Mininet Installation & Access Documentation

📌 Part 1: Download Mininet from mininet.org

1. Go to the official website:
🔗 https://mininet.org/download/

2. Under the "Mininet VM Images" section, download the latest .ova file:
Example: Mininet-2.3.0-210420.ova
(Contains Ubuntu + Mininet + POX controller pre-installed)

3. Save the file to a known location (e.g., Downloads).

📌 Part 2: Install Oracle VirtualBox

1. Download VirtualBox from the official site:
🔗 https://www.virtualbox.org/wiki/Downloads

2. Install it like any standard application:
• Accept default options
• After installation, open Oracle VM VirtualBox

📌 Part 3: Import Mininet VM into VirtualBox

1. Open VirtualBox → Click File → Import Appliance

2. Browse and select the downloaded .ova file

3. Click Next → then Import

4. Once imported, select the VM and click Start

📌 Part 4: Access Mininet VM from Command Prompt via SSH

🔸 A. Setup Networking (choose one):

• Bridged Adapter

o Go to VirtualBox → Mininet VM → Settings → Network

o Set Attached to: Bridged Adapter

o Inside the VM, run:

```
ip a
```

o Note the IP (e.g., 192.168.x.x)

Then from Windows CMD:

```
ssh mininet@<VM_IP_ADDRESS>
```

• NAT with Port Forwarding (Recommended for Windows)

1. Go to VM Settings → Network → Enable NAT

2. Click Advanced → Port Forwarding

3. Add rule:
   - Name: SSH
   - Protocol: TCP
   - Host Port: 2223
   - Guest Port: 22

From Windows CMD:

```
ssh mininet@127.0.0.1 -p 2223
```

o Password: mininet

First Steps After Installing & Logging In via SSH

- 1. Update System Packages

```
sudo apt update && sudo apt upgrade -y
```

- 2. Test Mininet Installation

```
sudo mn --test pingall
```

This command:
• Launches a basic Mininet topology
• Tests end-to-end connectivity with ping between all hosts

Expected output:
Successful ping results showing no packet loss.

📌 Part 5: Course Project – SDN: Dynamic Link Failure Recovery

Project Title:

Dynamic Link Failure Recovery in Software Defined Networking (SDN)

Project Objective:

To design and implement a system in SDN that automatically detects link failures in the network and dynamically reroutes the traffic using alternate paths, thereby minimizing downtime and packet loss.

Concept Overview:

In traditional networks, failure recovery is handled locally by routers and can be slow. SDN, however, separates the control plane from the data plane, allowing a central controller to manage the entire network with a global view. This centralized control enables intelligent, real-time, and automated failure recovery mechanisms.

Objectives of the Project:

1. Failure Detection

- The controller receives immediate notifications when any network link goes down.
- This is faster than traditional methods relying on routing protocol convergence.

2. Minimal Downtime

- The proposed system provides automated and rapid recovery from failures,
- Minimizes packet loss and avoids manual reconfiguration.

SDN Project Documentation: Handling Link Failures with Mininet

Project Goal

The goal of this project is to simulate a network using Mininet and the POX controller to demonstrate how a Software-Defined Network (SDN) can automatically recover from link failures. This showcases SDN's resilience and intelligent traffic control capabilities.

How the Project Works

Network Setup

Mininet is used to create a virtual topology with interconnected virtual switches and hosts.

POX Controller Logic

- Topology Discovery: Builds a real-time map from OpenFlow messages.
- Link Failure Detection: Detects link-down events.
- Flow Rule Updates: Installs new flow rules to re-establish connectivity.

Link Failure Simulation

```
To bring down a link:

link s1 s3 down

To restore a link:

link s1 s3 up
```

Setting Everything Up

Environment Preparation

```
Ensure Mininet and POX are installed. Verify with:

cd ~/pox
```

Project Files

Place the following scripts in /home/mininet/pox/ext/:

- triangle_topo.py
- failover_controller.py

Starting POX Controller

```
cd ~/pox

./pox.py openflow.discovery failover_controller
```

Creating the Network

```
sudo python /home/mininet/pox/ext/create_topology.py
```

Testing Link Failures

Simulate and restore links in Mininet CLI:

```
link s1 s3 down
link s1 s3 up
```

Testing Host Connectivity

Run ping test before and after failures:

```
h1 ping h2
```

Cleanup

```
sudo mn -c
```

Custom Topology and Controller for Link Failover Recovery

Custom Triangle Topology

To create a simple triangle-shaped topology with two hosts and three switches, use the following Python script:

