DAFTAR PUSTAKA

Afan, R., Virgono, A., & Rumani, M. (2018). Analisis Efek Penggunaan *Controller* Ryu Dan POX pada Performansi Jaringan SDN. *eProceedings of Engineering*, *5*(3).

Alotaibi, D., Thayananthan, V., & Yazdani, J. (2021). The 5G *network slicing* using SDN based technology for managing *network traffic*. *Procedia Computer Science*, *194*, 114-121.

Barakabitze, A. A., Ahmad, A., Mijumbi, R., & Hines, A. (2020). 5G *network slicing* using SDN and NFV: A survey of taxonomy, architectures and future challenges. *Computer Networks*, *167*, 106984.

Benzekki, K., El Fergougui, A., & Elbelrhiti Elalaoui, A. (2016). *Software-defined network*ing (SDN): a survey. *Security and communication networks*, *9*(18), 5803-5833.

Broadband Speed Guide. Federal Communications Commission. https://www.fcc.gov/consumers/guides/broadband-speed-guide

Caroline, R., Rahmat, B., & Dewanta, F. (2023). Implementasi Dan Analysis *Network Slicing* Berbasis *Software Defined Network*. *eProceedings of Engineering*, *9*(6).

Cintasari, E. P. (2018). *Analisis Kinerja Jaringan Software Defined Network (SDN) Dengan Protokol OpenFlow pada Mininet* (Bachelor's thesis, Fakultas Sains dan Teknologi UIN Syarif Hidayatullah Jakarta).

Friwansya, H. A., Irawati, I. D., & Hariyani, Y. S. (2018). Implementasi Protokol Routing Ebgp Pada *Software Defined Network* Berbasis Routeflow. *eProceedings of Applied Science*, *4*(3).

Hasbi, M., & Saputra, N. R. (2021). Analisis *Quality of service* (Qos) Jaringan Internet Kantor Pusat King Bukopin Dengan Menggunakan Wireshark. Universitas Muhammadiyah Jakarta, 12 (1), 1–7.

Hu, F., Hao, Q., & Bao, K. (2014). A survey on *software-defined network* and openflow: From concept to implementation. *IEEE Communications Surveys & Tutorials*, *16*(4), 2181-2206.

Islam, M. T., Islam, N., & Refat, M. A. (2020). Node to node performance evaluation through RYU SDN *controller*. *Wireless Personal Communications*, *112*, 555-570.

Khan, L. U., Yaqoob, I., Tran, N. H., Han, Z., & Hong, C. S. (2020). *Network slicing*: Recent advances, taxonomy, requirements, and open research challenges. *IEEE Access*, *8*, 36009-36028.

Kreutz, D., Ramos, F. M., Verissimo, P. E., Rothenberg, C. E., Azodolmolky, S., & Uhlig, S. (2015). *Software-defined networking*: A comprehensive survey. Proceedings of the IEEE, 103(1), 14-76.

Monita, V., Wendanto, W., & Anggiratih, E. (2023). *Network Slicing* Using FlowVisor for Enforcement of *Bandwidth* Isolation in SDN Virtual *Network*s. *Jurnal Ilmiah Teknik Elektro Komputer dan Informatika (JITEKI)*, *9*(3), 768-779.

Oracle. (2023, November). VM VirtualBox | Virtualization. https://www.oracle.com/virtualization/virtualbox/

Pelayo, G. M. E. (2023). *Bandwidth* prediction for adaptive video *streaming*.

Pratama, I. P. A. E., & Bakkara, K. C. (2021). Pengujian QoS Pada Implementasi SDN Berbasis Mininet dan OpenDaylight Menggunakan Topologi Tree. *Jurnal Sisfokom (Sistem Informasi dan Komputer)*, *10*(2), 170-175.

Purnomo, R., & Arisandi, P. R. (2019). Analisis QoS Dengan Virtual Tenant *Network* Pada *Software* Define *Networking*. *Jurnal Teknologi Informatika dan Komputer*, *5*(2), 33-42.

