Dynamic Round Robin Load Balancer with Fault Detection

[Topology]

H1 (10.0.1.1) -------(10.0.0.1) P4 (DRRLB) --------H2 (Simple HTTP Server, 10.0.2.2)

--------H3 (Simple HTTP Server, 10.0.3.3)

--------H4 (Simple HTTP Server, 10.0.4.4)

--------H5 (Simple HTTP Server, 10.0.5.5)

virtual ip :10.0.0.1

Originally, there are 4 backend webservers. When the user request load becomes lighter, we can reduce the number of serving web servers. When load becomes heavy, we can add the serving web servers during runtime.

Enhancement: we add a controller that can detect the status of backend servers. When a backend server fails, the controller will talk to P4 Load Balancer not to dispatch new requests to this failed server.

(basic16.p4)

```
#include <core.p4>
#include <v1model.p4>

struct meta_t {
    bit<1>   do_forward;
    bit<32> ipv4_sa;
    bit<32> ipv4_da;
    bit<16> tcp_sp;
    bit<16> tcp_dp;
    bit<32> nhop_ipv4;
    bit<32> if_ipv4_addr;
    bit<48> if_mac_addr;
    bit<1>   is_ext_if;
    bit<16> tcpLength;
    bit<8>   if_index;
}

struct mymetadata_t {
    bit<13> flowlet_map_index;
```

```
        bit<3>    flowlet_select;
        bit<3>    myselect;
        bit<3>    max;
        bit<3>    old_max;
        bit<1>    server1;
        bit<1>    server2;
        bit<1>    server3;
        bit<1>    server4;
}

header arp_t {
        bit<16> htype;
        bit<16> ptype;
        bit<8>    hlen;
        bit<8>    plen;
        bit<16> opcode;
        bit<48> hwSrcAddr;
        bit<32> protoSrcAddr;
        bit<48> hwDstAddr;
        bit<32> protoDstAddr;
}

header ethernet_t {
        bit<48> dstAddr;
        bit<48> srcAddr;
        bit<16> etherType;
}

header ipv4_t {
        bit<4>    version;
        bit<4>    ihl;
        bit<8>    diffserv;
        bit<16> totalLen;
        bit<16> identification;
        bit<3>    flags;
        bit<13> fragOffset;
        bit<8>    ttl;
        bit<8>    protocol;
```

```
        bit<16> hdrChecksum;
        bit<32> srcAddr;
        bit<32> dstAddr;
}

header tcp_t {
        bit<16> srcPort;
        bit<16> dstPort;
        bit<32> seqNo;
        bit<32> ackNo;
        bit<4>    dataOffset;
        bit<4>    res;
        bit<8>    flags;
        bit<16> window;
        bit<16> checksum;
        bit<16> urgentPtr;
}

header udp_t {
        bit<16> srcPort;
        bit<16> dstPort;
        bit<16> length_;
        bit<16> checksum;
}

struct metadata {
        @name(".meta")
        meta_t              meta;
        @name(".mymetadata")
        mymetadata_t mymetadata;
}

struct headers {
        @name(".arp")
        arp_t           arp;
        @name(".ethernet")
        ethernet_t ethernet;
        @name(".ipv4")
```

```
    ipv4_t      ipv4;
    @name(".tcp")
    tcp_t       tcp;
    @name(".udp")
    udp_t       udp;
}

parser ParserImpl(packet_in packet, out headers hdr, inout metadata
meta, inout standard_metadata_t standard_metadata) {
    @name(".parse_arp") state parse_arp {
        packet.extract(hdr.arp);
        transition accept;
    }
    @name(".parse_ethernet") state parse_ethernet {
        packet.extract(hdr.ethernet);
        transition select(hdr.ethernet.etherType) {
            16w0x800: parse_ipv4;
            16w0x806: parse_arp;
            default: accept;
        }
    }
    @name(".parse_ipv4") state parse_ipv4 {
        packet.extract(hdr.ipv4);
        meta.meta.ipv4_sa = hdr.ipv4.srcAddr;
        meta.meta.ipv4_da = hdr.ipv4.dstAddr;
        meta.meta.tcpLength = hdr.ipv4.totalLen - 16w20;
        transition select(hdr.ipv4.protocol) {
            8w6: parse_tcp;
            8w17: parse_udp;
            default: accept;
        }
    }
    @name(".parse_tcp") state parse_tcp {
        packet.extract(hdr.tcp);
        meta.meta.tcp_sp = hdr.tcp.srcPort;
        meta.meta.tcp_dp = hdr.tcp.dstPort;
        transition accept;
    }
```

```
        @name(".parse_udp") state parse_udp {
                packet.extract(hdr.udp);
                transition accept;
        }
        @name(".start") state start {
                meta.meta.if_index = (bit<8>)standard_metadata.ingress_port;
        meta.mymetadata.myselect=0;
                meta.mymetadata.max=4;
        meta.mymetadata.server1=0;
              meta.mymetadata.server2=0;
                meta.mymetadata.server3=0;
                meta.mymetadata.server4=0;
                transition parse_ethernet;
        }
}

control egress(inout headers hdr, inout metadata meta, inout
standard_metadata_t standard_metadata) {
        @name("._drop") action _drop() {
                mark_to_drop(standard_metadata);
        }
        @name(".rewrite_sip") action rewrite_sip(bit<32> sip) {
                hdr.ipv4.srcAddr = sip;
        }
        @name(".nop") action nop() {
        }
        @name(".send_frame") table send_frame {
                actions = {
                        _drop;
                        rewrite_sip;
                        nop;
                }
                key = {
                        standard_metadata.egress_port: exact;
                }
                size = 256;
        }
        apply {
```

```
            send_frame.apply();
        }
}

@name(".flowlet_select") register<bit<3>>(32w8192) flowlet_select;

@name(".myselect") register<bit<3>>(32w1) myselect;

register<bit<3>>(32w1) mymax;

control ingress(inout headers hdr, inout metadata meta, inout
standard_metadata_t standard_metadata) {
        @name("._drop") action _drop() {
            mark_to_drop(standard_metadata);
        }
        @name(".set_ecmp_select") action set_ecmp_select() {
            hash(meta.mymetadata.flowlet_map_index,
HashAlgorithm.crc16, (bit<13>)0, { hdr.ipv4.srcAddr, hdr.ipv4.dstAddr,
hdr.ipv4.protocol, hdr.tcp.srcPort, hdr.tcp.dstPort }, (bit<26>)8192);
            flowlet_select.read(meta.mymetadata.flowlet_select,
(bit<32>)meta.mymetadata.flowlet_map_index);
            myselect.read(meta.mymetadata.myselect, (bit<32>)0);
            meta.mymetadata.myselect = meta.mymetadata.myselect +
3w1;
            meta.mymetadata.flowlet_select =
(bit<3>)meta.mymetadata.myselect;
            //meta.mymetadata.flowlet_select =
meta.mymetadata.flowlet_select + 3w1;
            myselect.write((bit<32>)0,
(bit<3>)meta.mymetadata.myselect);

flowlet_select.write((bit<32>)meta.mymetadata.flowlet_map_index,
(bit<3>)meta.mymetadata.flowlet_select);
        }
        @name(".nop") action nop() {
        }
        @name(".set_ecmp_nhop") action set_ecmp_nhop(bit<48>
nhop_mac, bit<32> nhop_ipv4, bit<9> port) {
```

```
            standard_metadata.egress_spec = port;
            hdr.ipv4.dstAddr = nhop_ipv4;
            hdr.ethernet.dstAddr = nhop_mac;
            hdr.ipv4.ttl = hdr.ipv4.ttl + 8w255;
        }
        @name(".set_nhop") action set_nhop(bit<48> dmac, bit<9> port) {
            standard_metadata.egress_spec = port;
            hdr.ethernet.dstAddr = dmac;
            hdr.ipv4.ttl = hdr.ipv4.ttl + 8w255;
        }
        @name(".read_flowlet_select") action read_flowlet_select() {
            hash(meta.mymetadata.flowlet_map_index,
HashAlgorithm.crc16, (bit<13>)0, { hdr.ipv4.srcAddr, hdr.ipv4.dstAddr,
hdr.ipv4.protocol, hdr.tcp.srcPort, hdr.tcp.dstPort }, (bit<26>)8192);
            flowlet_select.read(meta.mymetadata.flowlet_select,
(bit<32>)meta.mymetadata.flowlet_map_index);
        }
        action _max(bit<3> max){
            meta.mymetadata.max=max;
        }

        action _fail1(bit<1> fail){
        meta.mymetadata.server1=fail;
        }

        action _fail2(bit<1> fail){
        meta.mymetadata.server2=fail;
        }

        action _fail3(bit<1> fail){
        meta.mymetadata.server3=fail;
        }

        action _fail4(bit<1> fail){
        meta.mymetadata.server4=fail;
        }

        @name(".ecmp_group") table ecmp_group {
```

```
        actions = {
            _drop;
            set_ecmp_select;
            nop;
        }
        key = {
            hdr.ipv4.dstAddr: lpm;
        }
        size = 1024;
    }
    @name(".ecmp_nhop") table ecmp_nhop {
        actions = {
            _drop;
            set_ecmp_nhop;
            nop;
        }
        key = {
            meta.mymetadata.flowlet_select: exact;
        }
        size = 1024;
    }
    @name(".forward") table forward {
        actions = {
            _drop;
            set_nhop;
            nop;
            read_flowlet_select;
        }
        key = {
            hdr.ipv4.dstAddr: lpm;
        }
        size = 1024;
    }
    table set_max {
        actions = {
            _max;
        }
        key = {
```

```
                hdr.ipv4.dstAddr: lpm;
        }
        size = 1;
 }
 table set_status1 {
actions = {
                _fail1;
        }
        key = {
                hdr.ipv4.dstAddr: lpm;
        }
        size = 1;
 }
 table set_status2 {
actions = {
                _fail2;
        }
        key = {
                hdr.ipv4.dstAddr: lpm;
        }
        size = 1;
 }
 table set_status3 {
actions = {
                _fail3;
        }
        key = {
                hdr.ipv4.dstAddr: lpm;
        }
        size = 1;
 }
 table set_status4 {
actions = {
                _fail4;
        }
        key = {
                hdr.ipv4.dstAddr: lpm;
        }
```

```
            size = 1;
        }
        apply {
            forward.apply();
            //set_max.apply();
            if( set_max.apply().hit ) {
                mymax.read(meta.mymetadata.old_max, (bit<32>)0);

                if(meta.mymetadata.old_max!=meta.mymetadata.max) {
                  meta.mymetadata.myselect=0;
