

PROJECT: MINI CLOUD DATA CENTER SIMULATION

ABSTRACT

This project presents the design and implementation of a Software-Defined Networking (SDN) based cloud data center that integrates intelligent management, automated operations, and real-time monitoring. The system provides a sophisticated simulation of modern cloud infrastructure by leveraging SDN principles to build a flexible and programmable network environment. Built on the Mininet emulation platform and controlled through a custom Ryu controller, the architecture showcases how centralized control and automation can effectively overcome traditional networking challenges.

The implementation features a multi-tier topology consisting of web, application, and database servers, supported by advanced capabilities such as dynamic load balancing and automated failure recovery through graph-based path recomputation. The project focuses on SDN-driven design and virtualization concepts to demonstrate a resilient, scalable, and efficiently managed network infrastructure aligned with modern cloud data center requirements.

Contents

NETWORK INFRASTRUCTURE DESIGN.....	3
RYU CONTROLLER FEATURES AND IMPLEMENTATION.....	5
LOAD BALANCING FEATURES IN CONTROLLER.....	7
FAILURE RECOVERY MODULE IN SDN CONTROLLER	8
HOST MIGRATION MODULE IN SDN CONTROLLER	9
CENTRALIZED LOGGING SYSTEM	11
FLASK BASED DASHBOARD	11
REFERENCES	12

NETWORK INFRASTRUCTURE DESIGN

THREE-TIER ARCHITECTURE

The project implements an enhanced three-tier data center architecture that clearly separates server and client functionalities, providing a realistic model of modern cloud environments. This structured design enables efficient traffic flow, improved scalability, and fault-tolerant communication across the network. The architecture is divided into **three primary layers** core, aggregation, and access each responsible for specific network operations. The system incorporates redundancy, hierarchical routing, and differentiated service tiers to accurately emulate real cloud data center behavior.

At the **core layer**, switch **s1** functions as the **central backbone** responsible for interconnecting all major segments of the topology. The **aggregation layer** consists of switches **s2** and **s3**, which aggregate traffic from the access layer and forward it to the core. A significant design enhancement is the redundant link between s2 and s3, implemented specifically to support failure recovery and ensure uninterrupted connectivity. In the **access layer**, hosts are directly connected to the aggregation switches. The server group **web1**, **web2**, **app1**, and **db1**—is configured with diverse bandwidth and latency parameters to represent different service tiers within a cloud infrastructure. Additionally, client nodes **c1**, **c2**, **c3**, and **c4** operate as end-users generating traffic and interacting with the service layers.

TOPOLOGY CODE SNIPPET:

The virtual network is developed using **Mininet**, which provides host, switch, and link emulation within a single system.

```
#!/usr/bin/python
from mininet.topo import Topo
from mininet.net import Mininet
from mininet.node import RemoteController, OVSSwitch
from mininet.cli import CLI
from mininet.link import TCLink
from mininet.log import setLogLevel
import time

class EnhancedDC(Topo):
    def build(self):
        # Switches
        core = self.addSwitch('s1')          # Core
        agg1 = self.addSwitch('s2')          # Aggregation 1
        agg2 = self.addSwitch('s3')          # Aggregation 2

        # Servers
        web1 = self.addHost('web1', ip='10.0.0.1')
        web2 = self.addHost('web2', ip='10.0.0.2')
        app1 = self.addHost('app1', ip='10.0.0.3')
        db1 = self.addHost('db1', ip='10.0.0.4')

        # Clients
        c1 = self.addHost('c1', ip='10.0.0.11')
        c2 = self.addHost('c2', ip='10.0.0.12')
        c3 = self.addHost('c3', ip='10.0.0.13')
        c4 = self.addHost('c4', ip='10.0.0.14')

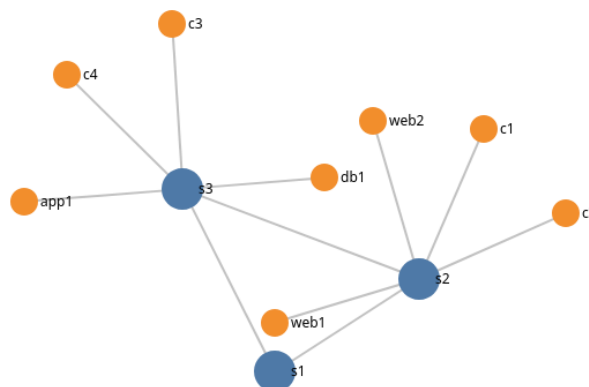
        # Core -> Aggregation
        self.addLink(core, agg1, cls=TCLink, bw=1000, delay='2ms', loss=0)
        self.addLink(core, agg2, cls=TCLink, bw=1000, delay='2ms', loss=0)
        # Redundant link (keep for failure recovery)
        self.addLink(agg1, agg2, cls=TCLink, bw=800, delay='3ms', loss=0)
```

```

# Aggregation -> Servers
self.addLink(agg1, web1, cls=TCLink, bw=500, delay='5ms', loss=0)
self.addLink(agg1, web2, cls=TCLink, bw=500, delay='5ms', loss=0)
self.addLink(agg2, app1, cls=TCLink, bw=300, delay='10ms', loss=0)
self.addLink(agg2, db1, cls=TCLink, bw=300, delay='15ms', loss=0)
# Aggregation -> Clients
self.addLink(agg1, c1, cls=TCLink, bw=100, delay='1ms', loss=0)
self.addLink(agg1, c2, cls=TCLink, bw=100, delay='1ms', loss=0)
self.addLink(agg2, c3, cls=TCLink, bw=100, delay='1ms', loss=0)
self.addLink(agg2, c4, cls=TCLink, bw=100, delay='1ms', loss=0)
def run_topo():
    topo = EnhancedDC()
    net = Mininet(
        topo=topo,
        controller=lambda name: RemoteController(name, ip='127.0.0.1', port=6653),
        link=TCLink,
        switch=OVSSwitch,
        autoSetMacs=True
    )
    net.start()
    time.sleep(10) # wait for all switches to register
    # Enable OpenFlow13 + STP
    for s in net.switches:
        s.cmd(f'ovs-vsctl set bridge {s.name} protocols=OpenFlow13')
        s.cmd(f'ovs-vsctl set bridge {s.name} stp_enable=true')
    print("\n Enhanced Cloud Data Center topology is running with STP enabled!")
    CLI(net)
    net.stop()
if __name__ == '__main__':
    setLogLevel('info')
    run_topo()

```

TOPOLOGY GRAPH:



RYU CONTROLLER FEATURES AND IMPLEMENTATION

The project leverages the **Ryu SDN framework** with full **OpenFlow 1.3 support**, enabling standardized and seamless communication with all **Open vSwitch (OVS)–based data-plane devices**. The OVS switches serve as the network’s forwarding plane, handling packet transmission according to the flow rules installed by the Ryu controller. They also support advanced features such as **STP for loop prevention, LLDP for topology discovery, and QoS/traffic shaping**. This design follows a centralized control plane approach, replacing distributed switch intelligence with a unified, global, and programmable network view. The controller maintains complete oversight of the data center topology, traffic flows, and network events, while OVS handles all packet forwarding efficiently.