Putra, F. F., Dewanta, F., & Hertiana, S. N. (2023). Analisis QoS Pengaplikasian *Network Slicing* Pada Topologi Abilene Jaringan SDN Menggunakan FlowVisor Dan POX *Controller*. *MULTINETICS*, *9*(1), 35-42.

Ryu-sdn. (2023, November). What’s Ryu?. https://ryu-sdn.org/

Sriastunti, A. K., Primananda, R., & Yahya, W. (2019). Implementasi Routing pada OpenFlow *Software-Defined Network* dengan Algoritme Depth-First Search dan Breadth-First Search. *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer*, *3*(8), 8112-8120.

Thomas, E. E., Palit, H., & Noertjahyana, A. (2018). Aplikasi Manajemen Jaringan Berbasis *Software Defined Networking*. *Jurnal Infra*, *6*(1), 195-199.

Ubuntu. (2023, November). Ubuntu: Enterprise Open Source and Linux. https://ubuntu.com/

Wireshark. (2023, November). Wireshark Frequently Asked Questions. https://www.wireshark.org/faq.html

Zhao, J., Jing, X., Yan, Z., & Pedrycz, W. (2021). *Network traffic* classification for data fusion: A survey. *Information Fusion*, *72*, 22-47.

LAMPIRAN

Lampiran 1 *Script* topologi jaringan 1 *switch* 4 *host*

```
from mininet.topo import Topo
from mininet.net import Mininet
from mininet.link import TCLink
from mininet.cli import CLI
from mininet.node import RemoteController

class MyTopo(Topo):
    def build(self):
        # Tambahkan 1 switch
        switch1 = self.addSwitch('s1')

        # Tambahkan 4 host
        host1 = self.addHost('h1')
        host2 = self.addHost('h2')
        host3 = self.addHost('h3')
        host4 = self.addHost('h4')


        # Hubungkan host ke switch
        self.addLink(host1, switch1, bw=10)
        self.addLink(host2, switch1, bw=10)
        self.addLink(host3, switch1, bw=10)
        self.addLink(host4, switch1, bw=10)


topos = { 'mytopo': ( lambda: MyTopo() ) }


if __name__ == '__main__':
    topo = MyTopo()
    net = Mininet(topo=topo, controller=RemoteController,
autoSetMacs=True)
    net.addController('ryu', controller=RemoteController,
ip='127.0.0.1', port=6653)

    net.start()
    CLI(net)
    net.stop()
```

Lampiran 2 *Script* topologi jaringan 2 *switch* 4 *host*

```
from mininet.topo import Topo
```

```
from mininet.net import Mininet
from mininet.link import TCLink
from mininet.cli import CLI
from mininet.node import RemoteController

class MyTopo(Topo):
    def build(self):
        # Tambahkan 2 switch
        switch1 = self.addSwitch('s1')
        switch2 = self.addSwitch('s2')

        # Tambahkan 2 host
        host1 = self.addHost('h1')
        host2 = self.addHost('h2')
        host3 = self.addHost('h3')
        host4 = self.addHost('h4')

        # Hubungkan host ke switch
        self.addLink(host1, switch1, bw=10)
        self.addLink(host2, switch1, bw=10)
        self.addLink(host3, switch2, bw=10)
        self.addLink(host4, switch2, bw=10)


        # Hubungkan switch ke switch
        self.addLink(switch1, switch2)


topos = { 'mytopo': ( lambda: MyTopo() ) }


if __name__ == '__main__':
    topo = MyTopo()
    net = Mininet(topo=topo, controller=RemoteController,
autoSetMacs=True)
    net.addController('ryu', controller=RemoteController,
ip='127.0.0.1', port=6653)

    net.start()
    CLI(net)
    net.stop()
```

