

SecureNet DC

Building and Defending a Software-Defined
Data Center Network

Lab Experiment Document

CPEG 460 - Computer Networks

Fall 2025

Dr. Mohammad Shaqfeh

Duration: 4-5 hours
Difficulty: Advanced
Prerequisites: TCP/IP, Python, Linux basics

Contents

1	Introduction	2
1.1	Overview	2
1.2	Learning Objectives	2
1.3	Background	2
2	Part A: Environment Setup (30 minutes)	3
2.1	Prerequisites	3
2.2	Operating Modes	3
2.3	Quick Setup (Recommended)	3
2.4	Manual Installation (Alternative)	4
2.5	Verify Installation	4
3	Part B: Topology Exploration (45 minutes)	5
3.1	Launch the Network	5
3.2	Explore the Topology	5
3.3	Understanding the Fat-Tree Structure	5
4	Part C: SDN Controller Basics (60 minutes)	7
4.1	Understanding OpenFlow	7
4.2	Examine Flow Tables	7
4.3	L2 Learning Switch Behavior	7
5	Part D: Load Balancer Implementation (45 minutes)	8
5.1	Concept	8
5.2	Configuration	8
5.3	Testing	8
6	Part E: DDoS Attack and Defense (60 minutes)	9
6.1	Attack Types	9
6.2	Detection Algorithm (v3.0)	9
6.3	Detection Thresholds	9
6.4	Recommended: Using the Integrated Demo Runner	9
6.5	Alternative: Manual Attack in Mininet CLI	10
6.6	Capture with Wireshark	10
7	Part F: QoS Configuration (45 minutes)	11
7.1	Traffic Classes	11
7.2	Testing QoS	11
8	Part G: Performance Analysis (30 minutes)	12
8.1	Automated Benchmarking	12
8.2	Expected Results	12
9	Deliverables	13
9.1	Grading Rubric	13
10	Appendix: Useful Commands	14
10.1	Mininet Commands	14
10.2	OpenFlow Commands	14
10.3	Network Tools	14

1 Introduction

1.1 Overview

This lab introduces Software-Defined Networking (SDN) through a comprehensive data center simulation using Mininet. You will explore a Fat-Tree topology, implement SDN controller features, experience DDoS attack detection, and configure Quality of Service (QoS) policies.

1.2 Learning Objectives

Upon completing this lab, you will be able to:

- Understand Fat-Tree data center topology and its advantages
- Program an SDN controller using the Ryu framework
- Implement load balancing using OpenFlow
- Detect and mitigate DDoS attacks in an SDN environment
- Configure QoS policies for traffic prioritization
- Analyze network performance using standard tools

1.3 Background

Software-Defined Networking separates the control plane (decision-making) from the data plane (packet forwarding). The SDN controller makes forwarding decisions and installs flow rules on switches via the OpenFlow protocol.

The Fat-Tree topology is widely used in data centers due to its:

- Full bisection bandwidth
- Multiple paths for fault tolerance
- Scalable, regular structure

2 Part A: Environment Setup (30 minutes)

2.1 Prerequisites

This lab requires WSL2 (Windows Subsystem for Linux 2) with Ubuntu. Ensure you have:

- Windows 10/11 with WSL2 enabled
- Ubuntu 20.04+ distribution installed in WSL2
- Python 3.11 (recommended)
- Mininet 2.3.0+
- Open vSwitch 2.13+

2.2 Operating Modes

The project supports two operating modes depending on your environment:

Table 1: Operating Mode Comparison

Aspect	Linux Bridge Mode (WSL2)	Full SDN Mode (VM)
Switch type	Linux bridges (<code>--switch lxbr</code>)	OVS (<code>--switch ovsk</code>)
Controller	Not required	Ryu controller required
Flow rules	Not supported	Full OpenFlow support
Best for	Traffic generation, detection testing	Complete SDN demonstration
Command	<code>sudo mn --switch lxbr --topo tree,2</code>	<code>sudo mn --controller remote --topo tree,2</code>

Note: WSL2 has issues establishing OpenFlow connections. Use Linux Bridge mode for quick testing, or set up a Linux VM (VirtualBox/VMware) for full SDN features.

