Lab 10 - Openflow based Custom Router
Group: Route-On

Group Member:
Hao Zhang, Vidhi Goel, Venkata Srinivas Chowdary Reddy
Oct 30th, 2015

In Custom Network we used Custom Router to route packets between nodes. In this lab we use Openflow switch in place of Custom Router to route packets. To make Openflow switch do routing based on flow tables we connect it to a controller which sends flow tables to Switch. The switch based on the flow tables does the packet routing.

A. DESIGN IMPLEMENTATION

As described in the problem statement we are building a 5 node network as shown below. Of these nodes center node is a router (switch), and 4 nodes being connected to it with a link. One of the 4 nodes is the controller and the other 3 are used as hosts for file transfers. A general visualization of our network is shown here:

Custom Packet and Custom Protocol:

We are using our Custom Packets and Protocols from previous labs. For convenience we are reviewing it here.

dest_mac	src_mac	protocol	saddr	daddr	ttl	protocol	size	check	dummy	port	dummy	check	seq	PAYLOAD
Ethernet Header			Routing Header							Reliable Header				

Ethernet header is briefly described as below:

Destination MAC - a0:36:9f:0a:5e:a6,
Source MAC - a0:36:9f:0a:5f:9a,
Protocol - 0x0000

Routing header is briefly described as below:

Source - 11,
Destination - 21,
TTL - 16,
Protocol - 02 (Reliable File transfer protocol).
Size - 1422 (Total size including headers),
Checksum - 0081 (This is checksum for routing protocol header alone. Router will verify this checksum)
Dummy - 0008 (This is used to simulate the protocol field of ethernet header. NIC card rejects the packet if this is not set correctly).

Reliable File transfer protocol header is described as below:

Port - 12 (This is the application port number)
Dummy - 00 (This field is optional and can be used for additional purposes)
Checksum - 0030 (This is checksum for reliable header and application payload. End system needs to verify this)
Sequence Number - 00000000 (This is the first sequence number sent by Sender to Receiver)

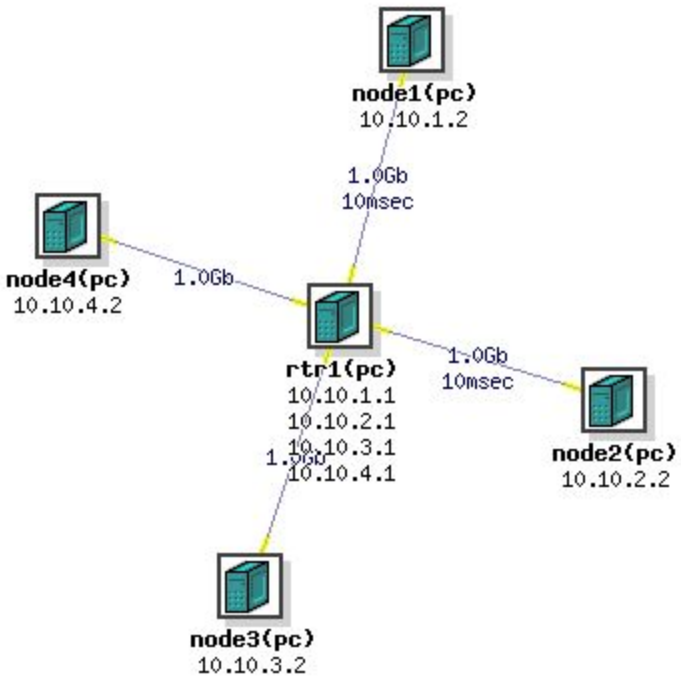
PAYLOAD correspond to the actual file data sent from Sender to Receiver

B. DETER IMPLEMENTATION

Our Network topology is made of 5 nodes. One center node where we have router (rtr1) and all other 4 nodes connected to it. Of the 4 nodes we consider one node (Node4) as the Controller. The NS file used to build this Network is as shown below:

```
set ns [new Simulator]
source tb_compat.tcl
# Nodes
set rtr1 [$ns node]
tb-set-node-os $rtr1 Ubuntu1204-64-STD
set node1 [$ns node]
tb-set-node-os $node1 Ubuntu1204-64-STD
set node2 [$ns node]
tb-set-node-os $node2 Ubuntu1204-64-STD
set node3 [$ns node]
tb-set-node-os $node2 Ubuntu1204-64-STD
set node4 [$ns node]
tb-set-node-os $node2 Ubuntu1204-64-STD
# Links
set link0 [$ns duplex-link $rtr1 $node1 1000000.0kb 10.0ms DropTail]
tb-set-ip-link $rtr1 $link0 10.10.1.1
tb-set-ip-link $node1 $link0 10.10.1.2
# tb-set-link-loss $link0 0.2
set link1 [$ns duplex-link $rtr1 $node2 1000000.0kb 10.0ms DropTail]
tb-set-ip-link $rtr1 $link1 10.10.2.1
tb-set-ip-link $node2 $link1 10.10.2.2
set link2 [$ns duplex-link $rtr1 $node3 1000000.0kb 0.0ms DropTail]
tb-set-ip-link $rtr1 $link2 10.10.3.1
tb-set-ip-link $node3 $link2 10.10.3.2
set link3 [$ns duplex-link $rtr1 $node4 1000000.0kb 0.0ms DropTail]
tb-set-ip-link $rtr1 $link3 10.10.4.1
tb-set-ip-link $node4 $link3 10.10.4.2
$ns rtproto Manual
$ns run
```

The DETER topology of the Network is as shown below:



