LAPORAN TUGAS AKHIR

Mendesain Jaringan SDN Dengan Mininet

Anggota

Anang Ma'rub (17183207047)
Faris Saifudin (17183207059)
Muhammad Ihza Rois Akbar (17183207055)
Muhammad Nabil Adani (17183207008)
Yusuf Yoga Bakti (17183207064)

Universitas Bhinneka PGRI
Tulungagung

2020

Kata Pengantar

Puji syukur kehadirat Allah SWT atas limpahan rahmat dan anugrah dari-Nya kami dapat menyelesaikan makalah tentang Mendesain Jaringan SDN Dengan Mininet ini. Sholawat dan salam semoga senantiasa tercurahkan kepada junjungan besar kita, Nabi Muhammad SAW yang telah menunjukkan kepada kita semua jalan yang lurus berupa ajaran agama islam yang sempurna dan menjadi anugrah terbesar bagi seluruh alam semesta.Puji syukur kehadirat Allah SWT atas limpahan rahmat dan anugrah dari-Nya kami dapat menyelesaikan makalah tentang Mendesain Jaringan SDN Dengan Mininet ini. Sholawat dan salam semoga senantiasa tercurahkan kepada junjungan besar kita, Nabi Muhammad SAW yang telah menunjukkan kepada kita semua jalan yang lurus berupa ajaran agama islam yang sempurna dan menjadi anugrah terbesar bagi seluruh alam semesta. Penulis sangat bersyukur karena dapat menyelesaikan makalah yang menjadi tugas pendidikan agama dengan judul Mendesain Jaringan SDN Dengan Mininet. Disamping itu, kami mengucapkan banyak terimakasih kepada semua pihak yang telah membantu kamu selama pembuatan makalan ini berlangsung sehingga dapat terealisasikanlah makalah ini.

Penulis sangat bersyukur karena dapat menyelesaikan makalah yang menjadi tugas pendidikan agama dengan judul Mendesain Jaringan SDN Dengan Mininet. Disamping itu, kami mengucapkan banyak terimakasih kepada semua pihak yang telah membantu kamu selama pembuatan makalan ini berlangsung sehingga dapat terealisasikanlah makalah ini.

Demikian yang dapat kami sampaikan, semoga makalah ini dapat bermanfaat bagi para pembaca. Kami mengharapkan kritik dan saran terhadap makalah ini agar kedepannya dapat kami perbaiki. Karena kami sadar, makalah yang kami buat ini masih banyak terdapat kekurangannya. Demikian yang dapat kami sampaikan, semoga makalah ini dapat bermanfaat bagi para pembaca. Kami mengharapkan kritik dan saran terhadap makalah ini agar kedepannya dapat kami perbaiki. Karena kami sadar, makalah yang kami buat ini masih banyak terdapat kekurangannya.

Tulungagung, 9 Juni 2020.

Penulis

BAB 1

PENDAHULUAN

1.1 Latar Belakang

Pada saat ini perkembangan teknologi informasi berkembang sangat pesat, tidak terkecuali pada jaringan komputer. Saat ini berkembang gagasan paradigma baru dalam mengelola jaringan komputer, yang disebut Software-Define Networking (SDN). Software-Define Networking (SDN) adalah sebuah konsep pendekatan baru untuk mendesain, membangun dan mengelola jaringan komputer dengan memisahkan control plane dan data plane. Konsep utama pada Software-Define Networking (SDN) adalah sentralisasi kendali jaringan dengan semua pengaturan berada pada control plane.

Konsep SDN ini sangat memudahkan operator dan network administrator dalam mengelola jaringannya. SDN juga mampu memberikan solusi untuk permasalahan-permasalahan jaringan yang ada sekarang ini, seperti sulitnya mengintegrasikan teknologi baru karena masalah perbedaan platform perangkat keras, kinerja yang buruk karena ada beberapa operasi yang berlebihan pada protokol layer dan sulitnya menyediakan layanan-layanan baru.

Konsep dari SDN sendiri dapat mempermudah dan mempercepat inovasi pada jaringan sehingga diharapkan muncul ide-ide baru yang lebih baik dan dapat dengan cepat diimplementasikan pada real network environment.

1.2 Tujuan

Tujuan ditulisnya makalah ini adalah untuk memahami software define network serta mampu membangun sebuah jaringan sdn dengan menggunakan controller floodlight.

1.3 Rumusan Masalah

1. Memahami software define network (SDN).
2. Membangun sebuah jaringan SDN pada mininet dengan controller floodlight.

BAB 2

MATERI

2.1 Software Define Network

Software Defined Network (SDN) adalah istilah yang merujuk pada konsep/paradigma baru dalam mendisain, mengelola dan mengimplementasikan jaringan, terutama untuk mendukung kebutuhan dan inovasi di bidang ini yg semakin lama semakin kompleks. Konsep dasar SDN adalah dengan melakukan pemisahan eksplisit antara control dan forwarding plane, serta kemudian melakukan abstraksi sistem dan meng-isolasi kompleksitas yg ada pada komponen atau sub-sistem dengan mendefinisikan antar-muka (interface) yg standard.

