Performance Evaluation of Software-Defined Networking using Minitnet and Ryu Controller

Bo-Hao Wu (bohaowu2@illinois.edu), Chun-Chieh Chang (cc132@illinois.edu)

ABSTRACT

Software-Defined Networking (SDN) has introduced significant changes to traditional networking by providing centralized control and programmability. OpenFlow plays an important role in managing SDN networks. It is important to review SDN's performance comprehensively before applying it to real-world scenarios in order to determine its practical benefits. A comparison between SDN networks, using simple and advanced controllers, and traditional OSPF networks is presented in this paper. Each of these network architectures is evaluated for its scalability, throughput, and latency, as well as its potential benefits and drawbacks.

1 INTRODUCTION

With SDN, network administrators have the ability to control and manage their networks programmatically. Controllers and network devices communicate using OpenFlow in SDN. On the other hand, traditional networks such as OSPF use distributed protocols to compute optimal routes. We compare the performance of SDN networks with OSPF networks in terms of scalability, throughput, and latency. In our research, we investigate the effect of network topology size, centralized control, and traffic patterns on network performance.

2 BACKGROUND AND RELATED WORK

Mininet[2] is a network emulation tool that creates a virtual network of hosts, switches, and links on a single machine. The Ryu[4] controller is a Python-based framework for building SDN applications. OpenFlow allows SDN controllers to manipulate flow tables in network switches. Traditional OSPF networks use a distributed routing protocol that computes routes based on link-state information. Previous studies have shown that SDN provides better network management and application control, but its performance is affected by controller processing capacity and topology size. Traditional networks tend to have faster convergence times due to decentralized decision-making. information.

3 PROBLEM STATEMENT

As modern networks become increasingly complex, SDN performance needs to be evaluated comprehensively. In order to measure performance, we focus on three key metrics: throughput, latency, and bandwidth. Our objective is to compare SDN against traditional OSPF networks to determine:

- The impact of topology size on performance.
- The difference in latency between SDN and OSPF.
- The scalability and bandwidth of each network type.

4 METHODS

4.1 Network Setup

- **Mininet**: We used Mininet to simulate linear and tree network topologies of varying sizes. Iperf is integrated into hosts in these mininet networks for testing bandwidth and latency.
- **Ryu Controllers**: Two controllers were implemented:
  - **Simple SDN Switch**: This controller works like a learning switch with basic forwarding logic.
  - **Advanced SDN Switch**: This controller tracks network topology and runs the Dijkstra algorithm for optimal routing. The pseudocode for this controller is detailed below.

4.2 Pseudocode for Simple SDN Switch

Algorithm 1 Core Functionality of SimpleSwitch

```
class SimpleSwitch inherits RyuApp
function SWITCH_FEATURES_HANDLER(ev)
    Get datapath, parser, and ofproto from event
    Create match: OFPMatch()
    Define actions: Send all unmatched packets to the controller
    Call add_flow with zero priority for the default rule
end function
function ADD_FLOW(ev)
    Define flow instructions
    Create and send FlowMod message to switch
end function
function PACKET_IN_HANDLER(ev)
    Extract in-port, MAC addresses from packet
    Update MAC-to-port mapping
    Determine output port (flood if unknown)
    If not flooding, install flow to handle future packets
    Create and send PacketOut message
end function
end class
```

4.3 Pseudocode for Advanced SDN Switch

Algorithm 2 Core Functionality of AdvancedSwitch

```
class AdvancedSwitch inherits RyuApp
function SWITCH_FEATURES_HANDLER(ev)
    Initialize with default flow for packet handling
end function
function PACKET_IN_HANDLER(ev)
    Extract MAC addresses and determine in-port
    Learn source MAC to avoid future floods
    Determine output port or flood if unknown
    Install flow to optimize packet routing
end function
function ADD_FLOW(datapath, priority, match, actions)
    Send flow modification to switch
end function
end class

class TopoStructure
function FIND_SHORTEST_PATH(source)
    Employ a Dijkstra algorithm to find shortest paths
end function
end class
```

4.4 OSPF Network

We configured a traditional network using OSPF with the Quagga[3] routing suite:

- **Zebra**: Manages routing tables and interfaces, facilitating communication between the kernel routing table and OSPFd.
- **OSPFd**: Implements OSPF, dynamically exchanging routing information for optimal packet forwarding.

This setup creates a dynamic and efficient routing environment that serves as a performance baseline for comparison with software-defined networks.