                  myselect.write((bit<32>)0,
(bit<3>)meta.mymetadata.myselect);
                   mymax.write((bit<32>)0,
(bit<3>)meta.mymetadata.max);
                }
            }

            if (hdr.tcp.flags & 8w2 != 8w0) {
                ecmp_group.apply();
                 if(meta.mymetadata.myselect==meta.mymetadata.max){
                   meta.mymetadata.myselect=0;
                    myselect.write((bit<32>)0,
(bit<3>)meta.mymetadata.myselect);
                 }
            }

      if( set_status1.apply().hit && hdr.tcp.flags & 8w2 != 8w0 ) {
         if(meta.mymetadata.server1 == 1 &&
meta.mymetadata.flowlet_select==1){
            hash(meta.mymetadata.flowlet_map_index,
HashAlgorithm.crc16, (bit<13>)0, { hdr.ipv4.srcAddr, hdr.ipv4.dstAddr,
hdr.ipv4.protocol, hdr.tcp.srcPort, hdr.tcp.dstPort }, (bit<26>)8192);
                flowlet_select.read(meta.mymetadata.flowlet_select,
(bit<32>)meta.mymetadata.flowlet_map_index);
                myselect.read(meta.mymetadata.myselect, (bit<32>)0);
                if(meta.mymetadata.server2 != 1){
                     meta.mymetadata.myselect = 2;
```

```
                        meta.mymetadata.flowlet_select =
(bit<3>)meta.mymetadata.myselect;
                        myselect.write((bit<32>)0,
(bit<3>)meta.mymetadata.myselect);

flowlet_select.write((bit<32>)meta.mymetadata.flowlet_map_index,
(bit<3>)meta.mymetadata.flowlet_select);
                } else if(meta.mymetadata.server3 != 1){
                        meta.mymetadata.myselect = 3;
                        meta.mymetadata.flowlet_select =
(bit<3>)meta.mymetadata.myselect;
                        myselect.write((bit<32>)0,
(bit<3>)meta.mymetadata.myselect);

flowlet_select.write((bit<32>)meta.mymetadata.flowlet_map_index,
(bit<3>)meta.mymetadata.flowlet_select);
                } else if(meta.mymetadata.server4 != 1){
                        meta.mymetadata.myselect = 0;
                        meta.mymetadata.flowlet_select = 4;
                        myselect.write((bit<32>)0,
(bit<3>)meta.mymetadata.myselect);

flowlet_select.write((bit<32>)meta.mymetadata.flowlet_map_index,
(bit<3>)meta.mymetadata.flowlet_select);
                } else {
                        _drop();
            }
        }
            }

            if( set_status2.apply().hit && hdr.tcp.flags & 8w2 != 8w0 ) {
        if(meta.mymetadata.server2 == 1 &&
meta.mymetadata.flowlet_select==2){
            hash(meta.mymetadata.flowlet_map_index,
HashAlgorithm.crc16, (bit<13>)0, { hdr.ipv4.srcAddr, hdr.ipv4.dstAddr,
hdr.ipv4.protocol, hdr.tcp.srcPort, hdr.tcp.dstPort }, (bit<26>)8192);
                flowlet_select.read(meta.mymetadata.flowlet_select,
(bit<32>)meta.mymetadata.flowlet_map_index);
```

```
                myselect.read(meta.mymetadata.myselect, (bit<32>)0);
                if(meta.mymetadata.server3 != 1){
                        meta.mymetadata.myselect = 3;
                        meta.mymetadata.flowlet_select =
(bit<3>)meta.mymetadata.myselect;
                        myselect.write((bit<32>)0,
(bit<3>)meta.mymetadata.myselect);

flowlet_select.write((bit<32>)meta.mymetadata.flowlet_map_index,
(bit<3>)meta.mymetadata.flowlet_select);
                } else if(meta.mymetadata.server4 != 1){
                        meta.mymetadata.myselect = 0;
                        meta.mymetadata.flowlet_select = 4;
                        myselect.write((bit<32>)0,
(bit<3>)meta.mymetadata.myselect);

flowlet_select.write((bit<32>)meta.mymetadata.flowlet_map_index,
(bit<3>)meta.mymetadata.flowlet_select);
                } else if(meta.mymetadata.server1 != 1){
                        meta.mymetadata.myselect = 1;
                        meta.mymetadata.flowlet_select =
(bit<3>)meta.mymetadata.myselect;
                        myselect.write((bit<32>)0,
(bit<3>)meta.mymetadata.myselect);

flowlet_select.write((bit<32>)meta.mymetadata.flowlet_map_index,
(bit<3>)meta.mymetadata.flowlet_select);
                } else {
                        _drop();
            }
        }
            }

            if( set_status3.apply().hit && hdr.tcp.flags & 8w2 != 8w0 ) {
               if(meta.mymetadata.server3 == 1 &&
meta.mymetadata.flowlet_select==3){
```

```
            hash(meta.mymetadata.flowlet_map_index,
HashAlgorithm.crc16, (bit<13>)0, { hdr.ipv4.srcAddr, hdr.ipv4.dstAddr,
hdr.ipv4.protocol, hdr.tcp.srcPort, hdr.tcp.dstPort }, (bit<26>)8192);
                    flowlet_select.read(meta.mymetadata.flowlet_select,
(bit<32>)meta.mymetadata.flowlet_map_index);
                    myselect.read(meta.mymetadata.myselect, (bit<32>)0);
                    if(meta.mymetadata.server4 != 1){
                            meta.mymetadata.myselect = 0;
                            meta.mymetadata.flowlet_select = 4;
                            myselect.write((bit<32>)0,
(bit<3>)meta.mymetadata.myselect);

flowlet_select.write((bit<32>)meta.mymetadata.flowlet_map_index,
(bit<3>)meta.mymetadata.flowlet_select);
                    } else if(meta.mymetadata.server1 != 1){
                            meta.mymetadata.myselect = 1;
                            meta.mymetadata.flowlet_select =
(bit<3>)meta.mymetadata.myselect;
                            myselect.write((bit<32>)0,
(bit<3>)meta.mymetadata.myselect);

flowlet_select.write((bit<32>)meta.mymetadata.flowlet_map_index,
(bit<3>)meta.mymetadata.flowlet_select);
                    } else if(meta.mymetadata.server2 != 1){
                            meta.mymetadata.myselect = 2;
                            meta.mymetadata.flowlet_select =
(bit<3>)meta.mymetadata.myselect;
                            myselect.write((bit<32>)0,
(bit<3>)meta.mymetadata.myselect);

flowlet_select.write((bit<32>)meta.mymetadata.flowlet_map_index,
(bit<3>)meta.mymetadata.flowlet_select);
                    } else {
                            _drop();
                }
            }
                }
```

```
            if( set_status4.apply().hit && hdr.tcp.flags & 8w2 != 8w0 ) {
        if(meta.mymetadata.server4 == 1 &&
meta.mymetadata.flowlet_select==4){
            hash(meta.mymetadata.flowlet_map_index,
HashAlgorithm.crc16, (bit<13>)0, { hdr.ipv4.srcAddr, hdr.ipv4.dstAddr,
hdr.ipv4.protocol, hdr.tcp.srcPort, hdr.tcp.dstPort }, (bit<26>)8192);
                    flowlet_select.read(meta.mymetadata.flowlet_select,
(bit<32>)meta.mymetadata.flowlet_map_index);
                    myselect.read(meta.mymetadata.myselect, (bit<32>)0);
                    if(meta.mymetadata.server1 != 1){
                            meta.mymetadata.myselect = 1;
                            meta.mymetadata.flowlet_select =
(bit<3>)meta.mymetadata.myselect;
                            myselect.write((bit<32>)0,
(bit<3>)meta.mymetadata.myselect);

flowlet_select.write((bit<32>)meta.mymetadata.flowlet_map_index,
(bit<3>)meta.mymetadata.flowlet_select);
                    } else if(meta.mymetadata.server2 != 1){
                            meta.mymetadata.myselect = 2;
                            meta.mymetadata.flowlet_select =
(bit<3>)meta.mymetadata.myselect;
                            myselect.write((bit<32>)0,
(bit<3>)meta.mymetadata.myselect);

flowlet_select.write((bit<32>)meta.mymetadata.flowlet_map_index,
(bit<3>)meta.mymetadata.flowlet_select);
                    } else if(meta.mymetadata.server3 != 1){
                            meta.mymetadata.myselect = 3;
                            meta.mymetadata.flowlet_select =
(bit<3>)meta.mymetadata.myselect;
                            myselect.write((bit<32>)0,
(bit<3>)meta.mymetadata.myselect);

flowlet_select.write((bit<32>)meta.mymetadata.flowlet_map_index,
(bit<3>)meta.mymetadata.flowlet_select);
                    } else {
                            _drop();
```

```
            }
        }
            }

            if(hdr.ipv4.isValid()){
                ecmp_nhop.apply();
            }
        }
}

control DeparserImpl(packet_out packet, in headers hdr) {
    apply {
        packet.emit(hdr.ethernet);
        packet.emit(hdr.arp);
        packet.emit(hdr.ipv4);
        packet.emit(hdr.udp);
        packet.emit(hdr.tcp);
    }
}

control verifyChecksum(inout headers hdr, inout metadata meta) {
    apply {
        verify_checksum(true, { hdr.ipv4.version, hdr.ipv4.ihl,
hdr.ipv4.diffserv, hdr.ipv4.totalLen, hdr.ipv4.identification, hdr.ipv4.flags,
hdr.ipv4.fragOffset, hdr.ipv4.ttl, hdr.ipv4.protocol, hdr.ipv4.srcAddr,
hdr.ipv4.dstAddr }, hdr.ipv4.hdrChecksum, HashAlgorithm.csum16);
        verify_checksum_with_payload(true, { hdr.ipv4.srcAddr,
hdr.ipv4.dstAddr, 8w0, hdr.ipv4.protocol, meta.meta.tcpLength,
hdr.tcp.srcPort, hdr.tcp.dstPort, hdr.tcp.seqNo, hdr.tcp.ackNo,
hdr.tcp.dataOffset, hdr.tcp.res, hdr.tcp.flags, hdr.tcp.window,
hdr.tcp.urgentPtr }, hdr.tcp.checksum, HashAlgorithm.csum16);
    }
}

control computeChecksum(inout headers hdr, inout metadata meta) {
    apply {
        update_checksum(true, { hdr.ipv4.version, hdr.ipv4.ihl,
hdr.ipv4.diffserv, hdr.ipv4.totalLen, hdr.ipv4.identification, hdr.ipv4.flags,
```

```
hdr.ipv4.fragOffset, hdr.ipv4.ttl, hdr.ipv4.protocol, hdr.ipv4.srcAddr,
hdr.ipv4.dstAddr }, hdr.ipv4.hdrChecksum, HashAlgorithm.csum16);
            update_checksum_with_payload(true, { hdr.ipv4.srcAddr,
hdr.ipv4.dstAddr, 8w0, hdr.ipv4.protocol, meta.meta.tcpLength,
hdr.tcp.srcPort, hdr.tcp.dstPort, hdr.tcp.seqNo, hdr.tcp.ackNo,
hdr.tcp.dataOffset, hdr.tcp.res, hdr.tcp.flags, hdr.tcp.window,
hdr.tcp.urgentPtr }, hdr.tcp.checksum, HashAlgorithm.csum16);
        }
}

V1Switch(ParserImpl(), verifyChecksum(), ingress(), egress(),
computeChecksum(), DeparserImpl()) main;
```

