Ns-3 Tutorial
Stein Kristiansen (steikr@ifi.uio.no)
09.03.2010

Outline
- Ns-3 Overview
- Installation
- Ns-3 Objects
- Scripting Step-by-Step
- Resources

Ns-3 Overview
- Free and open source discrete event network simulator
- Intended as a replacement for the popular ns-2
  - Clean slate implementation -- no reuse of ns-2 code
  - Easier to use, more flexible, faster, more accurate and flexible
- First version 3.1 in June 2008
  - Current version: 3.10
  - Planned for release spring of 2011
- Available for Linux, OS-X and Windows w/ Cygwin
- Written in C++
  - Scripts written in C++ with optional Python interface
  - Helper classes make "scripting" in C++ easy
- Well documented
  - Manual, tutorial, Doxygen and examples
  - Examples for wireless networks: ns-3.10/examples/wireless

Installing ns-3
- Scripted approach: download tarball, extract and build
- In Ubuntu, you can use build.sh:
  $ wget http://www.nsnam.org/releases/ns-allinone-3.10.tar.bz2
  $ tar xjf ns-allinone-3.10.tar.bz2
  $ cd ns-allinone-3.10
  $ ./build
  $ cd ns-3-dev
  $ ./waf
  --run first

Example Scripts
- manet-main.cc: mobile ad-hoc network, OLSR, random walk mobility
  - Presented here
- chain-main.cc: wireless ad-hoc network, chain topology, no mobility, static routes
  - Slides in the appendix
- Running the script:
  $ cp -r manet NS3FOLDER/scratch
  $ cd NS3FOLDER
  $ make ns-allinone=4 --nodes=4 --spacing="100"
  1. Searches examples, samples and scratch folders for scripts
  2. Compiles your script (+ any other modified files) into ns-3
  3. Runs the main() method in manet-main.cc

Ns-3 Objects
Ns-3 Objects

- Most objects inherit from ns3::Object
  - ... which inherits from ns3::ObjectBase
- Properties
  - Manageable via smart-pointers
    - Provides garbage-collection via reference-counting
  - Can be aggregated
- Must implement the GetTypeId() method
  - Provides run-time information on type
  - Allows objects to be located via object paths
  - Provides objects with attributes
  - Provides objects with trace-sources

Smart-Pointers

- Ns-3 objects are created by the CreateObject function, returning a smart-pointer:
  ```
  Ptr<PacketSocketFactory> factory = CreateObject<PacketSocketFactory>();
  ```
- Always check functions’ return-values and parameters: Ptr<T> or not?
  - Most often they are
    ```
    Ptr<Node> node = nodes.Get(0);
    ```

Object Aggregation

- Object can be aggregated to access each other, and for the user to easily access individual objects in an aggregation
- Object aggregation:
  - Avoid huge classes encompassing all possible functionality

```
node->AggregateObject (mobility);
```
- Retrieving an aggregated object:
  ```
  Ptr<MobilityModel> mob = node->GetObject<MobilityModel> ();
  ```

Typeld GetTypeld(void)

- Returns TypeId object to identify and characterize object
- Contains object type, constructor and parent object
- Defines the object’s attributes and trace-sources

```
TypeId GetTypeId (void)
{
static TypeId tid = TypeId ("ns3::olsr::RoutingProtocol")
  .SetParent<Ipv4RoutingProtocol> ()
  .AddConstructor<RoutingProtocol> ()
  .AddAttribute ("HelloInterval", "HELLO messages emission interval.",
    TimeValue (Seconds (2)),
    MakeTimeAccessor (&RoutingProtocol::m_helloInterval),
    MakeTimeChecker ()
  ...
  .AddTraceSource ("Rx", "Receive OLSR packet.",
    MakeTraceSourceAccessor (&RoutingProtocol::m_rxPacketTrace))
  return tid;
}
```