2.3 Quick Setup (Recommended)

The project includes an automated setup script that handles all dependencies:

```

1 # Open WSL Ubuntu terminal
2 cd ~/securenet_dc
3
4 # Run the setup script
5 chmod +x scripts/setup_wsl.sh
6 ./scripts/setup_wsl.sh

```

Listing 1: Automated Setup

The script automatically:

- Creates a Python 3.11 virtual environment
- Installs Ryu SDN Framework with correct dependencies
- Installs Flask and dashboard dependencies
- Applies required patches for eventlet compatibility
- Verifies all installations

2.4 Manual Installation (Alternative)

If you prefer manual installation:

```
1 # Update system and install base packages
2 sudo apt-get update
3 sudo apt-get install -y mininet openvswitch-switch python3.11 python3.11-venv
4
5 # Install network testing tools
6 sudo apt-get install -y iperf3 hping3 tcpdump
7
8 # Create and activate virtual environment
9 python3.11 -m venv venv
10 source venv/bin/activate
11
12 # Install Python packages (order matters!)
13 pip install wheel setuptools==57.5.0
14 pip install --no-build-isolation ryu
15 pip install flask flask-socketio flask-cors requests scapy matplotlib
16
17 # Apply eventlet patch (REQUIRED for Ryu to work)
18 WSGI_FILE=$(find venv -name "wsgi.py" -path "*/ryu/app/*" | head -1)
19 sed -i "s/from eventlet.wsgi import ALREADY_HANDLED/ALREADY_HANDLED = b''/" "$WSGI_FILE"
```

Listing 2: Manual Installation Commands

2.5 Verify Installation

```
1 # Activate virtual environment
2 source venv/bin/activate
3
4 # Test Mininet
5 sudo mn --test pingall
6
7 # Check OVS
8 sudo ovs-vsctl show
9
10 # Verify Ryu
11 python -c "import ryu; print(f'Ryu version: {ryu.__version__}')"
12
13 # Verify Flask
14 python -c "import flask; print(f'Flask version: {flask.__version__}')"
15
```

Listing 3: Verification

Question A.1: What version of Mininet is installed? What OpenFlow versions does it support?

3 Part B: Topology Exploration (45 minutes)

3.1 Launch the Network

Recommended: One-Click Launch (Windows)

Double-click `START_SECURENET.bat` in the project folder. This automatically:

- Syncs files to WSL
- Opens 3 Windows Terminal tabs with all components

Alternative: Manual Launch (3 WSL terminals)

```
1 # TERMINAL 1: Start Mininet with Demo Runner
2 cd ~/securenet_dc
3 sudo python3 scripts/run_demo.py
4
5 # TERMINAL 2: Start Stats Collector (for detection)
6 cd ~/securenet_dc
7 source venv/bin/activate
8 python3 scripts/network_stats_collector.py
9
10 # TERMINAL 3: Start the Dashboard
11 cd ~/securenet_dc
12 source venv/bin/activate
13 python3 dashboard/app.py
```

Listing 4: Starting the System Manually

Simple Test (without dashboard):

```
1 # Quick test with Linux Bridge mode
2 sudo mn --switch lxbr --topo tree,depth=2,fanout=3
```

3.2 Explore the Topology

In the Mininet CLI, execute:

```
1 # View all nodes
2 mininet> nodes
3
4 # View network connections
5 mininet> net
6
7 # View detailed information
8 mininet> dump
9
10 # Test connectivity
11 mininet> pingall
```

Listing 5: Topology Exploration Commands

3.3 Understanding the Fat-Tree Structure

The topology has three layers:

- **Core Layer:** 4 switches (c1-c4)
- **Aggregation Layer:** 8 switches (a1-a8), 2 per pod
- **Edge Layer:** 8 switches (e1-e8), 2 per pod

- **Hosts:** 16 hosts (h1-h16), distributed across 4 pods

Table 2: Host Configuration

Hosts	IP Range	Role
h1-h4	10.0.1.x	Web Servers
h5-h6	10.0.2.x	Database Servers
h7-h12	10.0.3.x	Client Hosts
h13	10.0.4.1	Attacker
h14	10.0.4.2	IDS Monitor
h15-h16	10.0.5.x	Streaming Servers

Question B.1: How many hops does a packet travel from h1 to h4 (same pod)? From h1 to h16 (different pod)?

Question B.2: Calculate the theoretical bisection bandwidth of this k=4 Fat-Tree.