Beberapa aspek penting dari SDN adalah :

1. Adanya pemisahan secara fisik/eksplisit antara forwarding/data-plane dan control-plane
2. Antarmuka standard (vendor-agnosic) untuk memprogram perangkat jaringan
3. Control-plane yang terpusat (secara logika) atau adanya sistem operasi jaringan yang mampu membentuk peta logika (logical map) dari seluruh jaringan dan kemudian memrepresentasikannya melalui (sejenis) API (Application Programming Interface)
4. Virtualisasi dimana beberapa sistem operasi jaringan dapat mengkontrol bagian-bagian (slices atau substrates) dari perangkat yang sama.

Dalam konsep SDN, tersedia open interface yg memungkinkan sebuah entitas software/aplikasi untuk mengendalikan konektivitas yg disediakan oleh sejumlah sumber-daya jaringan, mengendalikan aliran trafik yg melewatinya serta melakukan inspeksi terhadap atau memodifikasi trafik tersebut.

Arsitektur SDN dapat dilihat sebagai 3 lapis/bidang:

1. Infrastruktur (data-plane / infrastructure layer)

Terdiri dari elemen jaringan yg dapat mengatur SDN Datapath sesuai dengan instruksi yg diberikan melalui Control-Data-Plane Interface (CDPI)

2. Kontrol (control plane / layer)

Entitas kontrol (SDN Controller) mentranslasikan kebutuhan aplikasi dengan infrastruktur dengan memberikan instruksi yg sesuai untuk SDN Datapath serta memberikan informasi yg relevan dan dibutuhkan oleh SDN Application

3. Aplikasi (application plane / layer)

Berada pada lapis teratas, berkomunikasi dengan sistem via NorthBound Interface (NBI)

2.2 Mininet

Mininet merupakah salah satu emulator jaringan sdn yang bisa menjalankan sebuah jaringan virtual yang mana dalam jaringan tersebut terdapat host, switch, controller serta link antar virtual device yang ada. Host pada mininet menggunakan sebuah perangkat lunak jaringan linux. Switch mininet dapat mendukung protokol openflow dengan routing yang fleksible serta compatible dengan software define network.

Beberapa keuntungan dari penggunaan emulator mininet:

1. Dapat melakukan emulasi jaringan SDN dengan jumlah yang besar.
2. Mengaktifkan beberapa mengembang untuk berkerja secara independen pada topologi yang sama.
3. Memungkinkan pengujian topologi yang kompleks tanpa menggunakan jaringan fisik.
4. Mendukung custom topologi yang mudah dimodifikasi.
5. Menyediakan API python yang mudah dan dapat diperluas untuk pembuatan dan eksperimen jaringan

Beberapa kekurangan dari penggunaan emulator mininet:

1. Tidak memiliki GUI yang lebih interaktif dan ramah terhadap pemula.
2. Tidak mudah dalam memahaminya.
3. Pada node tidak jelas perangkat apa yang sedang digunakan.
4. Tidak adanya perangkat wirelles pada mininet.

2.3 Floodlight

Floodlight adalah sebuah controller SDN yang cukup populer, controller ini merupakan kontribusi dari Big Big Switch Networks ke komunitas open source. Floodlight adalah sebuah pengendali openflow yang dibangung menggunakan bahasa pemrograman java yang menggunakan lisensi Apache non-OSGI.

Arsitektur pada floodlight ini bersifat modular, maksudnya adalah komponen pada floodlight mudah dikembangkan sesuai dengan keinginan.

Komponen komponen pada floodlight ini meliputi

1. Manajemen topologi.
2. Manajemen perangkat meliputi MAC Address dan IP Address.
3. Path computation.
4. Infrastruktu dengan akses web untuk management.
5. Counter store.
6. Menggunakan database untuk menyimpan datanya defaultnya menggunakan memory.
7. Menyediakan REST API untuk pengembangan notifikasi dan sejenisnya.

