NS-3 LECTURE

Jimmy J. Nielsen - 17-11-2009

Overview

- ns-3 basic concepts brush-up
  - Based on < http://www.nsnam.org/tutorials/trilogy-summer-school.pdf>
- Installing/running ns-3
- Introduction of basic concept from toturial example
  - Conceptual slides from ns-3 presentation.
- Refactor
  - Wired→wireless (channel, phy, mac, node positions)
  - Flow traffic model
  - Logging - plot of throughput
- Exercises

Environment setup and download

- Ubuntu

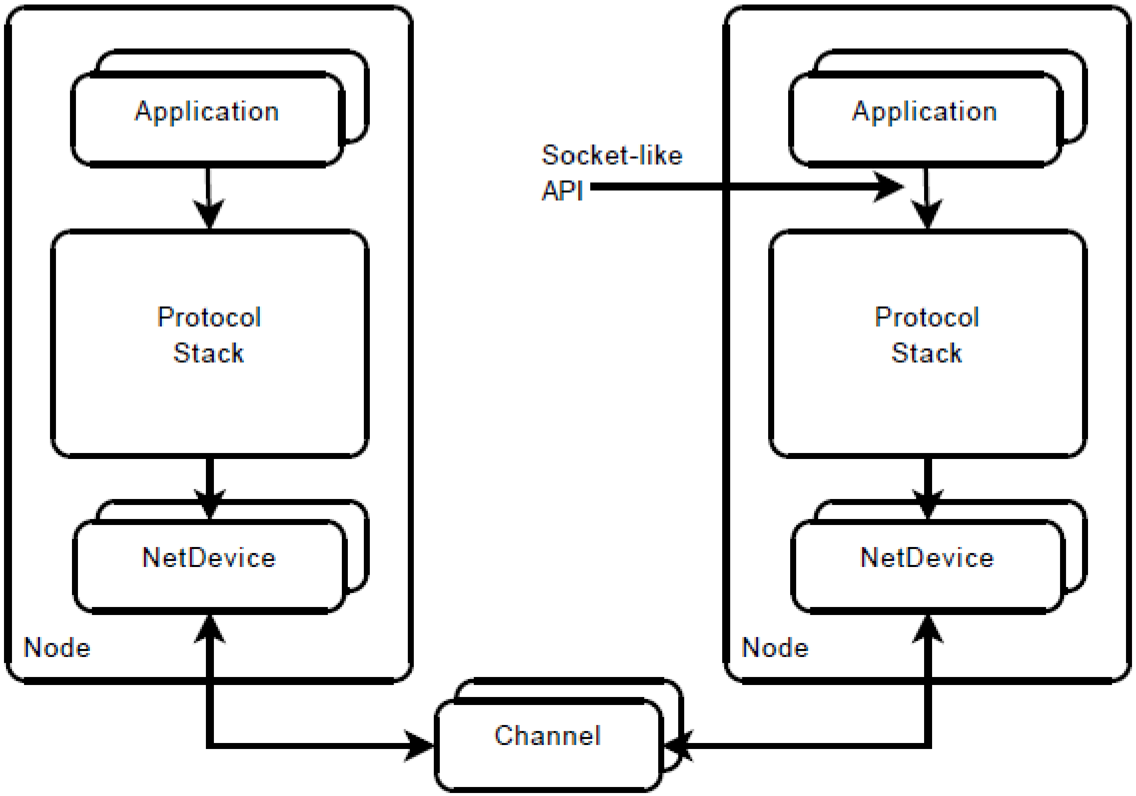
```
sudo apt-get install build-essential g++ python mercurial
```

- Windows
  A. cygwin, python, mercurial (but not all features work)
  B. install Sun Virtualbox and install Ubuntu.
- Availability:
  - Released tarballs: <http://www.nsnam.org/releases>
  - Development version: <http://code.nsnam.org/ns-3-dev>
  - The development version is usually stable: a lot of people use it for
  - daily work:

```
hg clone http://code.nsnam.org/ns-3-dev
```

- See tutorial <http://www.nsnam.org/tutorials.html> for build instructions.

