Implementing Policy-Based Routing in NS-3

Sudarshan S, Aditya Kamath, Bhargav Reddy
Department of Computer Science and Engineering
Indian Institute of Technology Hyderabad, India
Email:{cs10b036, cs11b001, cs11b012}@iith.ac.in

***Abstract*—NS-3 currently has a limited routing implementation that routes packets based solely on their destination IP address. While this is usually sufficient for most cases, it makes it essentially impossible to simulate things such as selective intercepting proxying that rely on redirecting connections with certain destination port numbers. We seek to enhance NS-3's routing framework to make it possible to route packets based on an arbitrary user-specified function of TCP headers and IP headers. This task is aimed to help the group that is implementing flow mobility for IPv6 flows, since it is impossible to perform redirection of individual flows with the existing API.**

I. Introduction

Although one would expect that the destination IP address provides enough information to unambiguously route packets, modern operating systems implement support for routing packets based on source and destination port numbers as well. There are some distinctive uses for this. For instance, the famous intercepting proxy redsocks, and the android applications built on top of it such as proxy droid all use this facility. Also, implementing mobility for individual flows necessarily requires this kind of support. Unfortunately, the current routing infrastructure in NS-3 does not support such complex routing schemes.

II. Goal

The goal for this project was to extend NS-3's routing mechanism in order to support routing based on an arbitrary, user defined combination of TCP headers and IP headers. This was to be achieved by allowing routing based on a callback that the user would implement. For this project we explicitly target IPv6 and do not worry about IPv4 support.

This project specifically aimed to allow individual TCP flows to the same destination to be dynamically re-routed through a different interface. For instance, as indicated by figure 1, mobile phones usually have multiple upstream connections to the Internet, and under some situations it would be useful to redirect some connections to a different interface.

III. Implementation

Over the course of this project, we made attempts to implement this in two different ways:

1) Using a virtualised network interface that would intercept all traffic to and from the real network cards
2) Modifying NS-3s routing code to support policy based routing

We shall now examine both of these approaches in detail:

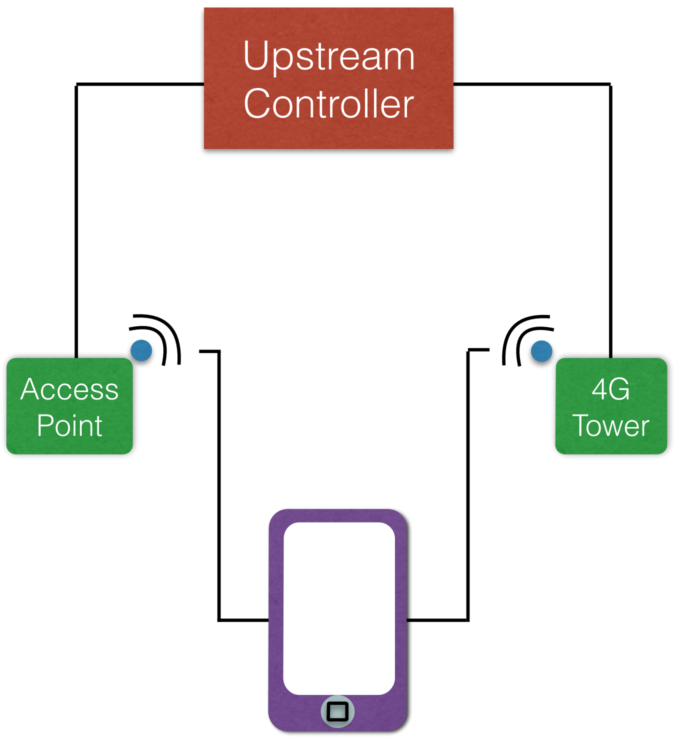


Fig. 1. The typical mobile phone has connections to the Internet through both 4G and wifi