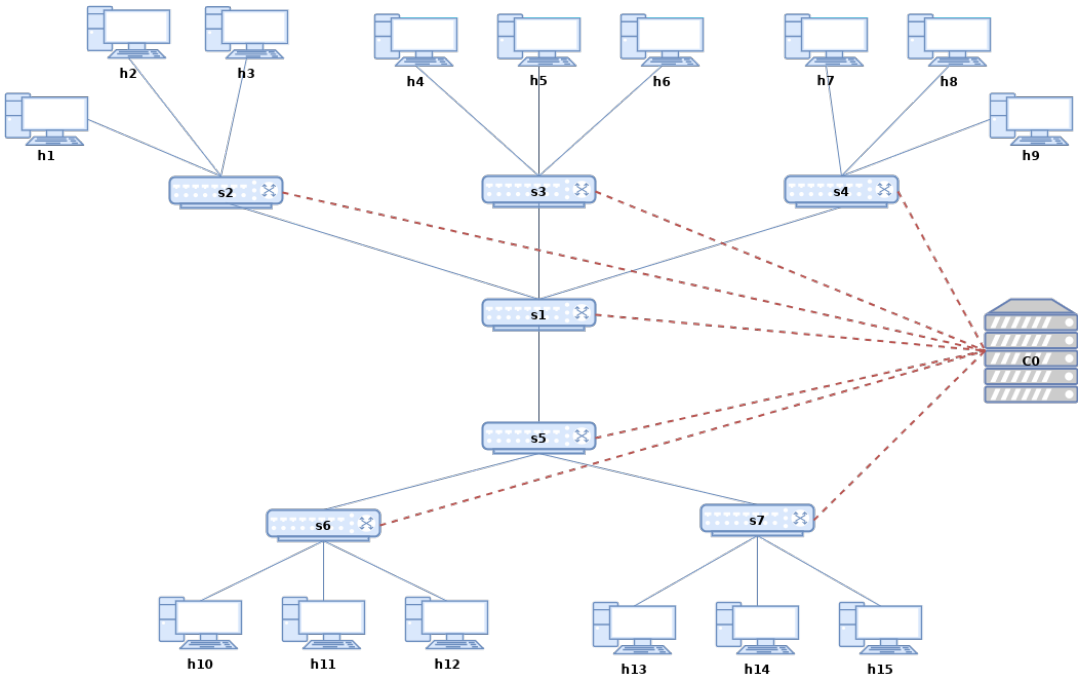
BAB 3

PEMBAHASAN

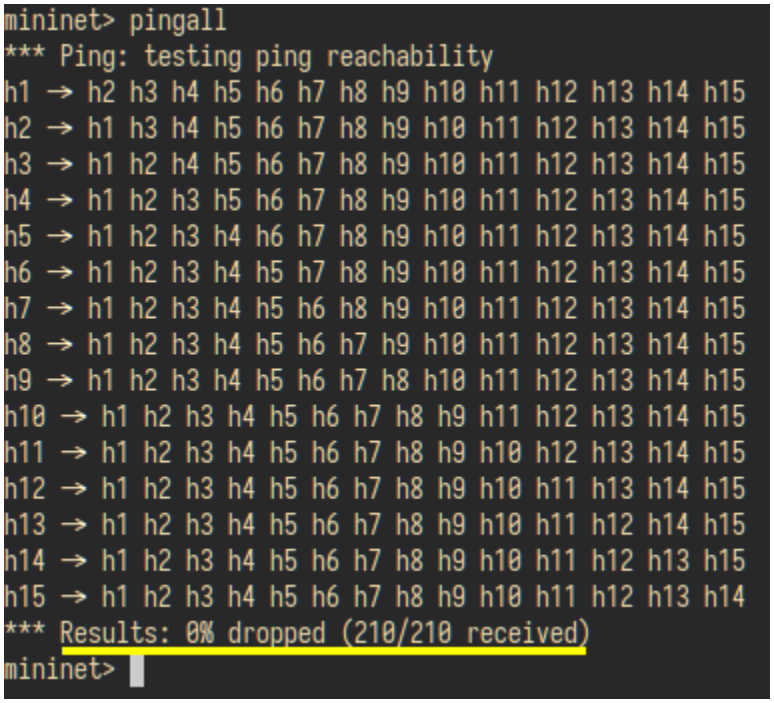
Software yang digunakan untuk mendesain jaringan SDN pada mininet:

1. Virtualbox/KVM (untuk menjalankaun os linux)
2. OS Linux (kami menggunakan ubuntu 18.04 LTS)
3. Docker (untuk menjalankan images floodlight)
4. Floodlight
5. Python

Topologi yang akan didesain dengan mininet



Pada topologi diatas adalah topologi yang kami gunakan untuk mengerjakan tugas uas arsitektur jaringan terkini, dengan menerapkan topologi tree yang menggunakan 15 buah client dan 7 buah switch sebagai penghubung antar clientnya, untuk mencoba apakah jaringan sudah saling terhubung dan tidak ada kendala pada progress pertama kami menerapkan test ping pada semua device yang ada apakah ada kesalahan configurasi atau tidak menggunakan perintah *pingall* pada console mininet dan menghasilkan hasil seperti gambar dibawah ini.



Dari hasil screenshot diatas dapat diketahui bahwa semua client bisa terkoneksi satu samalain akan tetapi dalam praktikum aslinya ping hanya bisa dilakukan pada device yang memiliki ip address saja maka dari itu pada pengujian yang kedua kami melakukan uji coba penambahan ip address pada client yang ada serta menerapkan controler floodlight pada topologi jaringan yang kami desain. Dengan sebarapan ip address seperti di bawah ini.

Host	**Ip Address**
h1	192.168.100.10/27
h2	192.168.100.11/27
h3	192.168.100.12/27
h4	192.168.100.13/27
h5	192.168.100.14/27
h6	192.168.100.15/27
h7	192.168.100.16/27
h8	192.168.100.17/27
h9	192.168.100.18/27
h10	192.168.100.19/27
h11	192.168.100.20/27
h12	192.168.100.21/27
h13	192.168.100.22/27
h14	192.168.100.23/27
h15	192.168.100.24/27

Setelah configurasi dibuat dengan bahasa pemorgraman python versi 2 maka dilanjutkan untuk uji coba konfigurasi pada ubuntu 18.04 yang kami gunakan sebagai server mininetnya dengan configurasi terlampir di lembar terakhir. Pada gambar dibawah ini kami melakukan uji coba test ping dari h1 ke h15 dan dari h1 ke h10.