(test_topo.py)

```
import os
from mininet.net import Containernet
from mininet.topo import Topo
from mininet.log import setLogLevel, info
from mininet.cli import CLI
from mininet.link import TCLink
from mininet.node import RemoteController
from mininet.node import Docker
from p4_mininet import P4Switch, P4Host

import argparse
from time import sleep

parser = argparse.ArgumentParser(description='Mininet demo')
parser.add_argument('--behavioral-exe', help='Path to behavioral executable',
                    type=str, action="store", required=False,
default='simple_switch' )
parser.add_argument('--thrift-port', help='Thrift server port for table updates',
                    type=int, action="store", default=9090)
parser.add_argument('--num-hosts', help='Number of hosts to connect to switch',
                    type=int, action="store", default=2)
parser.add_argument('--mode', choices=['l2', 'l3'], type=str, default='l3')
parser.add_argument('--json', help='Path to JSON config file',
                    type=str, action="store", required=True)
```

```
parser.add_argument('--pcap-dump', help='Dump packets on interfaces to pcap
files',
                                type=str, action="store", required=False, default=False)


args = parser.parse_args()

def main():
    net = Containernet(host = P4Host, link=TCLink, controller = None)
    switch1 = net.addSwitch('s1', sw_path = args.behavioral_exe, json_path =
args.json, thrift_port = args.thrift_port, cls = P4Switch, pcap_dump =
args.pcap_dump)

    host1 = net.addHost('h1', mac = '00:00:00:00:01:01',    ip="10.0.1.1/24")
    host2 = net.addDocker('h2', mac = '00:00:00:00:02:02', ip="10.0.2.2/24",
dimage="apache-php-mysql:v7",cpu_period=50000, cpu_quota=1000)
    host3 = net.addDocker('h3', mac = '00:00:00:00:03:03', ip="10.0.3.3/24",
dimage="apache-php-mysql:v7",cpu_period=50000, cpu_quota=1000)
    host4 = net.addDocker('h4', mac = '00:00:00:00:04:04', ip="10.0.4.4/24",
dimage="apache-php-mysql:v7",cpu_period=50000, cpu_quota=1000)
    host5 = net.addDocker('h5', mac = '00:00:00:00:05:05', ip="10.0.5.5/24",
dimage="apache-php-mysql:v7",cpu_period=50000, cpu_quota=1000)
    net.addLink(host1, switch1, port1 = 0, port2 = 1, cls=TCLink, bw=10)
    net.addLink(host2, switch1, port1 = 0, port2 = 2, cls=TCLink, bw=1)
    net.addLink(host3, switch1, port1 = 0, port2 = 3, cls=TCLink, bw=1)
    net.addLink(host4, switch1, port1 = 0, port2 = 4, cls=TCLink, bw=1)
    net.addLink(host5, switch1, port1 = 0, port2 = 5, cls=TCLink, bw=1)
    net.start()
    h1,h2,h3,h4,h5=net.get('h1','h2','h3','h4','h5')
    h1.cmd("arp -s 10.0.1.254 00:00:00:01:01:01")
    h1.cmd("ip route add default via 10.0.1.254")
    h1.cmd("ethtool -K eth0 tx off rx off")
    h2.cmd("arp -s 10.0.2.254 00:00:00:02:02:02")
    h2.cmd("ip route del default")
    h2.cmd("ip route add default via 10.0.2.254")
    h2.cmd("ethtool -K h2-eth0 tx off rx off")
    h2.cmd("cd /var/www/html; python -m SimpleHTTPServer 80 &")
    h3.cmd("arp -s 10.0.3.254 00:00:00:03:03:03")
```

```
        h3.cmd("ip route del default")
        h3.cmd("ip route add default via 10.0.3.254")
        h3.cmd("ethtool -K h3-eth0 tx off rx off")
        h3.cmd("cd /var/www/html; python -m SimpleHTTPServer 80 &")
        h4.cmd("arp -s 10.0.4.254 00:00:00:04:04:04")
        h4.cmd("ip route del default")
        h4.cmd("ip route add default via 10.0.4.254")
        h4.cmd("ethtool -K h4-eth0 tx off rx off")
        h4.cmd("cd /var/www/html; python -m SimpleHTTPServer 80 &")
        h5.cmd("arp -s 10.0.5.254 00:00:00:05:05:05")
        h5.cmd("ip route del default")
        h5.cmd("ip route add default via 10.0.5.254")
        h5.cmd("ethtool -K h5-eth0 tx off rx off")
        h5.cmd("cd /var/www/html; python -m SimpleHTTPServer 80 &")
        sleep(1)

        print('\033[0;32m'),
        print "Gotcha!"
        print('\033[0m')

        CLI(net)
        try:
                net.stop()
        except:
                print('\033[0;31m'),
                print('Stop error! Trying sudo mn -c')
                print('\033[0m')
                os.system('sudo mn -c')
                print('\033[0;32m'),
                print ('Stop successfully!')
                print('\033[0m')

if __name__ == '__main__':
        setLogLevel('info')
        main()
```

