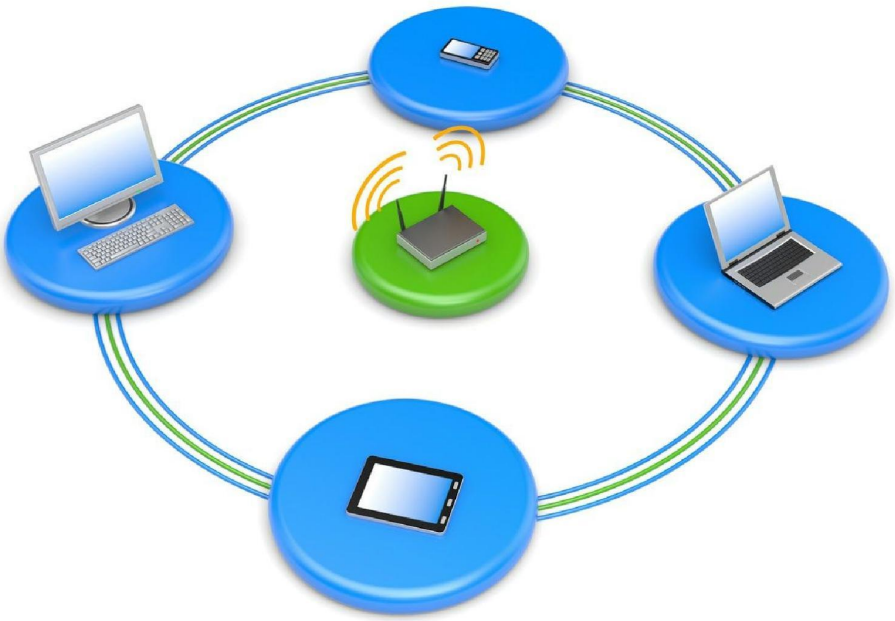
Advanced Computer Networks

Click Routing with NS3

Presented By Bofin B, Vivek S.



Introduction

Click modular router

Click is an open-source software architecture for building routers. It is assembled from packet processing modules called elements. Each elements implement common router functions like packet classification, queuing, scheduling etc. In this project I explore Click by learning Click configuration language in order to understand and evaluate its architecture.

A running Click router contains four important object categories: elements, router, packets, and timers. —Elements. The system contains an element object for each element in the current configuration, as well as prototype objects for every kind of primitive element that could be used. —Router. The single router object collects information relevant to a given router configuration. It configures the elements, checks that connections are valid, and puts the router on line; it also manages the task queue. —Packets. Click packet data is stored in a single block of memory, as opposed to a BSD mbuf-like structure. Packet data is copy-on-write—when copying a packet, the system copies the packet header but not the data. Annotations are stored in the packet header in a fixed static order; there is currently no way to dynamically add a new kind of annotation. In the Linux kernel, Click packet objects are equivalent to sk_buffs (Linux's packet abstraction). —Timers. In the Linux kernel, Click timers are implemented with Linux timer queues, which on Intel PCs have .01-second resolution.

NS3 Network Simulator

NS(from **network simulator**) is a name for series of discrete event network simulators, specifically **ns-1**, **ns-2** and **ns-3**. All of them are discrete-event computer network simulators, primarily used in researns-3 is built using C++ and Python with scripting capability. The ns-3 library is wrapped by Python thanks to the pybindgen library which delegates the parsing of the ns-3 C++ headers to gccxml and pygccxml to automatically generate the corresponding C++ binding glue. These automatically-generated C++ files are finally compiled into the ns-3 Python module to allow users to interact with the C++ ns-3 models and core through Python scripts. The ns-3 simulator features an integrated attribute-based system to manage default and per-instance values for simulation parameters. All of the configurable default values for parameters are managed by this system, integrated with command-line argument processing, Doxygen documentation, and an XML-based and optional GTK-based configuration subsystem.

