NS-3-Click: Click Modular Router Integration for NS-3

Lalith Suresh P.
Instituto Superior Tecnico
Lisbon, Portugal
lalith.puthalath@ist.utl.pt

Ruben Merz
Deutsche Telekom Laboratories / TU Berlin
Berlin, Germany
ruben.merz@telekom.de

ABSTRACT

The Click Modular Router provides a flexible platform for protocol development and testing. Integrating the Click Modular Router with a network simulator offers the advantage of bringing this flexibility into a simulation framework. The existing integration of Click with ns-2 (nsclick) has several limitations, namely it cannot be used with generic traffic generators, transport protocols and NetDevices. For the integration with ns-3, we address these limitations. Furthermore, the design of ns-3 makes it much better suited for embedding Click than ns-2 does. This includes the closer alignment of ns-3 with real world packet formats and handling multiple interfaces per node. In this paper, we describe ns-3-click, discuss its design and how it improves over nsclick. Our experiments suggest that ns-3-click does not incur significant performance hits as far as wall clock run time is concerned, but uses more memory than ns-3.

1. INTRODUCTION

The Click Modular Router [5] (or Click for short) is a framework for creating flexible and configurable routers. A user creates a router configuration by combining a group of packet processing units called *elements* to create a so-called Click graph. Click provides a platform for researchers to experiment with novel protocols.

There are several motivations behind bringing this flexibility to a simulation framework. Protocols developed using Click can be tested on various topologies and under different configurations as per the needs of the user. Once a Click configuration is validated, it can be deployed almost as-is on an actual network. Apart from bridging simulation and deployment, Click can allow nodes in a simulation to use an extensive library of Click elements, thus expanding the feature horizon for the simulator itself. Nsclick [7] is the original integration of Click with ns-2 [1] that enables several of these features. However, nsclick shows limitations that make it cumbersome to use: Nsclick works only with ns-2 raw packet formats because the ns-2 simulator does not implement real world packet formats. Consequently, nsclick can not work with all kinds of traffic generators or transport protocols. Nsclick also needs extensions to account for different kinds of interfaces (i.e. NetDevices).

On the other hand, the design of ns-3 is well suited for an integration with Click. Packets in ns-3 are serialized and deserialized as they move up and down the stack. This allows ns-3 packets to be passed to and from Click as they are, and greatly favors such an integration. It also makes it easy to run any traffic generator and transport protocol available in ns-3 on top of Click. Furthermore, very few changes to the ns-3 link layer (LL) and medium access control (MAC) code are required to allow Click to work seamlessly with any kind of ns-3 device.

Ns-3-click takes advantage of all the above aspects of ns-3 design. Thus, unlike nsclick, ns-3-click can work with all the existing applications and NetDevices of the simulator. Furthermore, it offers the flexible packet processing capabilities of Click, which would otherwise be harder to do with ns-3 or ns-2 alone.

The rest of the paper is organized as follows; In Section 2, we briefly explain the architecture of the Click modular router and of ns-3, and discuss different challenges of their integration. Section 3 describes the integration itself. In Section 4, we show results regarding the performance of ns-3-click. Section 5 explains related work. Section 7 concludes the paper and describes future work.

2. INTEGRATION OF CLICK AND NS-3

In this section, we briefly discuss Click and ns-3 individually, the two components of the integration. We then describe the integration with respect to architecture of ns-3.

2.1 The Click Modular Router

The Click Modular Router [5] is an architecture for designing and deploying highly configurable and flexible routers, written in C++. A router configuration describes an interaction between independent packet processing units known as *elements*. Click provides an extensive library of elements which can perform various functionalities ranging from IP TTL decrementing and WiFi header encapsulation, to implementing RED queues and performing traffic shaping. A router configuration is called a Click graph, and is described using a simple syntax in a file referred to as a *Click file*. Elements are configurable via arguments specified within the Click file. To initialize a Click router, a Click file has to be described. For ns-3-click, the simulation user would need to specify the Click graph a particular Click node is expected

WNS-3 2011, March 21, Barcelona, Spain

DOI 10.4108/icst.simutools.2011.245535

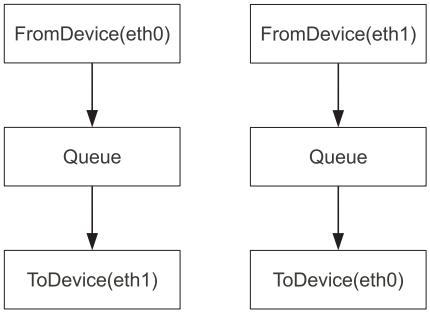

Figure 1: Representation of a simple Ethernet bridge as a Click graph, composed of individual packet processing elements from Click. The corresponding code is displayed in Figure 2.

```
//
// Ethernet-bridge.click
//
FromDevice(eth0) -> Queue -> ToDevice(eth1);
FromDevice(eth1) -> Queue -> ToDevice(eth0);
```

Figure 2: An Click file for the Ethernet bridge example in Figure 1. In ns-3-click, the Click file a particular Click node is to use will be specified within the simulation script using a helper function.

to run. We describe how this is done in Section 3.5.

