Introduction to NS-3 Part - I

UniS PGSDP Workshop
Konstantinos Katsaros
PhD Student
K.Katsaros@surrey.ac.uk

Overview

- NS-3 Vs. NS-2
- NS3 Features
- Download – Build – Use
- Current Modules
- Simulation Basics
- Abstractions – Simple example walkthrough
- Attributes
- Tracing
- Callbacks
- Examples-Lab
- Resources

NS-3 Vs. NS-2

1. **NS-2 uses OTcl as its scripting environment**
2. **NS-3 uses C++ programs or python scripts to define simulations.**

   Simulation programs are C++ executables or Python programs

   Python is often a glue language, in practice

   NS-3 is a GNU GPLv2-licensed project

   NS-3 is not backwards-compatible with NS-2

3. **Some ns-2 models that are mostly written in C++ have already been ported to ns-3. OTcl-based models can not be ported "as is". Need to re-write.**

NS3 Features - 1

- Scalability features
  - Packets can have ”virtual zero bytes” (or dummy bytes)
    - For dummy application data that we don't care about
    - No memory is allocated for virtual zero bytes
    - Reduces the memory footprint of the simulation
  - Nodes have optional features (sort of aspect-oriented programming (AOP))
    - No memory waste in IPv4 stack for nodes that don't need it
    - Mobility model may not be needed
      - E.g. wired netdevices do not need to know the node position at all
    - New features can be easily added in the future
      - For example, energy models

NS3 Features - 2

- Cross-layer features
  - Packet Tags
    - Small units of information attached to packets
  - Tracing
    - Allow to report events across non-contiguous layers
- Real world integration features
  - Packets can be saved to PCAP files, in a real format
    - Many tools can read PCAP files, e.g. Wireshark
  - Real-time scheduler
    - Simulation events synchronized to ”wall clock time”
  - ”Network Simulation Cradle”
    - Run Linux Kernel TCP/IP stack under simulation
  - POSIX Emulation (experimental)
    - Run unmodified POSIX programs under simulation
    - Running routing daemons on NS-3

Three Steps of NS3
Download - Install - Run

- **Cross platform (limited support for Windows)**
- **First download & install ALL dependencies**
- **Simple download and build project**

  Current stable version ns-3.13

  Next version planned for April '12

- **Run example!!**

Installation Wiki

Current Modules of NS3

Aodv	Applications	Bridge
Click	Config-store	Core
Csma	Csma-layout	Dsdv
Emu	Energy	Flow-monitor
Internet	Lte	Mesh
Mobility	Mpi	Netanim
Network	Nix-vector-routing	Olsr
Openflow	Point-to-point	Point-to-point-layout
Propagation	Spectrum	Stats
Tap-bridge	Test	Tools
Topology-read	Uan	Virtual-net-device
Visualizer	Wifi	Wimax

Model Library (PDF)

