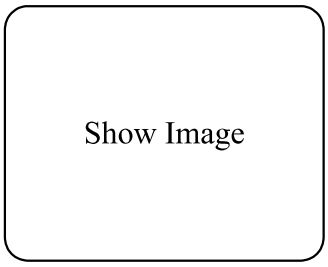
Day 4 - SDN Programming Lab Guide

Perfect Training Center - Software-Defined Networking Hands-On



📋 Lab Overview

- **Duration:** 30-35 minutes
- **Difficulty:** Intermediate
- **Focus:** SDN programming, flow control, network applications
- **Requirements:** Browser, Python basics helpful

🎯 Lab Objectives

By completing this lab, you will:

1. Understand SDN architecture practically
2. Program network behavior with Python
3. Create dynamic flow rules
4. Build a load balancer application
5. Implement security policies
6. Monitor and analyze traffic

🔬 Main Lab Exercise: SDN Network Programming

Part 1: Environment Setup (5 minutes)

Step 1: Access SDN Lab

1. **Open browser** (Chrome/Firefox recommended)
2. **Navigate to:** sdn-lab.perfect-tc.com

3. **Select workspace:** SDN-Programming-Lab

Step 2: Login

```
Username: student@perfect-tc
Password: SDN2024!
Lab ID: Day4-[YourName]
```

Step 3: Understand the Environment

You have access to:

- **Mininet:** Network emulator with virtual switches/hosts
- **ONOS Controller:** SDN controller with REST APIs
- **Python IDE:** For writing network applications
- **Wireshark:** For packet analysis

Initial Topology:

```
h1 ----          ---- h3
     \        /
      s1 --- s2
     /        \
h2 ----          ---- h4

h = host, s = switch
```

Part 2: Explore SDN Basics (5 minutes)

Step 1: Check Initial Connectivity

Open terminal 1:

```
bash

# Start Mininet with remote controller
sudo mn --topo tree,2,2 --controller remote,ip=127.0.0.1

# In Mininet prompt, test connectivity
mininet> pingall
```

Expected Output:

```
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)
```

Step 2: Examine Flow Tables

Open terminal 2:

```bash
# Check flows on switch s1
sudo ovs-ofctl dump-flows s1

# Initially empty or only default flows
```

Step 3: Access Controller UI

1. Open browser tab
2. Navigate to: `http://localhost:8181/onos/ui`
3. Login: `onos/rocks`
4. Explore:
   - Topology view
   - Device list
   - Flow tables

📸 **Screenshot Point:** Capture the ONOS topology view

Part 3: Program Your First SDN App (10 minutes)

Step 1: Simple Forwarding Application

Create file: `simple_forward.py`

```python
```

```
#!/usr/bin/env python3
"""
Simple SDN Forwarding Application
Perfect Training Center - Day 4 Lab
"""

from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import CONFIG_DISPATCHER, MAIN_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.ofproto import ofproto_v1_3
from ryu.lib.packet import packet, ethernet, ether_types

class SimpleForward(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]

    def __init__(self, *args, **kwargs):
        super(SimpleForward, self).__init__(*args, **kwargs)
        self.mac_to_port = {}  # MAC learning table

    @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
    def switch_features_handler(self, ev):
        """Handle switch connection"""
        datapath = ev.msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        # Install default flow to send packets to controller
        match = parser.OFPMatch()
        actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,
                                          ofproto.OFPCML_NO_BUFFER)]
        self.add_flow(datapath, 0, match, actions)
        self.logger.info("Switch %s connected", datapath.id)

    def add_flow(self, datapath, priority, match, actions, buffer_id=None):
        """Add flow to switch"""
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,
                                             actions)]
        if buffer_id:
            mod = parser.OFPFlowMod(datapath=datapath, buffer_id=buffer_id,
```

```
                    priority=priority, match=match,
                    instructions=inst)
        else:
            mod = parser.OFPFlowMod(datapath=datapath, priority=priority,
                    match=match, instructions=inst)
        datapath.send_msg(mod)

    @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
    def packet_in_handler(self, ev):
        """Handle packets sent to controller"""
        msg = ev.msg
        datapath = msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser
        in_port = msg.match['in_port']

        # Parse packet
        pkt = packet.Packet(msg.data)
        eth = pkt.get_protocols(ethernet.ethernet)[0]

        if eth.ethertype == ether_types.ETH_TYPE_LLDP:
            return  # Ignore LLDP packets

        dst = eth.dst
        src = eth.src
        dpid = datapath.id

        self.logger.info("Packet in %s %s %s %s", dpid, src, dst, in_port)

        # Learn MAC address
        self.mac_to_port[dpid] = self.mac_to_port.get(dpid, {})
        self.mac_to_port[dpid][src] = in_port

        # Decide output port
        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 flow if not flooding
        if out_port != ofproto.OFPP_FLOOD:
            match = parser.OFPMatch(in_port=in_port, eth_dst=dst, eth_src=src)
```

```
        self.add_flow(datapath, 1, match, actions, msg.buffer_id)
        self.logger.info("Flow added: %s -> %s via port %s", src, dst, out_port)

    # Send packet out
    data = None
    if msg.buffer_id == ofproto.OFP_NO_BUFFER:
        data = msg.data

    out = parser.OFPPacketOut(datapath=datapath, buffer_id=msg.buffer_id,
                              in_port=in_port, actions=actions, data=data)
    datapath.send_msg(out)
```