Every element has at least one port, which can be of type input or output. Ports allow connections between elements, and are the basis for packet flow through a Click graph. These connections can be of two types; a push connection or a pull connection. In a push connection, the upstream element hands a packet to the downstream element. In a pull connection, a downstream packet asks for a packet from an upstream element.

Figure 1 depicts how a simple bridge is described as a click graph. In this example, all packets arriving at *eth0* are transferred to *eth1* and vice-versa. In ns-3-click, as will be described in the following sections, *eth0* and *eth1* in a Click graph will correspond to the 0th and 1st net devices on the ns-3 node. Figure 2 shows the representation of the same Click graph in a Click file. The `FromDevice` and `ToDevice` elements are used to receive and send out packets on the network interfaces specified as their arguments. The `FromDevice` element has a push output whereas the `ToDevice` element has a pull input. A `Queue` element is used in between acting as a push-to-pull converter.

2.2 Network Simulator 3 (ns-3)

The network simulator 3 (ns-3) [2] is a discrete-event simulator, and the proposed eventual replacement for the ns-2 simulator. Nodes in an ns-3 simulation are connected to each other by means of *NetDevices* communicating over their respective channels. A node can have multiple NetDevices. Nodes may or may not have an IP stack (for instance, a Layer 2 switch or an Ethernet bridge) or a mobility model associated with it (stationary nodes). Packets in ns-3 are serialized/deserialised to/from actual packet formats as they traverse the network stack, making it well suited for real world integration. This is largely different from ns-2, where `structs` are passed between node objects. Furthermore, ns-3 has IPv4/IPv6 addressing schemes available for network specifications, something which ns-2 lacks and thus performs implicitly.

2.3 Technical Challenges of the Integration

The main challenges of integrating Click into ns-3 are:

- Creating an interface between the simulator and Click.
- Time synchronization between the simulator and Click.
- Maintaining compatibility with existing ns-3 traffic generators and transport protocols.
- Maintaining compatibility with the existing ns-3 NetDevices.

To interface between ns-3 and Click, we reuse the existing Click external simulator API originally developed for nsclick [7]. This accounts for the time management component as well. With this API, a Click router instance is created for a node which wishes to use Click for routing. The Click instance is aware of the simulation time as well. The simulator advances time as it is supposed to, and every time it has to run a Click router instance on a node, it informs Click of the current time. Furthermore, if Click wants the simulator to schedule a Click instance to run at a particular instant of time, it uses the API to schedule an event on the simulator. The API also enables the use of Click handlers from the simulator to read/write properties of different elements. Thus we write ns-3 specific implementations for the same API to interface between ns-3 and Click.

To maintain compatibility with ns-3 applications, transport protocols and NetDevices, we strive to have Click run entirely at layer 3 in the network stack. This is made possible by the fact that ns-3 design cleanly segregates the layer 3 functionality of a node into a class named *Ipv4L3Protocol*. The class *Ipv4L3Protocol* talks to a routing protocol class *Ipv4RoutingProtocol* in order to perform routing. Thus, by talking to Click through a routing protocol object, we can confine all our changes to the Layer 3 code itself. This allows ns-3 traffic generators, transport protocols and NetDevices to run with minor modifications in such a scenario.

3. DESIGN AND ARCHITECTURE OF NS-3-CLICK

In this section, we first briefly recall the architecture of ns-3. We then explain the architecture of ns-3-click, and finally proceed to outline the details of the implementation itself.

3.1 Architecture of ns-3

The flow of a packet through an IPv4 network stack in ns-3 is depicted in Figures 3 and 4. When running a traffic generator on top of a node, the socket abstraction *UdpL4SocketImpl/TcpL4SocketImpl* queries the routing protocol system to find a source address to match the required destination address for the packet. The routing protocol abstraction for IPv4 is called *Ipv4RoutingProtocol*, and the two important interfaces it provides are *RouteOutput()* and *RouteInput()*, which are analogous to *ip_route_output()* and *ip_route_input()* respectively on Linux.

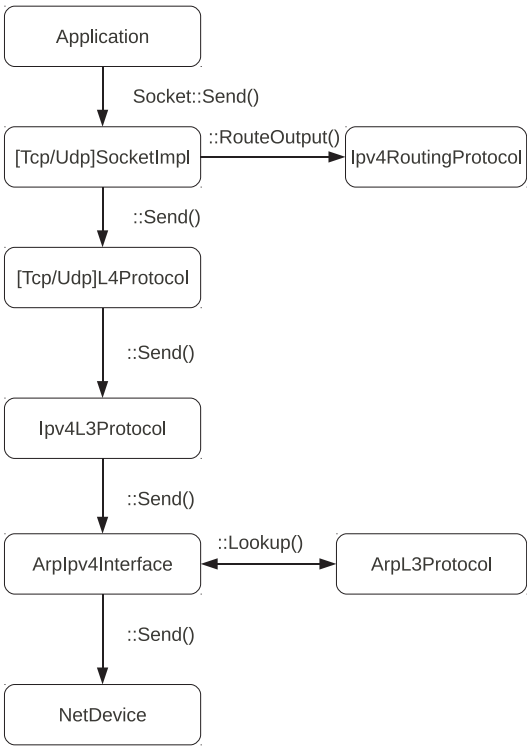

Figure 3: Path of a packet going down the stack in ns-3. The packet flows along the downward arrows, via the method invocations indicated on the arrows, with queries being made to the routing protocol and ARP protocol at appropriate stages.