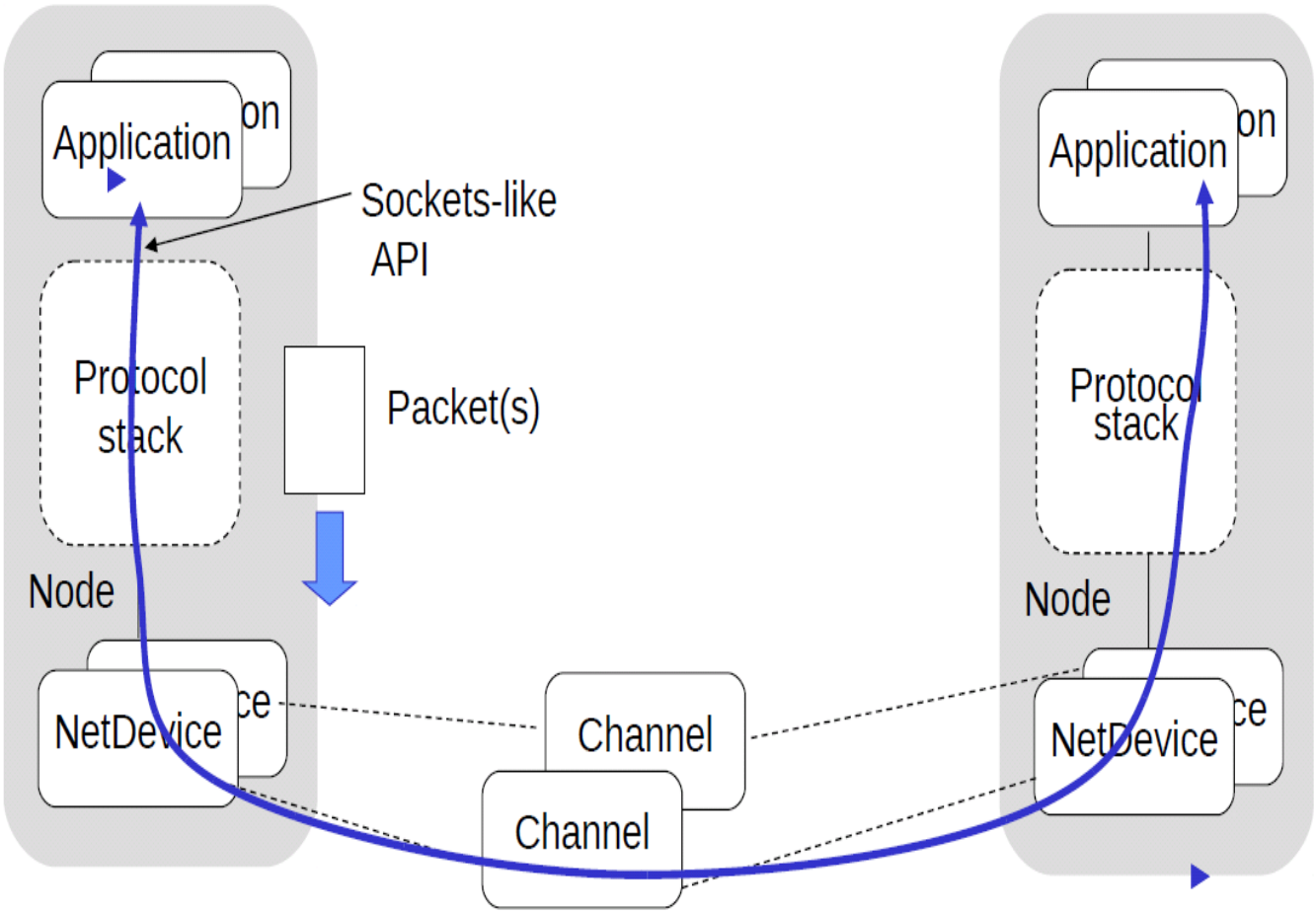
Simulation Basics

- **Simulation time moves in discrete jumps from event to event**
- **C++ functions schedule events to occur at specific simulation times**
- **A simulation scheduler orders the event execution**
- **Simulation::Run() gets it all started**
- **Simulation stops at specific time or when events end**