Key features of the Ryu controller in this project include **real-time topology discovery**, achieved using LLDP packets and Ryu’s built-in topology APIs, which automatically detect switches, hosts, inter-switch links, port states, and link attributes. The controller implements **global MAC learning**, recording MAC-to-(switch, port) mappings across the entire network rather than on a per-switch basis, improving routing accuracy and enabling controller-driven packet forwarding. A **reactive flow installation mechanism** allows the controller to handle Packet-In events dynamically and install corresponding flow entries on all switches along the path.

The controller also includes an **efficient packet-in processing pipeline** that inspects incoming packets, looks up source and destination addresses in global tables, learns unknown hosts, computes routing decisions, and propagates flow rules to the relevant OVS switches. All forwarding decisions are managed globally by the controller, with OVS switches acting as simple forwarding devices, ensuring a **controller-managed forwarding state**. Integrated **Spanning Tree Protocol (STP) compatibility** in OVS supports loop prevention and redundant path management without manual configuration.

The architecture follows an **event-driven design**, responding to critical Ryu events such as EventOFPPacketIn, EventSwitchEnter / EventSwitchLeave, EventLinkAdd / EventLinkDelete, and EventPortModify. The controller maintains **full logging and monitoring** of its operations, including flow installations, path computations, packet-in events, and topology changes. Finally, the system is designed with an **extensible modular structure**, allowing additional functionalities such as load balancing, auto-scaling, and failure recovery to be integrated without modifying the core controller logic.

Collectively, these features transform the Ryu controller and the OVS data-plane into an **intelligent, self-healing, and highly automated network**, providing centralized control, high reliability, and dynamic adaptability for the mini cloud data center.

```
vboxuser@CloudDataCenter: ~/mini-cloud
File Edit View Search Terminal Help
c2 -> app1 c1 c3 c4 db1 web1 web2
c3 -> app1 c1 c2 c4 db1 web1 web2
c4 -> app1 c1 c2 c3 db1 web1 web2
db1 -> app1 c1 c2 c3 c4 web1 web2
web1 -> app1 c1 c2 c3 c4 db1 web2
web2 -> app1 c1 c2 c3 c4 db1 web1
*** Results: 0% dropped (56/56 received)
mininet> sh ovs-ofctl dump-flows s2 -O OpenFlow13
cookie=0x0, duration=131.210s, table=0, n_packets=4725, n_bytes=283500, priority=65535,dl_dst=01:00:c2:00:00:0e,dl_type=0x88cc actions=CONTROLLER:65535
cookie=0x0, duration=28880.488s, table=0, n_packets=49, n_bytes=3626, priority=1,dl_src=00:00:00:00:03,dl_dst=00:00:00:00:02 actions=output:"s2-eth5"
cookie=0x0, duration=28880.483s, table=0, n_packets=42, n_bytes=3164, priority=1,dl_src=00:00:00:00:02,dl_dst=00:00:00:00:03 actions=output:"s2-eth6"
cookie=0x0, duration=28880.455s, table=0, n_packets=51, n_bytes=3878, priority=1,dl_src=00:00:00:00:04,dl_dst=00:00:00:00:02 actions=output:"s2-eth5"
cookie=0x0, duration=28880.445s, table=0, n_packets=44, n_bytes=3304, priority=1,dl_src=00:00:00:00:02,dl_dst=00:00:00:00:04 actions=output:"s2-eth2"
cookie=0x0, duration=28880.398s, table=0, n_packets=51, n_bytes=3878, priority=1,dl_src=00:00:00:00:05,dl_dst=00:00:00:00:02 actions=output:"s2-eth5"
cookie=0x0, duration=28880.391s, table=0, n_packets=44, n_bytes=3304, priority=1,dl_src=00:00:00:00:02,dl_dst=00:00:00:00:05 actions=output:"s2-eth2"
cookie=0x0, duration=28880.307s, table=0, n_packets=51, n_bytes=3878, priority=1,dl_src=00:00:00:00:06,dl_dst=00:00:00:00:02 actions=output:"s2-eth5"
cookie=0x0, duration=28880.302s, table=0, n_packets=44, n_bytes=3304, priority=1,dl_src=00:00:00:00:02,dl_dst=00:00:00:00:06 actions=output:"s2-eth2"
cookie=0x0, duration=28880.231s, table=0, n_packets=49, n_bytes=3682, priority=1,dl_src=00:00:00:00:07,dl_dst=00:00:00:00:02 actions=output:"s2-eth5"
cookie=0x0, duration=28880.226s, table=0, n_packets=44, n_bytes=3304, priority=1,dl_src=00:00:00:00:02,dl_dst=00:00:00:00:07 actions=output:"s2-eth3"
cookie=0x0, duration=28880.190s, table=0, n_packets=50, n_bytes=3724, priority=1,dl_src=00:00:00:00:08,dl_dst=00:00:00:00:02 actions=output:"s2-eth5"
cookie=0x0, duration=28880.182s, table=0, n_packets=45, n_bytes=3346, priority=1,dl_src=00:00:00:00:02,dl_dst=00:00:00:00:08 actions=output:"s2-eth4"
cookie=0x0, duration=28880.113s, table=0, n_packets=72, n_bytes=5376, priority=1,dl_src=00:00:00:00:01,dl_dst=00:00:00:00:03 actions=output:"s2-eth6"
cookie=0x0, duration=28880.108s, table=0, n_packets=68, n_bytes=4984, priority=1,dl_src=00:00:00:00:03,dl_dst=00:00:00:00:01 actions=output:"s2-eth2"
cookie=0x0, duration=28880.025s, table=0, n_packets=72, n_bytes=5432, priority=1,dl_src=00:00:00:00:04,dl_dst=00:00:00:00:03 actions=output:"s2-eth6"
cookie=0x0, duration=28879.017s, table=0, n_packets=68, n_bytes=5040, priority=1,dl_src=00:00:00:00:03,dl_dst=00:00:00:00:04 actions=output:"s2-eth3"
cookie=0x0, duration=28879.968s, table=0, n_packets=72, n_bytes=5432, priority=1,dl_src=00:00:00:00:05,dl_dst=00:00:00:00:03 actions=output:"s2-eth6"
cookie=0x0, duration=28879.964s, table=0, n_packets=68, n_bytes=5040, priority=1,dl_src=00:00:00:00:03,dl_dst=00:00:00:00:05 actions=output:"s2-eth2"
cookie=0x0, duration=28879.885s, table=0, n_packets=73, n_bytes=5474, priority=1,dl_src=00:00:00:00:06,dl_dst=00:00:00:00:03 actions=output:"s2-eth6"
cookie=0x0, duration=28879.880s, table=0, n_packets=69, n_bytes=5082, priority=1,dl_src=00:00:00:00:03,dl_dst=00:00:00:00:06 actions=output:"s2-eth2"
cookie=0x0, duration=28879.813s, table=0, n_packets=70, n_bytes=5236, priority=1,dl_src=00:00:00:00:07,dl_dst=00:00:00:00:03 actions=output:"s2-eth6"
cookie=0x0, duration=28879.806s, table=0, n_packets=68, n_bytes=5040, priority=1,dl_src=00:00:00:00:03,dl_dst=00:00:00:00:07 actions=output:"s2-eth3"
cookie=0x0, duration=28879.766s, table=0, n_packets=73, n_bytes=5366, priority=1,dl_src=00:00:00:00:08,dl_dst=00:00:00:00:03 actions=output:"s2-eth6"
cookie=0x0, duration=28879.761s, table=0, n_packets=71, n_bytes=5166, priority=1,dl_src=00:00:00:00:03,dl_dst=00:00:00:00:08 actions=output:"s2-eth4"
cookie=0x0, duration=28879.498s, table=0, n_packets=72, n_bytes=5432, priority=1,dl_src=00:00:00:00:07,dl_dst=00:00:00:00:04 actions=output:"s2-eth2"
cookie=0x0, duration=28879.476s, table=0, n_packets=69, n_bytes=5138, priority=1,dl_src=00:00:00:00:04,dl_dst=00:00:00:00:07 actions=output:"s2-eth3"
cookie=0x0, duration=28879.425s, table=0, n_packets=73, n_bytes=5474, priority=1,dl_src=00:00:00:00:08,dl_dst=00:00:00:00:04 actions=output:"s2-eth2"
cookie=0x0, duration=28879.399s, table=0, n_packets=69, n_bytes=5082, priority=1,dl_src=00:00:00:00:04,dl_dst=00:00:00:00:08 actions=output:"s2-eth4"
cookie=0x0, duration=28879.142s, table=0, n_packets=71, n_bytes=5390, priority=1,dl_src=00:00:00:00:07,dl_dst=00:00:00:00:05 actions=output:"s2-eth2"
```

