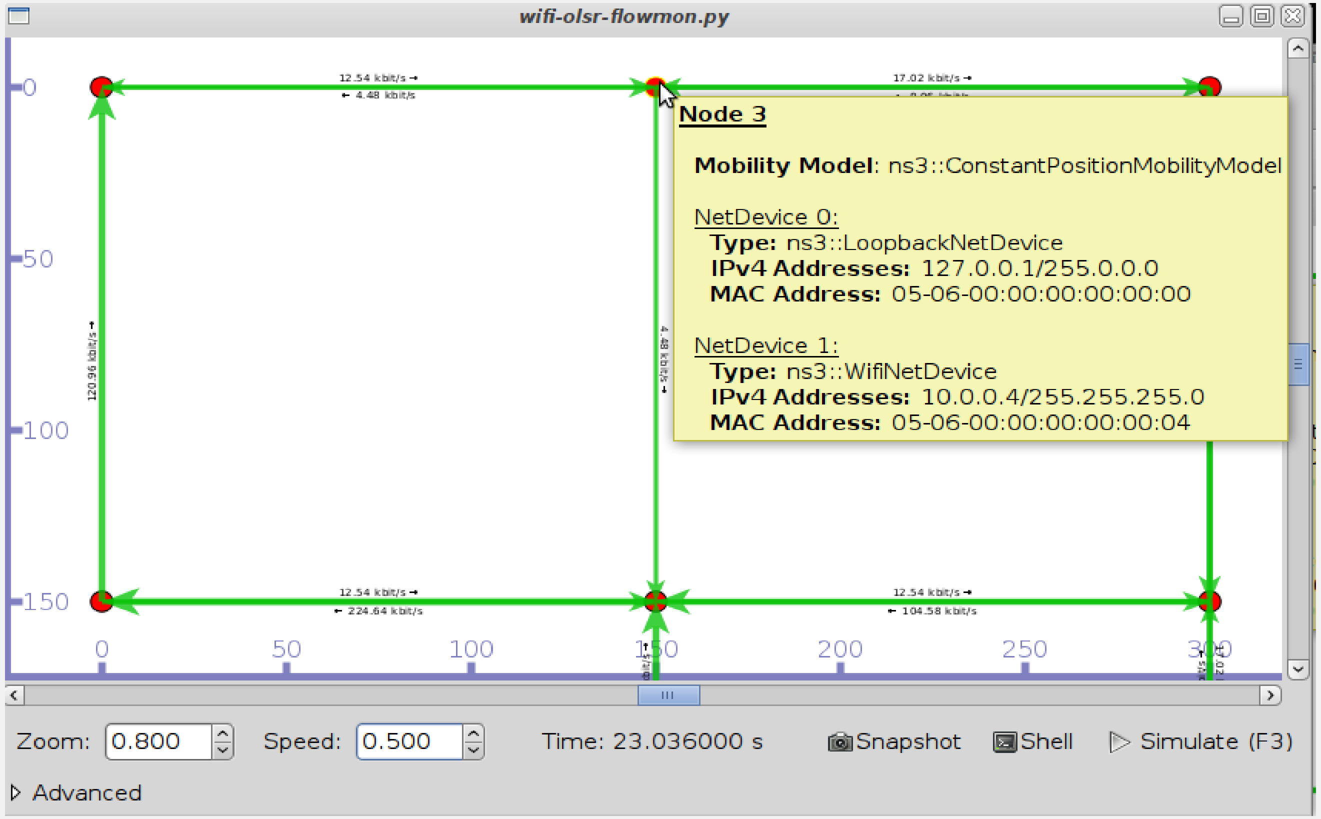
- Let’s install RedQueueDisc traffic on a device

```
TrafficControlHelper tch;
tch.SetRootQueueDisc ("ns3::RedQueueDisc");
QueueDiscContainer qdiscs = tch.Install (devices);
Ptr<QueueDisc> q = qdiscs.Get (1);
q ->TraceConnectWithoutContext ("PacketsInQueue", MakeCallback (&TcPacketsInQueueTrace));
Config::ConnectWithoutContext
("/NodeList/1/$ns3::TrafficControlLayer/RootQueueDiscList/0/SojournTime",
MakeCallback (&SojournTimeTrace));
Ptr<NetDevice> nd = devices.Get (0);
Ptr<PointToPointNetDevice> ptpnd = DynamicCast<PointToPointNetDevice> (nd);
Ptr<Queue<Packet> > queue = ptpnd->GetQueue ();
queue->TraceConnectWithoutContext ("PacketsInQueue", MakeCallback (&DevicePacketsInQueueTrace));
```