Learning Switch module on DETER

On one of the 4 nodes connected to Router (rtr1) we consider one as a Controller (Node 4 in our case). Before logging into that node, on user level (sc558bq) we download POX as described in Openflow tutorial (git clone http://github.com/noxrepo/pox)
Execute the following commands:

cd pox

./pox.py --verbose forwarding.l3_learning

Now the windows like this

```
sc558bq@node4:~$ cd pox
sc558bq@node4:~/pox$ ./pox.py --verbose forwarding.l3_learning
POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
DEBUG:core:POX 0.2.0 (carp) going up...
DEBUG:core:Running on CPython (2.7.6/Jun 22 2015 17:58:13)
DEBUG:core:Platform is Linux-3.13.0-62-generic-x86_64-with-Ubuntu-14.04-trusty
DEBUG:forwarding.l3_learning:Up...
INFO:core:POX 0.2.0 (carp) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
INFO:openflow.of_01:[00-25-90-6a-c9-75 1] connected
INFO:packet:(lldp tlv parse) warning TLV data too short to parse (84)
INFO:packet:(lldp parse) error parsing TLV
INFO:packet:(lldp tlv parse) warning TLV data too short to parse (85)
INFO:packet:(lldp parse) error parsing TLV
INFO:packet:(lldp tlv parse) warning TLV data too short to parse (85)
INFO:packet:(lldp parse) error parsing TLV
INFO:packet:(lldp tlv parse) warning TLV data too short to parse (84)
INFO:packet:(lldp parse) error parsing TLV
INFO:packet:(lldp tlv parse) warning TLV data too short to parse (84)
INFO:packet:(lldp parse) error parsing TLV
INFO:packet:(lldp tlv parse) warning TLV data too short to parse (85)
INFO:packet:(lldp parse) error parsing TLV
INFO:packet:(lldp tlv parse) warning TLV data too short to parse (85)
INFO:packet:(lldp parse) error parsing TLV
```

Openvswitch on DETER

In our deter topology, we deploy Openvswitch on Router (rtr1). Before we start with Openvswitch we need **sudo apt-get update** in case if we use bpc systems, but cpc systems don't need this step as they are already up-to-date.

sudo apt-get install openvswitch-switch

This command will install openvswitch in the node. To verify the installation check for the version using the command **ovs-vswitchd --version**

Now we check the ifconfig to know about the interfaces connected on this router.

```
sc558bq@rtr1:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr 00:25:90:6a:c9:74
          inet addr:192.168.0.32  Bcast:192.168.3.255  Mask:255.255.252.0
          inet6 addr: fe80::225:90ff:fe6a:c974/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:12866 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3198 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:16900052 (16.9 MB)  TX bytes:1470927 (1.4 MB)
          Memory:f7a80000-f7b00000

eth1      Link encap:Ethernet  HWaddr 00:25:90:6a:c9:75
          inet addr:10.10.2.1  Bcast:10.10.2.255  Mask:255.255.255.0
          inet6 addr: fe80::225:90ff:fe6a:c975/64 Scope:Link
```

```
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:6 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:468 (468.0 B)
          Memory:f7980000-f7a00000

eth2      Link encap:Ethernet  HWaddr a0:36:9f:0a:5c:bc
          inet addr:10.10.4.1  Bcast:10.10.4.255  Mask:255.255.255.0
          inet6 addr: fe80::a236:9fff:fe0a:5cbc/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:6 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:468 (468.0 B)
          Memory:f7400000-f7500000

eth3      Link encap:Ethernet  HWaddr a0:36:9f:0a:5c:bd
          inet addr:10.10.3.1  Bcast:10.10.3.255  Mask:255.255.255.0
          inet6 addr: fe80::a236:9fff:fe0a:5cbd/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:6 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:468 (468.0 B)
          Memory:f7300000-f7400000

eth4      Link encap:Ethernet  HWaddr a0:36:9f:0a:5c:be
          inet addr:10.10.1.1  Bcast:10.10.1.255  Mask:255.255.255.0
          inet6 addr: fe80::a236:9fff:fe0a:5cbe/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:6 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:468 (468.0 B)
          Memory:f7200000-f7300000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:104 errors:0 dropped:0 overruns:0 frame:0
          TX packets:104 errors:0 dropped:0 overruns:0 carrier:0
```

collisions:0 txqueuelen:0
RX bytes:11696 (11.6 KB) TX bytes:11696 (11.6 KB)

As mentioned above we considered Node 4 as Controller which has IP address of 10.10.4.2.

Configure the Software Switch (OVS Window)

We create an Ethernet bridge that will act as our software switch

sudo ovs-vsctl add-br br0

Add all the data interfaces to your switch (bridge).

sudo ovs-vsctl add-port br0 eth1
sudo ovs-vsctl add-port br0 eth2
sudo ovs-vsctl add-port br0 eth3
sudo ovs-vsctl add-port br0 eth4

```
sc558bq@rtr1:~$ sudo ovs-vsctl add-br br0
sc558bq@rtr1:~$ sudo  ovs-vsctl add-port br0 eth1
sc558bq@rtr1:~$ sudo  ovs-vsctl add-port br0 eth2
sc558bq@rtr1:~$ sudo  ovs-vsctl add-port br0 eth3
sc558bq@rtr1:~$ sudo  ovs-vsctl add-port br0 eth4
```

Use the below command to set the hwaddr of bridge:

sudo ovs-vsctl set bridge br0 other-config:hwaddr=00:15:17:57:c6:c8

Now we will remove the IP address of ethernet (eth2) connected to Controller and add this to the Bridge

sudo ifconfig eth2 0
sudo ifconfig br0 10.10.4.1 netmask 255.255.255.0

```
sc558bq@rtr1:~$ sudo ifconfig eth2 0
sc558bq@rtr1:~$ sudo ifconfig br0 10.10.4.1
```

Point your switch to a controller

An OpenFlow switch will not forward any packet unless instructed by a controller. Basically the forwarding table is empty, until an external controller inserts forwarding rules. The OpenFlow controller communicates with the switch over the control network and it can be anywhere in the Internet as long as it is reachable by the OVS host.