The table shows LLDP flows for topology discovery (sent to the controller) and reactive flows for host-to-host traffic. Each flow specifies source/destination MAC addresses and output ports, enabling efficient forwarding while the packet/byte counters indicate active traffic.

```
Applications Mini Cloud DC - Dashboard - Mozilla Firefox
vboxuser@CloudDataCenter: ~/mini-cloud
INFO: Pushed: [{'ts': 1763586990, 'requests': 0.0, 'bandwidth': 0.0}]
Pushed: [{'ts': 1763586992, 'requests': 0.0, 'bandwidth': 0.0}]
INFO: Pushed: [{'ts': 1763586992, 'requests': 0.0, 'bandwidth': 0.0}]
Switch 1 features handled
INFO: Switch 1 features handled
Switch 1 connected
INFO: Switch 1 connected
Switch connected: 1
INFO: Switch connected: 1
Switch connected: 1
INFO: Switch connected: 1
Switch 3 features handled
INFO: Switch 3 features handled
Switch connected: 3
INFO: Switch connected: 3
Switch connected: 3
INFO: Switch connected: 3
Switch connected: 3
INFO: Switch connected: 3
Topology updated (switch-enter): 1 switches, 0 links
INFO: Topology updated (switch-enter): 1 switches, 0 links
Switch 3 connected
INFO: Switch 3 connected
Topology updated (switch-enter): 2 switches, 0 links
INFO: Topology updated (switch-enter): 2 switches, 0 links
Topology updated (link-add): 2 switches, 1 links
INFO: Topology updated (link-add): 2 switches, 1 links
Topology updated (link-add): 2 switches, 1 links
INFO: Topology updated (link-add): 2 switches, 1 links
```

Ryu controller console showing switches connecting and flow installation events.

LOAD BALANCING FEATURES IN CONTROLLER

The SDN controller implements a **multi-tier load balancing engine**, fully integrated within the Ryu framework, providing independent load balancing for each service tier of the data center. The architecture ensures efficient traffic distribution and high availability across the network:

- **Web Tier:** web1, web2
- **Application Tier:** app1 and future application instances
- **Database Tier:** db1 and future database instances

To facilitate centralized and secure traffic management, the controller uses **Virtual IP (VIP) and Virtual MAC abstractions**. Each tier is assigned a VIP: Web VIP (10.0.0.100), Application VIP (10.0.0.110), and Database VIP (10.0.0.120). Clients access services exclusively through these VIPs, while the actual server IPs remain hidden, ensuring traffic flows are managed centrally by the controller.

The controller also provides **ARP proxy handling**, intercepting all ARP requests for VIPs and responding with a single virtual MAC address. This mechanism ensures that all service traffic passes through the controller, enabling **optimal load distribution, monitoring, and centralized routing control**.

Multiple **load balancing algorithms** are supported:

- **Round-Robin (RR):** Cyclically distributes new flows equally among available backend servers.
- **Weighted Round-Robin (WRR):** Distributes flows proportionally according to configurable server weights (e.g., web1: weight 2, web2: weight 1).
- **Dynamic / Least-Load / Least-Connections:** Selects the server with the lowest active connection count or real-time CPU/load metric, providing adaptive traffic distribution based on current network conditions.