The basic model

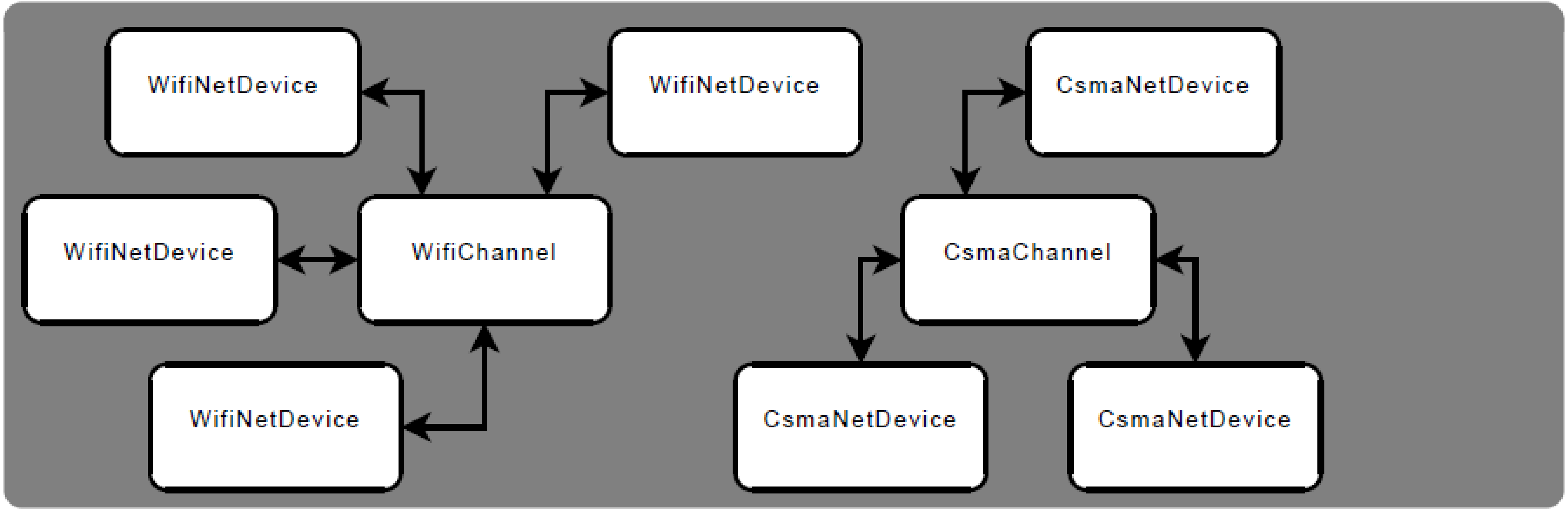


The fundamental objects

- Node
  - the motherboard of a computer with RAM, CPU, and, IO interfaces
- Application
  - a packet generator and consumer which can run on a Node and talk to a set of network stacks
- Socket
  - the interface between an application and a network stack
- NetDevice
  - a network card which can be plugged in an IO interface of a Node
- Channel
  - a physical connector between a set of NetDevice objects

Important remark

- NetDevices are strongly bound to Channels of a matching type:



Existing models

- Network stacks: arp, ipv4, icmpv4, udp, tcp (ipv6 under review)
- Devices: wifi, csma, point-to-point, bridge
- Error models and queues
- Applications: udp echo, on/off, sink
- Mobility models: random walk, etc.
- Routing: olsr, static global

WiFi models

- New model, written from 802.11 specification
  - Accurate model of the MAC
  - DCF, beacon generation, probing, association
  - A set of rate control algorithms (ARF, ideal, AARF, Minstrel, etc.)
  - Not-so-slow models of the 802.11a PHY

- New contributions from many developers:
  - University of Florence: 802.11n, EDCA, frame aggregation, block ack
  - Russian Academy of Sciences: 802.11s, HWMP routing protocol
  - Boeing: 802.11b channel models, validation
  - Deutsche Telekom Laboratories: PHY modelization, validation
  - Karlsruhe Institute of Technology: PHY modelization (Rayleigh,
  - Nakagami)

The Helper/Container API

- We want to:
  - Make it easy to build topologies with repeating patterns
  - Make the topology description more high-level (and less verbose) to make it easier to read and understand
- The idea is simple:
  - Sets of objects are stored in Containers
  - One operation is encoded in a Helper object and applies on a Container
- Helper operations:
  - Are not generic: different helpers provide different operations
  - Do not try to allow code reuse: just try to minimize the amount of code written
  - Provide syntactical sugar: make the code easier to read

