MINT 709 PROJECT REPORT

Optimal traffic load balancing algorithm for aggregated Ethernet links on Open vSwitch platform

Prepared By:

Sandesh Shrestha

Supervised By:

Mr. Pete Nanda

MSc, P. Eng, Sr. Engineer, TELUS

Master of Science In Internetworking

MINT 709 – Capstone Project

|

|

|

University of Alberta

Edmonton,Alberta,Canada

Table of Contents

Problem Description

Link Load balancing is major functionality to be performed by any switch. It is the process of making the decision of sending the packet to the link in such a way that the aggregated links carry fairly equal amount of traffic at all times. The 802.3 ad load balancing algorithm commonly used in today's servers does not take into account the traffic flow through the interfaces. It does not have any intelligence to check the amount of traffic flowing through the interfaces. This project addresses this issue by adding intelligence to the switch to check the traffic flow and take necessary action when it reaches a fixed limit.

Solution

Open vSwitch is an open source OpenFlow capable virtual switch that is typically used with hypervisors to interconnect virtual machines within a host and virtual machines between hosts across networks. In addition to it, it supports standard management interfaces and protocols (eg. NetFlow, sFlow, IPFIX, RSPAN, CLI, LACP, 802.1ag). It also supports distribution across multiple physical servers similar to VMWare's vNetwork distributed vswitch or Cisco's Nexus 1000V.

It differs from standard switches as it does not have a network operating system embedded to it. This lack of decision making engine in open vswitch is compensated by making it programmable using a remote or local controller. This feature of open vswitch makes it a multi-purpose switch which can work not only on layer 2 but also across layer 3 and 4. Not only this, it can be programmed to work as a router, firewall, IDS depending upon requirement.

The solution to the problem described above includes programming the controller to make decisions to send the traffic out one of the aggregated links of Open vSwitch. Traffic flow through the links is monitored at regular intervals. Depending upon the traffic movement, flows are adjusted for optimum traffic flow in each of the links. On events like link failure or interface down, the traffic is directed through one of the other aggregated links. Doing so maintains fairly equal amount of traffic on each of the aggregated links and adjusts traffic flow in case of link failure and high traffic thus achieving an optimum load balancing. On the other hand, when the traffic flow is normal, this solution directs the packets in such a way that the flow is balanced among the aggregated links.

Solution Setup

The setup for the solution includes the following components:

1. **Mininet:** Mininet creates a realistic virtual switch, running real kernel, switch and application code on a single machine in seconds with a single command. It also provides APIs to create complex network topologies which is not possible with standard mininet commands. This project uses those APIs as the topology required for this project cannot be created with a standard mininet command. Mininet provides environment to simulate the traffic flow using hosts and switches in a single VM.

2. **Ryu Controller:** Ryu is a component-based software defined networking framework. It provides software components with well defined API to make it easier for developers to build new network management and control applications. Ryu supports various protocols for managing network devices such as OpenFlow, NetConf, OF-Config. The protocol that is used for this project is OpenFlow. Ryu is used in this project to analyse the utilization statistics of ovs and re-define the path on the fly.

   There are many other controllers that can achieve the same functionality. They are POX, Trema, OpenDayLight, Floodlight. The reason for choosing Ryu is that it uses Python which has a less steep learning curve. Also, the online documentation available for Ryu is easier to understand thus we can focus more on the implementation rather than understanding the nuances of Ryu.