4.5 Topology Design

We set up two kinds of network topologies with different parameters to adjust network size.

- **Linear**: Linear network setup a sequence of switches and aligned them into a line. This topology has two parameters: (1) The first parameter numSwitch decides the length of the linear switch chain. (2) The second parameter numHostsPerSwitch specifies how many hosts are connected to each switch. An example of a linear network is presented in Figure 1.
- **Tree**: Tree network setup a tree topology of switches. Each leaf switch is connected to a host. This topology has two parameters: (1) The first parameter depth is the depth of the tree formed by switches. (2) The second parameter fanout specifies how many leaf nodes(switches) the root node or internal nodes have. An example of a tree network is presented in Figure 2.

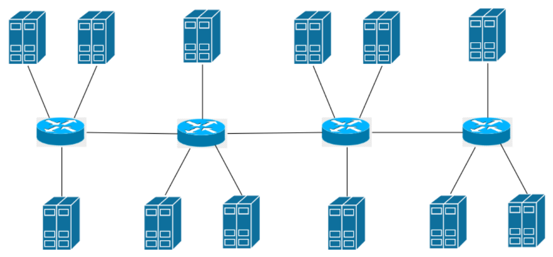

Figure 1: Topology of a linear network with numSwitches=4 and numHostsPerSwitch=3

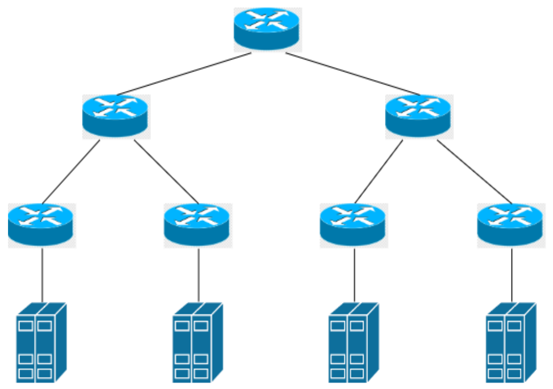

Figure 2: Topology of a tree network with depth=3 and fanout=2

4.6 Traffic Measurement and Metrics

- **Iperf[1]**: For measuring throughput and bandwidth.
- **Ping**: For latency measurements.
- **Metrics**: Round-trip Time (RTT), minimum, average, maximum, and mean deviation; Bandwidth measured in Gbps.

5 RESULTS

For the experiment, we set up six different network topologies with three different switches.

5.1 Average RTT

Figure 3 shows the result for the average RTT of the furthest hosts under the network with different setups. We can observe that the performance of OSPF is really consistent in all network topologies, while both SDN switches suffer from large tree topologies. This could be due to the complexity of the network. In our attempt to implement different switches, managing a big tree network requires a lot to keep track of and control from a central point. There could be a lot of packets arriving at different switches and all sent to the controller, which overwhelms the controller. Another reason for this to happen is that flooded packets between switches can circulate the network and consume a lot of network resources and bandwidth, causing it to slow down. This problem is known as a broadcast storm.

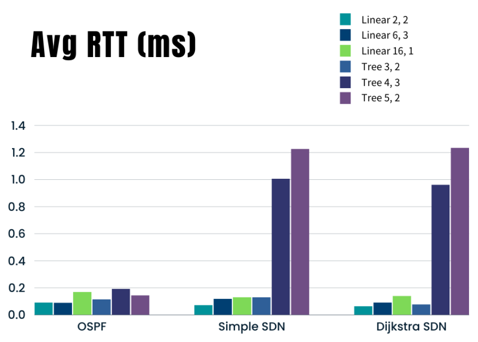

Figure 3: Average RTT between furthest hosts for switches under different network topologies