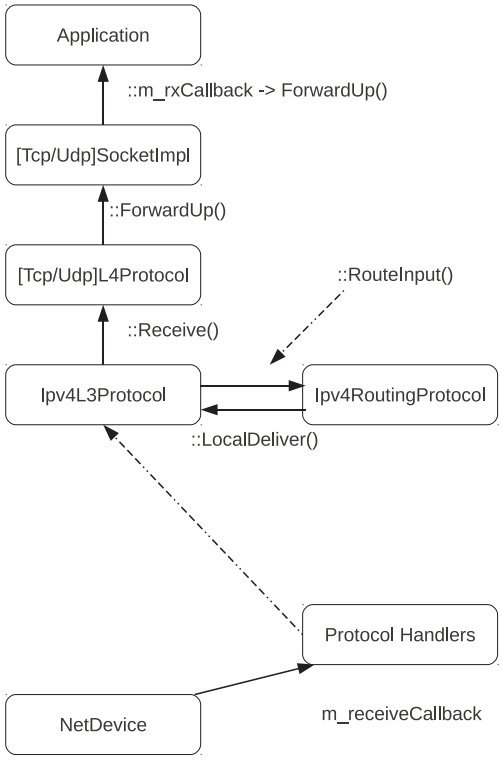

Figure 4: Path of packet going up the stack in ns-3. The packet flows along the upward arrows, with the protocol handler passing the packet to either the ARP instance or the IP layer (shown above), depending on the packet type. Forwarding decision is made at Layer 3 using a *RouteInput()* query to the routing protocol instance.

The socket independent protocol logic is implemented within *UdpL4Protocol/TcpL4Protocol*, after which the packet enters the *Ipv4L3Protocol* instance, which is the Layer 3 abstraction in ns-3. Depending on the route passed from the Layer 4 protocol, *Ipv4L3Protocol* passes the packet to the appropriate *Ipv4Interface*, which performs an ARP query if required, before sending the packet out via the corresponding network device (NetDevice object).

3.2 Architecture of ns-3-click

In ns-3-click, we need to provide a routing protocol abstraction that will talk to Click using the Simulator API discussed in section 2.3. This abstraction is implemented in the form of *Ipv4ClickRouting* (Figures 5 and 6). We also replace *Ipv4L3Protocol* with *Ipv4L3ClickProtocol*, which is a stripped down version of the former. This is because much of the routing functionality is dependent on the packet traversing the click graph corresponding to the node it is running on. This is in stark contrast to an *Ipv4RoutingProtocol* instance in ns-3, which merely responds to *RouteOutput()* and *RouteInput()* queries from the Layer 4 and Layer 3 components, as opposed to handling the packet itself. Thus, the ns-3-click Layer 3 subsystem needs to hand the packet to Click at appropriate points in the stack instead of merely querying Click. In effect, ns-3-click handles Layer 3 and routing functionality differently from ns-3 by performing them using Click. The following sections explain how this is achieved.

3.3 Implementation of the integration

In this section we describe the implementation details of the integration.

3.3.1 Ipv4ClickRouting and Ipv4L3ClickProtocol

As depicted in figures 5 and 6, *Ipv4ClickRouting* implements the interfaces required to communicate between Click and ns-3. If a user wants to run an application on top of a node running Click, a routing table element needs to be included in the Click graph and be specified in the simulation script as well. This is necessary for the *RouteOutput()* functionality, where the socket implementation queries the routing subsystem to find a matching source IP address for a given destination. For sending packets down the stack, *Ipv4L3ClickProtocol* hands the packet to *Ipv4ClickRouting* upon invocation of *Ipv4L3ClickProtocol::Send()*. *Ipv4ClickRouting* then passes the packet to Click, and the packet flows through the Click graph of the node it is running on. The packet is then received from one of the Click interfaces (say *eth0*). Depending on the receiving interface, *Ipv4ClickRouting* makes the decision of which NetDevice of the node the packet is to be sent out from. Forwarding of packets work in the same manner, except that the packet