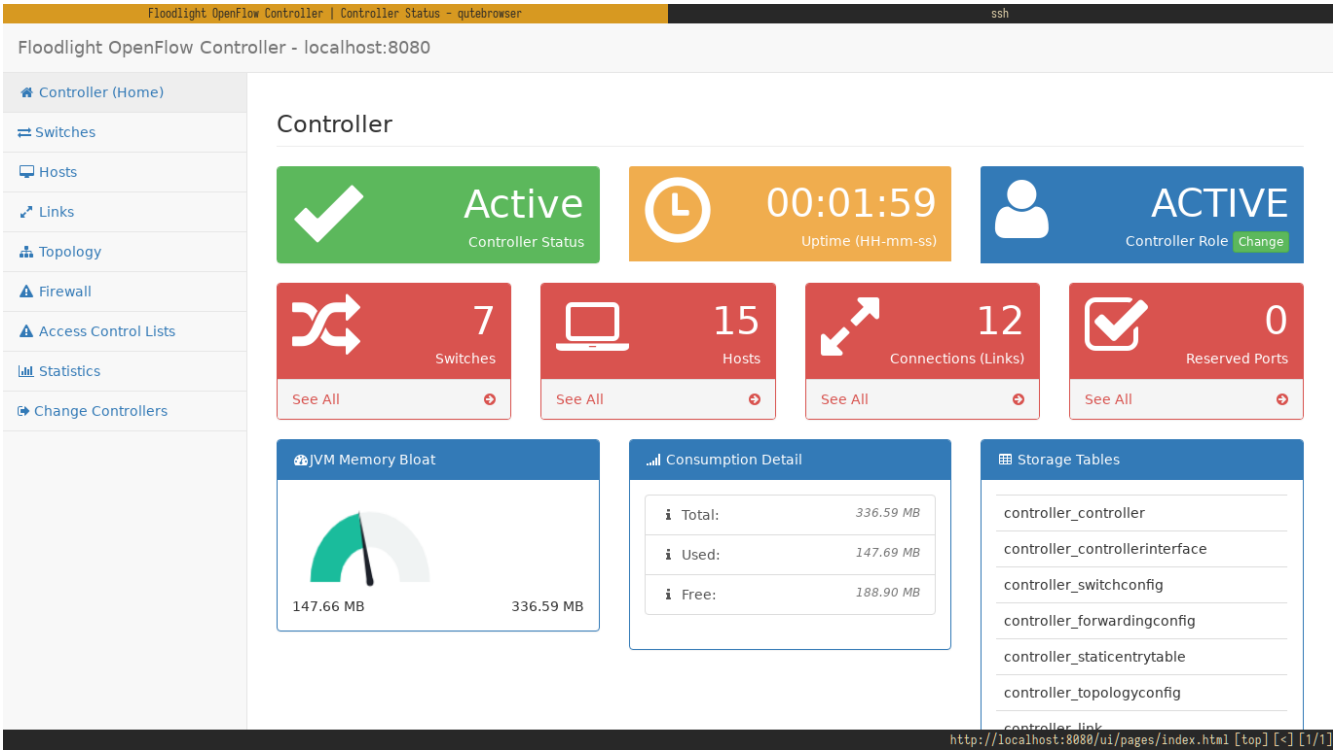
```
mininet> h1 ping -c 4 192.168.100.24
PING 192.168.100.24 (192.168.100.24) 56(84) bytes of data.
64 bytes from 192.168.100.24: icmp_seq=1 ttl=64 time=91.7 ms
64 bytes from 192.168.100.24: icmp_seq=2 ttl=64 time=22.9 ms
64 bytes from 192.168.100.24: icmp_seq=3 ttl=64 time=0.111 ms
64 bytes from 192.168.100.24: icmp_seq=4 ttl=64 time=0.091 ms

--- 192.168.100.24 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3029ms
rtt min/avg/max/mdev = 0.091/28.723/91.755/37.566 ms
mininet> h1 ping -c 4 192.168.100.19
PING 192.168.100.19 (192.168.100.19) 56(84) bytes of data.
64 bytes from 192.168.100.19: icmp_seq=1 ttl=64 time=983 ms
64 bytes from 192.168.100.19: icmp_seq=2 ttl=64 time=6.67 ms
64 bytes from 192.168.100.19: icmp_seq=3 ttl=64 time=0.097 ms
64 bytes from 192.168.100.19: icmp_seq=4 ttl=64 time=0.112 ms

--- 192.168.100.19 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3019ms
rtt min/avg/max/mdev = 0.097/247.701/983.923/425.066 ms
mininet>
```

Pengujian hanya dilakukan pada h1 ke h15 dan h1 ke h10 untuk memberikan gambaran apakah configurasi yang telah kami buat benar benar berjalan sedangkan hasil pengujian pada semua host yang ada di dalam topologi mininet terlampir di lembar terakhir setelah lembar configurasi.

Setelah tahap pengujian jaringan pada mininet dirasa sudah berhasil selanjutnya kami melakukan pengujian controller floodlight pada jaringan yang kami buat apakah sudah sesuai dengan keinginan kami atau belum. Tahapan awal yang kami ambil yakni melakukan instalasi floodlight pada docker container . Docker disini kami fungsikan sebagai host untuk floodlightnya yang terinstall pada laptop dengan sistem operasi archlinux, sedangkan untuk mininetnya tetap terinstall di ubuntu 18.04 yang terdapat pada kvm. dibawah ini adalah gambar dari docker container yang sedang menjalankan floodlight.

```
❯ docker container ls
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS                                                NAMES
aedd02ffd84e        glefevre/floodlight   "/bin/sh -c 'java -j…"   8 days ago          Up 7 seconds        0.0.0.0:6653->6653/tcp, 0.0.0.0:8080->8080/tcp   fl
~
❯
```

Setelah tahap pembuatan container pada docker selesai dijalankan, maka web-ui floodlight bisa diakses melalu url ttp://localhost:8080/ui/pages/index.html untuk melihat tampilan dari floodlight itu sendiri. Dibawah ini adalah tampilan dari awala floodlight.