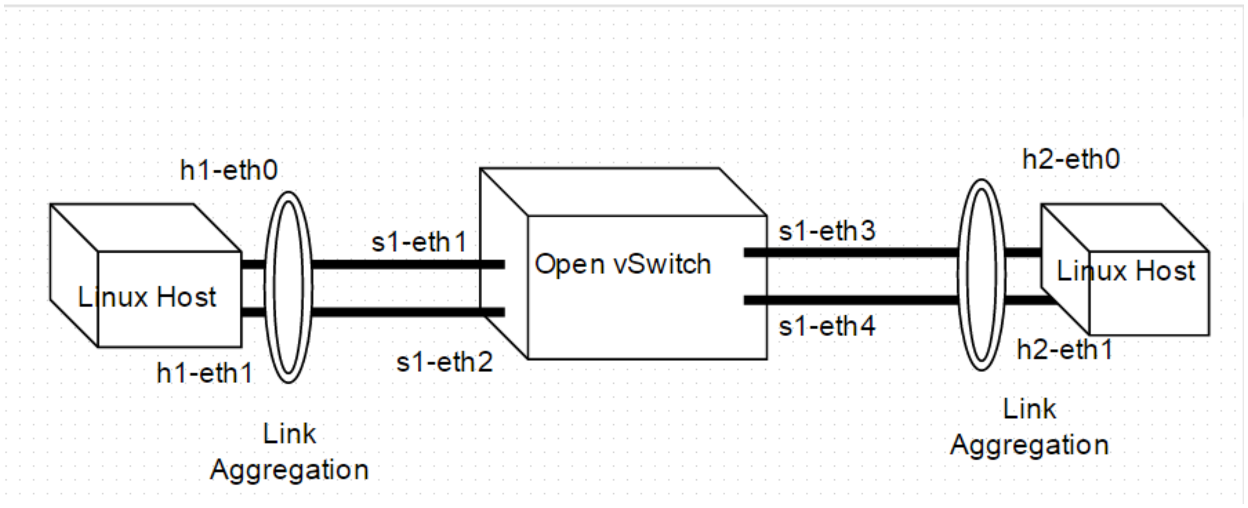
3. **Open vSwitch:** Open vSwitch is the main component of this project as this project is all about optimal load balancing on Open vSwitch. Open vSwitch is an open source OpenFlow capable virtual switch that is typically used with hypervisors to interconnect virtual machines within a host and virtual machines between hosts across networks It differs from standard switches as it does not have an operating system along with it. This lack of decision making

engine in open vswitch is compensated by making it programmable using a remote or local controller.

4. **Linux Hosts:** Mininet uses ubuntu hosts to simulate a host. The host and the open vswitch linux hosts share the filesystem so changes made to filesystem in open vswitch are reflected in the host.

5. **Virtualbox:** Virtualbox is a powerful virtualization platform for x86 and Intel64/AMD64 architecture. It is a professional solution and is available as an Open Source Software under the terms of GNU General Public License (GPL v2.0).Mininet runs on virtualbox as a VM. Due to the lack of graphical interface in Mininet, Xming is used to generate graphical interface required to provide graphical interface in host.

Network Topology



Test Procedure:

The test procedure include the following steps:

1. **L2 Switch Functionality:** Just like a standard switch, the primary function of Open vSwitch is to perform L2 switching. However, unlike normal switch, it does not have its own capability to perform switching. OVS needs to be programmed for L2 switching.

L2 Switching in OVS:

The first packet that comes to controller goes to the controller. The controller contains the logic to deal with the packet. Whenever a packet comes to OVS, they are matched against the flows. If there is no match, the packet is sent to controller. In this project, logic is implemented to store the mac- address and incoming port number of the incoming

```
def __init__(self, *args, **kwargs):
    super(SimpleSwitch, self).__init__(*args, **kwargs)
    self.mac_to_port = {}
```

packet and then flood the packet out all interfaces except input port. Next time, when there is a packet that is destined for that mac-address, the packet is send out through the interface that is stored in the controller. Also, a flow is added to the OVS, so that packet forwarding can then be performed solely by the ovs, without sending to the controller.

```
if dst in self.mac_to_port[dpid]:
        out_port = self.mac_to_port[dpid][dst]
    else:
        out_port = ofproto.OFPP_FLOOD

    actions = [parser.OFPActionOutput(out_port)]

    # install a flow to avoid packet_in next time
    if out_port != ofproto.OFPP_FLOOD:
        match = parser.OFPMatch(in_port=in_port, eth_dst=dst)
```

Testing L2 Switching:

With the topology as shown in the network topology section above, L2 switching can be considered to be implemented if h1 can ping h2 and vice-versa. The ping results are shown below.