Finally, the system maintains **comprehensive logging and analytics**, recording every load balancing decision, including server selection rationale, current weights, and active connection counts. This supports performance evaluation, debugging, and operational monitoring, ensuring the SDN-enabled mini cloud data center remains resilient, efficient, and scalable.

A screenshot of a terminal window titled 'Applications - Mini Cloud DC - Dashboard - Terminal'. The terminal shows a series of network tests. First, a 'pingall' command is executed, showing connectivity to various hosts (c1, c2, c3, c4, db1, web1, web2) through application (app1) and database (db1) tiers. The results show 0% dropped packets and 56/56 received. Then, a 'ping 10.0.0.100' command is executed, showing 64 bytes of data received from 10.0.0.1 with various ICMP sequence numbers and times. Finally, a 'ping statistics' command is shown, indicating 6 packets transmitted, 6 received, 0% packet loss, and a time of 5005ms. The round-trip time (rtt) statistics are also displayed: min/avg/max/mdev = 19.437/22.321/36.087/6.157 ms.

```
mininet> pingall
*** Ping: testing ping reachability
app1 -> c1 c2 c3 c4 db1 web1 web2
c1 -> app1 c2 c3 c4 db1 web1 web2
c2 -> app1 c1 c3 c4 db1 web1 web2
c3 -> app1 c1 c2 c4 db1 web1 web2
c4 -> app1 c1 c2 c3 db1 web1 web2
db1 -> app1 c1 c2 c3 c4 web1 web2
web1 -> app1 c1 c2 c3 c4 db1 web1
web2 -> app1 c1 c2 c3 c4 db1 web1
*** Results: 0% dropped (56/56 received)
mininet> c3 ping 10.0.0.100
PING 10.0.0.100 (10.0.0.100) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=36.1 ms (DIFFERENT ADDRESS!)
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=19.7 ms (DIFFERENT ADDRESS!)
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=19.4 ms (DIFFERENT ADDRESS!)
64 bytes from 10.0.0.1: icmp_seq=4 ttl=64 time=19.4 ms (DIFFERENT ADDRESS!)
64 bytes from 10.0.0.1: icmp_seq=5 ttl=64 time=19.7 ms (DIFFERENT ADDRESS!)
64 bytes from 10.0.0.1: icmp_seq=6 ttl=64 time=19.6 ms (DIFFERENT ADDRESS!)
^C
--- 10.0.0.100 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5005ms
rtt min/avg/max/mdev = 19.437/22.321/36.087/6.157 ms
```

```

Replied ARP for VIP 10.0.0.100 -> MAC 00:aa:bb:00:00:01 (to 10.0.0.11)
INFO: Replied ARP for VIP 10.0.0.100 -> MAC 00:aa:bb:00:00:01 (to 10.0.0.11)
🔍 Learned host: 10.0.0.11 -> S3:2
INFO: 🔍 Learned host: 10.0.0.11 -> S3:2
🔍 Learned host (ARP): 10.0.0.11 @ 3:2
INFO: 🔍 Learned host (ARP): 10.0.0.11 @ 3:2
Host seen via ARP: 10.0.0.11
INFO: Host seen via ARP: 10.0.0.11
LB assign: 10.0.0.11 -> 10.0.0.2 (policy=round_robin)
INFO: LB assign: 10.0.0.11 -> 10.0.0.2 (policy=round_robin)
Host seen via IPv4: 10.0.0.11
INFO: Host seen via IPv4: 10.0.0.11
Replied ARP for VIP 10.0.0.100 -> MAC 00:aa:bb:00:00:01 (to 10.0.0.11)
INFO: Replied ARP for VIP 10.0.0.100 -> MAC 00:aa:bb:00:00:01 (to 10.0.0.11)
🔍 Learned host: 10.0.0.11 -> S1:2
INFO: 🔍 Learned host: 10.0.0.11 -> S1:2
🔍 Learned host (ARP): 10.0.0.11 @ 1:2
INFO: 🔍 Learned host (ARP): 10.0.0.11 @ 1:2
Host seen via ARP: 10.0.0.11
INFO: Host seen via ARP: 10.0.0.11
🔍 Learned host: 10.0.0.100 -> S2:2
INFO: 🔍 Learned host: 10.0.0.100 -> S2:2
🔍 Learned host (ARP): 10.0.0.100 @ 2:2
INFO: 🔍 Learned host (ARP): 10.0.0.100 @ 2:2
LB assign: 10.0.0.11 -> 10.0.0.1 (policy=round_robin)
INFO: LB assign: 10.0.0.11 -> 10.0.0.1 (policy=round_robin)
..

```

Load Balancing in action, The first request to VIP 10.0.0.100 is routed to backend server 10.0.0.2 (round-robin policy), and the next request is automatically routed to 10.0.0.1, demonstrating sequential distribution of client requests among servers.

FAILURE RECOVERY MODULE IN SDN CONTROLLER

The **Failure Recovery module** in the SDN controller provides **automated, real-time detection and restoration** of network connectivity. Leveraging Ryu's built-in topology and port status events, such as **EventLinkDelete**, **EventSwitchLeave**, **EventSwitchEnter**, and **EventPortModify**, the controller continuously monitors the operational state of **switches, ports, and links**. Any network anomaly whether it is a failed link, downed port, or disconnected switch is detected immediately, allowing the controller to initiate corrective actions without manual intervention. This proactive monitoring ensures that the network can respond to failures dynamically and maintain high availability.

Upon detecting a failure, the controller maintains a **real-time network graph** and adjacency lists using the **NetworkX library**, which are continuously updated to reflect current network topology. It then triggers **Dijkstra's shortest-path algorithm** to compute **optimal alternative paths** for all affected source-destination pairs. The controller identifies all active flows impacted by the failure and proactively **installs new OpenFlow rules** along the recalculated paths, ensuring seamless traffic rerouting. This mechanism allows ongoing sessions to continue with minimal packet loss, reduced latency, and uninterrupted connectivity.