Step 2: Run Your SDN App

```bash
# Terminal 3: Run the controller application
ryu-manager simple_forward.py --verbose
```

Step 3: Test the Application

Back in Mininet terminal:

```bash
# Test connectivity again
mininet> pingall

# Check flows now
mininet> sh ovs-ofctl dump-flows s1
```

Expected: You'll see flow rules being installed dynamically!

Part 4: Build a Load Balancer (12 minutes)

Step 1: Create Load Balancer Application

Create file: `load_balancer.py`

```python
```

```python
#!/usr/bin/env python3
"""
SDN Load Balancer Application
Perfect Training Center - Day 4 Lab
"""

from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import CONFIG_DISPATCHER, MAIN_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.ofproto import ofproto_v1_3
from ryu.lib.packet import packet, ethernet, ether_types, ipv4, tcp
import random

class LoadBalancer(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]

    def __init__(self, *args, **kwargs):
        super(LoadBalancer, self).__init__(*args, **kwargs)

        # Virtual IP for load balancer
        self.virtual_ip = '10.0.0.100'
        self.virtual_mac = '00:00:00:00:00:FF'

        # Backend servers
        self.servers = [
            {'ip': '10.0.0.3', 'mac': '00:00:00:00:00:03', 'port': 3},
            {'ip': '10.0.0.4', 'mac': '00:00:00:00:00:04', 'port': 4}
        ]

        # Server selection (round-robin)
        self.server_index = 0

    def get_next_server(self):
        """Round-robin server selection"""
        server = self.servers[self.server_index]
        self.server_index = (self.server_index + 1) % len(self.servers)
        self.logger.info("Selected server: %s", server['ip'])
        return server

    @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
    def switch_features_handler(self, ev):
        """Handle switch connection"""
```

```
    datapath = ev.msg.datapath
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser

    # Default flow to controller
    match = parser.OFPMatch()
    actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,
                                      ofproto.OFPCML_NO_BUFFER)]
    self.add_flow(datapath, 0, match, actions)

def add_flow(self, datapath, priority, match, actions,
             idle_timeout=0, hard_timeout=0):
    """Add flow to switch"""
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser

    inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,
                                         actions)]
    mod = parser.OFPFlowMod(datapath=datapath, priority=priority,
                            match=match, instructions=inst,
                            idle_timeout=idle_timeout,
                            hard_timeout=hard_timeout)
    datapath.send_msg(mod)

@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
def packet_in_handler(self, ev):
    """Handle packets sent to controller"""
    msg = ev.msg
    datapath = msg.datapath
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser
    in_port = msg.match['in_port']

    # Parse packet
    pkt = packet.Packet(msg.data)
    eth = pkt.get_protocols(ethernet.ethernet)[0]

    # Check if IPv4 packet
    ipv4_pkt = pkt.get_protocol(ipv4.ipv4)
    tcp_pkt = pkt.get_protocol(tcp.tcp)

    if ipv4_pkt and tcp_pkt and ipv4_pkt.dst == self.virtual_ip:
        # This is a request to our virtual IP
        if tcp_pkt.dst_port == 80:  # HTTP traffic
```

```