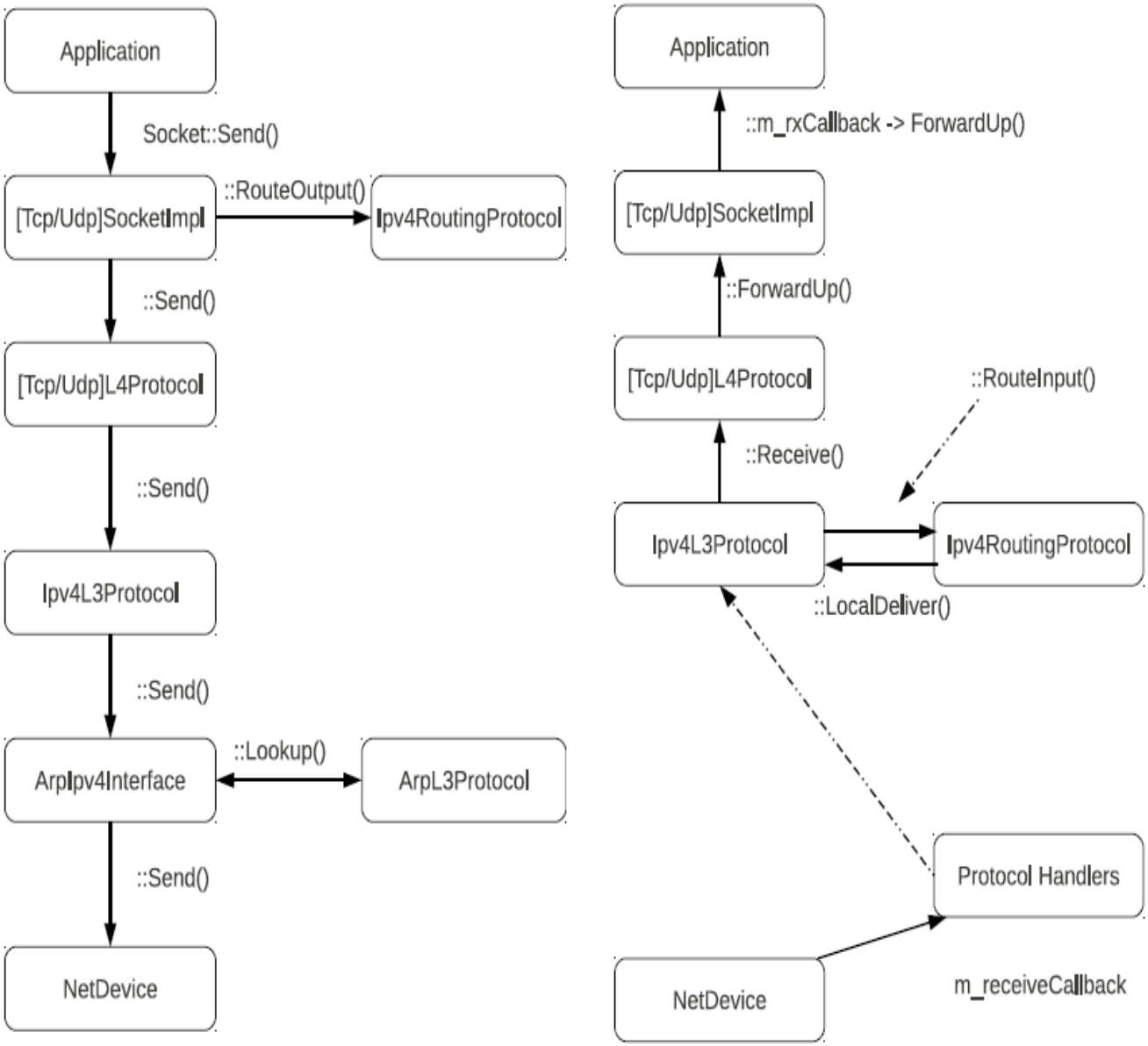
Abstractions

- **Node**
- **Application**
- **Channel**
- **NetDevice**
- **Packet**
- **Topology Helpers – aggregate functionality of modules to make common operations easier than using the low-level API**
  Consists of:
  - **container objects**
  - **helper classes**

Example - Conceptual



Example - NS3 Implementation



Example - NS3 Script

Starting your script with this line ensures that emacs editor will be able to indent your code correctly. The following lines ensure that your code is licensed under the GPLv2.

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil;
-*- */
// GPLv2 Licence ...
```

Include the proper header files. For simulation scripts, you can aggregate them with modules, but when developing your module, you have to include the specific header file (not the aggregated)

```
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/applications-module.h"
...
```

Use the ns3 project namespace

```
using namespace ns3;
...
```

enable and disable console message logging by reference to the name

```
NS_LOG_COMPONENT_DEFINE ("FirstScriptExample");

int main (int argc, char *argv[])
{
  LogComponentEnable ("UdpEchoClientApplication",
LOG_LEVEL_INFO);
  LogComponentEnable ("UdpEchoServerApplication",
LOG_LEVEL_INFO);
...
```

Topology Configuration

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

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

Set up Internet Stack

```
InternetStackHelper stack;
stack.Install (nodes);

Ipv4AddressHelper address;
address.SetBase ("10.1.1.0", "255.255.255.0");
Ipv4InterfaceContainer interfaces = address.Assign
(devices);
```

Set up applications

```
UdpEchoServerHelper echoServer (9);