2. Getting statistics from Open vSwitch:

Ryu framework provides API to get the statistics from open vswitch. This statistics is used to take decisions to change the out port thus achieving optimum load balance.

The code below gets the port details from open vswitch.

```
@set_ev_cls(ofp_event.EventOFPPortStatsReply, MAIN_DISPATCHER)

    def _port_stats_reply_handler(self, ev):

        body = ev.msg.body

        for stat in sorted(body, key=attrgetter('port_no')):

            self.logger.info('%016x %8x %8d %8d %8d %8d %8d %8d',

                             ev.msg.datapath.id, stat.port_no,
                             stat.rx_packets, stat.rx_bytes, stat.rx_errors,

                             stat.tx_packets, stat.tx_bytes, stat.tx_errors)
```

The code below fetches the flow details from open vswitch.

```
@set_ev_cls(ofp_event.EventOFPFlowStatsReply, MAIN_DISPATCHER)

    def _flow_stats_reply_handler(self, ev):

        body = ev.msg.body

        for stat in sorted([flow for flow in body if flow.priority == 1],

                           key=lambda flow: (flow.match['in_port'],

                                             flow.match['eth_dst'])):

            self.logger.info('%016x %8x %17s %8x %8d %8d',

                             ev.msg.datapath.id,

                             stat.match['in_port'], stat.match['eth_dst'],

                             stat.instructions[0].actions[0].port,

                             stat.packet_count, stat.byte_count)
```

As seen in the above code, we can get all the required statistics from open vswitch. By getting these statistics at a regular interval, we can ,

i. Know the status of the ports, whether they are up or they have gone down

ii. Know the amount of traffic that has traversed since the last the statistics was read.

iii. Know the flows in the open vswitch, which should be changed, depending on the status of the ports and traffic movement.

A typical statistics fetched from open vswitch is shown below:

Flow Statistics:

```
datapath         in-port  eth-dst          out-port packets  bytes
---------------- -------- ----------------- -------- -------- --------
0000000000000001        1 00:00:00:00:00:02        4       19     1806
0000000000000001        3 00:00:00:00:00:01        2       20     1904
```

Port Statistics

```
datapath         port     rx-pkts  rx-bytes rx-error tx-pkts  tx-bytes tx-error
---------------- -------- -------- -------- -------- -------- -------- --------
0000000000000001        1       33     2994        0       12     1064        0
0000000000000001        2        5      390        0       16     1512        0
0000000000000001        3       34     3092        0       12     1064        0
0000000000000001        4        6      480        0       15     1414        0
0000000000000001        5        7      558        0        1       42        0
0000000000000001 fffffffe        0        0        0        1       42        0
```

3. Standard Load Balancing:

In this project, first a standard load balancer was developed. Having tested the standard load balancer, optimization was done on the standard load balancer to get an optimum load balancer. For the load balancer to function, NIC bonding needs to be done on the host side. Since, mininet uses alias to create multiple interfaces within a host, only one interface will function when there is no NIC bonding. In the above topology, the two links in the Linux host are bonded together using the commands given in the appendix. The mode selected for NIC bonding is round-robin mode.