In order to point our software OpenFlow switch to the controller, in the ovs terminal window run this command:

sudo ovs-vsctl set-controller br0 tcp:10.10.4.2:6633

Now the Router is all set to function as an Openvswitch.
We will verify our OVS Configurations before we move further:

sudo ovs-vsctl show

```
sc558bq@rtr1:~$ sudo ovs-vsctl show
cdd73de0-6732-4389-bf21-281f6b12fef0
    Bridge "br0"
        Controller "tcp:10.10.4.2:6633"
            is_connected: true
        Port "eth3"
            Interface "eth3"
        Port "eth1"
            Interface "eth1"
        Port "eth4"
            Interface "eth4"
        Port "br0"
            Interface "br0"
                type: internal
        Port "eth2"
            Interface "eth2"
    ovs_version: "1.4.6"
sc558bq@rtr1:~$
```

After Configuration changes the ifconfig will look like this:

```
sc558bq@rtr1:~$ ifconfig
br0       Link encap:Ethernet  HWaddr 00:25:90:6a:c9:75
          inet addr:10.10.4.1  Bcast:10.255.255.255  Mask:255.0.0.0
          inet6 addr: fe80::225:90ff:fe6a:c975/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:11366825 errors:0 dropped:152 overruns:0 frame:0
          TX packets:359500 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:974902805 (974.9 MB)  TX bytes:842307258 (842.3 MB)

eth0      Link encap:Ethernet  HWaddr 00:25:90:6a:c9:74
          inet addr:192.168.0.32  Bcast:192.168.3.255  Mask:255.255.252.0
          inet6 addr: fe80::225:90ff:fe6a:c974/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:46672 errors:0 dropped:0 overruns:0 frame:0
          TX packets:60011 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:22344029 (22.3 MB)  TX bytes:7374562 (7.3 MB)
          Memory:f7a80000-f7b00000

eth1      Link encap:Ethernet  HWaddr 00:25:90:6a:c9:75
          inet addr:10.10.2.1  Bcast:10.10.2.255  Mask:255.255.255.0
          inet6 addr: fe80::225:90ff:fe6a:c975/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:49860578 errors:0 dropped:0 overruns:0 frame:0
          TX packets:11226270 errors:0 dropped:0 overruns:0 carrier:0
```

```
          collisions:0 txqueuelen:1000
          RX bytes:3014340001 (3.0 GB)  TX bytes:1014289338 (1.0 GB)
          Memory:f7980000-f7a00000

eth2      Link encap:Ethernet  HWaddr a0:36:9f:0a:5c:bc
          inet6 addr: fe80::a236:9fff:fe0a:5cbc/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:525567 errors:0 dropped:0 overruns:0 frame:0
          TX packets:11611648 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:322931470 (322.9 MB)  TX bytes:1522264116 (1.5 GB)
          Memory:f7400000-f7500000

eth3      Link encap:Ethernet  HWaddr a0:36:9f:0a:5c:bd
          inet addr:10.10.3.1  Bcast:10.10.3.255  Mask:255.255.255.0
          inet6 addr: fe80::a236:9fff:fe0a:5cbd/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:16785 errors:0 dropped:0 overruns:0 frame:0
          TX packets:10810491 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:3057306 (3.0 MB)  TX bytes:650584835 (650.5 MB)
          Memory:f7300000-f7400000

eth4      Link encap:Ethernet  HWaddr a0:36:9f:0a:5c:be
          inet addr:10.10.1.1  Bcast:10.10.1.255  Mask:255.255.255.0
          inet6 addr: fe80::a236:9fff:fe0a:5cbe/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1492221497 errors:0 dropped:0 overruns:41096563 frame:0
          TX packets:50676 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:94831088279 (94.8 GB)  TX bytes:3129615 (3.1 MB)
          Memory:f7200000-f7300000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:104 errors:0 dropped:0 overruns:0 frame:0
          TX packets:104 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:11696 (11.6 KB)  TX bytes:11696 (11.6 KB)
```

Configuring Hosts on Deter

On all the 3 remaining deter nodes we update the routing tables so that they all point to Router (rtr1).
Initial Routing Tables on hosts is:

```
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         192.168.1.254   0.0.0.0         UG    0      0        0 eth0
10.10.1.0       0.0.0.0         255.255.255.0   U     0      0        0 eth4
192.168.0.0     0.0.0.0         255.255.252.0   U     0      0        0 eth0
```

sudo route add default gw 10.10.X.1 // X is 1,2 and 3 in our case.
sudo route add -host 192.168.253.1 gw 192.168.1.254
sudo route del default gw 192.168.1.254

After using these commands our routing table will look like

```
 Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         10.10.1.1       0.0.0.0         UG    0      0        0 eth4
10.10.1.0       0.0.0.0         255.255.255.0   U     0      0        0 eth4
192.168.0.0     0.0.0.0         255.255.252.0   U     0      0        0 eth0
192.168.253.1   192.168.1.254   255.255.255.255 UGH   0      0        0 eth0
```

Pinging Host1 (Node1) to Host2 (Node2)

Before checking the ping test we check the flow table on openvswitch as shown:

```
sc558bq@rtr1:~$ sudo ovs-ofctl dump-flows br0
NXST_FLOW reply (xid=0x4):
```

Now we run ping test on Host1 as shown:

```
sc558bq@node1:~$ ping -c 10 10.10.2.2
PING 10.10.2.2 (10.10.2.2) 56(84) bytes of data.

--- 10.10.2.2 ping statistics ---
10 packets transmitted, 0 received, 100% packet loss, time 9072ms
```

When we do the ping test for the first time, the Controller doesn't know about the details of Host2 to send the packets to destination. At this point, the controller captures the MAC details of Host1 (sender).