            self.handle_load_balance(datapath, msg, pkt, in_port)
            return

        # Normal forwarding for other traffic
        self.simple_forward(datapath, msg, pkt, in_port)

    def handle_load_balance(self, datapath, msg, pkt, in_port):
        """Load balance incoming requests"""
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        # Get packet details
        eth = pkt.get_protocols(ethernet.ethernet)[0]
        ipv4_pkt = pkt.get_protocol(ipv4.ipv4)
        tcp_pkt = pkt.get_protocol(tcp.tcp)

        # Select backend server
        server = self.get_next_server()

        # Install flows for load balancing
        # 1. Client -> Server (DNAT)
        match = parser.OFPMatch(
            in_port=in_port,
            eth_type=ether_types.ETH_TYPE_IP,
            ipv4_dst=self.virtual_ip,
            ip_proto=6,  # TCP
            tcp_dst=80
        )

        actions = [
            parser.OFPActionSetField(ipv4_dst=server['ip']),
            parser.OFPActionSetField(eth_dst=server['mac']),
            parser.OFPActionOutput(server['port'])
        ]

        self.add_flow(datapath, 100, match, actions, idle_timeout=30)

        # 2. Server -> Client (SNAT)
        match_reverse = parser.OFPMatch(
            in_port=server['port'],
            eth_type=ether_types.ETH_TYPE_IP,
            ipv4_src=server['ip'],
            ip_proto=6,  # TCP
            tcp_src=80
```

```
        )

        actions_reverse = [
            parser.OFPActionSetField(ipv4_src=self.virtual_ip),
            parser.OFPActionSetField(eth_src=self.virtual_mac),
            parser.OFPActionOutput(in_port)
        ]

        self.add_flow(datapath, 100, match_reverse, actions_reverse,
                      idle_timeout=30)

        # Send first packet
        data = msg.data
        out = parser.OFPPacketOut(
            datapath=datapath, buffer_id=ofproto.OFP_NO_BUFFER,
            in_port=in_port, actions=actions, data=data
        )
        datapath.send_msg(out)

        self.logger.info("Load balanced to server %s", server['ip'])

    def simple_forward(self, datapath, msg, pkt, in_port):
        """Simple L2 forwarding for non-LB traffic"""
        # Simplified forwarding logic
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        eth = pkt.get_protocols(ethernet.ethernet)[0]
        dst = eth.dst

        # For simplicity, flood non-LB traffic
        actions = [parser.OFPActionOutput(ofproto.OFPP_FLOOD)]

        data = msg.data
        out = parser.OFPPacketOut(
            datapath=datapath, buffer_id=ofproto.OFP_NO_BUFFER,
            in_port=in_port, actions=actions, data=data
        )
        datapath.send_msg(out)
```

Step 2: Configure Virtual IP

In Mininet:

```bash
# Add virtual IP to h1 (client)
mininet> h1 ip addr add 10.0.0.100/24 dev h1-eth0

# Start simple web servers on h3 and h4
mininet> h3 python -m SimpleHTTPServer 80 &
mininet> h4 python -m SimpleHTTPServer 80 &
```

Step 3: Test Load Balancer

```bash
# Stop previous controller and start load balancer
# In Terminal 3:
Ctrl+C  # Stop previous app
ryu-manager load_balancer.py --verbose

# In Mininet, from h1, make multiple requests:
mininet> h1 curl 10.0.0.100
# Repeat several times - see traffic going to different servers!

# Check flows
mininet> sh ovs-ofctl dump-flows s1
```

📸 **Screenshot Point:** Capture the flow rules showing load balancing

---

Part 5: Add Security Policies (8 minutes)