The module is capable of handling single or multiple simultaneous failures, maintaining maximum network connectivity and graceful degradation where necessary. Active hosts and applications experience **minimal disruption**, as alternate paths are leveraged transparently. Additionally, the controller maintains **comprehensive centralized logging**, capturing every failure event, affected flows, original versus recovered paths, recovery durations, and post-recovery topology states.

```

<OVSSwitch s3: lo:127.0.0.1,s3-eth1:None,s3-eth2:None,s3-eth3:None,s3-eth4:None,s3-eth5:None,s3-eth6:None pld=43671>
<RemoteController c8: 127.0.0.1:6653 pld=43648>
mininet> link s1 s2 down
mininet> pingall
*** Ping: testing ping reachability
app1 -> c1 c2 c3 c4 db1 web1 web2
c1 -> app1 c2 c3 c4 db1 web1 web2
c2 -> app1 c1 c3 c4 db1 web1 web2
c3 -> app1 c1 c2 c4 db1 web1 web2
c4 -> app1 c1 c2 c3 db1 web1 web2
db1 -> app1 c1 c2 c3 c4 web1 web2
web1 -> app1 c1 c2 c3 c4 db1 web2
web2 -> app1 c1 c2 c3 c4 db1 web1
*** Results: 0% dropped (56/56 received)
mininet> link s1 s2 up
mininet> pingall
*** Ping: testing ping reachability
app1 -> c1 c2 c3 c4 db1 web1 web2
c1 -> app1 c2 c3 c4 db1 web1 web2
c2 -> app1 c1 c3 c4 db1 web1 web2
c3 -> app1 c1 c2 c4 db1 web1 web2
c4 -> app1 c1 c2 c3 db1 web1 web2
db1 -> app1 c1 c2 c3 c4 web1 web2
web1 -> app1 c1 c2 c3 c4 db1 web2
web2 -> app1 c1 c2 c3 c4 db1 web1
*** Results: 0% dropped (56/56 received)
mininet>

```

```

{'id': '10.0.0.3', 'ip': '10.0.0.3', 'status': 'UP'}, {'id': '10.0.0.4', 'ip': '10.0.0.4', 'status': 'UP'}}]]
INFO: Pushed to Flask: {'hosts': [{'id': '10.0.0.12', 'ip': '10.0.0.12', 'status': 'DOWN'}, {'id': '10.0.0.13', 'ip': '10.0.0.13', 'status': 'DOWN'}, {'id': '10.0.0.14', 'ip': '10.0.0.14', 'status': 'DOWN'}, {'id': '10.0.0.11', 'ip': '10.0.0.11', 'status': 'DOWN'}], 'servers': [{'id': '10.0.0.1', 'ip': '10.0.0.1', 'status': 'UP'}, {'id': '10.0.0.2', 'ip': '10.0.0.2', 'status': 'UP'}, {'id': '10.0.0.3', 'ip': '10.0.0.3', 'status': 'UP'}, {'id': '10.0.0.4', 'ip': '10.0.0.4', 'status': 'UP'}]]
Link down: 2 <-> 1
WARNING: Link down: 2 <-> 1
Topology updated (link-delete): 3 switches, 0 links
INFO: Topology updated (link-delete): 3 switches, 0 links
Recomputing all alternate paths...
WARNING: Recomputing all alternate paths...
Link down: 1 <-> 2
WARNING: Link down: 1 <-> 2
Topology updated (link-delete): 3 switches, 0 links
INFO: Topology updated (link-delete): 3 switches, 0 links
Recomputing all alternate paths...
WARNING: Recomputing all alternate paths...
Topology updated (link-delete): 3 switches, 0 links
INFO: Topology updated (link-delete): 3 switches, 0 links
Topology updated (link-delete): 3 switches, 0 links
INFO: Topology updated (link-delete): 3 switches, 0 links
Pushed to Flask: {'hosts': [{'id': '10.0.0.12', 'ip': '10.0.0.12', 'status': 'DOWN'}, {'id': '10.0.0.13', 'ip': '10.0.0.13', 'status': 'DOWN'}, {'id': '10.0.0.14', 'ip': '10.0.0.14', 'status': 'DOWN'}, {'id': '10.0.0.11', 'ip': '10.0.0.11', 'status': 'DOWN'}], 'servers': [{'id': '10.0.0.1', 'ip': '10.0.0.1', 'status': 'UP'}, {'id': '10.0.0.2', 'ip': '10.0.0.2', 'status': 'UP'}, {'id': '10.0.0.3', 'ip': '10.0.0.3', 'status': 'UP'}, {'id': '10.0.0.4', 'ip': '10.0.0.4', 'status': 'UP'}]]

```

Here even after manually bringing down the link between switches s1 and s2, the SDN controller successfully reroutes traffic along alternate paths, allowing all hosts to continue communicating as verified by pingall.

HOST MIGRATION MODULE IN SDN CONTROLLER

The **Host Migration Service** in the SDN controller provides **live relocation of hosts** or workloads between switches without interrupting ongoing services. This capability is essential for **maintenance, load optimization, and dynamic resource management** in cloud environments. Migration operations are triggered through a **REST API**, where administrators specify the **target host IP** and the **destination switch port**, allowing seamless control over host placement in the network.

The controller maintains an **up-to-date host mapping** by learning host attachments through **topology events** such as **EventHostAdd** and **ARP packet inspection**. Each host is tracked as a mapping from its IP to (dpid, port), enabling the controller to determine the host's current location and manage forwarding correctly. Upon a migration request, the system first **validates the target location** to ensure the destination switch and port are active and capable of hosting the host. It then performs **flow cleanup** on the previous switch, deleting all flows that forward traffic to the host's old location to prevent stale routing.

Afterwards, the controller updates the internal **host database** to reflect the new (dpid, port) assignment and logs the migration details for auditing and monitoring. Traffic forwarding continues seamlessly through **reactive flow installation** and the controller's **learning and failure recovery mechanisms**, ensuring **minimal disruption to active connections**. This design provides a **flexible, efficient, and self-healing network**.