ApplicationContainer serverApps = echoServer.Install
(nodes.Get (1));
serverApps.Start (Seconds (1.0));
serverApps.Stop (Seconds (10.0));
UdpEchoClientHelper echoClient (interfaces.GetAddress (1),
9);
echoClient.SetAttribute ("MaxPackets", UintegerValue (1));
echoClient.SetAttribute ("Interval", TimeValue (Seconds
(1.0)));
echoClient.SetAttribute ("PacketSize", UintegerValue
(1024));
ApplicationContainer clientApps = echoClient.Install
(nodes.Get (0));
clientApps.Start (Seconds (2.0));
clientApps.Stop (Seconds (10.0));
...
```

Run your scenario

```
  Simulator::Run ();
  Simulator::Destroy ();
  return 0;
}
```

Running Example

All simulation scenarions should be run under /scratch folder.

e.g., copy the first tutorial into the scratch folder, and re-build the project

```
% cp examples/tutorial/first.cc scratch/myfirst.cc
% ./waf
```

Now run the scenario

```
% ./waf --run /scratch/myfirst
% Waf: Entering directory ‘/scratch/ns3-workshop/ns-
allinone-3.13/ns-3.13/build’
Waf: Leaving directory ‘/scratch/ns3-workshop/ns-allinone-
3.13/ns-3.13/build’
’build’ finished successfully (1.218s)
Sent 1024 bytes to 10.1.1.2
Received 1024 bytes from 10.1.1.1
Received 1024 bytes from 10.1.1.2
```

Attributes

Problem: Researchers want to identify all of the values affecting the results of their simulations and configure them easily

- ns-3 solution: Each ns-3 object has a set of attributes:
  - A name, help text
  - A type
  - An initial value
- Control all simulation parameters for static objects
- Dump and read them all in configuration files
- Visualize them in a GUI
- Makes it easy to verify the parameters of a simulation
- An Attribute represents a value in our system
- An Attribute can be connected to an underlying variable or function
  - e.g., `TcpSocket::m_cwnd;`
  - or a trace source

How to handle attributes

- The traditional C++ way:
  - export attributes as part of a class's public API
  - walk pointer chains (and iterators, when needed) to find what you need
  - use static variables for defaults
- The attribute system provides a more convenient API to the user to do these things
- Attributes are exported into a string-based namespace, with filesystem-like paths
  - namespace supports regular expressions
- Attributes also can be used without the paths
  - e.g., "`ns3::WifiPhy::TxGain`"
- A Config class allows users to manipulate the attributes
- Set or get the current value of a variable
  Here, one needs the path in the namespace to the right instance of the object

```
Config::SetAttribute("/NodeList/5/DeviceList/3/Phy
/TxGain", DoubleValue(1.0));
DoubleValue d;
nodePtr->GetAttribute("/NodeList/5/NetDevice/3/Phy
/TxGain", d);
```

- Users can get Ptrs to instances also, and Ptrs to trace sources, in the same way

How to manipulate attributes

- Individual object attributes often derive from default values
  - Setting the default value will affect all subsequently created objects
  - Ability to configure attributes on a per-object basis
- Set the default value of an attribute from the command-line:
  - `CommandLine cmd;`
  - `cmd.Parse (argc, argv);`
- Set the default value of an attribute with `NS_ATTRIBUTE_DEFAULT`
- Set the default value of an attribute in C++
  - ```
    Config::SetDefault
    ("ns3::Ipv4L3Protocol::CalcChecksum", BooleanValue
    (true));
    ```
- Set an attribute directly on a specic object
  - ```
    Ptr<CsmaChannel> csmaChannel = ...;
    csmaChannel->SetAttribute ("DataRate", StringValue
    ("5Mbps"));
    ```

Tracing System

- Simulator provides a set of pre-configured trace sources
  - Users may edit the core to add their own
- Users provide trace sinks and attach to the trace source
  - Simulator core provides a few examples for common cases
- Multiple trace sources can connect to a trace sink
- Helper classes hide the tracing details from the user, for simple trace types
  - ascii or pcap traces of devices