Step 1: Enhance with Firewall Rules

Add to `load_balancer.py`:

```python
```

```
def __init__(self, *args, **kwargs):
    # ... existing code ...

    # Firewall rules (blacklist)
    self.blocked_ips = ['10.0.0.99']  # Example blocked IP
    self.blocked_ports = [22, 23]     # Block SSH and Telnet

def is_allowed(self, ipv4_pkt, tcp_pkt):
    """Check if traffic is allowed"""
    # Check IP blacklist
    if ipv4_pkt.src in self.blocked_ips or ipv4_pkt.dst in self.blocked_ips:
        self.logger.warning("Blocked IP: %s -> %s", ipv4_pkt.src, ipv4_pkt.dst)
        return False

    # Check port blacklist
    if tcp_pkt:
        if tcp_pkt.dst_port in self.blocked_ports:
            self.logger.warning("Blocked port: %d", tcp_pkt.dst_port)
            return False

    return True

# In packet_in_handler, add:
if ipv4_pkt and not self.is_allowed(ipv4_pkt, tcp_pkt):
    # Drop packet - no action
    return
```

Step 2: Test Security Policies

bash

```
# Try to SSH (should be blocked)
mininet> h1 ssh h3
# Connection should timeout

# Try from blocked IP
mininet> h1 ip addr add 10.0.0.99/24 dev h1-eth0
mininet> h1 curl -I 10.0.0.100 --bind-address 10.0.0.99
# Should be blocked
```

Part 6: Monitor and Analyze (5 minutes)

Step 1: Traffic Statistics

Add monitoring to your app:

```python
def __init__(self, *args, **kwargs):
    # ... existing code ...
    self.packet_count = 0
    self.byte_count = 0

def packet_in_handler(self, ev):
    # ... existing code ...
    self.packet_count += 1
    self.byte_count += len(msg.data)

    if self.packet_count % 10 == 0:
        self.logger.info("Stats - Packets: %d, Bytes: %d",
                         self.packet_count, self.byte_count)
```

Step 2: Visualize Flows

1. Go to ONOS UI
2. Click on "Flows" view
3. See your programmed flows
4. Watch them expire and refresh

Step 3: Performance Test

```bash
# Generate traffic
mininet> h1 ping -c 100 -f 10.0.0.100

# Check flow stats
mininet> sh ovs-ofctl dump-flows s1 --stats
```

📸 **Screenshot Point:** Capture flow statistics

🛠️ Bonus Exercises

Exercise 1: Quality of Service (QoS)

Add traffic prioritization:

```python
# Prioritize video traffic (port 554 - RTSP)
if tcp_pkt and tcp_pkt.dst_port == 554:
    actions.insert(0, parser.OFPActionSetQueue(1))  # High priority queue
```

Exercise 2: DDoS Protection

Implement rate limiting:

```python
# Track connections per source IP
self.connections = {}

# In packet handler:
if ipv4_pkt.src in self.connections:
    if self.connections[ipv4_pkt.src] > 100:  # Max 100 connections
        self.logger.warning("DDoS detected from %s", ipv4_pkt.src)
        return  # Drop
```

Exercise 3: Network Slicing

Create isolated virtual networks:

```python
# Assign VLAN based on source
if ipv4_pkt.src.startswith('10.0.1'):
    actions.append(parser.OFPActionPushVlan())
    actions.append(parser.OFPActionSetField(vlan_vid=100))
```

🐛 Troubleshooting Guide

Common Issues:

"Connection refused to controller"

```bash
# Check if controller is running
ps aux | grep ryu
# Restart if needed
ryu-manager your_app.py
```

"No flows installed"

- Check controller logs for errors
- Verify match fields are correct
- Ensure packet-in handler is called

"Ping doesn't work"

- Check ARP is handled
- Verify bidirectional flows
- Look for packet drops

"Load balancer not distributing"