In a standard load balancer, the traffic should be equally balanced among the aggregated links. The following flows of the switch s1 shows that behaviour as programmed in Ryu.

```
root@mininet-vm:~#ovs-ofctl dump-flows –O OpenFlow13 s1

cookie=0x0,duration=587.14s,table=0,n_packets=48,n_bytes=4480,priority=1,in_port=3,dl_dst=
00:00:00:00:00:11 actions=output:2

cookie=0x0,duration=587.14s,table=0,n_packets=52,n_bytes=4680,priority=1,in_port=4,dl_dst=
00:00:00:00:00:11 actions=output:1

cookie=0x0,duration=587.14s,table=0,n_packets=46,n_bytes=4280,priority=1,in_port=1,dl_dst=
00:00:00:00:00:22 actions=output:4

cookie=0x0,duration=587.14s,table=0,n_packets=48,n_bytes=4480,priority=1,in_port=2,dl_dst=
00:00:00:00:00:22 actions=output:3
```

4. Optimum Load Balancing:

After a standard load balancer is developed and tested, the load balancer is optimized. The optimization includes two main scenarios:

i. The open vswitch should be able to change the out port when the link or an interface is down. In the diagram above, if s1-eth3 is down , it should be able to direct all the traffic through s1-eth4. Similarly, if s1-eth1 is down, it should be able to direct all traffic through s1-eth2.

The following code fetches the port status from open vswitch:

```
@set_ev_cls(ofp_event.EventOFPFlowStatsReply, MAIN_DISPATCHER)

def _flow_stats_reply_handler(self, ev):

    body = ev.msg.body
```

```
msg = ev.msg

datapath = msg.datapath

ofproto = datapath.ofproto

parser = datapath.ofproto_parser

self.global_flow_details=[]

 for stat in sorted([flow for flow in body if flow.priority == 1],key=lambda flow:
            (flow.match['in_port'],flow.match['eth_dst'])):

     self.logger.info('%016x          %8x          %17s          %8x          %8d
%8d',ev.msg.datapath.id,stat.match['in_port'], stat.match['eth_dst'],

            stat.instructions[0].actions[0].port, stat.packet_count, stat.byte_count)
```

ii. The open vswitch should be able to change the out port when the traffic flow through a link exceeds the threshold limit. For eg: If the traffic in s1-eth3 is 95% of the bandwidth of the link, it should direct the traffic through s1-eth4 and vice-versa.

The following code checks the link utilization of the open vswitch ports at regular intervals.

```
for stat in sorted(body, key=attrgetter('port_no')):

      key = (ev.msg.datapath.id, stat.port_no)

      value = (stat.rx_bytes,stat.duration_sec, stat.duration_nsec)

      self._save_stats(self.port_stats, key, value,self.state_len)

      # Get port speed.

      pre = 0
```

```
                period = 2

                tmp = self.port_stats[key]

                if len(tmp) > 1:

                    pre = tmp[-2][0]

                    period = self._get_period(tmp[-1][1], tmp[-1][2],tmp[-2][1], tmp[-2][2])

                speed = self._get_speed(self.port_stats[key][-1][0], pre, period)

                self._save_stats(self.port_speed, key, speed, self.state_len)
```

If the above value which checks the link utilization of the ovs interfaces, exceeds the threshold limit, the out port is changed to one of the other aggregated links. The following code add the flow to change the out port.

```
for stat in self.port_speed:

        if len(self.port_speed[stat]) > 1:

            speed=self.port_speed[stat][1]

        else:

            continue

        self.logger.info('This is speed %s',self.port_speed)

        if(speed>0.9*self.bandwidth):

            self.logger.info('The speed has reached 10% of bandwidth, changing flows')

            self.logger.info('This is flow %s',self.global_flow_details)

            flows=[x for x in self.global_flow_details if x[2]==stat[1]]
        if self.global_flow_details and len(flows)!=0:

            self.logger.info('These are the flows %s',flows)

            out_port=self.balance_overloaded_link(stat[1])

            actions = [parser.OFPActionOutput(out_port)]

            for flow in flows:

                match = parser.OFPMatch(in_port=flow[0],eth_dst=flow[1])

                self.logger.info('Installing flow as the bandwidth limit is reached')

                self.add_flow(datapath, 1, match, actions)
```

Issues and challenges:

Aggregated links not shown as single interface:

As mentioned in the FAQ of Open vSwitch, the aggregated links in open vswitch are shown as individual interfaces and not shown as a single interface. So, a lot of programming needs to be done to achieve load balancing due to the lack of this feature.