Lampiran 3 *Script* topologi jaringan 3 *switch* 4 *host*

```
from mininet.topo import Topo
from mininet.net import Mininet
```

```
from mininet.link import TCLink
from mininet.cli import CLI
from mininet.node import RemoteController

class MyTopo(Topo):
    def build(self):
        # Tambahkan 3 switch
        switch1 = self.addSwitch('s1')
        switch2 = self.addSwitch('s2')
        switch3 = self.addSwitch('s3')

        # Tambahkan 4 host
        host1 = self.addHost('h1')
        host2 = self.addHost('h2')
        host3 = self.addHost('h3')
        host4 = self.addHost('h4')

        # Hubungkan host ke switch
        self.addLink(host1, switch1, bw=10)
        self.addLink(host2, switch2, bw=10)
        self.addLink(host3, switch2, bw=10)
        self.addLink(host4, switch3, bw=10)


        # Hubungkan switch ke switch
        self.addLink(switch1, switch2)
        self.addLink(switch2, switch3)


topos = { 'mytopo': ( lambda: MyTopo() ) }


if __name__ == '__main__':
    topo = MyTopo()
    net = Mininet(topo=topo, controller=RemoteController,
autoSetMacs=True)
    net.addController('ryu', controller=RemoteController,
ip='127.0.0.1', port=6653)

    net.start()
    CLI(net)
    net.stop()
```

Lampiran 4 *Script* topologi jaringan 4 *switch* 4 *host*

```
from mininet.topo import Topo
```

```
from mininet.net import Mininet
from mininet.link import TCLink
from mininet.cli import CLI
from mininet.node import RemoteController

class MyTopo(Topo):
    def build(self):
        # Tambahkan 4 switch
        switch1 = self.addSwitch('s1')
        switch2 = self.addSwitch('s2')
        switch3 = self.addSwitch('s3')
        switch4 = self.addSwitch('s4')

        # Tambahkan 4 host
        host1 = self.addHost('h1')
        host2 = self.addHost('h2')
        host3 = self.addHost('h3')
        host4 = self.addHost('h4')

        # Hubungkan host ke switch
        self.addLink(host1, switch1, bw=10)
        self.addLink(host2, switch2, bw=10)
        self.addLink(host3, switch3, bw=10)
        self.addLink(host4, switch4, bw=10)


        # Hubungkan switch ke switch
        self.addLink(switch1, switch2)
        self.addLink(switch2, switch3)
        self.addLink(switch3, switch4)
        self.addLink(switch4, switch1)


topos = { 'mytopo': ( lambda: MyTopo() ) }


if __name__ == '__main__':
    topo = MyTopo()
    net = Mininet(topo=topo, controller=RemoteController,
autoSetMacs=True)
    net.addController('ryu', controller=RemoteController,
ip='127.0.0.1', port=6653)

    net.start()
    CLI(net)
    net.stop()
```