4 Part C: SDN Controller Basics (60 minutes)

4.1 Understanding OpenFlow

OpenFlow defines how the controller communicates with switches:

- **Packet-In:** Switch sends packet to controller for decision
- **Flow-Mod:** Controller installs flow rules on switch
- **Packet-Out:** Controller tells switch how to forward a packet

4.2 Examine Flow Tables

```
1 # View flows on switch s1
2 mininet> sh ovs-ofctl dump-flows e1
3
4 # View flows with statistics
5 mininet> sh ovs-ofctl dump-flows e1 --stats
```

Listing 6: Viewing Flow Tables

4.3 L2 Learning Switch Behavior

Generate traffic and observe flow rule installation:

```
1 # Generate traffic
2 mininet> h1 ping -c 5 h7
3
4 # Check new flow rules
5 mininet> sh ovs-ofctl dump-flows e1
```

Question C.1: Describe the flow rules installed after the ping. What match fields are used?

Challenge C.1: Modify the controller to log every packet-in event with source and destination MAC addresses.

5 Part D: Load Balancer Implementation (45 minutes)

5.1 Concept

The load balancer uses a Virtual IP (VIP) to distribute traffic across multiple backend servers. When a client connects to the VIP, the controller selects a server and rewrites packet headers.

5.2 Configuration

- VIP: 10.0.0.100
- Server Pool: h1 (10.0.1.1), h2 (10.0.1.2), h3 (10.0.1.3), h4 (10.0.1.4)
- Algorithm: Round-robin

5.3 Testing

```
1 # Start web servers
2 mininet> h1 python3 -m http.server 80 &
3 mininet> h2 python3 -m http.server 80 &
4 mininet> h3 python3 -m http.server 80 &
5 mininet> h4 python3 -m http.server 80 &
6
7 # Send requests to VIP from client
8 mininet> h7 curl http://10.0.0.100/
9
10 # Send multiple requests
11 mininet> h7 for i in $(seq 1 8); do curl -s http://10.0.0.100/; done
```

Listing 7: Load Balancer Testing

Question D.1: How does the controller handle ARP requests for the VIP?

Question D.2: What OpenFlow actions are used to redirect traffic to backend servers?

Challenge D.1: Implement weighted round-robin where h1 gets 50% of traffic.

6 Part E: DDoS Attack and Defense (60 minutes)

6.1 Attack Types

The detection engine monitors for:

- **SYN Flood:** Many TCP SYN packets without completing handshake
- **ICMP Flood:** Excessive ping requests
- **UDP Flood:** High volume of UDP packets

6.2 Detection Algorithm (v3.0)

The detection system uses TX-based detection to accurately identify attackers:

- High RX on switch port = host is **SENDING** = ATTACKER
- High TX on switch port = host is **RECEIVING** = VICTIM (do not block)
- IP-based deduplication prevents duplicate blocking
- 5-second cooldown period prevents rapid re-detection

6.3 Detection Thresholds

Table 3: DDoS Detection Thresholds

Attack Type	Threshold (pps)
SYN Flood	100
ICMP Flood	50
UDP Flood	200
Total Traffic	500

6.4 Recommended: Using the Integrated Demo Runner

The easiest way to run attacks is using the integrated demo runner, which handles Mininet and attacks in one process:

```

1 # Start the integrated demo (from Windows)
2 # Double-click START_SECURENET.bat
3
4 # Or manually in WSL:
5 cd ~/securenet_dc
6 source venv/bin/activate
7 sudo python3 scripts/run_demo.py
8
9 # The interactive menu provides:
10 # 1. ICMP Flood Attack
11 # 2. SYN Flood Attack
12 # 3. UDP Flood Attack
13 # 4. Ping of Death
14 # 5. Multi-Vector Attack
15 # 6. Run Demo Scenario (recommended for presentation)
16 # 7. Test Connectivity
17 # 8. Show Status
18 # 9. Open Mininet CLI
19 # 0. Exit

```

Listing 8: Integrated Demo Runner

6.5 Alternative: Manual Attack in Mininet CLI

```
1 # Baseline: Normal traffic from h1
2 mininet> h1 ping h2
3
4 # Launch ICMP flood attack (simple, no extra tools needed)
5 mininet> h4 ping -f -c 1000 10.0.0.1
6
7 # For SYN flood (requires hping3 installed)
8 mininet> h4 hping3 -S --flood -p 80 10.0.0.1
9
10 # For UDP flood
11 mininet> h4 hping3 --udp --flood -p 53 10.0.0.1
12
13 # Alternative: Slow attack for better visibility
14 mininet> h4 ping -i 0.01 10.0.0.1
```

Listing 9: DDoS Simulation in Mininet

Note: The stats collector logs will show detection alerts like:

```
1 [DDoS] ATTACK DETECTED on s2-eth2
2 [DDoS] Attacker: h4 (10.0.0.4)
3 [DDoS] RX Rate: 15234 pps (threshold: 1000 pps)
4 [DDoS] BLOCKING host h4 (10.0.0.4)
```

6.6 Capture with Wireshark

```
1 # Start capture on edge switch interface
2 mininet> sh wireshark -i e1-eth1 -k &
3
4 # Rerun attack while capturing
```

Question E.1: How quickly was the attack detected? What was the packet rate?