Transcripts at the Controller end are as shown:

```
DEBUG:forwarding.l3_learning:161336707445 1 ARP request 192.168.0.21 => 10.10.2.
1
```

DEBUG:forwarding.l3_learning:161336707445 1 learned 192.168.0.21
DEBUG:forwarding.l3_learning:161336707445 1 flooding ARP request 192.168.0.21 => 10.10.2.1
DEBUG:forwarding.l3_learning:161336707445 65534 ARP reply 10.10.2.1 => 192.168.0.21
DEBUG:forwarding.l3_learning:161336707445 65534 learned 10.10.2.1
DEBUG:forwarding.l3_learning:161336707445 65534 flooding ARP reply 10.10.2.1 => 192.168.0.21
DEBUG:forwarding.l3_learning:161336707445 1 IP 192.168.0.21 => 192.168.252.1
INFO:packet:(lldp tlv parse) warning TLV data too short to parse (85)
INFO:packet:(lldp parse) error parsing TLV
INFO:packet:(lldp tlv parse) warning TLV data too short to parse (85)
INFO:packet:(lldp parse) error parsing TLV
DEBUG:forwarding.l3_learning:161336707445 4 ARP request 10.10.1.2 => 10.10.1.1
DEBUG:forwarding.l3_learning:161336707445 4 learned 10.10.1.2
DEBUG:forwarding.l3_learning:161336707445 4 flooding ARP request 10.10.1.2 => 10.10.1.1
DEBUG:forwarding.l3_learning:161336707445 65534 ARP reply 10.10.1.1 => 10.10.1.2
DEBUG:forwarding.l3_learning:161336707445 65534 learned 10.10.1.1
DEBUG:forwarding.l3_learning:161336707445 65534 flooding ARP reply 10.10.1.1 => 10.10.1.2
DEBUG:forwarding.l3_learning:161336707445 4 IP 10.10.1.2 => 10.10.2.2

Similarly if we do ping from host2 then ping test will show successful results as shown:

```
sc558bq@node2:~$ ping -c 10 10.10.1.2
PING 10.10.1.2 (10.10.1.2) 56(84) bytes of data.
64 bytes from 10.10.1.2: icmp_req=1 ttl=64 time=103 ms
64 bytes from 10.10.1.2: icmp_req=2 ttl=64 time=41.0 ms
64 bytes from 10.10.1.2: icmp_req=3 ttl=64 time=40.8 ms
64 bytes from 10.10.1.2: icmp_req=4 ttl=64 time=40.9 ms
64 bytes from 10.10.1.2: icmp_req=5 ttl=64 time=40.8 ms
64 bytes from 10.10.1.2: icmp_req=6 ttl=64 time=40.8 ms
64 bytes from 10.10.1.2: icmp_req=7 ttl=64 time=40.8 ms
64 bytes from 10.10.1.2: icmp_req=8 ttl=64 time=40.9 ms
64 bytes from 10.10.1.2: icmp_req=9 ttl=64 time=40.8 ms
64 bytes from 10.10.1.2: icmp_req=10 ttl=64 time=40.9 ms

--- 10.10.1.2 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9016ms
rtt min/avg/max/mdev = 40.879/47.202/103.760/18.854 ms
sc558bq@node2:~$
```

Now if we do the ping once again from host1, this time it will successful because the controller learnt about host2 in the previous ping from host2.

```
sc558bq@node1:~$ ping -c 10 10.10.2.2
PING 10.10.2.2 (10.10.2.2) 56(84) bytes of data.

--- 10.10.2.2 ping statistics ---
10 packets transmitted, 0 received, 100% packet loss, time 9072ms

sc558bq@node1:~$ ping -c 10 10.10.2.2
PING 10.10.2.2 (10.10.2.2) 56(84) bytes of data.
64 bytes from 10.10.2.2: icmp_req=1 ttl=64 time=94.8 ms
64 bytes from 10.10.2.2: icmp_req=2 ttl=64 time=41.0 ms
64 bytes from 10.10.2.2: icmp_req=3 ttl=64 time=40.9 ms
64 bytes from 10.10.2.2: icmp_req=4 ttl=64 time=40.9 ms
64 bytes from 10.10.2.2: icmp_req=5 ttl=64 time=40.9 ms
64 bytes from 10.10.2.2: icmp_req=6 ttl=64 time=40.8 ms
64 bytes from 10.10.2.2: icmp_req=7 ttl=64 time=40.9 ms
64 bytes from 10.10.2.2: icmp_req=8 ttl=64 time=40.8 ms
64 bytes from 10.10.2.2: icmp_req=9 ttl=64 time=40.8 ms
64 bytes from 10.10.2.2: icmp_req=10 ttl=64 time=40.9 ms

--- 10.10.2.2 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9016ms
rtt min/avg/max/mdev = 40.856/46.319/94.868/16.183 ms
```

Now if we check the flow table on Openvswitch is looks like this:

```
sc558bq@rtr1:~$ sudo ovs-ofctl dump-flows br0
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=5.727s, table=0, n_packets=6, n_bytes=588, idle_timeout=10
,priority=65535,icmp,in_port=4,vlan_tci=0x0000,dl_src=a0:36:9f:0a:5f:9a,dl_dst=0
0:25:90:6a:c9:75,nw_src=10.10.1.2,nw_dst=10.10.2.2,nw_tos=0,icmp_type=8,icmp_cod
e=0 actions=mod_dl_dst:a0:36:9f:0a:5e:a6,output:1
 cookie=0x0, duration=5.694s, table=0, n_packets=6, n_bytes=588, idle_timeout=10
,priority=65535,icmp,in_port=1,vlan_tci=0x0000,dl_src=a0:36:9f:0a:5e:a6,dl_dst=0
0:25:90:6a:c9:75,nw_src=10.10.2.2,nw_dst=10.10.1.2,nw_tos=0,icmp_type=0,icmp_cod
e=0 actions=mod_dl_dst:a0:36:9f:0a:5f:9a,output:4
sc558bq@rtr1:~$
```

Note: l3_learning switch works at Layer 3 and thus PING can be tested correctly.
For performing file transfer using our custom packet, we are modifying l2_learning switch as our custom packet does not contain OSI headers.