Experiments

Connecting a node with Click to a node without Click.

```
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/applications-module.h"
#include "ns3/wifi-module.h"
#include "ns3/click-internet-stack-helper.h"
#include "ns3/log.h"
#include "ns3/mobility-helper.h"
using namespace ns3;
void ReceivePacket (Ptr<Socket> socket)
 {
  NS_LOG_UNCOND ("Received one packet!");
 }

int main (int argc, char *argv[])
{
#ifdef NS3_CLICK
 double rss = -80;

 // Setup nodes
 NodeContainer wifiNodes;
 wifiNodes.Create (2);
// Get Wifi devices installed on both nodes.
// Adapted from examples/wireless/wifi-simple-adhoc.cc
 std::string phyMode ("DsssRate1Mbps");

 // disable fragmentation for frames below 2200 bytes
 Config::SetDefault
("ns3::WifiRemoteStationManager::FragmentationThreshold",
StringValue ("2200"));
 // turn off RTS/CTS for frames below 2200 bytes
 Config::SetDefault
("ns3::WifiRemoteStationManager::RtsCtsThreshold",
StringValue ("2200"));
 // Fix non-unicast data rate to be the same as that of
unicast
 Config::SetDefault
("ns3::WifiRemoteStationManager::NonUnicastMode",
 StringValue (phyMode));

 WifiHelper wifi;
 wifi.SetStandard (WIFI_PHY_STANDARD_80211b);

 YansWifiPhyHelper wifiPhy = YansWifiPhyHelper::Default
();
```

```
 // This is one parameter that matters when using
FixedRssLossModel
 // set it to zero; otherwise, gain will be added
 wifiPhy.Set ("RxGain", DoubleValue (0) );
 // ns-3 supports RadioTap and Prism tracing extensions
for 802.11b
 wifiPhy.SetPcapDataLinkType
(YansWifiPhyHelper::DLT_IEEE802_11_RADIO);

 YansWifiChannelHelper wifiChannel;
 wifiChannel.SetPropagationDelay
("ns3::ConstantSpeedPropagationDelayModel");
 // The below FixedRssLossModel will cause the rss to be
fixed regardless
 // of the distance between the two stations, and the
transmit power
 wifiChannel.AddPropagationLoss
("ns3::FixedRssLossModel","Rss",DoubleValue (rss));
 wifiPhy.SetChannel (wifiChannel.Create ());

 // Add a non-QoS upper mac, and disable rate control
 NqosWifiMacHelper wifiMac = NqosWifiMacHelper::Default
();
 wifi.SetRemoteStationManager
("ns3::ConstantRateWifiManager",
 "DataMode",StringValue (phyMode),
 "ControlMode",StringValue (phyMode));
  // Set it to adhoc mode
 wifiMac.SetType ("ns3::AdhocWifiMac");
 NetDeviceContainer wifiDevices = wifi.Install (wifiPhy,
wifiMac, wifiNodes);

 // Setup mobility models
 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 (wifiNodes);

 // Install normal internet stack on node B
 InternetStackHelper internet;
 internet.Install (wifiNodes.Get (1));

 // Install Click on node A
 ClickInternetStackHelper clickinternet;
 clickinternet.SetClickFile (wifiNodes.Get (0),
"src/click/examples/nsclick-wifi-single-interface.click");
 clickinternet.SetRoutingTableElement (wifiNodes.Get (0),
"rt");
 clickinternet.Install (wifiNodes.Get (0));
```

```
 // Configure IP addresses
 Ipv4AddressHelper ipv4;
  ipv4.SetBase ("172.16.1.0", "255.255.255.0");
  ipv4.Assign (wifiDevices);

 // Setup traffic application and sockets
  Address LocalAddress (InetSocketAddress
(Ipv4Address::GetAny (), 50000));
  PacketSinkHelper packetSinkHelper
("ns3::TcpSocketFactory", LocalAddress);
  ApplicationContainer recvapp = packetSinkHelper.Install
(wifiNodes.Get (1));
  recvapp.Start (Seconds (5.0));
  recvapp.Stop (Seconds (10.0));

  OnOffHelper onOffHelper ("ns3::TcpSocketFactory",
Address ());
  onOffHelper.SetAttribute ("OnTime", StringValue
("ns3::ConstantRandomVariable[Constant=1]"));
  onOffHelper.SetAttribute ("OffTime", StringValue
("ns3::ConstantRandomVariable[Constant=0]"));
  ApplicationContainer appcont;

 AddressValue remoteAddress (InetSocketAddress
(Ipv4Address ("172.16.1.2"), 50000));
 onOffHelper.SetAttribute ("Remote", remoteAddress);
 appcont.Add (onOffHelper.Install (wifiNodes.Get (0)));

 appcont.Start (Seconds (5.0));
 appcont.Stop (Seconds (10.0));

 // For tracing
 wifiPhy.EnablePcap ("nsclick-raw-wlan", wifiDevices);

 Simulator::Stop (Seconds (20.0));
 Simulator::Run ();

  Simulator::Destroy ();
 return 0;
 #else
  NS_FATAL_ERROR ("Can't use ns-3-click without NSCLICK
compiled in");
 #endif
 }
```