VISUALIZER

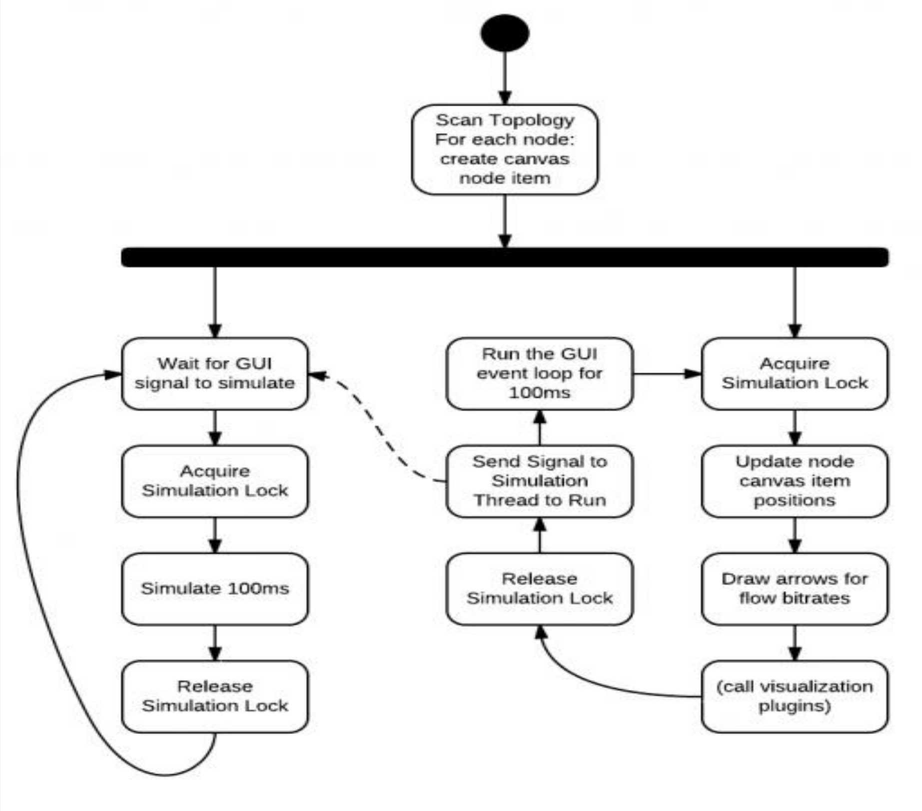
- NS-3 PyViz is a live simulation visualizer integrated into NS-3 since v3.10
- It uses no trace files and can be most useful for debugging purposes, i.e. to figure out if mobility models are what you expect, where packets are being dropped, etc.
- Built-in interactive python console that can be used to debug the state of the running objects
- Works both with Python and pure C++ simulations
- LTE devices do not support visualizer yet
- Visualizer can not be used with simulations that require emulation (EmuNetDevice, TapNetDevice) or real-time scheduler (RealTimeSimulator)