```
std::ofstream ascii;
ascii.open ("wns3-helper.tr");
CsmaHelper::EnableAsciiAll (ascii);
CsmaHelper::EnablePcapAll ("wns3-helper");
YansWifiPhyHelper::EnablePcapAll ("wsn3-helper");
```

Multiple Levels of Tracing

- **Highest-level**: Use built-in trace sources and sinks and hook a trace file to them
  - Setting the default value will affect all subsequently created objects
  - Ability to configure attributes on a per-object basis

```
// Also configure some tcpdump traces each interface
will be traced.
// The output files will be named:
// simple-point-to-point.pcap-<nodeId>-<interfaceId>
// and can be read by the "tcpdump -r" command (use
"-// tt" option to display timestamps correctly)

PcapTrace pcaptrace ("simple-point-to-point.pcap");
pcaptrace.TraceAllIp ();
```

- **Mid-level**: Customize trace source/sink behaviour using the tracing namespace

```
void
PcapTrace::TraceAllIp (void)
{
  NodeList::Connect ("/nodes/*/ipv4/(tx|rx)",
  MakeCallback (&PcapTrace::LogIp, this));
}
```

- **Low-level**: Add trace sources to the tracing namespace

```
Config::Connect ("/NodeList/.../Source",
MakeCallback (&ConfigTest::ChangeNotification, this));
```

NS3 Callbacks

- NS3 Callback class implements **function objects**
  - Type safe callbacks, manipulated by value
  - Used for example in sockets and tracing

- Example:

```
Class MyClass {
public:
   double MyFunc (int x, float y) {
     return double (x + y) / 2;
   }
[...]