To address this issue, the controller was designed to check the statistics of the ports and flows and take decisions based on the status of interface and traffic flow.

Understanding the concepts of SDN:

Open vSwitch is a dumb but programmable switch. A controller is used to program the flows in the open vswitch. This concept of programming the switch using a remote controller is a feature of SDN(Software defined Networking). SDN which is a new concept has a lot of aspects attached to it. Since, this project uses the concept of SDN, a considerable amount of time was spent to learn the concepts of SDN and start working on the project requirements.

Taking a SDN course at the university helped me a lot in the understanding of SDN. Also, coursera course on the same topic provided insights into this new field.

Shared file system in Mininet:

Mininet is implemented within a single VM. The switch and the hosts are both implemented in a single machine using multiple network namespaces. Also, they have a shared file system. The files in one host are exactly the same as that in other hosts and the switch. So, testing traffic flow using data transfer from one host to another is not possible in this simulation environment. The only traffic possible here is using ping commands.

Simulation of high traffic was done using parameters in controller program.

Understanding the concepts of Open vSwitch:

Open vSwitch was introduced in 2009. Quite new, this virtual switch is still in development and is mainly used in implementation in data centres where Xen or KVM is used as the virtual server. The number of people who are working on open vswitch are far less than the number of people working in other technology like ubuntu and android. A small problem would take weeks to get solved which largely slowed down the progress of the project.

Mailing list was the only way where we could email our problems. The replies from people subscribed to the mailing list helped a lot to solve the problems and complete the project. Also, the official open vswitch website has some documentation but they are limited to basic concepts of open vswitch only.

Results Analysis:

Choosing the load balancer mode:

The problem that is addressed in this project is load balancing of traffic in open vswitch.

There are a lot of methods that are used for load-balancing as described below:

Mode 0:balance-rr

Round-robin policy: Transmit packets in sequential order from the first available slave through the last. This mode provides load balancing and fault tolerance.

Mode 1:active-backup

Active-backup policy: Only one slave in the bond is active. A different slave becomes active if, and only if, the active slave fails. The bond's MAC address is externally visible on only one port (network adapter) to avoid confusing the switch. This mode provides fault tolerance. The primary option affects the behavior of this mode.

Mode 2:balance-xor

XOR policy: Transmit based on selectable hashing algorithm. The default policy is a simple source+destination MAC address algorithm. Alternate transmit policies may be selected via the xmit_hash_policy option, described below. This mode provides load balancing and fault tolerance.

Mode 3:broadcast

Broadcast policy: transmits everything on all slave interfaces. This mode provides fault tolerance.

Mode 4:802.3ad

IEEE 802.3ad Dynamic link aggregation. Creates aggregation groups that share the same speed and duplex settings. Utilizes all slaves in the active aggregator according to the 802.3ad specification.

Mode 5:balance-tlb

Adaptive transmit load balancing: channel bonding that does not require any special switch support. The outgoing traffic is distributed according to the current load (computed relative to the speed) on each slave. Incoming traffic is received by the current slave. If the receiving slave fails, another slave takes over the MAC address of the failed receiving slave.

Mode 6:balance-alb

Adaptive load balancing: includes balance-tlb plus receive load balancing (rlb) for IPV4 traffic, and does not require any special switch support. The receive load balancing is achieved by ARP negotiation. The bonding driver intercepts the ARP Replies sent by the local system on their way out and overwrites the source hardware address with the unique hardware address of one of the slaves in the bond such that different peers use different hardware addresses for the server.