(cmd.txt)

```
table_set_default forward nop
```

```
table_set_default ecmp_group nop
table_set_default ecmp_nhop nop
table_set_default send_frame nop
table_add forward set_nhop 10.0.1.1/32 => 00:00:00:00:01:01 1
table_add forward set_nhop 10.0.2.2/32 => 00:00:00:00:02:02 2
table_add forward set_nhop 10.0.3.3/32 => 00:00:00:00:03:03 3
table_add forward set_nhop 10.0.4.4/32 => 00:00:00:00:04:04 4
table_add forward set_nhop 10.0.5.5/32 => 00:00:00:00:05:05 5
table_add set_max _max 10.0.0.1/32 => 4
table_add set_status1 _fail1 10.0.0.1/32 => 0
table_add set_status2 _fail2 10.0.0.1/32 => 0
table_add set_status3 _fail3 10.0.0.1/32 => 0
table_add set_status4 _fail4 10.0.0.1/32 => 0
table_add forward read_flowlet_select 10.0.0.1/32 =>
table_add ecmp_group set_ecmp_select 10.0.0.1/32 =>
table_add ecmp_nhop set_ecmp_nhop 1 => 00:00:00:00:02:02 10.0.2.2 2
table_add ecmp_nhop set_ecmp_nhop 2 => 00:00:00:00:03:03 10.0.3.3 3
table_add ecmp_nhop set_ecmp_nhop 3 => 00:00:00:00:04:04 10.0.4.4 4
table_add ecmp_nhop set_ecmp_nhop 4 => 00:00:00:00:05:05 10.0.5.5 5
table_add send_frame rewrite_sip 1 => 10.0.0.1
```