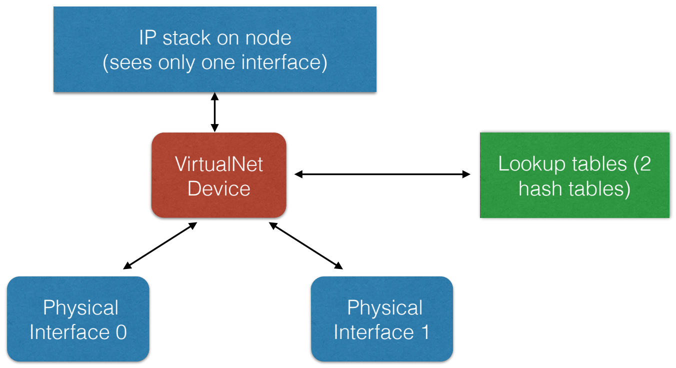
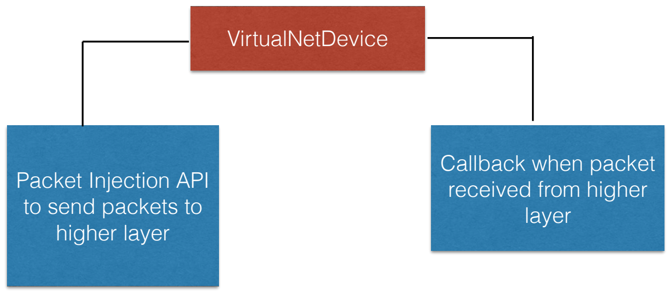

IP stack on node (sees only one interface)
VirtualNet Device
Lookup tables (2 hash tables)
Physical Interface 0
Physical Interface 1

Fig. 2. The proposed architecture- the virtual net device is between the nodes IP stack and physical network cards


VirtualNetDevice
Packet Injection API to send packets to higher layer
Callback when packet received from higher layer
Fig. 3. The two core features provided by the NS-3 VirtualNetDevice

A. *Virtualised network interface*

As indicated in figure 2, this approach was based on adding a virtual network device and hiding all physical network devices from the nodes IP stack. The idea was that the IP stack would only see the virtual net device, and hence all IP packets irrespective of their destinations would be sent to the virtual device which would then forward them to the appropriate physical device.

As illustrated in figure 3 NS-3 provides a device called the VirtualNetDevice, which provides two important pieces of functionality:

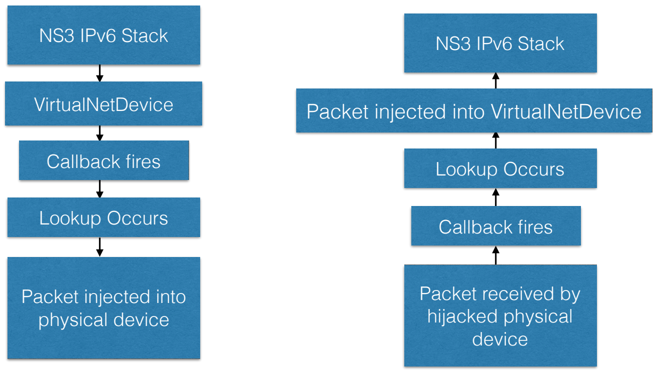
- Reception of packets from the higher layers triggers a call back, which can then decide what to do with the packet. In our case, we planned to use lookup tables that would use both IP and TCP headers to decide where to send the packet
- The virtual device is not connected to a channel and never receives any packets. However the code that created the virtual net device can inject packets into it. These packets appear to the higher layers as if they were received on the virtual device

We shall now examine in detail the sequence through which packets from the higher layer would be intercepted, routed and send to the physical devices.

- The IP stack on the node would forward a packet to the virtual device, since it is the only network device on the system visible to it
- The virtual device would fire the callback. Our callback would be provided the entire ethernet packet to be sent
- Callback would use the TCP and IP headers to decide where the packet should be sent
- The low level frame injection API would be used to send a packet to the real physical device

On the other hand, the sequence that would be followed when packets arrive at one of the nodes physical interfaces would be as follows:

- The packet would be received by the physical device
- There would be no higher layer protocol associated with the device since the device has been intentionally hidden from the IP stack
- Therefore, we would be able to intercept the packet with a callback


NS3 IPv6 Stack
VirtualNetDevice
Callback fires
Lookup Occurs
Packet injected into physical device
NS3 IPv6 Stack
Packet injected into VirtualNetDevice
Lookup Occurs
Callback fires
Packet received by hijacked physical device
Fig. 4. Summary of the techniques used to intercept packets both while sending and while receiving

- We would use the aforementioned packet injection API on the virtual net device to make the higher layer believe that the packet was received on the virtual device

This process is summarised in figure 4

Although this approach seemed promising, we had to abandon it for the following reasons:

- The ip stack sees only one device, and this device is either broadcast (ARP not required) or unicast (ARP required). On the other hand, some of the physical devices connected to the node could be unicast and some could be broadcast. Therefore, our code would have to handle the ARP as well.
- The virtual device has a different MAC address from the physical devices. Since IPv6 addresses are assigned based on the mac address, this could lead to wrong addresses.
- IPv6 requires neighbour and router solicitation packets to be generated on every physical interface. Since the IP stack only sees one interface, our code would have to generate and process all these packets

B. *Modifying NS-3 routing code*

The core of NS-3s IPv6 routing is present in the file ipv6-static-routing.cc, in the RouteOutput function. Here, the destination address is matched against the routing table in order to decide the gateway and interface to send the packet through.