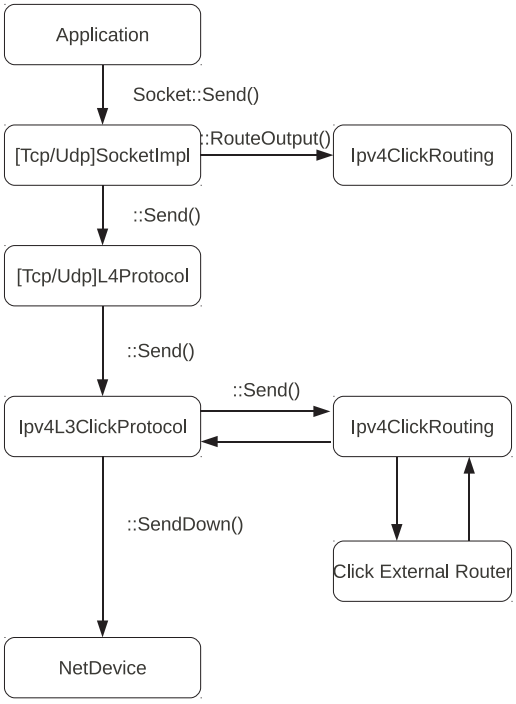

Figure 5: Path of a packet going down the stack in ns-3-click. The flow is similar to Figure 3 except at Layer 3, where the packet is actually passed to the Click router and back via *Ipv4L3ClickProtocol* and *Ipv4ClickRouting*, so as to have the packet traverse the Click graph. All Layer 3 functionality including ARP is handled by Click.

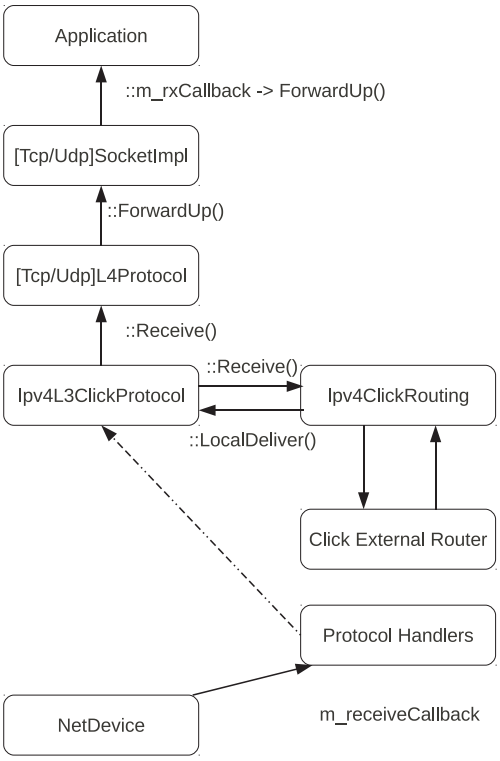

Figure 6: Path of a packet going up the stack in ns-3-click. *ns-3-click* handles all Layer 3 functionality including ARP via Click itself, so the protocol handler always passes packets up to *Ipv4L3ClickProtocol*. Instead of a call to *RouteInput()* as in Figure 4, the forwarding is performed by passing the packet through the Click graph.

hand-off between *Ipv4L3ClickProtocol* and *Ipv4ClickRouting* occurs at *Ipv4L3ClickProtocol::Receive()*. In this case, depending on the packet and the Click graph, a packet can be delivered locally as well. This happens when a packet sent to Click is received back via Click *tap0* interface. Thus *Ipv4L3ClickProtocol* defers all forwarding decisions to Click, and works around the need for an *Ipv4ClickRouting* specific *RouteInput()* implementation.

The Click interface naming convention has been fixed arbitrarily, with *tap0* corresponding to the Operating System, and *eth0, eth1* and so forth referring to the network interfaces of the node.

3.3.2 Packet Hand-off between ns-3 and Click

Packet hand-off between ns-3 and Click happens at four points:

- Layer 4 to Layer 3
- Layer 3 to Layer 4
- Layer 3 to Layer 2
- Layer 2 to Layer 3

In the above, the first two kinds of hand off have to ensure that ns-3 applications run unmodified on top of Click, while the last two have to be performed such that any ns-3 NetDevice can run underneath Click.

3.3.3 Application/Transport Compatibility

The socket implementations in ns-3-click continue to perform the calls to *RouteOutput()*, and at Layer 3, *Ipv4L3ClickProtocol* passes an IP encapsulated packet to Click, which then flows through the click graph. For a packet going up the stack, a packet that Click decides to deliver locally is delivered to the appropriate Layer 4 protocol and then to the application. Unlike nsclick [7], which was restricted to only raw sockets, ns-3-click thus allows for all kinds of traffic generators to run on top of Click.

3.3.4 NetDevice Compatibility

A packet going down the stack, received from one of Click interfaces, is passed to the appropriate NetDevice in order to be sent out to the channel. The use of MAC layer specific elements in the Click graph is avoided, and the Ethernet header attached to a packet by Click interfaces is stripped off before making the call to *NetDevice::Send()*. Packets going up the stack make their way through the ns-3 MAC models and NetDevice code, before being received at *Ipv4L3ClickProtocol*. An Ethernet header is then appended to the packet before being handed over to Click via

Ipv4ClickRouting. This architecture allows the same Click graph to run unmodified on top of any ns-3 NetDevice.

3.4 Packet Flow Example

We now provide a walk through of how a packet would flow through the stack in case of an application running on one Click based node sending a UDP packet to another instance of the application on a another Click based node. As shown in Figures 5 and 6, running TCP in this case would be exactly the same, except for the kind of socket implementation and Layer 4 instances running on the node.