C. RESULTS

1.Single Transmission (Node1 to Node2)

Controller Window:

DEBUG:forwarding.l2_learning:installing flow for 00:04:23:bb:10:aa.2 -> 0:04:23:bb:1f:61.1
DEBUG:forwarding.l2_learning:installing flow for 00:04:23:bb:10:aa.2 -> 00:04:23:bb:1f:61.1
DEBUG:forwarding.l2_learning:installing flow for 00:04:23:bb:1f:61.1 -> 00:04:23:bb:10:aa.2

DEBUG:forwarding.l2_learning:installing flow for 00:04:23:bb:1f:61.1 -> 00:04:23:bb:10:aa.2

Flow Table at Switch

cookie=0x0, duration=7.672s, table=0, n_packets=106216, n_bytes=152100312, idle_timeout=10,hard_timeout=60,priority=65535,in_port=1,vlan_tci=0x0000,dl_src=00:04:23:bb:1f:61,dl_dst=00:04:23:bb:10:aa,dl_type=0x05ff actions=output:2

cookie=0x0, duration=6.436s, table=0, n_packets=33419, n_bytes=2339330, idle_timeout=10,hard_timeout=60,priority=65535,in_port=2,vlan_tci=0x0000,dl_src=00:04:23:bb:10:aa,dl_dst=00:04:23:bb:1f:61,dl_type=0x05ff actions=output:1

cookie=0x0, duration=8.521s, table=0, n_packets=1, n_bytes=42, idle_timeout=10,hard_timeout=60,priority=65535,arp,in_port=65534,vlan_tci=0x0000,dl_src=00:04:23:bb:1c:ce,dl_dst=00:04:23:bb:1f:61,nw_src=10.10.1.1,nw_dst=10.10.1.2,arp_op=2 actions=output:1

cookie=0x0, duration=2.928s, table=0, n_packets=1, n_bytes=42, idle_timeout=10,hard_timeout=60,priority=65535,arp,in_port=65534,vlan_tci=0x0000,dl_src=00:04:23:bb:1c:ce,dl_dst=00:04:23:bb:10:aa,nw_src=10.10.2.1,nw_dst=10.10.2.2,arp_op=2 actions=output:2

cookie=0x0, duration=2.969s, table=0, n_packets=1, n_bytes=60, idle_timeout=10,hard_timeout=60,priority=65535,arp,in_port=2,vlan_tci=0x0000,dl_src=00:04:23:bb:10:aa,dl_dst=00:04:23:bb:1c:ce,nw_src=10.10.2.2,nw_dst=10.10.2.1,arp_op=1 actions=LOCAL

Sender Window (Filesize ~100MB):

Csc558ar@node1:/tmp/lab10/ftsudo ./sender /tmp/data100.bin node2 12
sudo: unable to resolve host node1.route-on-4.usc558l.isi.deterlab.net
Find iface eth1, ip 10.10.1.2
Trying to open device eth1 to send ... OPEN DONE
Filesize is 104857600

TOTAL PACKETS TRANSMITTED: 74899
TIME FOR DATA TRANSMISSION: 9.048489

Receiver Window:

sc558ar@node2:/tmp/lab10/ft$ sudo ./receiver 12 test.bin 104857600
sudo: unable to resolve host node2.route-on-4.usc558l.isi.deterlab.net
Find iface eth0, ip 10.10.2.2
Trying to open device eth0 to receive ... OPEN DONE
FILE WRITING DONE
SEND END

Throughput Calculation:11.052 MB/s

2. Full Duplex Transmission (Node1/Node2/Node3)

To perform full duplex, we are using round robin mechanism to send packets to multiple nodes. In the same process, we send each packet to two other nodes at different times. This helps to deal with the collision of packets that might occur at the sending node, switch or the receiving node. At the receiver, we receive packets from the other two nodes and differentiate between them using the port number of our reliable custom header.

```
sudo ./sender_n /tmp/data10.bin node2 node3
sudo ./sender_n /tmp/data10.bin node3 node1
sudo ./sender_n /tmp/data10.bin node1 node2

sudo ./receiver_n test1.bin test2.bin 10485760
```