Ns-3 Object Paths

- Paths define location(s) of objects or their attributes
- Objects can be reached via paths as long as they are attributes of, or aggregated to, another object reachable via paths
- All paths through which an object can be reached listed in doxygen
- Can select sub-set of objects in lists by qualifiers
- Examples (from the Traits section available on the Ns-3 web-page):
  - NodeList::matches nodes index 0, 1, 3, 4, 5, 8
  - NodeList::matches all nodes
  - NodeList::NodeList::ipv4 matches object of type ns3::Ipv4 aggregated to node number 3
  - NodeList::NodeList::NodeList::ethertype matches all devices of type ns3::EthertypeDevice matches all devices in node number 3
- Can e.g., use Config::LookupMatches to access the objects directly

Attributes

- Attributes represent the different parameters of the models
- Attributes are defined in the class implementation

```
TypeId RoutingProtocol::GetTypeId (void)
{
static TypeId tid = TypeId ("ns3::olsr::RoutingProtocol")
  .SetParent<Ipv4RoutingProtocol> ()
  .AddConstructor<RoutingProtocol> ()
  .AddAttribute ("HelloInterval", "HELLO messages emission interval.",
    TimeValue (Seconds (2)),
    MakeTimeAccessor (&RoutingProtocol::m_helloInterval),
    MakeTimeChecker ()
  ...
  .AddTraceSource ("Rx", "Receive OLSR packet.",
    MakeTraceSourceAccessor (&RoutingProtocol::m_rxPacketTrace))
  return tid;
}
```
Setting Attribute Values

- Default attribute values can be set via Config::SetDefault
- Set attribute values for all subsequently instantiated objects of this

Config::SetDefault("ns3::YansWifiPhy::TxGain", DoubleValue(1));

- Attributes for individual object can be set with Config::Set, or
directly on the object via SetAttribute

Config::Set("ns3::YansWifiPhy::TxGain", DoubleValue(1));

Simulation Scripts Step-by-Step

General Structure of a Script:

1. Handle command line arguments
2. Set default attribute values and random seed
3. Create nodes
4. Configure physical and MAC layers
5. Enable PCAP tracing
6. Set up network stack, routing and addresses
7. Configure and install applications
8. Set up initial positions and mobility
9. Connect trace sources and sinks
10. Schedule user-defined events and start simulation

Step 1: Command line Arguments

- Allows configuration from command line
- E.g., $ ./waf --run "manet --spacing=100"

```cpp
int main(int argc, char *argv[]) {
    uint32_t rows = 4, cols = 4, nodeSpacing = 90, duration = 900, seed = 1;
    std::string phyMode("DsssRate11Mbps");
    CommandLine cmd;
    cmd.AddValue("phymode", "Physical transmission mode", phyMode);
    cmd.AddValue("rows", "Rows of nodes", rows);
    cmd.AddValue("cols", "Columns of nodes", cols);
    cmd.AddValue("spacing", "Spacing between neighbouring nodes", nodeSpacing);
    cmd.AddValue("duration", "Duration of simulation", duration);
    cmd.AddValue("seed", "Random seed for simulation", seed);
    cmd.Parse(argc, argv);
    uint32_t numNodes = rows * cols;
    ...
}
```

Step 2: Set Attribute Values and Random Seed

- Set default attribute values
- Remember to set random seed to different values between runs

```cpp
Config::SetDefault("ns3::WifiRemoteStationManager::FragmentationThreshold", StringValue("2200"));
Config::SetDefault("ns3::WifiRemoteStationManager::RtsCtsThreshold", StringValue("2200"));
Config::SetDefault("ns3::ConstantRateWifiManager::DataMode", StringValue(phyMode));
Config::SetDefault("ns3::ConstantRateWifiManager::ControlMode", StringValue(phyMode));
Config::SetDefault("ns3::YansWifiPhy::RxGain", DoubleValue(-10));
Config::SetDefault("ns3::YansWifiPhy::TxGain", DoubleValue(1));
```

// Set seed for pseudorandom number generator
SeedManager::SetSeed(seed);

Step 3: Create nodes

- Most components in ns-3 is managed by containers
- Simulations usually consist of many components
- Used by e.g. helper classes to install components
- Individual entities accessible via the Get()-method

```cpp
NodeContainer nodes;
nodes.Create(numNodes);
```
Steps 4-7: Configuring the Nodes