The load-balancing mode that was used in this project is mode 0 which is round-robin mode. In this mode, equal amount of traffic is sent through each of the aggregated link thus keeping the links equally utilized at all times. Fault-tolerance is achieved using the controller which checks the statistics of the ports at regular intervals and changes flow accordingly. Thus the mode that is selected for this project is the best suited mode according to our requirement.

Choosing the controller:

There are different types of controllers available in market.

- **POX**

NOX's successor, POX, was built as a friendlier alternative and has been used and implemented by a number of SDN developers and engineers. Compared to NOX, POX has an easier development environment to work with and a reasonably well written API and documentation. It also provides a web based GUI and is written in Python, which typically shortens its experimental and developmental cycles.

- **Floodlight**

Next came Floodlight, a fork off of Beacon that is managed by Big Switch Networks. While its beginning was based on Beacon it was built using Apache Ant which is a very popular software build tool that makes the development of Floodlight easier and more flexible. Floodlight has a very active community and has a large number of features that can be added to create a system that best meets the requirements of a specific organization. Both a web based and Java based GUI are available and most of its functionality is exposed through a REST API.

- **Ryu**

Ryu is a component-based software defined networking framework.Ryu provides software components with well defined API that make it easy for developers to create new network management and control applications. Ryu supports various protocols for managing network devices, such as OpenFlow, Netconf, OF-config, etc. About OpenFlow, Ryu supports fully 1.0, 1.2, 1.3, 1.4 and Nicira Extensions. All of the code is freely available under the Apache 2.0 license. Ryu is fully written in Python.

- **OpenDaylight**

OpenDaylight is a Linux Foundation collaborative project that has been highly supported by Cisco, Big Switch, and several other networking companies. Like Floodlight, OpenDaylight is written in Java and is a popular, well-supported SDN controller. It also includes exposure with a REST API and a web based GUI. The second release of OpenDaylight (Helium) includes support for SDN, NV (Network Virtualization) and NFV (Network Functions Virtualization) and is intended to be scaled to very large sizes. Like Floodlight it also has a number of pluggable modules (interfaces, protocols, and applications) that can be used to alter it to the needs of an organization. OpenDaylight is a little different from the other offerings because it allows for other non-OpenFlow southbound protocols.

Among these entire available SDN controllers, Ryu was used as the controller of choice. The primary reason was that it uses python. Python is easier to learn that other programming languages and also I had experience programming in Python. Secondly, Ryu has great online documentation which increased the rate at which the application was developed.

Conclusion:

Open source software are trending these days. We can see many communities working on open source software. Along with that we can see plethora of boot-camps and developer meets that is fuelling the open source community. The notion of creating a product as a team, with people all over the world contributing to it, is far more economical and faster to take to market than being created by a single company. Many companies are dealing with a similar problem. So, it is wise to look for a common solution with everyone contributing to it. Anyone who has a basic understanding of the topic can join the community. So, it is open to everyone, regardless of their age, location and background. Therefore, a major takeaway for me from this project is the ability to contribute to the open community of open vswitch and similar other open source products.

Among the major technologies that were researched in this project is open vswitch. I gained extensive knowledge about programming the openvswitch which includes using the event handlers, getting the statistics from switch, processing that information and sending the openflow command to switch as necessary. . Open vSwitch being a dumb switch needs a controller to define and change its forwarding table as required. The open vswitch was programmed using a sdn controller called Ryu which is based on Python. So, I got an opportunity to hone my Python programming skills. In addition to this, I got an in-depth understanding of networking in Linux.

Future work:

We have used 802.3 ad link aggregation group (LAG) to aggregate the links on Open vSwitch. However, as of now, Open vSwitch does not see the aggregated links as a single logical interface. Rather, it shows them as individual interfaces although it puts them under a bond. It will be more logical to show them as a single logical interface as it is a bond. We cannot direct the traffic to the bond. It can only be sent to individual interfaces. The development of this feature requires a deep understanding of Linux kernel and a profound knowledge of C programming language. Working on this feature can be a stepping stone towards contribution to the Open vSwitch community.