(cmd_add.txt)

```
import os
os.system('sudo /home/vagrant/behavioral-
model/targets/simple_switch/simple_switch_CLI --thrift-port=9090 < cmd.txt')
```

(start_test_topo.py)

```
import os
os.system("sudo python test_topo.py --behavioral-exe /home/vagrant/behavioral-model/targets/simp
```

(p4_mininet.py)

```
# Copyright 2013-present Barefoot Networks, Inc.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#   http://www.apache.org/licenses/LICENSE-2.0
```

```
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#

from mininet.net import Mininet
from mininet.node import Switch, Host
from mininet.log import setLogLevel, info, error, debug
from mininet.moduledeps import pathCheck
from sys import exit
import os
import tempfile
import socket

class P4Host(Host):
   def config(self, **params):
      r = super(Host, self).config(**params)

      self.defaultIntf().rename("eth0")

      for off in ["rx", "tx", "sg"]:
         cmd = "/sbin/ethtool --offload eth0 %s off" % off
         self.cmd(cmd)

      # disable IPv6
      self.cmd("sysctl -w net.ipv6.conf.all.disable_ipv6=1")
      self.cmd("sysctl -w net.ipv6.conf.default.disable_ipv6=1")
      self.cmd("sysctl -w net.ipv6.conf.lo.disable_ipv6=1")

      return r

   def describe(self):
      print "**********"
      print self.name
```

```
        print "default interface: %s\t%s\t%s" %(
            self.defaultIntf().name,
            self.defaultIntf().IP(),
            self.defaultIntf().MAC()
        )
        print "**********"

class P4Switch(Switch):
    """P4 virtual switch"""
    device_id = 0

    def __init__(self, name, sw_path = None, json_path = None,
                 thrift_port = None,
                 pcap_dump = False,
                 log_console = True,
                 verbose = True,
                 device_id = None,
                 enable_debugger = False,
                 **kwargs):
        Switch.__init__(self, name, **kwargs)
        assert(sw_path)
        assert(json_path)
        # make sure that the provided sw_path is valid
        pathCheck(sw_path)
        # make sure that the provided JSON file exists
        if not os.path.isfile(json_path):
            error("Invalid JSON file.\n")
            exit(1)
        self.sw_path = sw_path
        self.json_path = json_path
        self.verbose = verbose
        logfile = "/tmp/p4s.{}.log".format(self.name)
        self.output = open(logfile, 'w')
        self.thrift_port = thrift_port
        self.pcap_dump = pcap_dump
        self.enable_debugger = enable_debugger
        self.log_console = log_console
        if device_id is not None:
```

```
        self.device_id = device_id
        P4Switch.device_id = max(P4Switch.device_id, device_id)
    else:
        self.device_id = P4Switch.device_id
        P4Switch.device_id += 1
    self.nanomsg = "ipc:///tmp/bm-{}-log.ipc".format(self.device_id)

@classmethod
def setup(cls):
    pass

def check_switch_started(self, pid):
    """While the process is running (pid exists), we check if the Thrift
    server has been started. If the Thrift server is ready, we assume that
    the switch was started successfully. This is only reliable if the Thrift
    server is started at the end of the init process"""
    while True:
        if not os.path.exists(os.path.join("/proc", str(pid))):
            return False
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.settimeout(0.5)
        result = sock.connect_ex(("localhost", self.thrift_port))
        if result == 0:
            return  True

def start(self, controllers):
    "Start up a new P4 switch"
    info("Starting P4 switch {}.\n".format(self.name))
    args = [self.sw_path]
    for port, intf in self.intfs.items():
        if not intf.IP():
            args.extend(['-i', str(port) + "@" + intf.name])

    #wuwzhs edit in 2017/11/10
    #args.extend(['-i 3@veth1'])

    if self.pcap_dump:
        args.append("--pcap")
```

```
        # args.append("--useFiles")
    if self.thrift_port:
        args.extend(['--thrift-port', str(self.thrift_port)])
    if self.nanomsg:
        args.extend(['--nanolog', self.nanomsg])
    args.extend(['--device-id', str(self.device_id)])
    P4Switch.device_id += 1
    args.append(self.json_path)
    if self.enable_debugger:
        args.append("--debugger")
    if self.log_console:
        args.append("--log-console")
    logfile = "/tmp/p4s.{}.log".format(self.name)
    info(' '.join(args) + "\n")

    pid = None
    with tempfile.NamedTemporaryFile() as f:
        # self.cmd(' '.join(args) + ' > /dev/null 2>&1 &')
        self.cmd(' '.join(args) + ' >' + logfile + ' 2>&1 & echo $! >> ' + f.name)
        pid = int(f.read())
    debug("P4 switch {} PID is {}.\n".format(self.name, pid))
    if not self.check_switch_started(pid):
        error("P4 switch {} did not start correctly.\n".format(self.name))
        exit(1)
    info("P4 switch {} has been started.\n".format(self.name))

def stop(self):
    "Terminate P4 switch."
    self.output.flush()
    self.cmd('kill %' + self.sw_path)
    self.cmd('wait')
    self.deleteIntfs()

def attach(self, intf):
    "Connect a data port"
    assert(0)

def detach(self, intf):
```

```
    "Disconnect a data port"
    assert(0)
```