3.4.1 Transmission of a Packet in ns-3-click

In Figure 5, a packet generated by an application will be sent out via a socket using *Socket::Send()*. The *UdpSocketImpl* instance of the node queries the Click router running on the node for the matching source address of the node for the destination of the packet. This query is performed using *Ipv4ClickRouting::RouteOutput()*, which internally uses a Click read handler to read the required route from the Click instance. Any Click node which runs an application has to use a routing table element for this step. Once *UdpSocketImpl* receives a suitable response for the route query, it follows up with a call to *UdpL4Protocol::Send()*, where the UDP header is added to the packet. Next, a call is made to *Ipv4L3ClickProtocol::Send()*, which adds an IP header to the packet (using the source and destination addresses specified in the route passed down by the UDP layer). A call is then made to *Ipv4ClickRouting::Send()* which sends the packet to Click to be passed down via the *FromSimDevice(tap0)* element. This corresponds to a Click graph receiving a packet from the Operating System. The packet then traverses the Click graph, and is ultimately received via a Click interface (ultimately via *Ipv4ClickProtocol::HandlePacketFromClick()*). This method then identifies the appropriate interface for sending out the packet, and makes a call to *Ipv4L3ClickProtocol::SendDown()*. Based on the information received from *Ipv4ClickProtocol*, *Ipv4L3ClickProtocol* uses the *NetDevice::Send()* method of the appropriate network device to ultimately send a packet out of the node. All Layer 3 functionality is implemented via Click, including ARP.

3.4.2 Reception of a Packet in ns-3-click

At the receiving end (Figure 6), a packet received at Layer 2 is directly passed to *Ipv4L3ClickProtocol* as explained in Section 3.3.4. This holds for all kinds of Layer 3 packets, including ARP. The method that receives the packet is *Ipv4L3ClickProtocol::Receive()*, which passes the packet to the appropriate Click interface via *Ipv4ClickRouting::Receive()*. The Click graph is then traversed, and the packet is received at one of Click interfaces again. Based on the interface ID of this Click interface, *Ipv4ClickRouting* then decides whether the packet is to be locally delivered or forwarded. In case of the former, *Ipv4L3ClickProtocol::LocalDeliver()* is then used to pass the packet up the stack, to the application via the UDP layer. Otherwise, the packet is sent out of a network interface as specified in the previous section.

3.5 ns-3-click Usage

Users of ns-3-click can setup a Click node using the *ClickInternetStackHelper* class. The most important configuration to be done is to specify a Click file for each Click node in the simulation. This is done through the ns-3 simulation script. An example snippet is shown below:

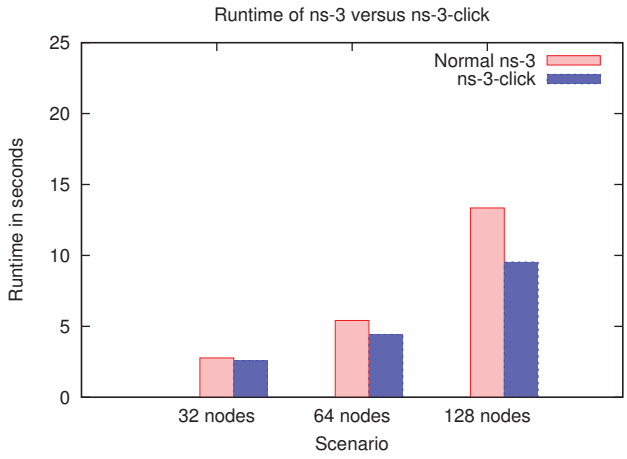

Figure 7: Average wall clock running time of ns-3 versus ns-3-click for running an IP Router configuration. Results indicate that ns-3-click benefits from improved running time by performing Layer 3 functionality in Click.

```
NodeContainer n;
...
ClickInternetStackHelper clickinternet;
clickinternet.SetClickFile (n, "router.click");
clickinternet.SetRoutingTableElement (n, "rt");
clickinternet.Install (n);
```

This example shows the use of the *SetClickFile()* method of the *ClickInternetStackHelper* class to specify a Click file named "router.click" for a group of nodes. The method is also overloaded so as to specify a Click file for individual nodes as well. The path to the Click file is specified as an absolute path, or a path relative to ns-3-click top-level directory. Furthermore, every Click node which is to run an ns-3 traffic generator has to have a routing table element in Click so as to enable an *Ipv4RoutingProtocol::RouteOutput()* query from the socket layers. This is faciliated through the *SetRoutingTableElement()* method provided by the helper method.

4. SIMULATION RESULTS

In this section, we perform several simulations to compare the performance of ns-3 and of ns-3-click. In both cases, we simulate an IP router configuration. We used the latest revision of ns-3-click (revision 6543 as of this time) [3], running on Ubuntu 10.04 on an x86 architecture with a 2.10 GHz Intel(R) Core(TM)2 Duo CPU T6500 processor and 4GB of RAM. We consider three scenarios where we perform simulations using either a CSMA NetDevice or a WiFi NetDevice. For each simulation result, we conducted ten runs. The wall-clock running times, memory consumption and throughput results are average results. The wall-click running times and memory consumption statistics for each simulation run was measured using the `/usr/bin/time` utility in Linux.