Typical containers and helpers

- Example containers:
  - NodeContainer
  - NetDeviceContainer
  - Ipv4AddressContainer

- Example helper classes:
  - InternetStackHelper
  - WifiHelper
  - MobilityHelper
  - OlsrHelper
  - etc. Each model provides a helper class

The traditional approach

- In C++, if you want to call methods on an object, you need a pointer to this object. To get a pointer, you need to:
  - keep local copies of pointers to every object you create
  - walk pointer chains to get access to objects created within other objects

- For example, in ns-3, you could do this:

```
Ptr<NetDevice> dev = NodeList::Get (5)->GetDevice (0);
Ptr<WifiNetDevice> wifi = dev->GetObject<WifiNetDevice> ();
Ptr<WifiPhy> phy = dev->GetPhy ();
phy->SetAttribute ("TxGain", ...);
phy->ConnectTraceSource (...);
```

- It’s not fun to do...

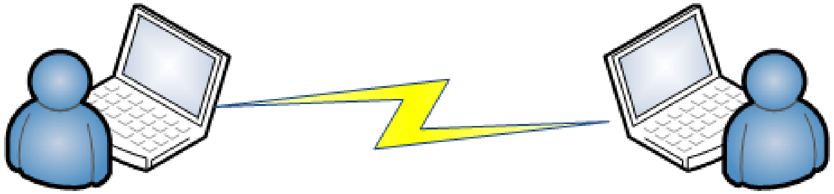
Look at the code

- Wired example from ns-3 tutorial
- See `first.cc`
- Build and run simulation:

```
./waf –run wifi_example_prototyping_2
```

Wired → wireless : changes

- Replace wired channel with wireless channel



- Install wifi protocol stack (PHY, MAC)
  - 802.11 ad hoc mode
  - rate adaptation algorithm
    - Adaptive Auto Rate Fallback (AARF), designed by one of the ns-3 creators, so implemented and tested.
- Install mobility model
  - Wireless transmission conditions depend on node positions.

Wired → wireless : Removed code

- Starting point is `first.cc`

Remove point-to-point channel.

```
//   PointToPointHelper pointToPoint;
//   pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
//   pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));
//   NetDeviceContainer devices;
//   devices = pointToPoint.Install (nodes);
```

Wired → wireless : Added code


```
#include "ns3/mobility-module.h"
#include "ns3/contrib-module.h"
#include "ns3/wifi-module.h"
...

  WifiHelper wifi = WifiHelper::Default (); //Needs PHY, MAC and nodes.
  wifi.SetRemoteStationManager ("ns3::AarfWifiManager");

  YansWifiChannelHelper channel = YansWifiChannelHelper::Default ();
  YansWifiPhyHelper wifiPhy = YansWifiPhyHelper::Default ();
  wifiPhy.SetChannel (channel.Create ());

  NqosWifiMacHelper wifiMac = NqosWifiMacHelper::Default ();
  wifiMac.SetType ("ns3::AdhocWifiMac");

  NetDeviceContainer devices = wifi.Install (wifiPhy, wifiMac, nodes);

  MobilityHelper mobility;
  Ptr<ListPositionAllocator> positionAlloc = CreateObject<ListPositionAllocator> ();
  positionAlloc->Add (Vector (0.0, 0.0, 0.0));
  positionAlloc->Add (Vector (5.0, 0.0, 0.0));
  mobility.SetPositionAllocator (positionAlloc);
  mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
  mobility.Install (nodes);
```

Demo

- Build and run simulation:

```
./waf -run wifi_example_prototyping_2
```

Flow traffic model

- ON/OFF traffic model (t_{OFF}=0 gives CBR)
  - Packet size = 1024 bytes
  - Data rate = 5 kb/s
- 10 s simulation → $\dfrac{10s \cdot 5kb/s}{1024B/packet} \approx \dfrac{50000}{8000} = 6{,}25\,packets$

Flow traffic : Removed code


```
//  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.)));
//  echoClient.SetAttribute ("PacketSize", UintegerValue (1024));
//
//  ApplicationContainer clientApps = echoClient.Install (nodes.Get (0));
//  clientApps.Start (Seconds (2.0));
//  clientApps.Stop (Seconds (10.0));
```