infrastructure, enabling dynamic host placement while maintaining uninterrupted service availability and high operational reliability.

```
mininet> sh curl -X POST http://127.0.0.1:8080/migrate -H "Content-Type: application/json" -d '{"ip":"10.0.0.1","new_dpId":2,"new_port":2}'
{"result":"Host migrated","ip":"10.0.0.1","new_dpId":2,"new_port":2}mininet>
```

```
{'id': '10.0.0.3', 'ip': '10.0.0.3', 'status': 'UP'}, {'id': '10.0.0.4', 'ip': '10.0.0.4', 'status': 'UP'}}]
INFO: Pushed to Flask: {'hosts': [{'id': '10.0.0.12', 'ip': '10.0.0.12', 'status': 'DOWN'}, {'id': '10.0.0.13', 'ip': '10.0.0.13', 'status': 'DOWN'}, {'id': '10.0.0.14', 'ip': '10.0.0.14', 'status': 'DOWN'}, {'id': '10.0.0.11', 'ip': '10.0.0.11', 'status': 'DOWN'}], 'servers': [{'id': '10.0.0.1', 'ip': '10.0.0.1', 'status': 'UP'}, {'id': '10.0.0.2', 'ip': '10.0.0.2', 'status': 'UP'}, {'id': '10.0.0.3', 'ip': '10.0.0.3', 'status': 'UP'}, {'id': '10.0.0.4', 'ip': '10.0.0.4', 'status': 'UP'}]}
Pushed to Flask: {'hosts': [{'id': '10.0.0.12', 'ip': '10.0.0.12', 'status': 'DOWN'}, {'id': '10.0.0.13', 'ip': '10.0.0.13', 'status': 'DOWN'}, {'id': '10.0.0.14', 'ip': '10.0.0.14', 'status': 'DOWN'}, {'id': '10.0.0.11', 'ip': '10.0.0.11', 'status': 'DOWN'}], 'servers': [{'id': '10.0.0.1', 'ip': '10.0.0.1', 'status': 'UP'}, {'id': '10.0.0.2', 'ip': '10.0.0.2', 'status': 'UP'}, {'id': '10.0.0.3', 'ip': '10.0.0.3', 'status': 'UP'}, {'id': '10.0.0.4', 'ip': '10.0.0.4', 'status': 'UP'}]}
INFO: Pushed to Flask: {'hosts': [{'id': '10.0.0.12', 'ip': '10.0.0.12', 'status': 'DOWN'}, {'id': '10.0.0.13', 'ip': '10.0.0.13', 'status': 'DOWN'}, {'id': '10.0.0.14', 'ip': '10.0.0.14', 'status': 'DOWN'}, {'id': '10.0.0.11', 'ip': '10.0.0.11', 'status': 'DOWN'}], 'servers': [{'id': '10.0.0.1', 'ip': '10.0.0.1', 'status': 'UP'}, {'id': '10.0.0.2', 'ip': '10.0.0.2', 'status': 'UP'}, {'id': '10.0.0.3', 'ip': '10.0.0.3', 'status': 'UP'}, {'id': '10.0.0.4', 'ip': '10.0.0.4', 'status': 'UP'}]}
(43629) accepted ('127.0.0.1', 51528)
INFO: (43629) accepted ('127.0.0.1', 51528)
Migration request: 10.0.0.1 1:2 -> 2:2
INFO: Migration request: 10.0.0.1 1:2 -> 2:2
Flows for 10.0.0.1 deleted from old switch 1
INFO: Flows for 10.0.0.1 deleted from old switch 1
Host 10.0.0.1 migrated to 2:2
INFO: Host 10.0.0.1 migrated to 2:2
127.0.0.1 - - [19/Nov/2025 22:17:57] "POST /migrate HTTP/1.1" 200 176 0.003143
INFO: 127.0.0.1 - - [19/Nov/2025 22:17:57] "POST /migrate HTTP/1.1" 200 176 0.003143
```