- Nodes are currently empty

Step 4: Physical Layer

- Helpers make scripting easier
- Here used to configure the physical layer
  - "Antenna"
  - Set capturing format (explained later)
- $TxGain = 1$ and $RxGain = -10$
- With $11$ Mbps DSSS this yields $160-190$ meters range
- Can be adjusted to obtain a range more realistic for a given scenario

```c
YansWifiPhyHelper wifiPhy = YansWifiPhyHelper::Default();
wifiPhy.Set("RxGain", DoubleValue(-10));
wifiPhy.SetPcapDataLinkType(YansWifiPhyHelper::DLT_IEEE802_11_RADIO);
```

Step 4: Channel

- Use of helpers to configure the channel
- Can also use e.g.,
  ```c
  ns3::TwoRayGroundPropagationLossModel
  ```

```c
YansWifiPhyHelper wifiPhy = YansWifiPhyHelper::Default();
wifiPhy.Set("RxGain", DoubleValue(-10));
wifiPhy.SetPcapDataLinkType(YansWifiPhyHelper::DLT_IEEE802_11_RADIO);
```

Step 4: MAC layer

- Non-QoS, $11$ Mbps $802.11$ used in the example
- Constant rate on transmissions and retransmissions
- Many alternatives, e.g., minstrel which is often used in real life
- Data mode set to "DsssRate11Mbps"

```c
NqosWifiMacHelper wifiMac = NqosWifiMacHelper::Default();
wifi.SetRemoteStationManager("ns3::ConstantRateWifiManager",
  "DataMode", StringValue(phyMode));
wifiMac.SetType("ns3::AdhocWifiMac");
```

Step 4: MAC layer

- Use helper to install devices
- Set the wireless interface into ad-hoc mode
- Select $802.11$ b standard
- Install this MAC layer into all nodes in our container "nodes", and connect them with the channel

```c
NqosWifiMacHelper wifiMac = NqosWifiMacHelper::Default();
wifi.SetRemoteStationManager("ns3::ConstantRateWifiManager",
  "DataMode", StringValue(phyMode));
wifiMac.SetType("ns3::AdhocWifiMac");
```

Step 5: Enable Tracing

- Must be done after setting up MAC
- Will produce one PCAP-file per interface per node in the current directory
  - Prefix is set to "MANET"
  - Resulting filenames: MANET-X.Y.pcap
  - Can later be analysed with e.g., tcpdump or wireshark:
    ```
    $ tcpdump -r MANET-X.Y.pcap
    ```

```c
wifiPhy.EnablePcap("MANET", devices);
```
Step 6: Stack, routing and addresses

- Select routing
  - Ns-3 currently supports OLSR, AODV and DSDV
  - Explained here: OLSR
  - In the appendix: manual configuration of static routes
- Install the network stack
  - Protocols installed: IP, TCP, UDARP and ICMP

```
OlsrHelper olsr;
InternetStackHelper internet;
internet.SetRoutingHelper (olsr);
internet.Install (nodes);
```

Step 6: Assign Network Addresses

- Nodes 1 to N get addresses 10.0.0.1 through 10.0.0.N
  - i.e., the address of node nodes.Get(X) has address 10.0.0.(X + 1)
- MAC addresses are set to 00:00:00:00:00:(X + 1)

```
Ipv4AddressHelper address;
address.SetBase ("10.0.0.0", "255.255.255.0");
Ipv4InterfaceContainer interfaces = address.Assign (devices);
```

Step 7: Configure and Install Applications

- This tutorial: UDP trace-client and -server
  - For specific applications, consult supervisors
- Set attributes, and start and stop applications according to duration of simulation

```
// Server/Receiver
UdpServerHelper server (4000);
ApplicationContainer apps = server.Install (nodes.Get(1));
apps.Start (Seconds (1.0));
apps.Stop (Seconds (10.0));

// Client/Sender
UdpClientHelper client (interfaces.GetAddress (1), 4000);
client.SetAttribute ("MaxPackets", UintegerValue (320));
client.SetAttribute ("Interval", TimeValue (Seconds (0.05)));
client.SetAttribute ("PacketSize", UintegerValue (1024));
apps = client.Install (nodes.Get (0));
apps.Start (Seconds (2.0));
apps.Stop (Seconds (9.0));
```