check_server.sh (controller)

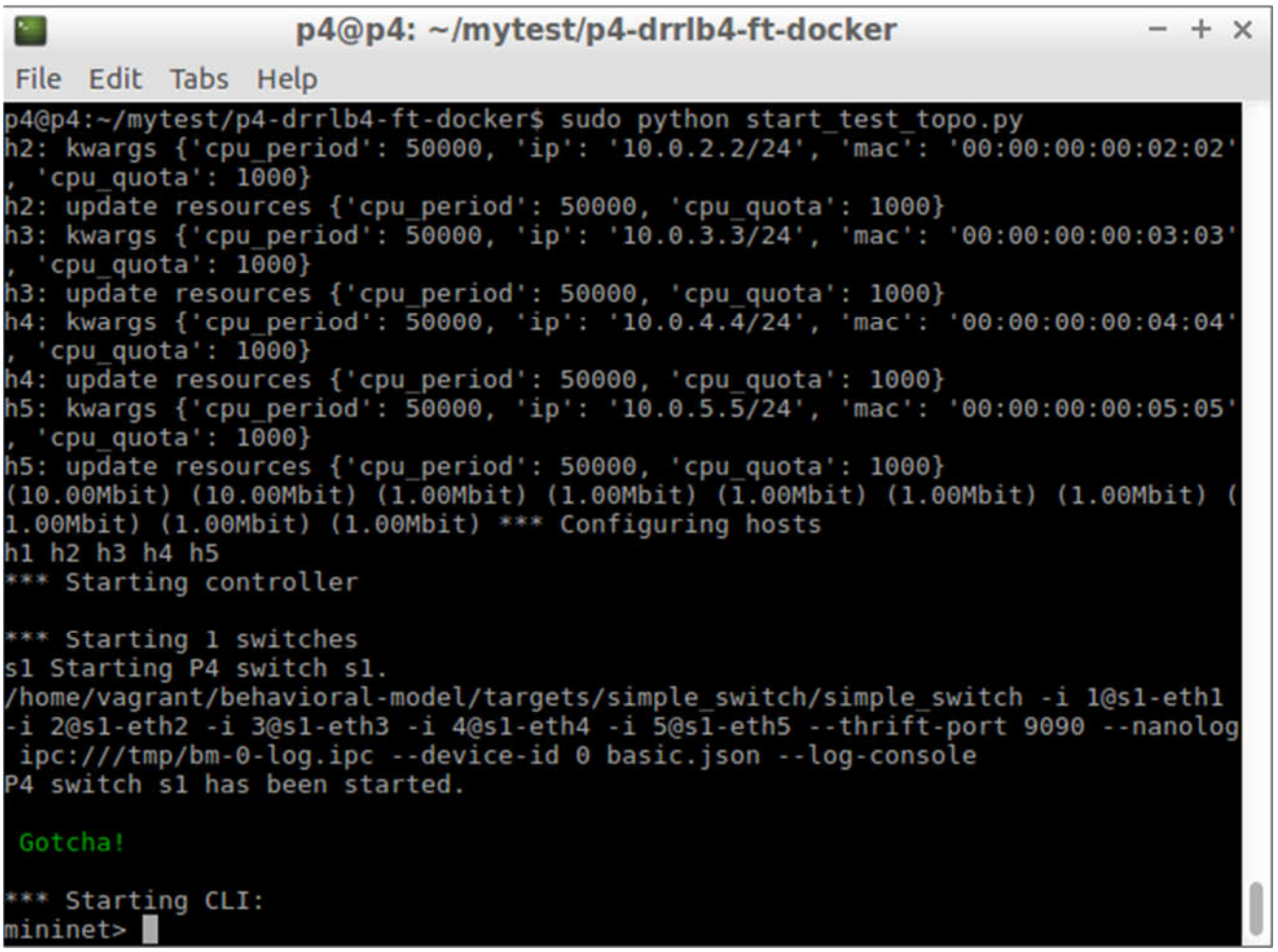
```
#!/bin/bash
CLI_PATH=/home/vagrant/behavioral-
model/targets/simple_switch/simple_switch_CLI
while true
do
   >fail.txt
   >ok.txt
   for ip in `cat ip.txt`
   do
    {
       ping -c1 -W1 $ip &>/dev/null
       if [ $? -ne 0 ]; then
            echo $ip >> fail.txt
       else
            echo $ip >> ok.txt
       fi
    }&
   done
   wait

   if [ -s fail.txt ];then
      for ip in `cat fail.txt`
      do
        #echo $ip
        if [ "$ip" = "172.17.0.2" ];then
            echo "server1 fails"
      echo "table_modify set_status1 _fail1 0 1" | $CLI_PATH --thrift-port
9090 &>/dev/null
        elif [ "$ip" = "172.17.0.3" ];then
      echo "server2 fails"
      echo "table_modify set_status2 _fail2 0 1" | $CLI_PATH --thrift-port
9090 &>/dev/null
        elif [ "$ip" = "172.17.0.4" ];then
            echo "server3 fails"
```

```
      echo "table_modify set_status3 _fail3 0 1" | $CLI_PATH --thrift-port
9090 &>/dev/null
        elif [ "$ip" = "172.17.0.5" ];then
            echo "server4 fails"
      echo "table_modify set_status4 _fail4 0 1" | $CLI_PATH --thrift-port
9090 &>/dev/null
        fi
       done
   fi

   if [ -s ok.txt ];then
       for ip in `cat ok.txt`
       do
        #echo $ip
        if [ "$ip" = "172.17.0.2" ];then
            echo "server1 ok"
      echo "table_modify set_status1 _fail1 0 0" | $CLI_PATH --thrift-port
9090 &>/dev/null
        elif [ "$ip" = "172.17.0.3" ];then
      echo "server2 ok"
      echo "table_modify set_status2 _fail2 0 0" | $CLI_PATH --thrift-port
9090 &>/dev/null
        elif [ "$ip" = "172.17.0.4" ];then
            echo "server3 ok"
      echo "table_modify set_status3 _fail3 0 0" | $CLI_PATH --thrift-port
9090 &>/dev/null
        elif [ "$ip" = "172.17.0.5" ];then
            echo "server4 ok"
      echo "table_modify set_status4 _fail4 0 0" | $CLI_PATH --thrift-port
9090 &>/dev/null
        fi
       done
   fi

   sleep 1
done
```