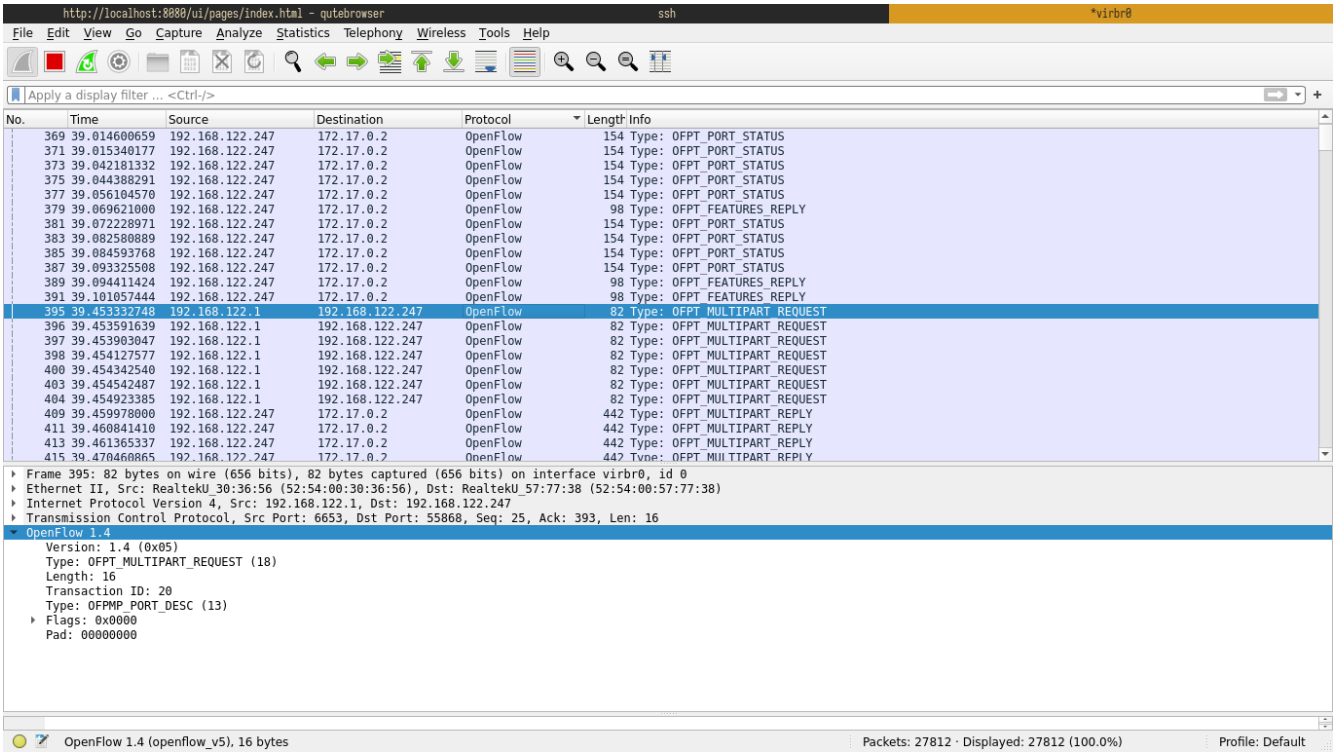
Setelah kami rasa floodlight sukses dijalankan maka kami mencoba menjalankan mininet menggunakan floodlight sebagai controllernya menggunakan perintah.

sudo mn --custom learn.py --topo g1 --controller remote,ip=192.168.122.1,port=6653