VISUALIZER


wifi-olsr-flowmon.py
0
50
100
150
0
50
100
150
200
250
300
Node 3
Mobility Model: ns3::ConstantPositionMobilityModel
NetDevice 0:
Type: ns3::LoopbackNetDevice
IPv4 Addresses: 127.0.0.1/255.0.0.0
MAC Address: 05-06-00:00:00:00:00:00
NetDevice 1:
Type: ns3::WifiNetDevice
IPv4 Addresses: 10.0.0.4/255.255.255.0
MAC Address: 05-06-00:00:00:00:00:04
Zoom: 0.800
Speed: 0.500
Time: 23.036000 s
Snapshot
Shell
Simulate (F3)
Advanced

VISUALIZER

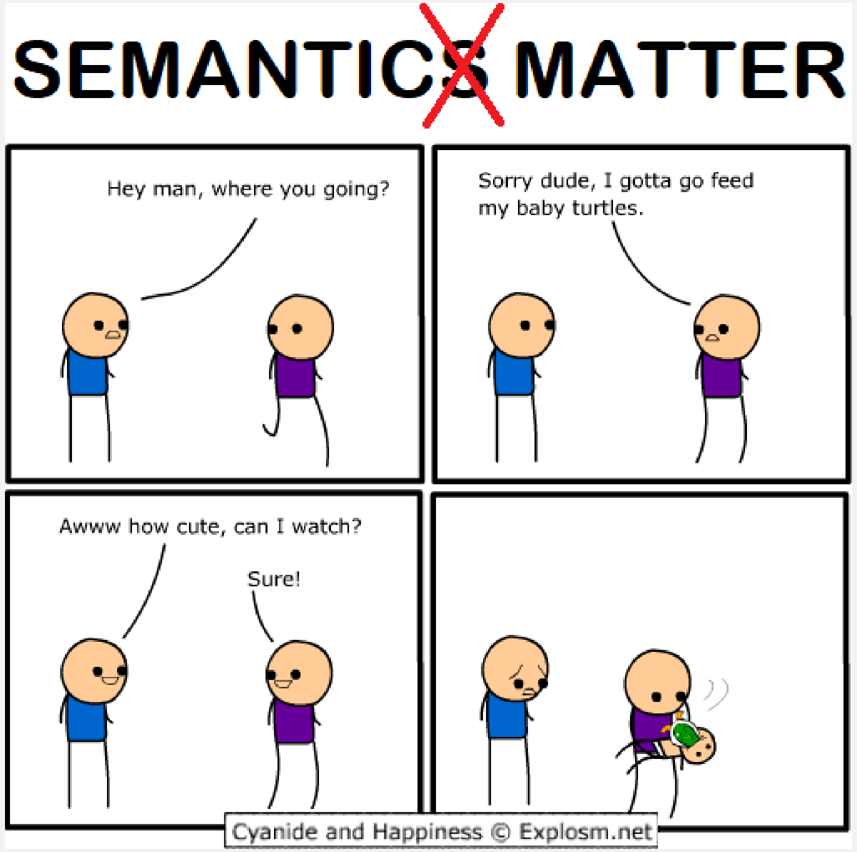
- PyViz runs two threads:
  - In the main thread, the GUI is kept updated
  - In a separate thread the simulation is run
  - Updates happen in slices of 100 ms



REFERENCES

- Documentation
  - www.nsnam.org/documentation
- API documentation (using Doxygen)
  - www.nsnam.org/doxygen
- Models documentation (using Doxygen)
  - www.nsnam.org/docs/models/html
- Wiki
  - www.nsnam.org/wiki
- Installation instructions
  - www.nsnam.org/wiki/Installation
- Q&A Google Group
  - https://groups.google.com/g/ns-3-users
- PyViz
  - www.nsnam.org/wiki/PyViz
- Code repository
  - www.gitlab.com/nsnam

THANK YOU



SEMANTICS MATTER
Hey man, where you going?
Sorry dude, I gotta go feed my baby turtles.
Awww how cute, can I watch?
Sure!
Cyanide and Happiness © Explosm.net