The host with IP 10.0.0.11 was initially connected to switch s1. After executing the migration via the SDN controller, the host was seamlessly relocated to switch s2 without disrupting ongoing traffic. Flow rules were automatically updated on both the source and destination switches, ensuring uninterrupted connectivity.

```
Applications  Controller Logs - Mozilla  Terminal
Controller Logs - Mozilla Firefox
vboxuser@CloudDataCenter:~/mini-cloud
mininet> sh ovs-ofctl -O OpenFlow13 dump-flows s1
cookie=0x0, duration=3541.387s, table=0, n_packets=3701, n_bytes=222060, priority=65535,dl_dst=01:80:c2:00:00:0e,dl_type=0x88cc actions=CONTROLLER:65535
cookie=0x0, duration=3541.398s, table=0, n_packets=9113, n_bytes=823025, priority=0 actions=CONTROLLER:65535
mininet> sh ovs-ofctl -O OpenFlow13 dump-flows s2
cookie=0x0, duration=3547.592s, table=0, n_packets=5008, n_bytes=300480, priority=65535,dl_dst=01:80:c2:00:00:0e,dl_type=0x88cc actions=CONTROLLER:65535
cookie=0x0, duration=3516.938s, table=0, n_packets=16, n_bytes=1288, priority=1,in_port="s2-eth6",dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:02 actions=output:"s2-eth5"
cookie=0x0, duration=3516.933s, table=0, n_packets=14, n_bytes=1092, priority=1,in_port="s2-eth5",dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:03 actions=output:"s2-eth6"
cookie=0x0, duration=3516.896s, table=0, n_packets=18, n_bytes=1484, priority=1,in_port="s2-eth2",dl_src=00:00:00:00:00:04,dl_dst=00:00:00:00:00:02 actions=output:"s2-eth5"
cookie=0x0, duration=3516.887s, table=0, n_packets=14, n_bytes=1092, priority=1,in_port="s2-eth5",dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:04 actions=output:"s2-eth2"
cookie=0x0, duration=3516.835s, table=0, n_packets=18, n_bytes=1484, priority=1,in_port="s2-eth2",dl_src=00:00:00:00:00:05,dl_dst=00:00:00:00:00:02 actions=output:"s2-eth5"
cookie=0x0, duration=3516.831s, table=0, n_packets=14, n_bytes=1092, priority=1,in_port="s2-eth5",dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:05 actions=output:"s2-eth2"
cookie=0x0, duration=3516.769s, table=0, n_packets=18, n_bytes=1484, priority=1,in_port="s2-eth2",dl_src=00:00:00:00:00:06,dl_dst=00:00:00:00:00:02 actions=output:"s2-eth5"
cookie=0x0, duration=3516.753s, table=0, n_packets=14, n_bytes=1092, priority=1,in_port="s2-eth5",dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:06 actions=output:"s2-eth3"
cookie=0x0, duration=3516.694s, table=0, n_packets=20, n_bytes=1568, priority=1,in_port="s2-eth3",dl_src=00:00:00:00:00:07,dl_dst=00:00:00:00:00:02 actions=output:"s2-eth5"
cookie=0x0, duration=3516.677s, table=0, n_packets=18, n_bytes=1372, priority=1,in_port="s2-eth5",dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:07 actions=output:"s2-eth3"
cookie=0x0, duration=3516.636s, table=0, n_packets=16, n_bytes=1288, priority=1,in_port="s2-eth4",dl_src=00:00:00:00:00:08,dl_dst=00:00:00:00:00:02 actions=output:"s2-eth5"
cookie=0x0, duration=3516.632s, table=0, n_packets=14, n_bytes=1092, priority=1,in_port="s2-eth5",dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:08 actions=output:"s2-eth4"
cookie=0x0, duration=3516.573s, table=0, n_packets=17, n_bytes=1386, priority=1,in_port="s2-eth2",dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:03 actions=output:"s2-eth6"
cookie=0x0, duration=3516.568s, table=0, n_packets=13, n_bytes=994, priority=1,in_port="s2-eth6",dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:01 actions=output:"s2-eth2"
cookie=0x0, duration=3516.498s, table=0, n_packets=18, n_bytes=1484, priority=1,in_port="s2-eth2",dl_src=00:00:00:00:00:04,dl_dst=00:00:00:00:00:03 actions=output:"s2-eth6"
cookie=0x0, duration=3516.487s, table=0, n_packets=14, n_bytes=1092, priority=1,in_port="s2-eth6",dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:04 actions=output:"s2-eth2"
cookie=0x0, duration=3516.437s, table=0, n_packets=18, n_bytes=1484, priority=1,in_port="s2-eth2",dl_src=00:00:00:00:00:05,dl_dst=00:00:00:00:00:03 actions=output:"s2-eth6"
cookie=0x0, duration=3516.433s, table=0, n_packets=14, n_bytes=1092, priority=1,in_port="s2-eth6",dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:05 actions=output:"s2-eth2"
cookie=0x0, duration=3516.348s, table=0, n_packets=18, n_bytes=1484, priority=1,in_port="s2-eth2",dl_src=00:00:00:00:00:06,dl_dst=00:00:00:00:00:03 actions=output:"s2-eth6"
cookie=0x0, duration=3516.341s, table=0, n_packets=14, n_bytes=1092, priority=1,in_port="s2-eth6",dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:06 actions=output:"s2-eth2"
cookie=0x0, duration=3516.278s, table=0, n_packets=21, n_bytes=1666, priority=1,in_port="s2-eth3",dl_src=00:00:00:00:00:07,dl_dst=00:00:00:00:00:03 actions=output:"s2-eth6"
cookie=0x0, duration=3516.269s, table=0, n_packets=19, n_bytes=1478, priority=1,in_port="s2-eth6",dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:07 actions=output:"s2-eth3"
cookie=0x0, duration=3516.227s, table=0, n_packets=16, n_bytes=1288, priority=1,in_port="s2-eth6",dl_src=00:00:00:00:00:06,dl_dst=00:00:00:00:00:03 actions=output:"s2-eth6"
cookie=0x0, duration=3516.226s, table=0, n_packets=14, n_bytes=1092, priority=1,in_port="s2-eth6",dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:08 actions=output:"s2-eth4"
cookie=0x0, duration=3515.935s, table=0, n_packets=39, n_bytes=3486, priority=1,in_port="s2-eth3",dl_src=00:00:00:00:00:07,dl_dst=00:00:00:00:00:04 actions=output:"s2-eth2"
cookie=0x0, duration=3515.913s, table=0, n_packets=35, n_bytes=3094, priority=1,in_port="s2-eth2",dl_src=00:00:00:00:00:04,dl_dst=00:00:00:00:00:07 actions=output:"s2-eth3"
cookie=0x0, duration=3515.856s, table=0, n_packets=18, n_bytes=1484, priority=1,in_port="s2-eth4",dl_src=00:00:00:00:00:08,dl_dst=00:00:00:00:00:04 actions=output:"s2-eth2"
cookie=0x0, duration=3515.833s, table=0, n_packets=15, n_bytes=1190, priority=1,in_port="s2-eth2",dl_src=00:00:00:00:00:04,dl_dst=00:00:00:00:00:08 actions=output:"s2-eth4"
cookie=0x0, duration=3515.587s, table=0, n_packets=18, n_bytes=1484, priority=1,in_port="s2-eth3",dl_src=00:00:00:00:00:07,dl_dst=00:00:00:00:00:05 actions=output:"s2-eth2"
cookie=0x0, duration=3515.554s, table=0, n_packets=14, n_bytes=1092, priority=1,in_port="s2-eth2",dl_src=00:00:00:00:00:05,dl_dst=00:00:00:00:00:07 actions=output:"s2-eth3"
cookie=0x0, duration=3515.501s, table=0, n_packets=24, n_bytes=2016, priority=1,in_port="s2-eth4",dl_src=00:00:00:00:00:08,dl_dst=00:00:00:00:00:05 actions=output:"s2-eth2"
cookie=0x0, duration=3515.484s, table=0, n_packets=20, n_bytes=1624, priority=1,in_port="s2-eth2",dl_src=00:00:00:00:00:05,dl_dst=00:00:00:00:00:08 actions=output:"s2-eth4"
cookie=0x0, duration=3515.136s, table=0, n_packets=18, n_bytes=1484, priority=1,in_port="s2-eth3",dl_src=00:00:00:00:00:07,dl_dst=00:00:00:00:00:06 actions=output:"s2-eth2"
```

Host 10.0.0.11 successfully migrated from switch s1 to s2. Verification via `ovs-ofctl -O OpenFlow13 dump-flows` shows flows updated on s2 and removed from s1.

CENTRALIZED LOGGING SYSTEM

The **logging system** provides comprehensive **log management** for the SDN controller, capturing detailed operational data from all modules, including load balancing, host migration and failure recovery components. Upon initialization, the system automatically creates a dedicated `ryu_logs/` directory within the project structure, ensuring all logs are stored in a centralized and organized location. Each log file is **timestamped**, facilitating easy retrieval and historical analysis of controller events.

A **unified log format** is maintained across all modules, providing consistent structure for events, errors, and performance metrics. Logs are simultaneously written to **console output** for real-time monitoring and **log files** for long-term storage, preserving both immediate visibility and durable records. The system captures specific events, network anomalies, flow updates, and detailed error contexts, giving operators the insight required to diagnose issues effectively.

This robust logging framework not only enhances visibility into the controller's operation but also supports **auditing**. By systematically recording controller activity across all modules, the system ensures that administrators can **understand system behavior, track historical events, and optimize the overall performance** of the cloud environment.