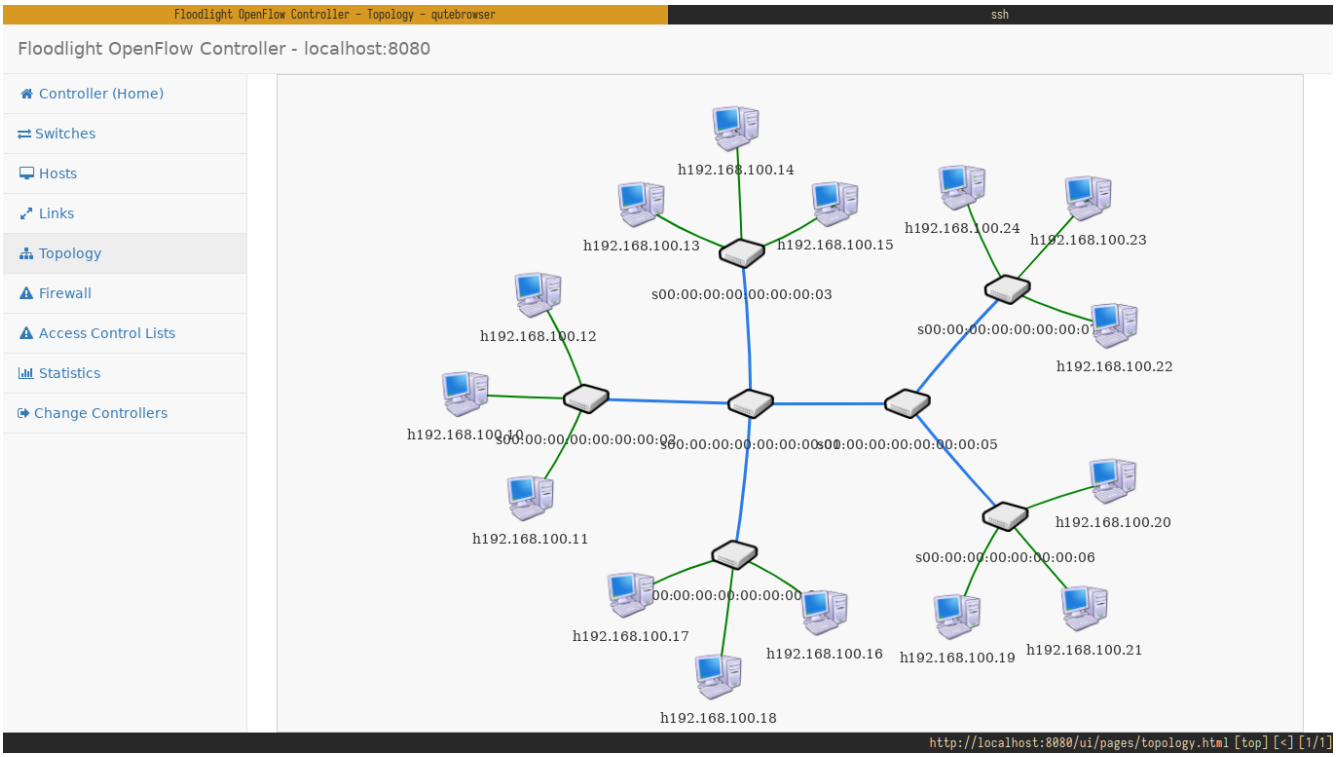
pada perintah diatas dapat diketahui bahwa mininet menggunakan controller floodlight yang memiliki ip address 192.168.122.1 yang mengarah ke interface kvmnya dan berjalan pada port 6653 dan menjalankan custom topologi dari configurasi yang telah kami buat. Dibawah ini adalah console log dari minet ketika dijalankan dengan controller floodlight.

```
user@lubuntu:~/uas$ sudo mn --custom config.py --topo g1 --controller remote,ip=192.168.122.1
*** Creating network
*** Adding controller
Connecting to remote controller at 192.168.122.1:6653
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15
*** Adding switches:
s1 s2 s3 s4 s5 s6 s7
*** Adding links:
(h1, s2) (h2, s2) (h3, s2) (h4, s3) (h5, s3) (h6, s3) (h7, s4) (h8, s4) (h9, s4) (h10, s6) (h11, s6) (h12, s6) (h13, s7) (h14, s7) (h15, s7) (s2, s1) (s3, s1) (s4, s1) (s5, s1) (s6, s5) (s7, s5)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15
*** Starting controller
c0
*** Starting 7 switches
s1 s2 s3 s4 s5 s6 s7 ...
*** Starting CLI:
mininet>
```

Pada saat mininet terkoneski dengan controller floodlight kita dapat mengetahui paket apa saja yang dikirimkan dari mininet ke controller melalui aplikasi wireshark.



Pada log wireshark di atas ini kita dapat mengetahui bahwa controller floodlight dengan ip address 192.168.122.1 mencoba terkoneksi dengan server mininet yang memiliki ip address 192.168.122.247 dengan cara mengirimkan OFPT_MULTIPART_REPLY. Kemudia data yang diterima dari mininet akan ditampilkan ke web-ui floodlight yang bisa diakses melalui browser pada sistem operasi yang digunakan. Sedangkan untuk melihat topologi pada jaringan mininet bisa dilihat melalui menu topologi yang ada di web-ui controller floodlight. Gambar dibawah ini adalah gambar ketika floodlight sukses menampilkan topologi yang ada di mininet.



BAB 4

PENUTUP

4.1 Kesimpulan

Setelah melakukan beberapa kali pengujian dari emulator SDN openflow yakni mininet dengan bantuan controller floodlight dapat diambil kesimpulan bahwa emulator minet dapat membantu mahasiswa memahami serta memberikan gambaran lebih mengenai infrastruktur pada SDN openflow dengan biaya yang relatif murah karena memungkinkan pengujian topologi yang kompleks dan menyediakan custom topologi tanpa menggunakan jaringan fisik.