2. UDP Flow from two LANs

```
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/applications-module.h"
#include "ns3/csma-module.h"
#include "ns3/ipv4-click-routing.h"
#include "ns3/ipv4-l3-click-protocol.h"
#include "ns3/click-internet-stack-helper.h"

using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("NsclickRouting");

int
main (int argc, char *argv[])
{
#ifdef NS3_CLICK

//
// Explicitly create the nodes required by the topology (shown above).
//
 NS_LOG_INFO ("Create nodes.");
 NodeContainer n;
 n.Create (3);

//
// Install Click on the nodes
//
 ClickInternetStackHelper clickinternet;
 clickinternet.SetClickFile (n.Get (0),
"src/click/examples/nsclick-routing-node0.click");
 clickinternet.SetClickFile (n.Get (1),
"src/click/examples/nsclick-ip-router.click");
 clickinternet.SetClickFile (n.Get (2),
"src/click/examples/nsclick-routing-node2.click");
 clickinternet.SetRoutingTableElement (n.Get (0), "kernel/rt");
 clickinternet.SetRoutingTableElement (n.Get (1), "u/rt");
 clickinternet.SetRoutingTableElement (n.Get (2), "kernel/rt");
 clickinternet.Install (n);

 NS_LOG_INFO ("Create channels.");
//
// Explicitly create the channels required by the topology (shown
above).
//
 CsmaHelper csma;
 csma.SetChannelAttribute ("DataRate", DataRateValue (DataRate
(5000000)));
 csma.SetChannelAttribute ("Delay", TimeValue (MilliSeconds (2)));
```

```
  csma.SetDeviceAttribute ("Mtu", UintegerValue (1400));
  NetDeviceContainer d01 = csma.Install (NodeContainer (n.Get (0), n.Get
(1)));
  NetDeviceContainer d12 = csma.Install (NodeContainer (n.Get (1), n.Get
(2)));

  Ipv4AddressHelper ipv4;
//
 // We've got the "hardware" in place. Now we need to add IP addresses.
 //
  NS_LOG_INFO ("Assign IP Addresses.");
  ipv4.SetBase ("172.16.1.0", "255.255.255.0");
  Ipv4InterfaceContainer i01 = ipv4.Assign (d01);

  ipv4.SetBase ("172.16.2.0", "255.255.255.0");
  Ipv4InterfaceContainer i12 = ipv4.Assign (d12);

  NS_LOG_INFO ("Create Applications.");
 //
 // Create one udpServer applications on node one.
//
  uint16_t port = 4000;
  UdpServerHelper server (port);
  ApplicationContainer apps = server.Install (n.Get (2));
  apps.Start (Seconds (1.0));
  apps.Stop (Seconds (10.0));

 //
 // Create one UdpClient application to send UDP datagrams from node zero
to
 // node one.
 //
  uint32_t MaxPacketSize = 1024;
  Time interPacketInterval = Seconds (0.05);
  uint32_t maxPacketCount = 320;
  UdpClientHelper client (i12.GetAddress (1), port);
  client.SetAttribute ("MaxPackets", UintegerValue (maxPacketCount));
  client.SetAttribute ("Interval", TimeValue (interPacketInterval));
  client.SetAttribute ("PacketSize", UintegerValue (MaxPacketSize));
  apps = client.Install (NodeContainer (n.Get (0)));
  apps.Start (Seconds (2.0));
  apps.Stop (Seconds (10.0));

  csma.EnablePcap ("nsclick-routing", d01, false);
  csma.EnablePcap ("nsclick-routing", d12, false);
  // Now, do the actual simulation.
  NS_LOG_INFO ("Run Simulation.");
  Simulator::Stop (Seconds (20.0));
  Simulator::Run ();
  Simulator::Destroy ();
  NS_LOG_INFO ("Done.");
 #else
  NS_FATAL_ERROR ("Can't use ns-3-click without NSCLICK compiled in");
 #endif
```

```
}
```