Filename: triangle_topo.py

```
from mininet.topo import Topo

class TriangleTopo(Topo):
    """Simple triangle topology with 3 switches and 2 hosts.
       Creates a triangle with two paths between h1 and h2:

       h1 --- s1 --- s2 --- h2
              |      |
              +------+

       This creates two paths: h1->s1->s2->h2 and h1->s1->s3->s2->h2
    """

    def build(self):
        # Add hosts
        h1 = self.addHost('h1')
        h2 = self.addHost('h2')

        # Add switches
        s1 = self.addSwitch('s1')
        s2 = self.addSwitch('s2')
        s3 = self.addSwitch('s3')

        # Add links
        self.addLink(h1, s1)
        self.addLink(h2, s2)
        self.addLink(s1, s2)  # Primary path
        self.addLink(s1, s3)  # Backup path part 1
        self.addLink(s3, s2)  # Backup path part 2

topos = {'triangle': (lambda: TriangleTopo())}
```

Run with Mininet:

```
sudo mn --custom triangle_topo.py --topo triangle --controller=remote
```

POX Failover Controller Logic

This custom POX controller installs flooding rules and prepares the network to reroute in the event of a failure.

Filename: failover_controller.py

```
from pox.core import core
import pox.openflow.libopenflow_01 as of
from pox.lib.revent import *
from pox.lib.util import dpidToStr

log = core.getLogger()

class FailoverController(EventMixin):

    def _init_(self):
        self.listenTo(core.openflow)
        log.info("Failover Controller initialized")

    def _handle_ConnectionUp(self, event):
        log.info("Switch %s has connected", dpidToStr(event.dpid))
        msg = of.ofp_flow_mod()
        msg.match = of.ofp_match()
        msg.actions.append(of.ofp_action_output(port=of.OFPP_FLOOD))
        event.connection.send(msg)

def launch():
    core.registerNew(FailoverController)
```

Run the controller:

```
cd ~/pox
./pox.py failover_controller
```

Helpful References

- Mininet: https://mininet.org/download/
- VirtualBox: https://www.virtualbox.org/wiki/Downloads
- POX Controller GitHub: https://github.com/noxrepo/pox