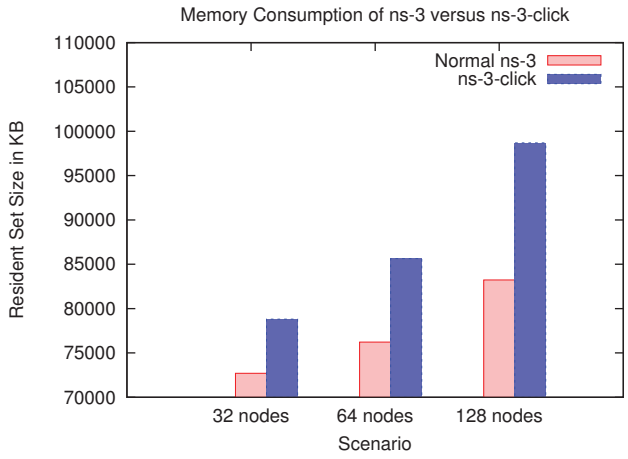

Figure 8: Average memory Consumption of ns-3 versus ns-3-click for running an IP Router configuration. Ns-3-click consumes slightly more memory than normal ns-3.

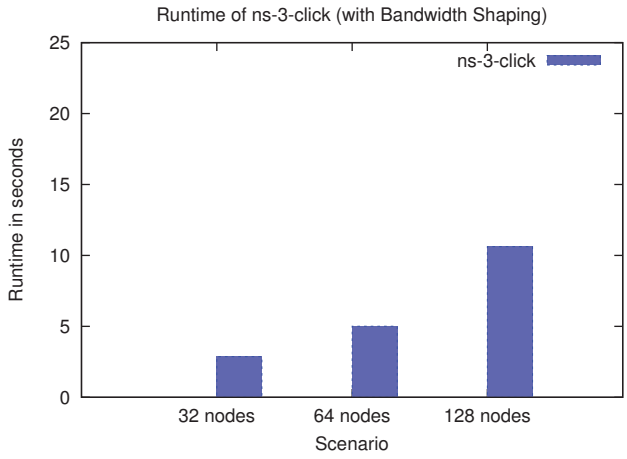

Figure 10: Average wall clock running time of ns-3-click for running an IP Router configuration with bandwidth shaping.

Scenario 1: CSMA NetDevice with Star Topology.

For our first scenario, we begin with a simple star topology (see Figure 9), with the IP router routing traffic between end-points of the topology. The links between the leaf nodes and the router are CSMA links. We run the *UdpClientServer* application, with *N/2* nodes acting as clients, and *N/2* nodes acting as servers, each node acting either as a client or as a server (but not both). Clients generate packets of size 1024 bytes every 0.05 seconds (with an equivalent data rate of 20 KB/s), and send up to 320 packets in total. The total simulation time of the simulations is 100 seconds.

In the ns-3-click version, we used the IP Router configuration used in [5], adapted for ns-3-click (by replacing *FromDevice* and *ToDevice* elements with *FromSimDevice* and *ToSimDevice* elements respectively). Routing was performed statically in either case. For ns-3-click, this is done using the *StaticIPLookup* element, and in ns-3, this corresponds to an *Ipv4StaticRouting* instance initialized as the routing protocol on the node. We consider a varying number of leaf nodes in the star topology, with 32, 64 and 128 nodes.

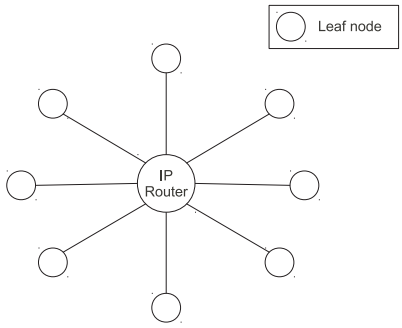

Figure 9: A simple star topology with eight leaf nodes.

Figure 8 shows the total memory consumption of the simulation for each case. Ns-3-click consumes more memory because of the Click router initialized in the simulation. Depending on the size of the Click graph, more elements are initialized, and this leads to increased memory usage. Figure 7 shows the total wall clock running time of the simulations. The results in Figure 7 suggests that the wall clock running time of ns-3-click is better than ns-3, suggesting that Click *StaticIPLookup* performs better than *Ipv4StaticRouting* in ns-3. Hence, this shows that ns-3-click benefits from improved performance by using Click for routing, and at the same time providing the flexibility associated with Click.

Scenario 2: CSMA NetDevice with Star Topology and a Bandwidth Shaping Element in ns-3-click.

For our second scenario, we add bandwidth shaping functionalities into the IP router configuration. This is done using Click *BandwidthShaper* element, which accepts a rate value as an argument. All packet flows through this element are limited to the specified rate. By inserting this element in the output path of a packet in the Click graph, we ensure that all data flows through the IP router are bandwidth shaped to 15 KB/s. Traces show that the rate at which packets are received at the receiver end is indeed limited to 75% as expected (15 KB/s after shaping versus 20 KB/s before the same). The wall clock running time and memory measurements of the simulation are shown in Figures 10 and 11. We note that the wall clock running time for running an IP Router with bandwidth shaping does not vary much from running a simple IP Router on ns-3-click (Figure 7 versus Figure 10). To perform the same operation using normal ns-3, we would have to implement rate controlling queues at appropriate points in the stack. On the other hand, with ns-3-click, it is merely a matter of inserting the required element at the desired point in the Click graph. It is exactly this kind of flexibility that Click brings into ns-3, along with the extensive library of packet processing elements.