3. Connecting a node with click enabled with CSMA to a non click node.

```
#include "re-module.h"
#include "ns3/network-module.h"
#include "ns3/csma-module.h"
#include "ns3/internet-module.h"
#include "ns3/applications-module.h"
#include "ns3/click-internet-stack-helper.h"
#include "ns3/log.h"

using namespace ns3;

void ReceivePacket (Ptr<Socket> socket)
{
 NS_LOG_UNCOND ("Received one packet!");
}

int main (int argc, char *argv[])
{
#ifdef NS3_CLICK
 NodeContainer csmaNodes;
 csmaNodes.Create (2);

 // Setup CSMA channel between the nodes
 CsmaHelper csma;
 csma.SetChannelAttribute ("DataRate", DataRateValue (DataRate
(5000000)));
 csma.SetChannelAttribute ("Delay", TimeValue (MilliSeconds (2)));
 NetDeviceContainer csmaDevices = csma.Install (csmaNodes);

 // Install normal internet stack on node B
 InternetStackHelper internet;
 internet.Install (csmaNodes.Get (1));

 // Install Click on node A
 ClickInternetStackHelper clickinternet;
 clickinternet.SetClickFile (csmaNodes.Get (0),
"src/click/examples/nsclick-lan-single-interface.click");
 clickinternet.SetRoutingTableElement (csmaNodes.Get (0), "rt");
 clickinternet.Install (csmaNodes.Get (0));

 // Configure IP addresses for the nodes
 Ipv4AddressHelper ipv4;
 ipv4.SetBase ("172.16.1.0", "255.255.255.0");
 ipv4.Assign (csmaDevices);

 // Configure traffic application and sockets
 Address LocalAddress (InetSocketAddress (Ipv4Address::GetAny (),
50000));
```

```
  PacketSinkHelper packetSinkHelper ("ns3::TcpSocketFactory",
LocalAddress);
  ApplicationContainer recvapp = packetSinkHelper.Install (csmaNodes.Get
(1));
  recvapp.Start (Seconds (5.0));
  recvapp.Stop (Seconds (10.0));

  OnOffHelper onOffHelper ("ns3::TcpSocketFactory", Address ());
  onOffHelper.SetAttribute ("OnTime", StringValue
("ns3::ConstantRandomVariable[Constant=1]"));
  onOffHelper.SetAttribute ("OffTime", StringValue
("ns3::ConstantRandomVariable[Constant=0]"));

  ApplicationContainer appcont;

  AddressValue remoteAddress (InetSocketAddress (Ipv4Address
("172.16.1.2"), 50000));
  onOffHelper.SetAttribute ("Remote", remoteAddress);
  appcont.Add (onOffHelper.Install (csmaNodes.Get (0)));

  appcont.Start (Seconds (5.0));
  appcont.Stop (Seconds (10.0));

  // For tracing
  csma.EnablePcap ("nsclick-simple-lan", csmaDevices, false);

  Simulator::Stop (Seconds (20.0));
  Simulator::Run ();

  Simulator::Destroy ();
  return 0;
 #else
  NS_FATAL_ERROR ("Can't use ns-3-click without NSCLICK compiled in");
 #endif
 }
```