Lampiran

config.py

Configurasi mininet dengan 15 computer dan 7 buah switch configurasi bisa diakses melalui https://github.com/mnabila/learningMininet

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

class GedungSatu(Topo):
    def __init__(self, **opts):
        Topo.__init__(self, **opts)

        # tambah client sebanyak 15 komputer
        h1 = self.addHost(name="h1", mac="00:00:00:00:0h:01", ip="192.168.100.10/27")
        h2 = self.addHost(name="h2", mac="00:00:00:00:0h:02", ip="192.168.100.11/27")
        h3 = self.addHost(name="h3", mac="00:00:00:00:0h:03", ip="192.168.100.12/27")
        h4 = self.addHost(name="h4", mac="00:00:00:00:0h:04", ip="192.168.100.13/27")
        h5 = self.addHost(name="h5", mac="00:00:00:00:0h:05", ip="192.168.100.14/27")
        h6 = self.addHost(name="h6", mac="00:00:00:00:0h:06", ip="192.168.100.15/27")
        h7 = self.addHost(name="h7", mac="00:00:00:00:0h:07", ip="192.168.100.16/27")
        h8 = self.addHost(name="h8", mac="00:00:00:00:0h:08", ip="192.168.100.17/27")
        h9 = self.addHost(name="h9", mac="00:00:00:00:0h:09", ip="192.168.100.18/27")
        h10 = self.addHost(name="h10", mac="00:00:00:00:0h:10", ip="192.168.100.19/27")
        h11 = self.addHost(name="h11", mac="00:00:00:00:0h:11", ip="192.168.100.20/27")
        h12 = self.addHost(name="h12", mac="00:00:00:00:0h:12", ip="192.168.100.21/27")
        h13 = self.addHost(name="h13", mac="00:00:00:00:0h:13", ip="192.168.100.22/27")
        h14 = self.addHost(name="h14", mac="00:00:00:00:0h:14", ip="192.168.100.23/27")
        h15 = self.addHost(name="h15", mac="00:00:00:00:0h:15", ip="192.168.100.24/27")

        # tambah switch sebanyak 7 buah
        s1 = self.addSwitch(name="s1", cls=OVSKernelSwitch, mac="00:00:00:00:0s:01",)
        s2 = self.addSwitch(name="s2", cls=OVSKernelSwitch, mac="00:00:00:00:0s:02",)
        s3 = self.addSwitch(name="s3", cls=OVSKernelSwitch, mac="00:00:00:00:0s:03",)
        s4 = self.addSwitch(name="s4", cls=OVSKernelSwitch, mac="00:00:00:00:0s:04",)
        s5 = self.addSwitch(name="s5", cls=OVSKernelSwitch, mac="00:00:00:00:0s:05",)
        s6 = self.addSwitch(name="s6", cls=OVSKernelSwitch, mac="00:00:00:00:0s:06",)
        s7 = self.addSwitch(name="s7", cls=OVSKernelSwitch, mac="00:00:00:00:0s:07",)

        # membuat topologi tree
        # menghubungkan switch bagian atas
        # menghubungkan switch s1,s2,s3 ke switch s1
        self.addLink(s2, s1)
        self.addLink(s3, s1)
        self.addLink(s4, s1)

        # menguhubungkan kedua switch untuk jaringan atas dan bawah
        self.addLink(s5, s1)

        # menghubungkan switch bagian bawah
        # menghubungkan switch s6,s7 ke switch s5
        self.addLink(s6, s5)
        self.addLink(s7, s5)

        # menghubungkan client ke setiap switch bagian atas
        # menghubungkan client ke switch s2
        self.addLink(h1, s2)
```

```
        self.addLink(h2, s2)
        self.addLink(h3, s2)

        # menghubungkan client ke switch s3
        self.addLink(h4, s3)
        self.addLink(h5, s3)
        self.addLink(h6, s3)

        # menghubungkan client ke switch s4
        self.addLink(h7, s4)
        self.addLink(h8, s4)
        self.addLink(h9, s4)

        # menguhubungkan client ke setiap switch bagian bawah
        # menghubungkan client ke switch s6
        self.addLink(h10, s6)
        self.addLink(h11, s6)
        self.addLink(h12, s6)

        # menghubungkan client ke switch s7
        self.addLink(h13, s7)
        self.addLink(h14, s7)
        self.addLink(h15, s7)

topos = {"g1": (lambda: GedungSatu())}
```

Uji Coba

console log ping pada semua host di topologi jaringan mininet kami dengan mengirimkan dua paket saja untuk menghemat durasi dan agar log pingnya tidak terlalu panjang
test ping dilakukan dari h1 menuju host lainnya

```
mininet> h1 ping -c 2 192.168.100.10
PING 192.168.100.10 (192.168.100.10) 56(84) bytes of data.
64 bytes from 192.168.100.10: icmp_seq=1 ttl=64 time=0.048 ms
64 bytes from 192.168.100.10: icmp_seq=2 ttl=64 time=0.057 ms

--- 192.168.100.10 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1003ms
rtt min/avg/max/mdev = 0.048/0.052/0.057/0.008 ms
mininet> h1 ping -c 2 192.168.100.11
PING 192.168.100.11 (192.168.100.11) 56(84) bytes of data.
64 bytes from 192.168.100.11: icmp_seq=1 ttl=64 time=16.6 ms
64 bytes from 192.168.100.11: icmp_seq=2 ttl=64 time=0.831 ms

--- 192.168.100.11 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1002ms
rtt min/avg/max/mdev = 0.831/8.748/16.666/7.918 ms
mininet> h1 ping -c 2 192.168.100.12
PING 192.168.100.12 (192.168.100.12) 56(84) bytes of data.
64 bytes from 192.168.100.12: icmp_seq=1 ttl=64 time=16.6 ms
64 bytes from 192.168.100.12: icmp_seq=2 ttl=64 time=3.64 ms