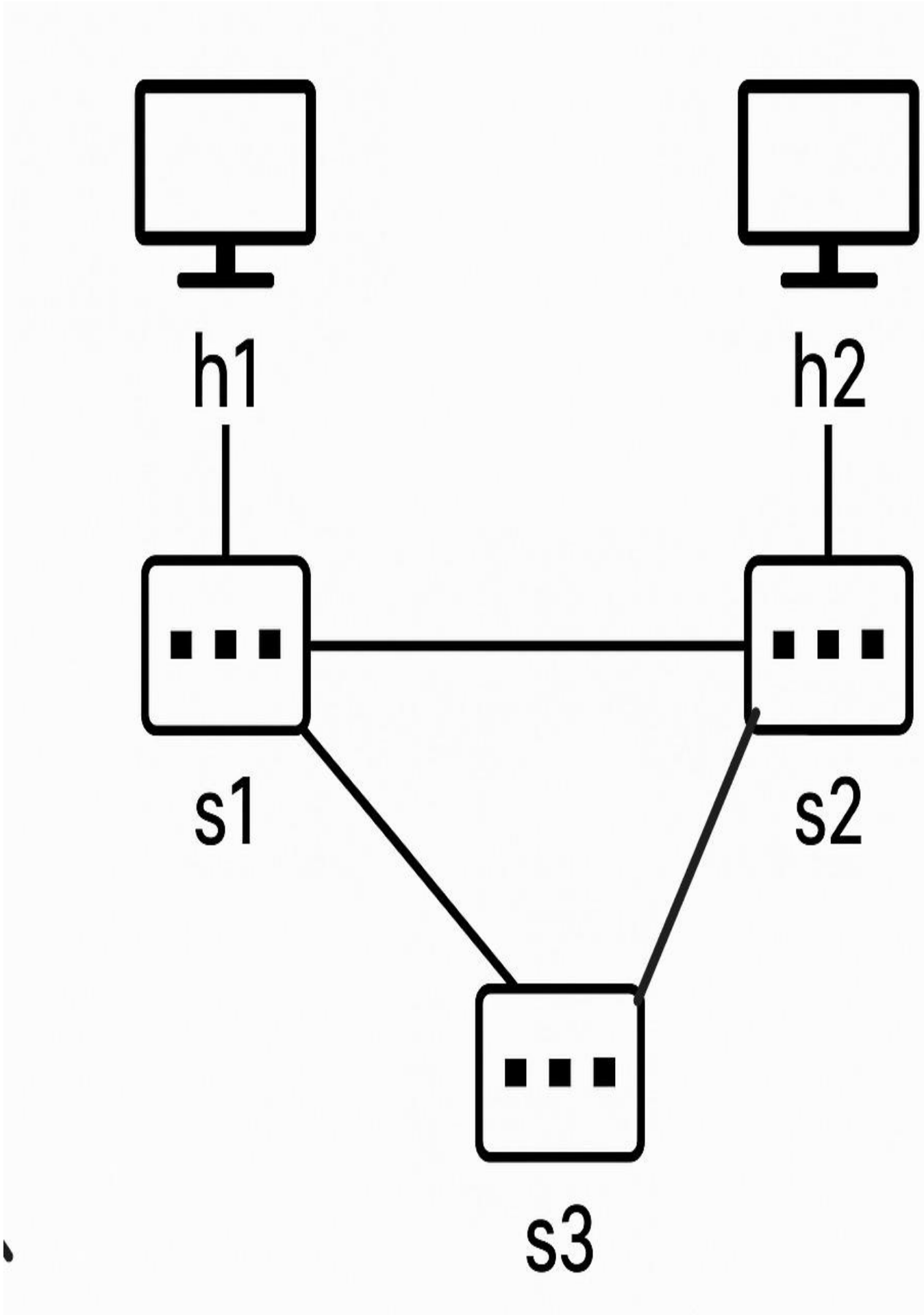
Summary

- Topology: Triangle with redundant links
- Controller: Listens for switches and installs failover rules
- Test Setup:

```
link s1 s2 down

h1 ping h2
```

Topology Diagram


h1
h2
s1
s2
s3

Output

```
mininet@mininet-vm: ~/pox    mininet@mininet-vm: ~
h1 h2
*** Adding switches:
s1 s2 s3
*** Adding links:
(h1, s1) (h2, s2) (s1, s2) (s1, s3) (s3, s2)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 3 switches
s1 s2 s3 ...
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
mininet> link s1 s2 down
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
mininet> link s2 s3 down
mininet> pingall
*** Ping: testing ping reachability
h1 -> X
h2 -> X
*** Results: 100% dropped (0/2 received)
mininet>
```

```
mininet@mininet-vm:~/pox$ ./pox.py openflow.discovery final_dijkstra
POX 0.7.0 (gar) / Copyright 2011-2020 James McCauley, et al.
INFO:final_dijkstra:Failover Controller initialized
WARNING:version:POX requires one of the following versions of Python: 3.6 3.7 3.8 3.9
WARNING:version:You're running Python 3.5.
WARNING:version:If you run into problems, try using a supported version.
INFO:core:POX 0.7.0 (gar) is up.
INFO:openflow.of_01:[00-00-00-00-00-06 2] connected
INFO:final_dijkstra:Switch 00-00-00-00-00-06 has connected
INFO:openflow.of_01:[00-00-00-00-00-04 1] connected
INFO:final_dijkstra:Switch 00-00-00-00-00-04 has connected
INFO:openflow.of_01:[00-00-00-00-00-03 4] connected
INFO:final_dijkstra:Switch 00-00-00-00-00-03 has connected
INFO:openflow.of_01:[00-00-00-00-00-01 3] connected
INFO:final_dijkstra:Switch 00-00-00-00-00-01 has connected
INFO:openflow.of_01:[00-00-00-00-00-05 6] connected
INFO:final_dijkstra:Switch 00-00-00-00-00-05 has connected
INFO:openflow.of_01:[00-00-00-00-00-02 5] connected
INFO:final_dijkstra:Switch 00-00-00-00-00-02 has connected
INFO:openflow.discovery:link detected: 00-00-00-00-00-01.3 -> 00-00-00-00-00-03.1
INFO:openflow.discovery:link detected: 00-00-00-00-00-01.2 -> 00-00-00-00-00-02.2
INFO:openflow.discovery:link detected: 00-00-00-00-00-03.2 -> 00-00-00-00-00-02.3
INFO:openflow.discovery:link detected: 00-00-00-00-00-03.1 -> 00-00-00-00-00-01.3
INFO:openflow.discovery:link detected: 00-00-00-00-00-02.2 -> 00-00-00-00-00-01.2
INFO:openflow.discovery:link detected: 00-00-00-00-00-02.3 -> 00-00-00-00-00-03.2
INFO:openflow.discovery:link timeout: 00-00-00-00-00-01.2 -> 00-00-00-00-00-02.2
INFO:openflow.discovery:link timeout: 00-00-00-00-00-02.2 -> 00-00-00-00-00-01.2
INFO:openflow.discovery:link timeout: 00-00-00-00-00-03.2 -> 00-00-00-00-00-02.3
INFO:openflow.discovery:link timeout: 00-00-00-00-00-02.3 -> 00-00-00-00-00-03.2
```

Expected Behavior

- Full connectivity before failure
- Temporary packet loss during reroute
- Logs from controller show detection and rerouting

Future Improvements

- Detailed Logging
- Network Visualization (e.g., Ryu GUI)
- OpenFlow packet analysis with Wireshark