Question E.2: What flow rule was installed to block the attacker?

Challenge E.1: Modify the detection threshold and test different values.

7 Part F: QoS Configuration (45 minutes)

7.1 Traffic Classes

Table 4: QoS Queue Configuration

Queue	Class	Min BW	Ports
0	Critical	50%	SSH (22), DNS (53)
1	Real-time	30%	RTSP (554), Streaming (5001)
2	Interactive	15%	HTTP (80, 443)
3	Bulk	5%	FTP (21), General

7.2 Testing QoS

```

1 # Start iperf servers
2 mininet> h15 iperf3 -s &
3 mininet> h5 iperf3 -s -p 5002 &
4
5 # Generate competing traffic
6 # High-priority (streaming)
7 mininet> h7 iperf3 -c 10.0.5.1 -u -b 50M -t 30 &
8
9 # Low-priority (bulk)
10 mininet> h8 iperf3 -c 10.0.2.1 -p 5002 -t 30 &
11
12 # Observe that streaming maintains throughput

```

Listing 10: QoS Testing

Question F.1: What DSCP values are assigned to each traffic class?

Question F.2: How does the HTB qdisc enforce bandwidth guarantees?

Challenge F.1: Create a custom QoS policy prioritizing your own application.

8 Part G: Performance Analysis (30 minutes)

8.1 Automated Benchmarking

```

1 # Throughput test
2 mininet> iperf h7 h1
3
4 # Latency test
5 mininet> h7 ping -c 20 h1
6
7 # Cross-pod latency
8 mininet> h7 ping -c 20 h16

```

Listing 11: Performance Testing

8.2 Expected Results

Table 5: Expected Performance Metrics

Test	Metric	Expected Value
Same-pod throughput	Bandwidth	94 Mbps
Cross-pod throughput	Bandwidth	90 Mbps
Same-pod latency	RTT	2-3 ms
Cross-pod latency	RTT	4-5 ms

Question G.1: Compare intra-pod vs inter-pod latency. Explain the difference.

Challenge G.1: Generate performance comparison graphs using the provided tools.

9 Deliverables

Submit the following:

1. **Written Answers:** Responses to all questions (A.1, B.1-B.2, C.1, D.1-D.2, E.1-E.2, F.1-F.2, G.1)
2. **Wireshark Captures:**
 - OpenFlow handshake messages
 - Attack traffic capture with annotations
 - Load balancer flow rules
3. **Performance Data:**
 - Throughput measurements (screenshot or data)
 - Latency measurements
 - Generated graphs
4. **Challenge Implementations** (for bonus points):
 - Code modifications
 - Test results
5. **Analysis Report:** 2-page report discussing:
 - Key observations from each part
 - Challenges encountered
 - Comparison of expected vs actual results

9.1 Grading Rubric

Table 6: Lab Grading Rubric

Component	Points	Criteria
Part A: Setup	10	Environment verified, questions answered
Part B: Topology	15	Exploration complete, calculations correct
Part C: SDN Basics	15	Flow rules explained, OpenFlow understood
Part D: Load Balancer	15	VIP tested, NAT mechanism explained
Part E: DDoS Defense	20	Attack simulated, detection observed
Part F: QoS Config	15	Traffic classes tested, DSCP understood
Part G: Performance	10	Measurements taken, analysis provided
Total	100	
Challenge Bonus	+20	Successful challenge implementations

10 Appendix: Useful Commands

10.1 Mininet Commands

```
1 nodes          # List all nodes
2 net            # Show network topology
3 dump           # Detailed node info
4 pingall        # Test all-to-all connectivity
5 iperf h1 h2     # Bandwidth test
6 xterm h1       # Open terminal for h1
```

10.2 OpenFlow Commands

```
1 ovs-ofctl dump-flows s1      # View flow table
2 ovs-ofctl dump-ports s1     # View port statistics
3 ovs-vsctl show               # View OVS configuration
```

10.3 Network Tools

```
1 ping -c 10 <ip>             # Latency test
2 iperf3 -s                   # Start server
3 iperf3 -c <ip> -t 10        # Bandwidth test
4 tcpdump -i <interface>      # Packet capture
```