Importance of the research:

The links interconnecting the systems may be not up all the time due to port failure or link disconnection. This means, it’s highly probable that they might go down once in a while. Under such circumstances, redundant links keeps the overall system running. Open vSwitch is used as a switch that connects the VMs to the outside world in a virtual environment. So, it is very important that there is a working link to these virtual machines as the services are present in these VMs. Load balancing needs to be done to make sure the services are available all the time. The program developed in this project can be applied to Open vSwitch in industry systems. This will help the live system to achieve an uptime near to hundred percent. This program takes into account the status of ports and links. In addition to this, it checks the traffic that is following through them in a certain time interval which can be set to any value depending upon the requirement. The research and development done in this project has given a solution that can be implemented in real world scenario to make sure that the VMs that provide services to the clients are always available.

Appendices:

CAPSTONE PROJECT DEMO STEPS

Start mininet topology.

```
sudo mn --custom ~/mininet/custom/mytopo.py --topo=mytopo --controller=remote --mac --
switch=ovsk,protocols=OpenFlow13
```

Start Ryu controller

```
cd /home/mininet/ryu && ./bin/ryu-manager --verbose ryu/app/switchandmonitor13.py
```

Configure bonding in switch interfaces using the following command.

Bond configuration for h1:

```
h1 ip link add bond0 type bond
h1 ip link set bond0 address 02:01:02:03:04:08
h1 ip link set h1-eth0 down
h1 ip link set h1-eth0 address 00:00:00:00:00:11
h1 ip link set h1-eth0 master bond0
h1 ip link set h1-eth1 down
h1 ip link set h1-eth1 address 00:00:00:00:00:12
h1 ip link set h1-eth1 master bond0
h1 ip addr add 10.0.0.5/8 dev bond0
h1 ip addr del 10.0.0.1/8 dev h1-eth0

h1 ip link set bond0 up
```

Bond configuration for h2:

```
h2 ip link add bond1 type bond
h2 ip link set bond1 address 02:01:02:03:04:10

h2 ip link set h2-eth0 down
```

```
h2 ip link set h2-eth0 address 00:00:00:00:00:21
h2 ip link set h2-eth0 master bond1
h2 ip link set h2-eth1 down
h2 ip link set h2-eth1 address 00:00:00:00:00:22
h2 ip link set h2-eth1 master bond1

h2 ip addr add 10.0.0.6/8 dev bond1
h2 ip addr del 10.0.0.2/8 dev h2-eth0

h2 ip link set bond1 up
```

TESTING A NORMAL LOADBALANCER:

Check the distribute functionality by checking the flow.

Send ping packets from h1 and h2 by pinging each other and check the flows.
The load should be balanced between input and output bond-pairs:

```
Command: ovs-ofctl –O OpenFlow13 dump-flows s1 dl_dst=02:01:02:03:04:08
Command: ovs-ofctl –O OpenFlow13 dump-flows s1 dl_dst=02:01:02:03:04:10
```

TESTING THE OPTIMIZATION OF LOADBALANCER:

Bring an interface down and check the pings and flows in s1

```
Command: sudo ifconfig s1-eth3 down
```
Also, bring down other interfaces and check the pings and flows.
The flows should not have down interface but the other interface of the bond pair.

Bring the interfaces up and check the pings and flows.

The flows should be distributed among the bond interfaces.

Send traffic from one host such that it crosses the rate at which the interface can send the packet.

```
Command on h1: ping 10.0.0.6 –c 100
Command on h2: ping 10.0.0.5 –c 100
```

The traffic generated will be at a speed of 12 kbps. So, if we sent the bandwidth as 12 kbps, the limit will be reached and we can test the functionality.

References:

1. Open vSwitch Official Site(www.openvswitch.org)
2. Ryu official site (http://osrg.github.io/ryu/)
3. Ryu documentation (http://ryu.readthedocs.org/en/latest/index.html)