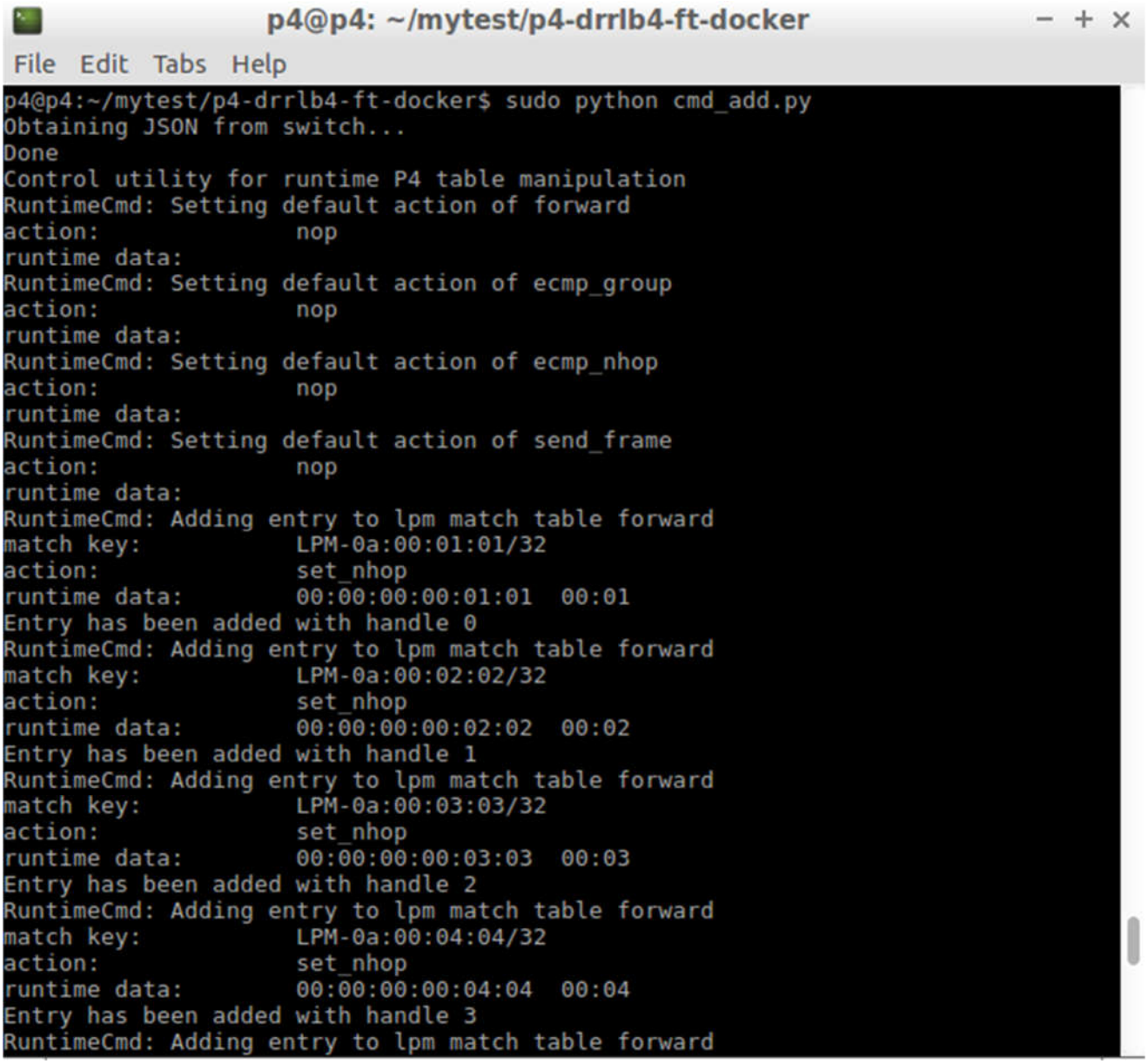
p.s. You need to prepare a ip.txt file. In this file. Add the server ip address in it. For example,

```
172.17.0.2
172.17.0.3
172.17.0.4
172.17.0.5
```