Callback<double, int, float> cb1;
MyClass myobj;
cb1 = MakeCallback(&MyClass::MyFunc, &myobj);
double result = cb1 (2,3); // result receives 2.5
```

Lab 1: Simple Client/Server

Level: Introductory

Expected learning outcome: NS-3 simulation basics. Basic client server paradigm. Reading pcap traces.

Experiment:

1. Create a simple topology of two nodes (Node1, Node2) separated by a point-to-point link.
2. Setup a UdpClient on one Node1 and a UdpServer on Node2. Let it be of a fixed data rate Rate1.
3. Start the client application, and measure end to end throughput whilst varying the latency of the link
4. Now add another client application to Node1 and a server instance to Node2. What do you need to configure to ensure that there is no conflict?
5. Repeat step 3 with the extra client and server application instances. Show screenshots of pcap traces which indicate that delivery is made to the appropriate server instance.

Example Solution

Lab 2: TCP Variants

Level: Introductory

Expected learning outcome: TCP internals and the difference between each of the variants. NS-3 tracing mechanism.

Experiment:

Create a simple dumbbell topology, two client Node1 and Node2 on the left side of the dumbbell and server nodes Node3 and Node4 on the right side of the dumbbell. Let Node5 and Node6 form the bridge of the dumbbell. Use point to point links.

Install a TCP socket instance on Node1 that will connect to Node3.

Install a UDP socket instance on Node2 that will connect to Node4.

Start the TCP application at time 1s.

Start the UDP application at time 20s at rate Rate1 such that it clogs half the dumbbell bridge's link capacity.

Increase the UDP application's rate at time 30s to rate Rate2 such that it clogs the whole of the dumbbell bridge's capacity.

Use the ns-3 tracing mechanism to record changes in congestion window size of the TCP instance over time. Use gnuplot/matplotlib to visualise plots of cwnd vs time.

Mark points of fast recovery and slow start in the graphs.

Perform the above experiment for TCP variants Tahoe, Reno and New Reno, all of which are available with ns-3

Example Solution

Lab 3: TCP and Router Queues

Level: Introductory

Expected learning outcome: Queues, packet drops and their effect on congestion window size

Experiment:

As in previous exercise, Create a simple dumbbell topology, two client Node1 and Node2 on the left side of the dumbbell and server nodes Node3 and Node4 on the right side of the dumbbell. Let Node5 and Node6 form the bridge of the dumbbell. Use point to point links

Add drop tail queues of size QueueSize5 and QueueSize6 to Node5 and Node6, respectively

Install a TCP socket instance on Node1 that will connect to Node3

Install a TCP socket instance on Node2 that will connect to Node3.

Install a TCP socket instance on Node2 that will connect to Node4

Start Node1--Node3 flow at time 1s, then measure it's throughput. How long does it take to fill link's entire capacity?

Start Node2--Node3 and Node2--Node4 flows at time 15s, measure their throughput

Measure packet loss and cwnd size, and plot graphs throughput/time, cwnd/time and packet loss/time for each of the flows

Plot graph throughput/cwnd and packet loss/cwnd for the first flow. Is there an optimal value for cwnd?

Vary QueueSize5 and QueueSize6. Which one has immediate effect on cwnd size of the first flow? Explain why.

Example Solution - not available yet

Lab 4: OLSR routing

Level: Introductory

Expected learning outcome: What are MANETs and how they work. OLSR basics. Routing issues associated with MANETs.

Experiment:

Create a wireless mobile ad-hoc network with three nodes Node1, Node2 and Node3. Install the OLSR routing protocol on these nodes

Place them such that Node1 and Node3 are just out of reach of each other

Create a UDP client on Node1 and the corresponding server on Node3

Schedule Node1 to begin sending packets to Node3 at time 1s

Verify whether Node1 is able to send packets to Node3

Make Node2 move between Node1 and Node3 such that Node2 is visible to both A and C. This should happen at time 20s. Ensure that Node2 stays in that position for another 15s

Verify whether Node1 is able to send packets to Node3

At time 35s, move Node2 out of the region between Node1 and Node3 such that it is out of each other's transmission ranges again.

Verify whether Node1 is able to send packets to Node3

To verify whether data transmissions occur in the above scenarios, use either the tracing mechanism or a RecvCallback() for Node3's socket.

Plot the number of bytes received versus time at Node3

Show the pcap traces at Node 2's Wifi interface, and indicate the correlation between Node2's packet reception timeline and Node2's mobility

Example Solution -- myapp.h

Lab 5: WiFi RTS/CTS

Level: Introductory

Expected learning outcome: How 802.11 works with and without RTS/CTS. An insight into why its hard to setup efficient wireless networks

Experiment:

Setup a 5x5 wireless adhoc network with a grid. You may use examples/wireless/wifi-simple-adhoc-grid.cc as a base

Install the OLSR routing protocol.

Setup three UDP traffic flows, one along each diagonal and one along the middle (at high rates of transmission).

Setup the ns-3 flow monitor for each of these flows.

Now schedule each of the flows at times 1s, 1.5s, and 2s.

Now using the flow monitor, observe the throughput of each of the UDP flows. Furthermore, use the tracing mechanism to monitor the number of packet collisions/drops at intermediary nodes. Around which nodes are most of the collisions/drops happening?

Now repeat the experiment with RTS/CTS enabled on the wifi devices.

Show the difference in throughput and packet drops if any

Example Solution -- myapp.h

Lab 6: WiFi Channels

Level: Intermediate

Expected learning outcome: How Radio channel models affect transmission. An insight into why its important to correctly model the channel.

Experiment:

Setup a 2-nodes wireless adhoc network. Place the nodes at a fixed distance in a 3d scenario

Install all the relevant network stacks, up to and including UDP

Setup a CBR transmission between the nodes, one acting as a server and one as a client. Take the iperf [1] behaviour as an example.

Setup counters and outputs for packets sent and received.

Schedule the simulation to run for enough time to obtain statistically relevant results (suggestion: analyze some test results and reduce the simulation time accordingly).

Repeat the simulation varying the distance between the nodes from a minimum of 1meter to the point where the nodes can't transmit/receive anymore

Repeat the above varying the channel models and the transmission/receive parameters like node's position above the ground, transmission power, etc

Show the differences between the various channel models, and comment them. Identify the channel model that is more appropriate for each case (indoor, outdoor, LoS, NLoS, etc.)

Example Solution -- not available yet

Resources

- Main Website
- Wiki Page
- Source Repository
- Google Group Discussion
- NS3 Users mailing list

Acknowledgements

Special thanks to:

- Mathieu Lacage
- Tom Henderson
- Gustavo Carneiro
  for borrowing parts of their slides

Thank You

Please fill in the survey regarding NS3 here