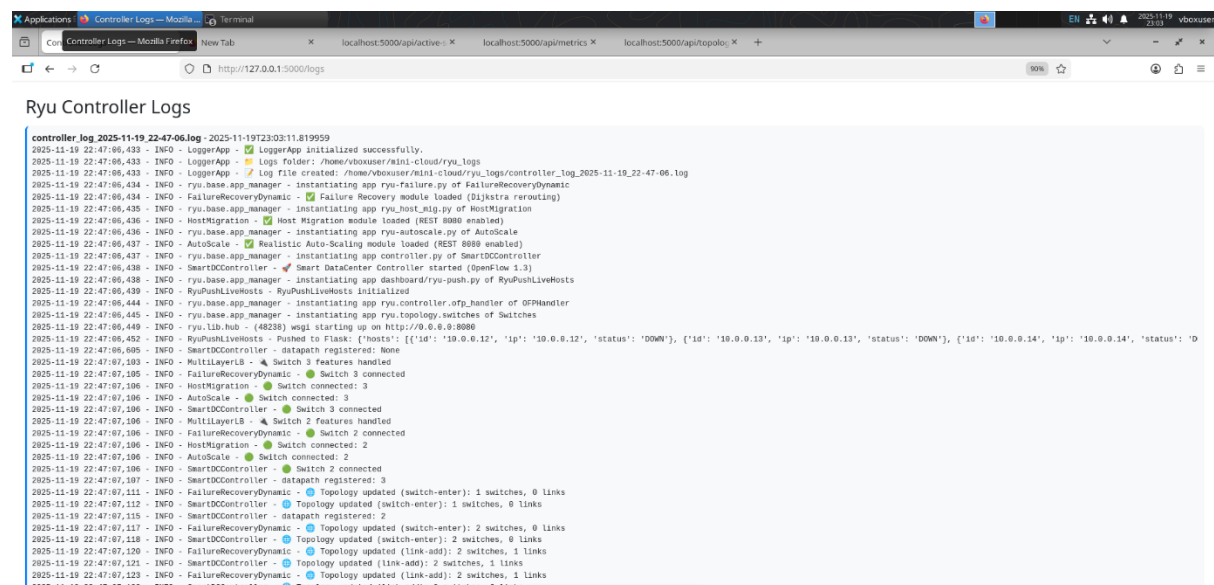
```
(ryu39-env) vboxuser@CloudDataCenter:~/mint-cloud$ ls
controller.py  __pycache__  ryu-autoscale.py  ryu-host_mig.py  ryu_logs
dashboard      ryu           ryu-controller    ryu-lb1.py
dashboard-int.py  ryu312-env   ryu-env           ryu-lb.py
enhanced_topo.py  ryu39-env    ryu-failure.py    ryu-log.py
(ryu39-env) vboxuser@CloudDataCenter:~/mint-cloud$ cd ryu_logs
(ryu39-env) vboxuser@CloudDataCenter:~/mint-cloud/ryu_logs$ ls
controller_log_2025-11-13_06-24-41.log  controller_log_2025-11-16_07-55-59.log
controller_log_2025-11-13_08-00-25.log  controller_log_2025-11-16_10-47-25.log
controller_log_2025-11-13_08-06-47.log  controller_log_2025-11-16_10-53-23.log
controller_log_2025-11-13_08-19-16.log  controller_log_2025-11-17_18-30-25.log
controller_log_2025-11-13_08-27-55.log  controller_log_2025-11-18_04-48-42.log
controller_log_2025-11-15-15-28-34.log  controller_log_2025-11-18_04-50-50.log
controller_log_2025-11-15-16-17-19.log  controller_log_2025-11-19_12-50-17.log
controller_log_2025-11-15-16-29-30.log  controller_log_2025-11-19_12-53-56.log
controller_log_2025-11-15-16-33-58.log  controller_log_2025-11-19_12-54-19.log
controller_log_2025-11-15-16-39-04.log  controller_log_2025-11-19_15-58-29.log
controller_log_2025-11-15-16-39-22.log  controller_log_2025-11-19_16-25-46.log
controller_log_2025-11-15-18-41-55.log  controller_log_2025-11-19_20-53-19.log
controller_log_2025-11-15-18-54-41.log  controller_log_2025-11-19_20-53-56.log
controller_log_2025-11-15-19-00-19.log  controller_log_2025-11-19_21-01-24.log
controller_log_2025-11-15-19-06-57.log  controller_log_2025-11-19_21-17-16.log
controller_log_2025-11-15-19-13-07.log  controller_log_2025-11-19_21-18-50.log
controller_log_2025-11-15-19-20-44.log  controller_log_2025-11-19_21-19-04.log
controller_log_2025-11-16-06-37-00.log  controller_log_2025-11-19_21-20-56.log
controller_log_2025-11-16-06-39-14.log  controller_log_2025-11-19_21-27-11.log
controller_log_2025-11-16-07-13-52.log  controller_log_2025-11-19_21-29-15.log
controller_log_2025-11-16-07-36-42.log  controller_log_2025-11-19_22-47-06.log
controller_log_2025-11-16-07-47-04.log
(ryu39-env) vboxuser@CloudDataCenter:~/mint-cloud/ryu_logs$
```

FLASK BASED DASHBOARD

The **mini cloud dashboard** is a centralized visualization interface designed for real-time monitoring and management of the SDN-based mini cloud environment. It provides a live view of all **active hosts and servers**, including their IP addresses, along with operational status. The information is continuously updated via REST APIs and controller push mechanisms, ensuring administrators always have an accurate representation of the network state.

In addition to host tracking, the dashboard features **dynamic graphs for server load and per-link bandwidth utilization**. Server load graphs are derived from traffic statistics and packet-in events, enabling identification of overloaded or underutilized servers. Similarly, bandwidth graphs display traffic through individual switch ports, providing insight into

A dedicated **logging panel** aggregates controller events, load balancing, failure recovery and host migration records. Logs are timestamped, categorized, and searchable, giving administrators the ability to track historical events and analyze system behavior over time. Together, these features make the dashboard a comprehensive monitoring and management tool, offering operational visibility, performance insights, and event auditing for the mini cloud infrastructure.



- Mininet Official Documentation <http://mininet.org/>
- Ryu Controller Documentation <https://ryu-sdn.org/>
- OpenFlow Switch Specification v1.3
- Flask Web Development framework <https://flask.palletsprojects.com/>