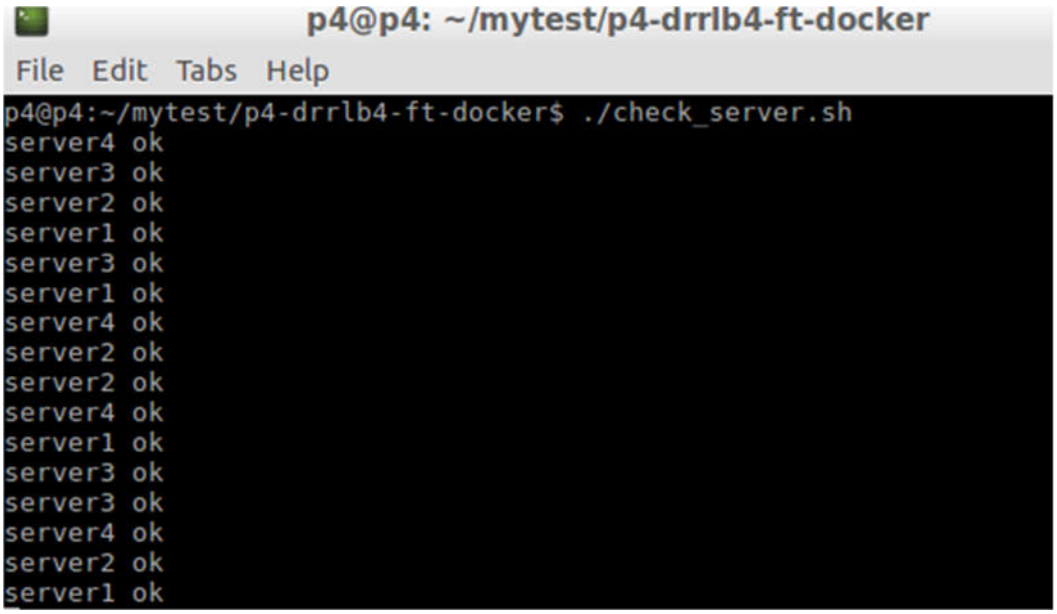
Execution



Add the rules to the P4 Load Balancer Switch (open another terminal)

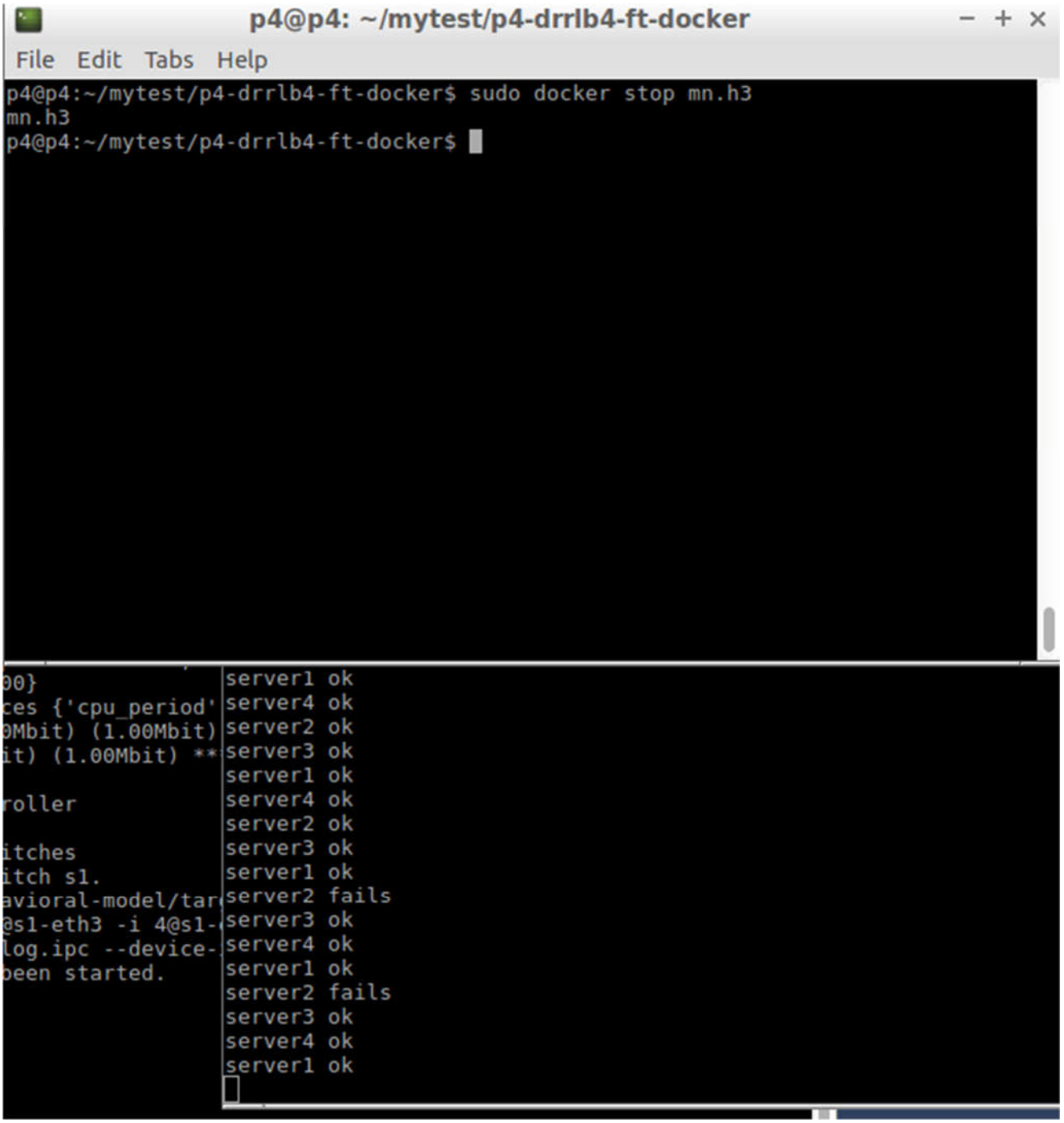


Open another terminal to execute the controller (monitor servers' status)



In mininet, open a terminal for h1 (client)

Open another terminal to make server2 down and see the status of controller.



In mininet h1, you can still get the webpage. But P4 LB won't dispatch the request to server 2.

Dr. Chih-Heng Ke (smallko@gmail.com)
Department of Computer Science and Information Engineering,
National Quemoy University, Kinmen, Taiwan.