4. UDP Client Server with CSMA (with 3 nodes - all Click enabled)

```
#include <fstream>
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/csma-module.h"
#include "ns3/applications-module.h"
#include "ns3/ipv4-click-routing.h"
#include "ns3/click-internet-stack-helper.h"

using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("NsclickUdpClientServerCsma");

int
```

```
main (int argc, char *argv[])
{
#ifdef NS3_CLICK
//
// Enable logging for UdpClient and
//
 LogComponentEnable ("NsclickUdpClientServerCsma", LOG_LEVEL_INFO);

//
// Explicitly create the nodes required by the topology (shown above).
//
 NS_LOG_INFO ("Create nodes.");
 NodeContainer n;
 n.Create (3);

 NS_LOG_INFO ("Create channels.");
//
// Explicitly create the channels required by the topology (shown
above).
//
 CsmaHelper csma;
 csma.SetChannelAttribute ("DataRate", DataRateValue (DataRate
(5000000)));
 csma.SetChannelAttribute ("Delay", TimeValue (MilliSeconds (2)));
 csma.SetDeviceAttribute ("Mtu", UintegerValue (1400));
 NetDeviceContainer d = csma.Install (n);

//
// Install Click on the nodes
//
 ClickInternetStackHelper clickinternet;
 clickinternet.SetClickFile (n,
"src/click/examples/nsclick-lan-single-interface.click");
 clickinternet.SetRoutingTableElement (n, "rt");
 clickinternet.Install (n);

 Ipv4AddressHelper ipv4;
//
// We've got the "hardware" in place. Now we need to add IP addresses.
//
 NS_LOG_INFO ("Assign IP Addresses.");
 ipv4.SetBase ("172.16.1.0", "255.255.255.0");
 Ipv4InterfaceContainer i = ipv4.Assign (d);

 NS_LOG_INFO ("Create Applications.");
//
// Create one udpServer applications on node one.
//
 uint16_t port = 4000;
 UdpServerHelper server (port);
 ApplicationContainer apps = server.Install (n.Get (1));
 apps.Start (Seconds (1.0));
 apps.Stop (Seconds (10.0));
```

```
 //
 // Create one UdpClient application to send UDP datagrams from node zero
to
 // node one.
 //
  uint32_t MaxPacketSize = 1024;
  Time interPacketInterval = Seconds (0.05);
  uint32_t maxPacketCount = 320;
  UdpClientHelper client (i.GetAddress (1), port);
  client.SetAttribute ("MaxPackets", UintegerValue (maxPacketCount));
  client.SetAttribute ("Interval", TimeValue (interPacketInterval));
  client.SetAttribute ("PacketSize", UintegerValue (MaxPacketSize));
  apps = client.Install (NodeContainer (n.Get (0), n.Get (2)));
  apps.Start (Seconds (2.0));
  apps.Stop (Seconds (10.0));

  csma.EnablePcap ("nsclick-udp-client-server-csma", d, false);

 //
 // Now, do the actual simulation.
 //
  NS_LOG_INFO ("Run Simulation.");
  Simulator::Stop (Seconds (20.0));
  Simulator::Run ();
  Simulator::Destroy ();
  NS_LOG_INFO ("Done.");
 #else
  NS_FATAL_ERROR ("Can't use ns-3-click without NSCLICK compiled in");
 #endif
 }
```