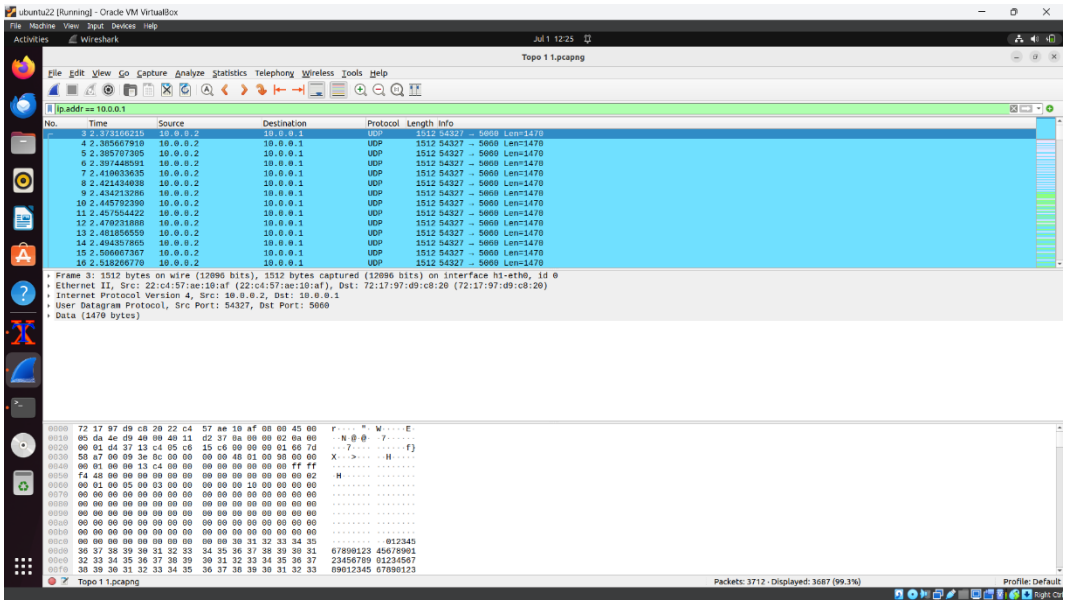
Lampiran 5 *Script* ryu *controller* (ryu-*manager*)

```
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 import dpid as dpid_lib
from ryu.lib import stplib
from ryu.lib.packet import packet
from ryu.lib.packet import ethernet
from ryu.app import simple_switch_13


class SimpleSwitch13(simple_switch_13.SimpleSwitch13):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]
    _CONTEXTS = {'stplib': stplib.Stp}

    def __init__(self, *args, **kwargs):
        super(SimpleSwitch13, self).__init__(*args, **kwargs)
        self.mac_to_port = {}
        self.stp = kwargs['stplib']

        # Sample of stplib config.
        #  please refer to stplib.Stp.set_config() for details.
        config = {dpid_lib.str_to_dpid('0000000000000001'):
                  {'bridge': {'priority': 0x8000}},
                  dpid_lib.str_to_dpid('0000000000000002'):
                  {'bridge': {'priority': 0x9000}},
                  dpid_lib.str_to_dpid('0000000000000003'):
                  {'bridge': {'priority': 0xa000}}}
        self.stp.set_config(config)

    def delete_flow(self, datapath):
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        for dst in self.mac_to_port[datapath.id].keys():
            match = parser.OFPMatch(eth_dst=dst)
            mod = parser.OFPFlowMod(
                datapath, command=ofproto.OFPFC_DELETE,
                out_port=ofproto.OFPP_ANY,
out_group=ofproto.OFPG_ANY,
                priority=1, match=match)
```

```
            datapath.send_msg(mod)

    @set_ev_cls(stplib.EventPacketIn, MAIN_DISPATCHER)
    def _packet_in_handler(self, ev):
        msg = ev.msg
        datapath = msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser
        in_port = msg.match['in_port']

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

        dst = eth.dst
        src = eth.src

        dpid = datapath.id
        self.mac_to_port.setdefault(dpid, {})

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

        # learn a mac address to avoid FLOOD next time.
        self.mac_to_port[dpid][src] = in_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 a flow to avoid packet_in next time
        if out_port != ofproto.OFPP_FLOOD:
            match = parser.OFPMatch(in_port=in_port, eth_dst=dst)
            self.add_flow(datapath, 1, match, actions)

        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)
```

```
    @set_ev_cls(stplib.EventTopologyChange, MAIN_DISPATCHER)
    def _topology_change_handler(self, ev):
        dp = ev.dp
        dpid_str = dpid_lib.dpid_to_str(dp.id)
        msg = 'Receive topology change event. Flush MAC table.'
        self.logger.debug("[dpid=%s] %s", dpid_str, msg)

        if dp.id in self.mac_to_port:
            self.delete_flow(dp)
            del self.mac_to_port[dp.id]

    @set_ev_cls(stplib.EventPortStateChange, MAIN_DISPATCHER)
    def _port_state_change_handler(self, ev):
        dpid_str = dpid_lib.dpid_to_str(ev.dp.id)
        of_state = {stplib.PORT_STATE_DISABLE: 'DISABLE',
                    stplib.PORT_STATE_BLOCK: 'BLOCK',
                    stplib.PORT_STATE_LISTEN: 'LISTEN',
                    stplib.PORT_STATE_LEARN: 'LEARN',
                    stplib.PORT_STATE_FORWARD: 'FORWARD'}
        self.logger.debug("[dpid=%s][port=%d] state=%s",
                          dpid_str, ev.port_no,
of_state[ev.port_state])
```

Lampiran 6 *Traffic packet* pengirim



Lampiran 7 *Traffic packet* penerima