Flow traffic : Added code


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

...
  // Create, setup and install on/off traffic generator
  OnOffHelper onoff ("ns3::UdpSocketFactory",
                     Address (InetSocketAddress (interfaces.GetAddress (1), 9)));
  onoff.SetAttribute ("OnTime", RandomVariableValue (ConstantVariable (1)));
  onoff.SetAttribute ("OffTime", RandomVariableValue (ConstantVariable (0)));
  onoff.SetAttribute ("PacketSize", StringValue ("1024"));
  onoff.SetAttribute ("DataRate", StringValue ("5kb/s"));

  ApplicationContainer apps = onoff.Install (nodes.Get (0));
  apps.Start (Seconds (1.0));
  apps.Stop (Seconds (11.0));

  // Create a packet sink to receive these packets
  PacketSinkHelper sink ("ns3::UdpSocketFactory",
                          InetSocketAddress (Ipv4Address::GetAny (), 9));
  apps = sink.Install (nodes.Get (1));
  apps.Start (Seconds (1.0));
```

Demo

- Build and run simulation:

```
./waf –run wifi_example_prototyping_3
```

Logging – plot throughput

- How do we keep track of the throughput?
  - On packet reception – increment counter
- We need to create a callback handler: `ThroughputCounter`
  - `ThroughputCounter.ReceivePacket()` is called every time a packet is received, and increments byte counter.
    → See next slide
- Further increase data rate to 10Mbit/s
  - Simulation duration: 5-10 s

Througput counter


```
void
ThroughputCounter::ReceivePacket (Ptr<Socket> socket)
{
  Ptr<Packet> packet;
  while (packet = socket->Recv ())
  {
    bytesTotal += packet->GetSize();
  }
}
```

- in main function:

```
  // Create socket and bind to address+port
  ThroughputCounter* throughputCounter = new ThroughputCounter();
  TypeId tid = TypeId::LookupByName ("ns3::UdpSocketFactory");
  Ptr<Socket> sink = Socket::CreateSocket (nodes.Get (1), tid);
  InetSocketAddress local = InetSocketAddress (Ipv4Address::GetAny (), 9);
  sink->Bind (local);

  // Register callback function
  sink->SetRecvCallback (MakeCallback (&ThroughputCounter::ReceivePacket,
   throughputCounter));
```

Throughput logging


```
void
ThroughputCounter::CheckThroughput()
{
  double mbs = ((bytesTotal * 8.0) /1000000);
  bytesTotal = 0;
  m_output.Add ((Simulator::Now ()).GetSeconds (), mbs);
  Simulator::Schedule (Seconds (1.0), &ThroughputCounter::CheckThroughput,
   this);
}
```

- in main function:

```
  ...
  throughputCounter->CheckThroughput();
  Simulator::Run ();
  Simulator::Destroy ();

  // Output to GNU plot file
  std::ofstream outfile ("throughput.plt");
  Gnuplot gnuplot;
  gnuplot.AddDataset (throughputCounter->GetDatafile());
  gnuplot.GenerateOutput (outfile);
```

Demo

- Build and run simulation:

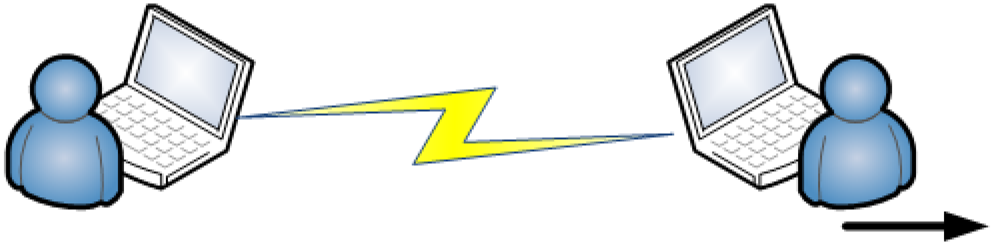
```
./waf –run wifi_example_prototyping_4
```

- Plot throughput:

```
gnuplot -persist throughput.plt
```

Exercises

1. Add a mobility model and observe how the throughput decreases with longer transmit distance.



2. Can ns-3 be useful for your project?