Flow table at Switch Window

Below, we show the six different flows that were inserted during the full duplex transmission between 3 nodes in bold.

```
sc558ar@rtr1:~$ sudo ovs-ofctl dump-flows br0
NXST_FLOW reply (xid=0x4):
cookie=0x0, duration=3.784s, table=0, n_packets=6575, n_bytes=9415400,
idle_timeout=10,hard_timeout=60,priority=65535,in_port=2,vlan_tci=0x0000,dl_src=00:04:23:c7:cb
:dc,dl_dst=00:18:8b:41:60:15,dl_type=0x05ff actions=output:3
 cookie=0x0, duration=5.885s, table=0, n_packets=7324, n_bytes=10487728,
idle_timeout=10,hard_timeout=60,priority=65535,in_port=1,vlan_tci=0x0000,dl_src=00:04:23:bb:1c
:24,dl_dst=00:18:8b:41:60:15,dl_type=0x05ff actions=output:3
 cookie=0x0, duration=3.784s, table=0, n_packets=6573, n_bytes=9412536,
idle_timeout=10,hard_timeout=60,priority=65535,in_port=2,vlan_tci=0x0000,dl_src=00:04:23:c7:cb
:dc,dl_dst=00:04:23:bb:1c:24,dl_type=0x05ff actions=output:1
 cookie=0x0, duration=0.756s, table=0, n_packets=1, n_bytes=1432,
idle_timeout=10,hard_timeout=60,priority=65535,in_port=3,vlan_tci=0x0000,dl_src=00:18:8b:41:60
:15,dl_dst=00:04:23:c7:cb:dc,dl_type=0x05ff actions=output:2
 cookie=0x0, duration=0.793s, table=0, n_packets=1, n_bytes=1432,
idle_timeout=10,hard_timeout=60,priority=65535,in_port=3,vlan_tci=0x0000,dl_src=00:18:8b:41:60
:15,dl_dst=00:04:23:bb:1c:24,dl_type=0x05ff actions=output:1
 cookie=0x0, duration=5.848s, table=0, n_packets=7324, n_bytes=10487728,
idle_timeout=10,hard_timeout=60,priority=65535,in_port=1,vlan_tci=0x0000,dl_src=00:04:23:bb:1c
:24,dl_dst=00:04:23:c7:cb:dc,dl_type=0x05ff actions=output:2
 cookie=0x0, duration=8.434s, table=0, n_packets=2, n_bytes=162,
idle_timeout=10,hard_timeout=60,priority=65535,udp,in_port=3,vlan_tci=0x0000,dl_src=00:18:8b:41
:60:15,dl_dst=00:04:23:bb:25:96,nw_src=10.10.3.2,nw_dst=192.168.252.1,nw_tos=0,tp_src=35190,tp
_dst=53 actions=LOCAL
```

cookie=0x0, duration=10.252s, table=0, n_packets=1, n_bytes=42, idle_timeout=10,hard_timeout=60,priority=65535,arp,in_port=65534,vlan_tci=0x0000,dl_src=00:04:23:bb:25:96,dl_dst=00:04:23:c7:cb:dc,nw_src=10.10.2.1,nw_dst=10.10.2.2,arp_op=2 actions=output:2
cookie=0x0, duration=10.255s, table=0, n_packets=1, n_bytes=60, idle_timeout=10,hard_timeout=60,priority=65535,arp,in_port=2,vlan_tci=0x0000,dl_src=00:04:23:c7:cb:dc,dl_dst=00:04:23:bb:25:96,nw_src=10.10.2.2,nw_dst=10.10.2.1,arp_op=1 actions=LOCAL
cookie=0x0, duration=3.452s, table=0, n_packets=1, n_bytes=42, idle_timeout=10,hard_timeout=60,priority=65535,arp,in_port=65534,vlan_tci=0x0000,dl_src=00:04:23:bb:25:96,dl_dst=00:18:8b:41:60:15,nw_src=10.10.3.1,nw_dst=10.10.3.2,arp_op=2 actions=output:3
cookie=0x0, duration=3.454s, table=0, n_packets=1, n_bytes=60, idle_timeout=10,hard_timeout=60,priority=65535,arp,in_port=3,vlan_tci=0x0000,dl_src=00:18:8b:41:60:15,dl_dst=00:04:23:bb:25:96,nw_src=10.10.3.2,nw_dst=10.10.3.1,arp_op=1 actions=LOCAL

Controller Window

```
DEBUG:forwarding.l2_learning:installing flow for 00:04:23:bb:1c:24.1 -> 00:18:8b:41:60:15.3
DEBUG:forwarding.l2_learning:installing flow for 00:04:23:bb:1c:24.1 -> 00:04:23:c7:cb:dc.2
DEBUG:forwarding.l2_learning:installing flow for 00:04:23:bb:1c:24.1 -> 00:18:8b:41:60:15.3
DEBUG:forwarding.l2_learning:installing flow for 00:04:23:bb:1c:24.1 -> 00:18:8b:41:60:15.3
DEBUG:forwarding.l2_learning:installing flow for 00:04:23:bb:1c:24.1 -> 00:18:8b:41:60:15.3
DEBUG:forwarding.l2_learning:installing flow for 00:18:8b:41:60:15.3 -> 00:04:23:bb:1c:24.1
DEBUG:forwarding.l2_learning:installing flow for 00:18:8b:41:60:15.3 -> 00:04:23:c7:cb:dc.2
DEBUG:forwarding.l2_learning:installing flow for 00:18:8b:41:60:15.3 -> 00:04:23:bb:1c:24.1
DEBUG:forwarding.l2_learning:installing flow for 00:18:8b:41:60:15.3 -> 00:04:23:c7:cb:dc.2
DEBUG:forwarding.l2_learning:installing flow for 00:18:8b:41:60:15.3 -> 00:04:23:bb:1c:24.1
DEBUG:forwarding.l2_learning:installing flow for 00:18:8b:41:60:15.3 -> 00:04:23:c7:cb:dc.2
DEBUG:forwarding.l2_learning:installing flow for 00:18:8b:41:60:15.3 -> 00:04:23:bb:1c:24.1
```

Sender Window Node1

```
sc558ar@node1:/tmp/lab10/ft$ sudo ./sender_n /tmp/data10.bin node2 node3
sudo: unable to resolve host node1.route-on-5.usc558l.isi.deterlab.net
Find iface eth0, ip 10.10.1.2
Trying to open device eth0 to send ... OPEN DONE
Filesize is 10485760
Current time: Fri Oct 30 19:06:18 2015
Sending ...

ALL DONE
```