- Confirm servers are reachable
- Check round-robin logic
- Verify flow timeouts

📊 Lab Report Template

Complete and submit:

```markdown
```

```
# SDN Lab Report

**Name:** _____
**Date:** _____

## Application Built
- [ ] Simple Forwarding
- [ ] Load Balancer
- [ ] Firewall Rules
- [ ] Traffic Monitor

## Key Flows Installed
1. _____
2. _____
3. _____

## Performance Metrics
- Packets Processed: _____
- Flows Installed: _____
- Latency Added: _____ms
- Throughput: _____Mbps

## Challenges & Solutions
1. _____
2. _____

## Key Learnings
1. _____
2. _____
3. _____

## Screenshots
- [ ] ONOS Topology View
- [ ] Flow Table
- [ ] Load Balancing in Action
- [ ] Traffic Statistics
```

💡 Key Takeaways

After this lab, you've:

✅ **Mastered SDN Concepts**

- Understood control/data plane separation
- Programmed network behavior
- Seen OpenFlow in action

✅ **Built Real Applications**

- L2 learning switch
- L4 load balancer
- Security policies
- Traffic monitoring

✅ **Gained Practical Skills**

- Python for networking
- Flow table management
- Network debugging
- Performance analysis

✅ **Prepared for Future**

- Ready for SDN roles
- Understand programmable networks
- Can build network apps

🚀 Next Steps

This Week:

1. **Enhance Applications**
   - Add more sophisticated algorithms
   - Implement east-west LB
   - Create stateful firewall
2. **Learn More Controllers**
   - Try OpenDaylight
   - Explore ONOS apps
   - Use P4 programming

3. **Build Portfolio**
    - GitHub your code
    - Blog about experience
    - Create video demo

Career Path:

1. **SDN Certifications**
    - ONF Certified SDN Engineer
    - Cisco SDN specialist
    - VMware NSX
2. **Advanced Topics**
    - P4 programming
    - Service mesh
    - Intent-based networking
3. **Real Deployments**
    - Home lab with Open vSwitch
    - Contribute to open source
    - Internship opportunities

🤝 Support Resources

Need Help?

- **During Lab:** Instructor support
- **After Hours:** sdn@perfect-tc.com
- **Community:** Join SDN Slack channel
- **Office Hours:** Tuesday 3-5 PM

Documentation:

- Ryu Docs: ryu-sdn.org
- OpenFlow Spec: opennetworking.org
- ONOS Guides: onosproject.org
- Mininet Walkthrough: mininet.org

🎯 Challenge Extensions

Advanced Challenge 1: Stateful Load Balancer

- Track TCP sessions
- Ensure session persistence
- Handle connection failures

Advanced Challenge 2: ML-Based Routing

- Collect traffic patterns
- Predict optimal paths
- Implement adaptive routing

Innovation Challenge:
Design SDN solution for:

- Egyptian smart highway system
- Nile River monitoring network
- Desert IoT connectivity

Prize: Best solution presented at tech conference!

🎉 Congratulations!

You've successfully:
✅ Programmed an SDN network
✅ Built multiple applications
✅ Implemented security policies
✅ Created load balancing
✅ Monitored traffic
✅ Debugged flow tables

You're now an SDN Developer!

📸 Social Sharing

Don't forget to:

1. Post your achievement on LinkedIn
2. Tag @PerfectTrainingCenter

3. Use #SDNProgrammer #Day4Complete

4. Share your load balancer demo

5. Connect with classmates

Perfect Training Center - Mastering Tomorrow's Networks

Version: 4.0
Updated: November 2024
Next Review: February 2025

🔗 Quick Commands Reference

Mininet Commands:

```bash
pingall             # Test all connectivity
h1 ping h2          # Specific ping
sh ovs-ofctl dump-flows s1  # Show flows
link s1 s2 down     # Simulate failure
iperf               # Performance test
```

Controller Commands:

```bash
ryu-manager app.py --verbose  # Run app
curl http://localhost:8080/stats/flow/1  # REST API
```

Debugging:

```bash
tcpdump -i s1-eth1  # Packet capture
ovs-appctl ofproto/trace  # Trace packet path
tail -f /var/log/ryu/ryu.log  # Controller logs
```

Ready to Program the Future of Networking? Let's SDN! 🚀