--- 192.168.100.12 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1002ms
rtt min/avg/max/mdev = 3.641/10.143/16.646/6.503 ms
mininet> h1 ping -c 2 192.168.100.13
```

PING 192.168.100.13 (192.168.100.13) 56(84) bytes of data.
64 bytes from 192.168.100.13: icmp_seq=1 ttl=64 time=474 ms
64 bytes from 192.168.100.13: icmp_seq=2 ttl=64 time=1.02 ms

--- 192.168.100.13 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 1.021/237.726/474.431/236.705 ms
mininet> h1 ping -c 2 192.168.100.14
PING 192.168.100.14 (192.168.100.14) 56(84) bytes of data.
64 bytes from 192.168.100.14: icmp_seq=1 ttl=64 time=51.1 ms
64 bytes from 192.168.100.14: icmp_seq=2 ttl=64 time=1.69 ms

--- 192.168.100.14 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1000ms
rtt min/avg/max/mdev = 1.693/26.428/51.164/24.736 ms
mininet> h1 ping -c 2 192.168.100.15
PING 192.168.100.15 (192.168.100.15) 56(84) bytes of data.
64 bytes from 192.168.100.15: icmp_seq=1 ttl=64 time=42.5 ms
64 bytes from 192.168.100.15: icmp_seq=2 ttl=64 time=1.10 ms

--- 192.168.100.15 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 1.105/21.826/42.548/20.722 ms
mininet> h1 ping -c 2 192.168.100.16
PING 192.168.100.16 (192.168.100.16) 56(84) bytes of data.
64 bytes from 192.168.100.16: icmp_seq=1 ttl=64 time=39.8 ms
64 bytes from 192.168.100.16: icmp_seq=2 ttl=64 time=1.55 ms

--- 192.168.100.16 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 1.557/20.714/39.872/19.158 ms
mininet> h1 ping -c 2 192.168.100.17
PING 192.168.100.17 (192.168.100.17) 56(84) bytes of data.
64 bytes from 192.168.100.17: icmp_seq=1 ttl=64 time=53.8 ms
64 bytes from 192.168.100.17: icmp_seq=2 ttl=64 time=1.59 ms

--- 192.168.100.17 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 1.598/27.720/53.842/26.122 ms
mininet> h1 ping -c 2 192.168.100.18
PING 192.168.100.18 (192.168.100.18) 56(84) bytes of data.
64 bytes from 192.168.100.18: icmp_seq=1 ttl=64 time=39.7 ms
64 bytes from 192.168.100.18: icmp_seq=2 ttl=64 time=1.40 ms

--- 192.168.100.18 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 1.406/20.567/39.729/19.162 ms
mininet> h1 ping -c 2 192.168.100.19
PING 192.168.100.19 (192.168.100.19) 56(84) bytes of data.
64 bytes from 192.168.100.19: icmp_seq=1 ttl=64 time=60.0 ms
64 bytes from 192.168.100.19: icmp_seq=2 ttl=64 time=4.29 ms

--- 192.168.100.19 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 4.290/32.191/60.093/27.902 ms
mininet> h1 ping -c 2 192.168.100.20
PING 192.168.100.20 (192.168.100.20) 56(84) bytes of data.

64 bytes from 192.168.100.20: icmp_seq=1 ttl=64 time=50.6 ms
64 bytes from 192.168.100.20: icmp_seq=2 ttl=64 time=1.69 ms

--- 192.168.100.20 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1002ms
rtt min/avg/max/mdev = 1.698/26.154/50.610/24.456 ms
mininet> h1 ping -c 2 192.168.100.21
PING 192.168.100.21 (192.168.100.21) 56(84) bytes of data.
64 bytes from 192.168.100.21: icmp_seq=1 ttl=64 time=52.6 ms
64 bytes from 192.168.100.21: icmp_seq=2 ttl=64 time=2.09 ms

--- 192.168.100.21 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1002ms
rtt min/avg/max/mdev = 2.095/27.364/52.633/25.269 ms
mininet> h1 ping -c 2 192.168.100.22
PING 192.168.100.22 (192.168.100.22) 56(84) bytes of data.
64 bytes from 192.168.100.22: icmp_seq=1 ttl=64 time=39.9 ms
64 bytes from 192.168.100.22: icmp_seq=2 ttl=64 time=1.42 ms

--- 192.168.100.22 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 1.425/20.705/39.985/19.280 ms
mininet> h1 ping -c 2 192.168.100.23
PING 192.168.100.23 (192.168.100.23) 56(84) bytes of data.
64 bytes from 192.168.100.23: icmp_seq=1 ttl=64 time=35.9 ms
64 bytes from 192.168.100.23: icmp_seq=2 ttl=64 time=1.26 ms

--- 192.168.100.23 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 1.268/18.633/35.999/17.366 ms
mininet> h1 ping -c 2 192.168.100.24
PING 192.168.100.24 (192.168.100.24) 56(84) bytes of data.
64 bytes from 192.168.100.24: icmp_seq=1 ttl=64 time=43.7 ms
64 bytes from 192.168.100.24: icmp_seq=2 ttl=64 time=1.13 ms

--- 192.168.100.24 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 1.131/22.460/43.790/21.330 ms
mininet>