5.2 Maximum RTT

Figure 4 shows the result for the maximum RTT of the furthest hosts under a network with different setups. Maximum RT follows the same distribution as average RTT. We suspect the underlying reason is also similar. Noted that maximum RTT for SDN switches is significantly larger than average RTT. This is because that when a network is setup, switches do not have any information about hosts. Our SDN switches use the rule of learning switch to update the forward table as packets arrive at the switch. Thus, the first RTT is significantly longer than the rest. We show an example output for a tree network to demonstrate the learning behavior of our SDN switches.

```
mininet> h1 ping -c 10 h4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_seq=1 ttl=64 time=0.317
     ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=0.042
     ms
64 bytes from 10.0.0.4: icmp_seq=3 ttl=64 time=0.082
     ms
64 bytes from 10.0.0.4: icmp_seq=4 ttl=64 time=0.071
     ms
64 bytes from 10.0.0.4: icmp_seq=5 ttl=64 time=0.075
     ms
64 bytes from 10.0.0.4: icmp_seq=6 ttl=64 time=0.077
     ms
64 bytes from 10.0.0.4: icmp_seq=7 ttl=64 time=0.068
     ms
64 bytes from 10.0.0.4: icmp_seq=8 ttl=64 time=0.067
     ms
64 bytes from 10.0.0.4: icmp_seq=9 ttl=64 time=0.068
     ms
64 bytes from 10.0.0.4: icmp_seq=10 ttl=64 time
    =0.072 ms

--- 10.0.0.4 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss,
     time 9198ms
rtt min/avg/max/mdev = 0.042/0.093/0.317/0.075 ms
```

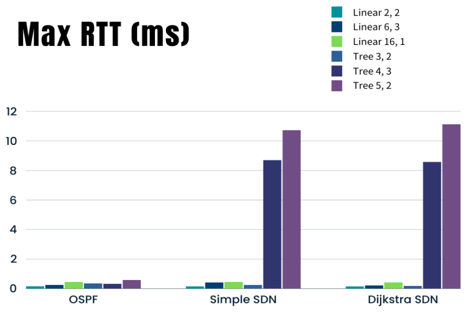

Figure 4: Maximum RTT between furthest hosts for switches under different network topologies

5.3 Bandwidth

Figure 5 shows the overall bandwidth of networks. Even though we set up our network to have a link speed of 1 Mbps. Iperf evaluates the network's bandwidth to have the capacity of Gbps. After adjusting different evaluation times and some retesting, we concluded that this discrepancy might be due to Mininet and iperf running on the same machine. In this case, iperf traffic does not actually go over the network but is instead passed between processes. This could cause iperf to report a much higher bandwidth than the actual link speed. We can observe that the bandwidth is consistent and similar in all different networks topologies. The only factor that affects bandwidth is the maximum path length. It could be shown that even though Tree topology with parameter (4, 2) is a more complex network than with parameter (5, 2), the letter one still has lower bandwidth since the longest path is longer.

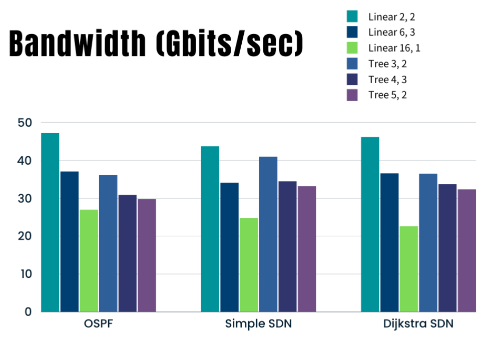

Figure 5: Bandwidth for switches under different network topologies

6 DISCUSSION AND COMPARISON

6.1 Performance

- **Latency**: Traditional OSPF network has lower latency compared to both simple and advanced SDN switches. It makes routing decisions faster due to decentralized computation.
- **Throughput and Bandwidth**: A simple SDN switch has similar throughput to OSPF but struggles in larger networks. Advanced switches performed better than simple switches due to their pre-computed optimized routing using the Dijkstra algorithm.

6.2 Scalability

SDN switches have higher response times and degrades as network size increases. This is likely due to centralized controller delays.

6.3 Security

SDN provides a customizable security policy that can block traffic with programmable rules.

7 CONCLUSION AND FUTURE WORK

This study demonstrates that even though SDN have centralized control and better policy management, traditional networks still often outperform SDN due to decentralized routing. We also encountered challenges like broadcast storms and difficulties in running our SDN controller on certain topologies like torus and mesh, which both contain loops. Although we tried the Spanning Tree Protocol (STP) switch provided on the official Ryu website, we still failed to solve this problem. Future work includes implementing a more optimized controller, implementing various different routing algorithms, and testing SDN switches with STP on more complicated networks that contain loops. By solving these problems, SDN can realize its potential of low latency and robustness under dynamic network topologies.

8 METADATA

The code/data of the project can be found at:

https://github.com/Patrick8894/SDN

REFERENCES

[1] iperf. https://iperf.fr/.
[2] Mininet. https://mininet.org.
[3] Quagga. https://www.nongnu.org/quagga/.
[4] Ryu sdn framework. https://github.com/faucetsdn/ryu.