The Ns-3 Node

- Node provides methods to retrieve pointers to devices and applications
  - For specific applications, consult supervisors
  - Set attributes, and start and stop applications according to duration of simulation

```
Ptr<Application> app = node->GetApplication(0);
Ptr<NetDevice> nic = node->GetDevice(0);
```

- Aggregated with stack, mobility and energy-model

```
MobilityHelper mobility;
mobility.SetPositionAllocator ("ns3::GridPositionAllocator",
  "MinX", DoubleValue (0.0),
  "MinY", DoubleValue (0.0),
  "DeltaX", DoubleValue (nodeSpacing),
  "DeltaY", DoubleValue (nodeSpacing),
  "GridWidth", UintegerValue (cols));
```

Step 8: Set up Initial Positions

- Several options available, including grid, disc, random placement and user-defined locations
  - Explained here: grid
  - In the appendix: user-defined locations
  - For the usage of other alternatives, consult manual, tutorial and doxygen

```
// Server/Receiver
UdpServerHelper server (4000);
ApplicationContainer apps = server.Install (nodes.Get(1));
apps.Start (Seconds (1.0));
apps.Stop (Seconds (10.0));

// Client/Sender
UdpClientHelper client (interfaces.GetAddress (1), 4000);
client.SetAttribute ("MaxPackets", UintegerValue (320));
client.SetAttribute ("Interval", TimeValue (Seconds (0.05)));
client.SetAttribute ("PacketSize", UintegerValue (1024));
apps = client.Install (nodes.Get (0));
apps.Start (Seconds (2.0));
apps.Stop (Seconds (9.0));
```

Step 8: The Grid

- Various grid options available
  - Explained here: grid
  - In the appendix: user-defined locations
  - For the usage of other alternatives, consult manual, tutorial and doxygen
Step 8: Set up Mobility

- Several alternatives
  - Random mobility: random waypoint, random walk, ...
  - User-defined: constant position/velocity/acceleration

Explained here: random walk

- In the appendix: constant position (no mobility)

Step 8: Random Walk

Random walk:
- Walks in random direction for fixed time/distance + reflect on boundaries
- Upon arrival, select new random direction
- Select speed with a random variable
  - Default: "Uniform:2-4"
  - Select traveling distance or time

Step 9: Connecting Trace Sources and Sinks

- No 3 Objects can have a set of trace sources
  - All listed in the debugger
  - Invoked by methods in the object upon certain events
    - E.g., for the routing protocol whenever there is a change in the route table
  - User can connect sources to their own sinks
    - Sink function defined by the user
      - Use Config::Connect with paths
      - Invoked each time the source object invokes the corresponding source
      - Function prototype of source and sink must match
        - First argument: text string identifying the trace source

Step 9: Connecting Trace Sources and Sinks

- No 3 Objects can have a set of trace sources
  - All listed in the debugger
  - Invoked by methods in the object upon certain events
    - E.g., for the routing protocol whenever there is a change in the route table
  - User can connect sources to their own sinks
    - Sink function defined by the user
      - Use Config::Connect with paths
      - Invoked each time the source object invokes the corresponding source
      - Function prototype of source and sink must match
        - First argument: text string identifying the trace source

#include routing-protocol.cc

RoutingProtocol GetTypeId (void)
{
  return tid;
}

RouteChange AddTraceSource (uint32_t size)
{
  
}

RoutingT ableChanged

public:

class RoutingProtocol : public Ipv4RoutingProtocol
{
public:

  TraceConnect (uint32_t id, const char * trace)
  {
    
  }

  
}

Step 9: Connecting Trace Sources and Sinks

- No 3 Objects can have a set of trace sources
  - All listed in the debugger
  - Invoked by methods in the object upon certain events
    - E.g., for the routing protocol whenever there is a change in the route table
  - User can connect sources to their own sinks
    - Sink function defined by the user
      - Use Config::Connect with paths
      - Invoked each time the source object invokes the corresponding source
      - Function prototype of source and sink must match
        - First argument: text string identifying the trace source

#include routing-protocol.cc

RoutingProtocol GetTypeId (void)
{
  return tid;
}

RouteChange AddTraceSource (uint32_t size)
{
  
}

RoutingT ableChanged

public:

class RoutingProtocol : public Ipv4RoutingProtocol
{
public:

  TraceConnect (uint32_t id, const char * trace)
  {
    
  }

  
}

sample output:

RouteChange at /NodeList/1/$ns3::olsr::RoutingProtocol/RoutingTableChanged, new size: 4
Step 10: Schedule Events and Start Simulation

- Can schedule your own events before execution
  - Event: run a function with arguments at a given time
- Can schedule new events
- Duration of simulation set by Simulator::Stop

```cpp
void printTime(int interval) {
    std::cout << Simulator::Now() << std::endl;
    Simulator::Schedule(Seconds(interval), &printTime, interval);
}
```

```cpp
int main(int argc, char *argv[]) {
    ...
    Simulator::Schedule(Seconds(1), &printTime, 1);
    Simulator::Stop (Seconds (duration));
    Simulator::Run ();
    Simulator::Destroy ();
    return 0;
}
```

## Bienvisto: Enabling Bienvisto-Tracing

- Bienvisto: Extensible MANET visualizer developed as a master project by Miguel Santirso
- Flash (web)application taking as input a trace from Ns-3
- Based on the trace source/sink mechanism
- Currently traces events related to mobility, routing and data transmission
- Put visualizer-trace-helper.cc/h into folder with script

```cpp
#include "visualizer-trace-helper.h"
int main(int argc, char *argv[]) {
    ...
    Simulator::Schedule(Seconds(1), &printTime, 1);
    Simulator::Stop (Seconds (duration));
    VisualizerTraceHelper traceHelper(duration*1000, nodes);
    traceHelper.StartWritingFile("tracefile.txt");
    Simulator::Finish ();
    traceHelper.EndWritingFile();
    Simulator::Destroy ();
    return 0;
}
```

## Resources

- The Ns-3 Website: http://www.nsnam.org
  - Wiki
  - Manual
  - Tutorials
  - Presentations
  - Doxygen
- Ns-3 examples
  - `<ns3 folder>/examples`
  - `<ns3 folder>/samples`
  - chain and manet

## That’s All

Questions?
Good luck with the assignments!

### Static routes
- Static routes
- User defined locations
- Constant positions
- The Ns-3 logging facility

### Setting Static Routing

- Use Ipv4StaticRoutingHelper
- We provide a function to manipulate table
Step 6: Static Routing

- Setting static routes
  - UseIpv4StaticRoutingHelper
  - We provide a function to manipulate table

```c
void SetStaticRoute(Ptr<Node> n, const char* destination, const char* nextHop, uint32_t interface)
{
  Ipv4StaticRoutingHelper staticRouting;
  Ptr<Ipv4> ipv4 = n->GetObject<Ipv4>();
  Ptr<Ipv4StaticRouting> a = staticRouting.GetStaticRouting(ipv4);
  a->AddHostRouteTo(Ipv4Address(destination), Ipv4Address(nextHop), interface);
}
```

Step 6: Configuring Static Routes

- Setting static routes:

```c
// Set addresses
SetStaticRoute(nodes.Get(0), "10.0.0.3", "10.0.0.2", 1);
SetStaticRoute(nodes.Get(0), "10.0.0.2", "10.0.0.2", 1);
SetStaticRoute(nodes.Get(1), "10.0.0.1", "10.0.0.1", 1);
SetStaticRoute(nodes.Get(1), "10.0.0.3", "10.0.0.3", 1);
SetStaticRoute(nodes.Get(2), "10.0.0.1", "10.0.0.2", 1);
SetStaticRoute(nodes.Get(2), "10.0.0.2", "10.0.0.2", 1);
```

The logging facility

- Ns-3 has an extensive logging facility
  - Seven levels: error, warn, debug, info, function, logic, all

```c
NS_LOG_COMPONENT_DEFINE("MANET");
...
NS_LOG_INFO("Area width: "+(rows-1)"
NS_LOG_INFO("Area height: "+(cols-1)"
```

- Can activate component from script or from shell
  - LogComponentEnable("MANET", LOG_LEVEL_INFO);
  - $ export NS_LOG="MANET=level_info"