Scenario/ Number of Nodes	*Avg. Throughput: ns-3-click*			*Avg. Throughput: ns-3*		
	32	64	128	32	64	128
CSMA	20.07	20.07	20.07	20.07	20.07	20.07
B/w shaping	14.11	14.11	14.11	N/A	N/A	N/A
WiFi	20.10	20.14	20.24	20.10	20.14	20.23

Figure 14: Validation tests: Measured average throughput in for the CSMA, bandwidth shaping and WiFi scenarios with ns-3-click and ns-3. Reported results are average results over ten simulation runs.

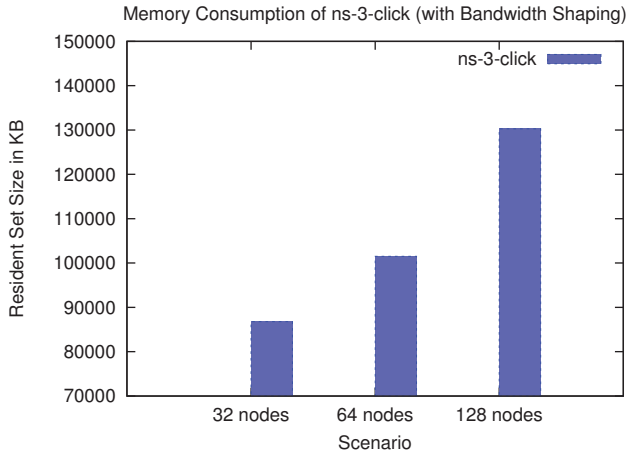

Figure 11: Average memory consumption of ns-3-click for running an IP Router configuration with bandwidth shaping.

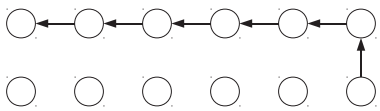

Figure 12: Simulation topology for a wireless ad hoc network. Routing is static, with the packet being forwarded in the Y direction first, and then along the X direction.

Scenario 3: WiFi NetDevice with Two Dimensional Grid Topology.

To demonstrate the working of the WiFi NetDevice working on ns-3-click, we perform simulations on a wireless ad hoc network. The topology we use is that of a two dimensional grid, with all the nodes split across two lines. We use static routing to model a simple X-Y routing scheme. Figure 12 describes the topology and shows routing done first along the Y direction, and then along the X direction. We perform static routing in ns-3-click using the *StaticIPLookup* element, and the same is done in ns-3 using each node *Ipv4StaticRouting* instance. As indicated in the figure, a single UDP source sends packets to the node situated diametrically across it in the topology. Packets with a length of 1024 byte are generated by the source at a rate of 20 KB/s.

Figure 13 compares the wall clock running times of ns-3-click and ns-3 for the simulations. We observe that for

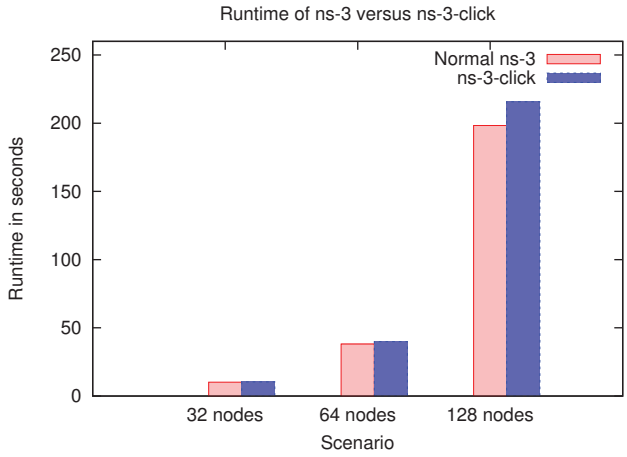

Figure 13: Average wall clock running time of ns-3-click for running a wireless ad hoc network.

lightly loaded traffic scenarios such as this, the wall clock running times of ns-3-click are close to that of ns-3 itself. Figure 15 compares the memory consumption of ns-3-click with ns-3. We note from these results that ns-3-click consumes more memory than ns-3 as was the case with the CSMA simulations as well.

Throughput for Each Scenario.

Figure 14 indicates the observed throughput at the receiver end measured at the end of each simulation scenario. These results validate the functioning of ns-3-click. In case of CSMA, the simulation traces are exactly the same for ns-3 and ns-3-click. But with the WiFi simulations, there are slight differences stemming from the fact that *ns-3-click* uses Click *ARPQuerier/ARPResponder* elements for handling ARP, whereas normal *ns-3* uses its own ARP implementation. Inspection of the code explains the observed difference. Click *ARPQuerier* sends out a maximum of only one ARP query for each packet entering its input for which a destination MAC address isn't available in the ARP table. On the other hand, the ARP module in *ns-3* performs more than one attempt for the same packet in case a response is not received immediately. This leads to a minor difference in the traces in the WiFi scenario.