Our modification consisted of adding a user routing callback and a user data pointer as members of this class. This callback would be set to NULL by default and it would only fire if it were changed from this default value. The original lookup code was replaced with the following:

1) Check if the callback has been set. If it has not been set go ahead to step 5
2) Check that the packet is a TCP packet. If it is not go ahead to step 5
3) Execute the callback and check if it returns a valid route. If it returns NULL then go ahead to step 5
4) Return the route provided by the callback

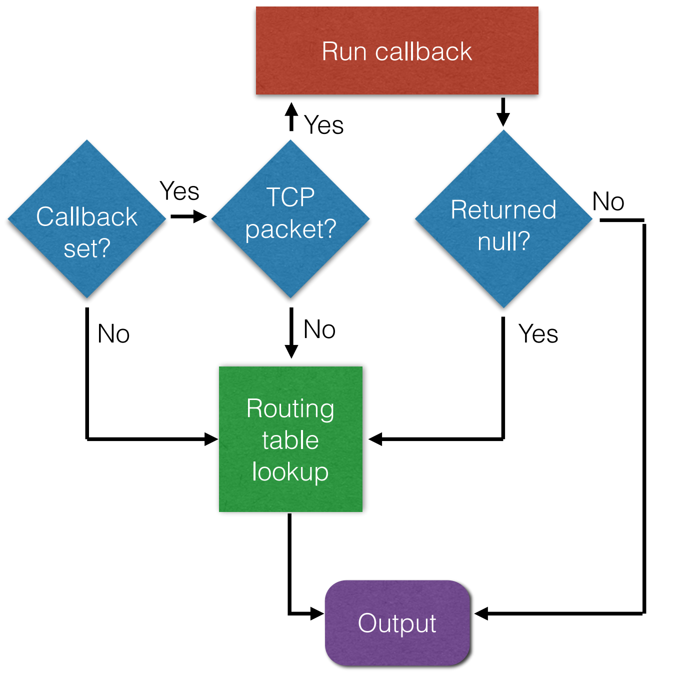


Fig. 5. Summary of the algorithm used to implement policy based TCP routing in ns3

5) Perform a lookup on the routing table and return the appropriate route

The algorithm that was used is summarised in figure 5. The callback would be provided both the IP and the TCP headers, and could use any arbitrary technique to decide which node to send the packet to. Our demo will be with a callback that redirects all tcp packets to a different node after 10 seconds of simulation time

IV. Experimental Setup

As indicated in figure 6, the experimental setup on which the code was tested consisted of three nodes, with interconnections as indicated. All links were 8Kbps CSMA links, similar to ethernet.

The events that happened during the experiment can be briefly summarised as follows:

- Node 1 starts sending TCP traffic to node 3 via node 2 using NS-3s built in bulk send application
- At 10 seconds, policy based routing migrates the live TCP connection to flow directly to node 3. It is to be noted that the connection is migrated live, with the same source and destination addresses and ports.

V. Results and conclusion

In this project, we successfully implemented policy based routing in the NS-3 network simulator. In the process, we also investigated how to create a virtual net device in ns3 that can intercept all communication to and from higher layers

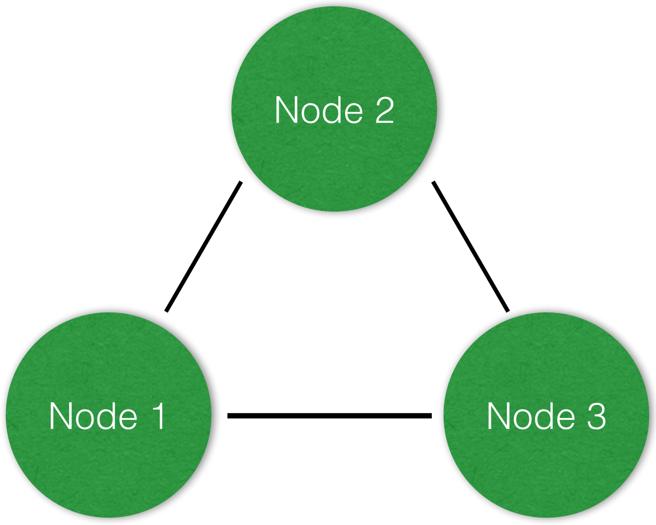


Fig. 6. Summary of the experimental setup used to test policy based TCP routing