Sender Window Node2

```
sc558ar@node2:/tmp/lab10/ft$ sudo ./sender_n /tmp/data10.bin node3 node1
sudo: unable to resolve host node2.route-on-5.usc558l.isi.deterlab.net
Find iface eth0, ip 10.10.2.2
Trying to open device eth0 to send ... OPEN DONE
Filesize is 10485760
Current time: Fri Oct 30 19:06:19 2015
Sending ...

ALL DONE
```

Sender Window Node3

```
sc558ar@node3:/tmp/lab10/ft$ sudo ./sender_n /tmp/data10.bin node1 node2
sudo: unable to resolve host node3.route-on-5.usc558l.isi.deterlab.net
Find iface eth5, ip 10.10.3.2
Trying to open device eth5 to send ... OPEN DONE
Filesize is 10485760
Current time: Fri Oct 30 19:06:20 2015
Sending ...

ALL DONE
```

Receiver window Node1

```
sc558ar@node1:/tmp/lab10/ft$ sudo ./receiver_n file1.bin file2.bin 10485760
sudo: unable to resolve host node1.route-on-5.usc558l.isi.deterlab.net
Find iface eth0, ip 10.10.1.2
Trying to open device eth0 to receive ... OPEN DONE
FILE 2 WRITING DONE
Current time: Fri Oct 30 19:06:23 2015
FILE 1 WRITING DONE
Current time: Fri Oct 30 19:06:24 2015
```

Throughput Calculation for Stream 1:2.364 MB/s
Throughput Calculation for Stream 2: 2.5MB/s

Receiver Window Node2

```
sc558ar@node2:/tmp/lab10/ft$ sudo ./receiver_n file1.bin file2.bin 10485760
sudo: unable to resolve host node2.route-on-5.usc558l.isi.deterlab.net
Find iface eth0, ip 10.10.2.2
Trying to open device eth0 to receive ... OPEN DONE
FILE 1 WRITING DONE
Current time: Fri Oct 30 19:06:22 2015
FILE 2 WRITING DONE
Current time: Fri Oct 30 19:06:24 2015
```

Throughput Calculation for Stream 3: 3.33MB/s
Throughput Calculation for Stream 4: 2.85MB/s

Receive Window Node3

```
sc558ar@node3:/tmp/lab10/ft$ sudo ./receiver_n file1.bin file2.bin 10485760
sudo: unable to resolve host node3.route-on-5.usc558l.isi.deterlab.net
Find iface eth5, ip 10.10.3.2
Trying to open device eth5 to receive ... OPEN DONE
FILE 2 WRITING DONE
Current time: Fri Oct 30 19:06:22 2015
FILE 1 WRITING DONE
Current time: Fri Oct 30 19:06:23 2015
```

Throughput Calculation for Stream 5: 2.25MB/s.
Throughput Calculation for Stream 6: 2.42MB/s.

3. Conclusion

We are able to perform full-duplex transmission successfully with significant throughput. We are using a round-robin mechanism to send packets to multiple nodes.

D. SELF-DEFINED EXPERIMENT: FLOW REDIRECTION

In self-defined experiment, we would like to simulate a situation that would take advantage of the flexibility of OpenFlow. In a data center, a server might be down suddenly. As for redundancy, or disaster recovery, there could be some alternative server that takes its responsibility. However, as the clients (end users) might not be noticed that the server they are connecting to is down, they might still send packets to the broken server. With SDN, we can let the controller perform a flow redirection: to redirect the flows to the alternative server, and therefore the client would not notice that its server is down.

We did the experiment in our old topology. Assume node1 is sending packets to node2, and node2 is down. The controller should redirect packets sent from node1 to node3. As the controller is making the switch like a L2-learning switch, it only tells the switch to modify the flow's destination MAC address, and the receiver can still use our self-defined protocol for routing and other actions.

1. Experiment Setup

Node 1 sends 1000 packets to node2. Both node2 and node3 starts the receiver code. The expected result is that the receiver on node2 receives none of the 1000 packets, but the receiver on node3 receives everything.

After the receiver receives all 1000 packets, it will send an ACK back.

As we implemented checksum in our self-defined packet headers, we reject any packet whose checksum does not match. So whatever receiver receives should be whatever sender sends.

2. Modify Controller

We make the controller act as flow-redirect:

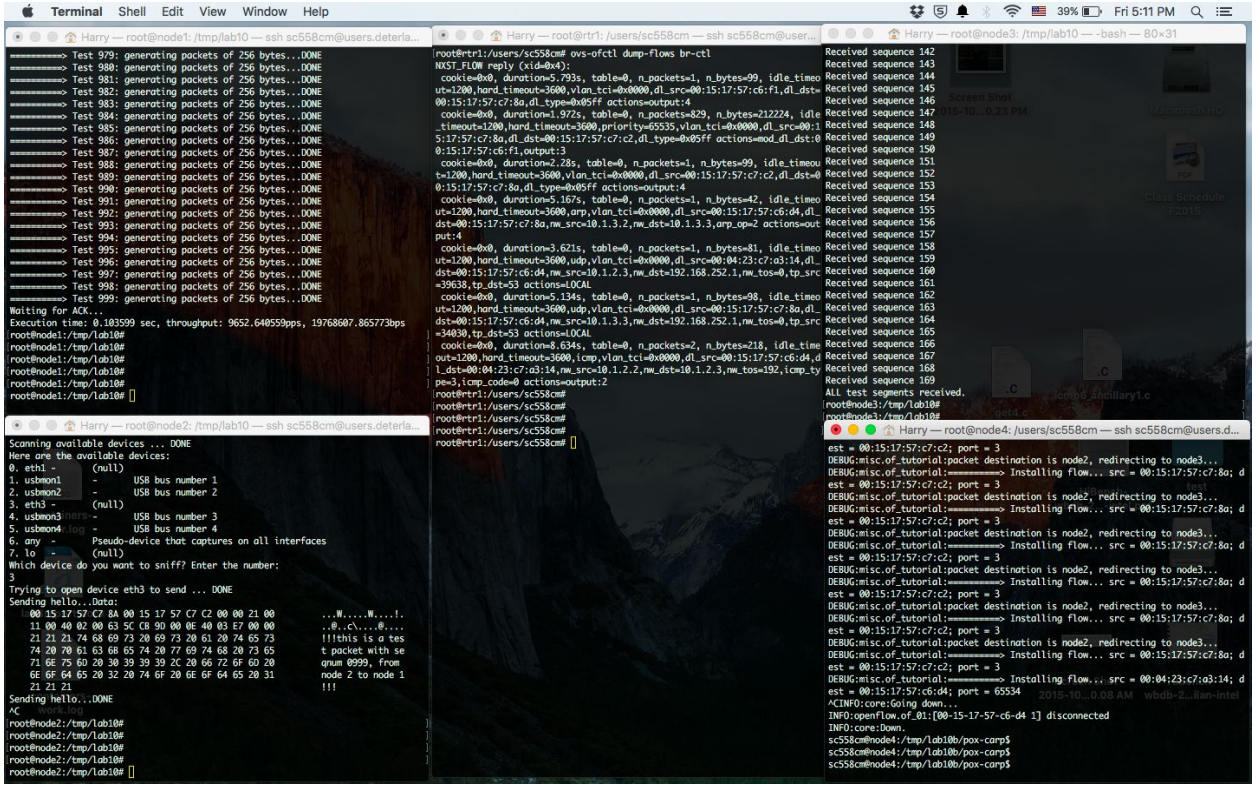
```
def flow_redirect (self, packet, packet_in):
    if packet.src not in self.mac_to_port:
      log.debug("==========> adding mac_to_port_entry: src = " + str(packet.src) + "; 
in_port = " + str(packet_in.in_port))
      self.mac_to_port[packet.src] = packet_in.in_port

    if packet.dst == EthAddr("00:15:17:57:c7:c2"):
      log.debug("packet destination is node2, redirecting to node3...")
      msg = of.ofp_flow_mod()
      msg.match = of.ofp_match.from_packet(packet)
      msg.idle_timeout = 1200
      msg.hard_timeout = 3600
      msg.buffer_id = packet_in.buffer_id
      msg.priority = 65535
      msg.actions.append(of.ofp_action_dl_addr.set_dst(EthAddr("00:15:17:57:c6:f1")))
      msg.actions.append(of.ofp_action_output(port = 3))
      # msg.data = event.ofp
      log.debug("==========> Installing flow... src = " + str(packet.src) + "; dest = 
" + str(packet.dst) + "; port = 3")
      #packet.dst = EthAddr("00:15:17:57:c6:f1")
```

```
        self.connection.send(msg)
        # self.resend_packet(packet_in, 3)
    elif packet.dst in self.mac_to_port:
        log.debug("==========> Installing flow... src = " + str(packet.src) + "; dest =
" + str(packet.dst) + "; port = " + str(self.mac_to_port[packet.dst]))
        msg = of.ofp_flow_mod()
        msg.match = of.ofp_match.from_packet(packet)
        msg.idle_timeout = 1200
        msg.hard_timeout = 3600
        msg.buffer_id = packet_in.buffer_id
        msg.actions.append(of.ofp_action_output(port = self.mac_to_port[packet.dst]))
        self.connection.send(msg)
    else:
        self.resend_packet(packet_in, of.OFPP_ALL)
```

The highlighted line adds a mod-flow action to the controller

3. Experiment Output



Topleft: node1; Bottomleft: node2; Topright: node3; Bottomright: node4; Middle: router

a. We can see that node1 sent all the packets, node3 receives all the packets, and node2 receives nothing, as expected.
b. The controller has printouts showing that it sees packets with destination to node2, and is injecting redirection flow to node3

```
DEBUG:misc.of_tutorial:packet destination is node2, redirecting to node3...
DEBUG:misc.of_tutorial:==========> Installing flow... src = 00:15:17:57:c7:8a; dest =
00:15:17:57:c7:c2; port = 3
```

c. When we do a dump-flows on the router we can see the corresponding flow:

```
cookie=0x0, duration=1.972s, table=0, n_packets=829, n_bytes=212224,
idle_timeout=1200,hard_timeout=3600,priority=65535,vlan_tci=0x0000,dl_src=00:15:17:57:
c7:8a,dl_dst=00:15:17:57:c7:c2,dl_type=0x05ff
actions=mod_dl_dst:00:15:17:57:c6:f1,output:3
```

Note: node1 has mac address `00:15:17:57:c7:8a` ; node2 has mac address `00:15:17:57:c7:c2` ; node3 has mac address `00:15:17:57:c6:f1`

d. The receiver is not receiving packets in order, which is expected with Open vSwitch

4. Performance

Due to the limited way we can modify the flow behavior in the switch, we did a dummy performance calculation: we put a timestamp in sender just before it sends packets, and another timestamp after it receives ACK from receiver. It calculates time spent, packet-per-second, and bits-per-second.

```
Execution time: 0.103599 sec, throughput: 9652.640559pps, 19768607.865773bps
```

Note that this performance calculation is very dummy as it takes packet generation time into consideration and it is not pure flow-redirecting throughput (on 100Mbps link).

5. Conclusion

This self-designed experiment showed two things:

a. Our self-designed protocol is fully compatible with OpenFlow
b. Router is functioning with not only packet forwarding, but can also do flow-redirecting