5. RELATED WORK

Nsclick [7] was developed by integrating the Click Modular Router with ns-2 [1]. The integration was performed by developing a glue layer between ns-2 and Click. Some of the

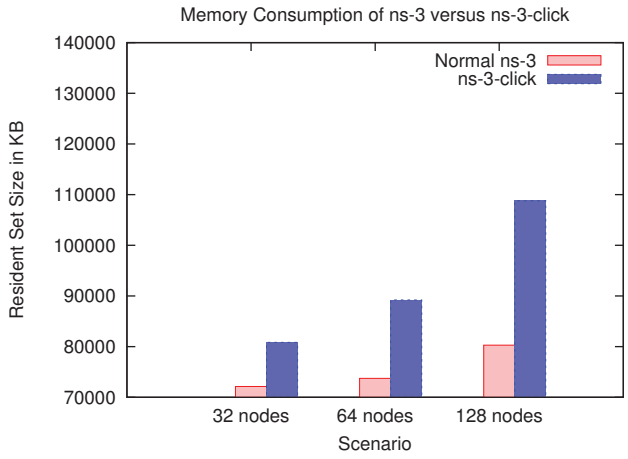

Figure 15: Average memory consumption of ns-3-click for running a wireless ad hoc network.

incompatibilities between the two tools are the following:

- ns-2 packets are represented as `structs` being exchanged between nodes whereas Click, being a real software router, processes packets in wire frame format.
- Nodes in ns-2 are not assigned IP and MAC addresses explicitly, but Click requires IP and MAC addresses to function correctly. Thus in nsclick, the simulation script has to configure IP and MAC addresses for each node using some extra *Tcl* commands, added for this very purpose.
- Extensions had to be written to allow nsclick to work with different kinds of network devices [6] (for instance, IEEE 802.11).

One of ns-3 design goals is to be transparent to real world protocols and networks. As a result of this objective, packets in ns-3 are always serialized/deserialized into real world packet formats depending on the direction of the packet flow in the network stack. Furthermore, nodes in ns-3 are assigned IP and MAC addresses, which further facilitate an integration with Click. By striving to implement ns-3 fully at Layer 3, we avoid having to write entire extensions so as to keep ns-3-click compatible with different kinds underlying of network devices.

Another example of embedding router software into simulators is the GNS3 Network Simulator [4], which enables users to experiment with Cisco IOS and Juniper JunOS router configurations.

6. CURRENT LIMITATIONS

A current limitation of ns-3-click is that some of Click MAC specific elements cannot be used yet. This is because ns-3-click focuses on running the same Click graph unmodified on any kind of ns-3 NetDevice. Furthermore, some of Click Wifi elements expect radiotap or atheros descriptor headers, which are features that ns-3 does not yet support. One possible step towards supporting this would be to allow ns-3 to support monitor mode for Wifi devices, and thus pass L2 packets directly to Click.

7. CONCLUSIONS AND FUTURE WORK

In this work, we have integrated Click with ns-3, and have enabled a flexible platform for network protocols development, testing and experimentation. Furthermore, it allows us to expand the feature horizon of ns-3, by enabling the use of Click wide range of elements within ns-3 simulation environment. Experimental tests suggest that adding Click to ns-3 scenarios can be done without significant run-time performance hits, but the improved flexibility comes at the cost of increased memory consumption, and therefore more requirements. We are currently working to extend this to support Click MAC layer functionality as well: this would involve making Click directly bind to ns-3 network devices, with Click handling the MAC high layer models. This would be a stronger framework for protocol experimentation. In addition, because both Click and ns-3 support IPv6, extending ns-3-click to support IPv6 is another direction to follow. Finally, the integration of ns-3-click with the emulation features of ns-3 may create a powerful platform for transparent protocol development in both simulation environment and testbeds.

8. ACKNOWLEDGMENTS

The work in this paper was supported by the Google Summer of Code 2010 (GSOC) program.

9. REFERENCES

[1] Network Simulator 2 (ns-2), http://www.isi.edu/nsnam/ns, November, 2010.

[2] Network Simulator 3 (ns-3), http://www.nsnam.org, November, 2010.

[3] NS-3 Click Integration (ns-3-click), http://code.nsnam.org/lalith/ns-3-click, December, 2010.

[4] The GNS3 Graphical Network Simulator, http://www.gns3.net/, February, 2010.

[5] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The click modular router. *ACM Transactions on Computer Systems*, 18(3):263–297, August 2000.

[6] N. Letor, P. De Cleyn, and C. Blondia. Enabling cross layer design: adding the madwifi extensions to nsclick. In *Proceedings of the 2nd international conference on Performance evaluation methodologies and tools*, ValueTools '07, pages 19:1–19:10, ICST, Brussels, Belgium, Belgium, 2007. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[7] M. Neufeld, A. Jain, and D. Grunwald. Nsclick:: bridging network simulation and deployment. In *MSWiM '02: Proceedings of the 5th ACM international workshop on Modeling analysis and simulation of wireless and mobile systems*, pages 74–81, New York, NY, USA, 2002. ACM.