IMPLEMENTASI JARINGAN *WI-FI* PADA *SOFTWARE-DEFINED NETWORKING (SDN)* DENGAN MENGGUNAKAN ONOS *CONTROLLER* DAN OPEN vSWITCH

SKRIPSI

Diajukan sebagai salah satu syarat untuk menyelesaikan Pendidikan Diploma Empat (D-4) Program Studi Teknik Komputer dan Jaringan
Jurusan Teknik Elektro
Politeknik Negeri Ujung Pandang

NINDA DWIYANA

425 17 014

PROGRAM STUDI D-4 TEKNIK KOMPUTER DAN JARINGAN

JURUSAN TEKNIK ELEKTRO

POLITEKNIK NEGERI UJUNG PANDANG

MAKASSAR

2021

HALAMAN PENGESAHAN

Skripsi dengan judul "**Implementasi Jaringan *Wi-Fi* pada *Software-Defined Networking (SDN)* dengan menggunakan ONOS *Controller* dan Open vSwitch**" oleh **Ninda Dwiyana (425 17 014)** telah diterima dan disahkan sebagai salah satu syarat untuk memperoleh gelar Diploma IV (D-4/ S1 Terapan) pada Program Studi Teknik Komputer dan Jaringan Jurusan Teknik Elektro Politeknik Negeri Ujung Pandang.

Makassar, September 2021

Menyetujui,

Pembimbing I,	Pembimbing II,
Ir.Dahlia,M.T.	**Zawiyah Saharuna, S.T., M.Eng.**
NIP. 196412311 991032 003	**NIP. 19830903 201404 2 001**

Mengetahui,
Koordinator Program Studi
Teknik Komputer dan Jaringan
Politeknik Negeri Ujung Pandang

Eddy Tungadi, S.T., M.T.
NIP. 19790823 201012 1 001

HALAMAN PENERIMAAN

Pada hari ini, Senin 20 September 2021, Tim Penguji Sidang Tugas Akhir, telah menerima dengan baik hasil skripsi oleh mahasiswa: **Ninda Dwiyana** Nomor Induk Mahasiswa **425 17 014** dengan judul **"Implementasi Jaringan *Wi-Fi* pada *Software-Defined Networking (SDN)* dengan menggunakan ONOS *Controller* dan Open vSwitch"**

Makassar, September 2021

1. Drs. Kasim, M.T.	Ketua	(.....)
2. Iin Karmila Yusri, S.ST., M.Eng., Ph.D.	Sekretaris	(.....)
3. Rini Nur, S.T., M.T.	Anggota	(.....)
4. Eddy Tungadi, S.T., M.T.	Anggota	(.....)
5. Ir. Dahlia, M.T.	Pembimbing I	(.....)
6. Zawiyah Saharuna, S.T., M.Eng.	Pembimbing II	(.....)

KATA PENGANTAR

Puji syukur penulis panjatkan kehadirat Allah Subhanahu Wa Ta'Ala atas rahmat dan hidayah-Nya yang telah memberikan kesehatan dan keselamatan kepada penulis sehingga penulis dapat menyelesaikan skripsi ini dengan baik. Salawat dan salam kepada baginda rasul Muhammad Sallallahu Alaihi Wasallam sebagai sebaik-baik panutan bagi seluruh manusia.

Skripsi ini disusun guna memenuhi salah satu syarat untuk menyelesaikan studi serta dalam rangka memperoleh gelar Diploma IV (D4/S1 Terapan) pada Program Studi Teknik Komputer dan Jaringan Jurusan Teknik Elektro Politeknik Negeri Ujung Pandang.

Penyusunan skripsi ini bukanlah hasil kerja penulis sendiri, melainkan juga berkat bantuan dari berbagai pihak baik secara langsung maupun tidak langsung. Oleh karena itu, dengan rendah hati penulis mengucapkan rasa terima kasih yang tak terhingga kepada kedua orang tua penulis terkhusus ibunda tercinta Hj. Asma. Ibu yang selalu memberikan dukungan moral maupun materi, semangat, kasih sayang, pengorbanan, motivasi, kepercayaan, bimbingan dan doa restu. Dan kepada ayahanda Sudarman yang selalu memberikan dukungan moral dan materi, motivasi, bimbingan dan doa restu serta menguatkan penulis untuk tetap semangat, sabar, dan tabah. Tak lupa penulis ucapkan terima kasih kepada kakak dan adik-adikku Muh. Ikbal, Nilpa Triana dan Ghina Reski Adelyana serta keluarga besar penulis yang telah memberikan dukungan, motivasi, dan doa kepada penulis untuk menyelesaikan skripsi ini. Kesempatan ini penulis menyampaikan penghargaan dan ucapan terima kasih yang sebesar-besarnya kepada:

1. Bapak Prof. Ir. Muhammad Anshar, M.Si., Ph.D. selaku Direktur Politeknik Negeri Ujung Pandang.
2. Bapak Ahmad Rizal Sultan, S.T., M.T., Ph.D. selaku Ketua Jurusan Elektro Politeknik Negeri Ujung Pandang.
3. Bapak Eddy Tungadi, S.T., M.T. selaku Koordinator Program Studi Teknik Komputer dan Jaringan Politeknik Negeri Ujung Pandang.

4. Ibu Ir. Dahlia, M.T. selaku pembimbing I dan Ibu Zawiyah Saharuna, S.T., M.Eng. selaku pembimbing II yang berkenan menyiapkan waktu untuk membantu, mengarahkan, memberikan masukan dalam membimbing penulis hingga selesainya penelitian ini.
5. Teman-teman seperjuangan Teknik Komputer dan Jaringan angkatan 2017 yang telah memberikan banyak pembelajaran hidup tentang kebersamaan dan persaudaraan selama menyelesaikan studi di Politeknik Negeri Ujung Pandang.
6. Teman-teman anggota PLI *IT CENTER* PNUP yang telah memberikan banyak bantuan, semangat dan motivasi selama proses pengerjaan penelitian ini.
7. Sahabat-sahabat kelas TKJ B 2017 yang telah banyak memberikan bantuan, semangat, dan motivasi selama proses penelitian ini. Terima kasih untuk 4 tahun yang luar biasa.
8. Semua pihak yang telah memberikan bantuan moril maupun materiil yang tidak dapat disebutkan satu per satu.

Penulis menyadari bahwa masih banyak kekurangan baik dari penulisan maupun penyajian skripsi ini, untuk itu segala kritik dan saran yang membangun sangat diharapkan untuk perbaikan di masa mendatang. Semoga skripsi ini dapat bermanfaat bagi pembaca maupun penulis sendiri. Akhirnya, semoga Allah SWT. memberikan perlindungan kepada kita semua.

Makassar, September 2021

Penulis

DAFTAR ISI

DAFTAR TABEL

Hal.

DAFTAR GAMBAR

Hal.

DAFTAR LAMPIRAN

Hal.

SURAT PERNYATAAN

Saya yang bertanda tangan di bawah ini:

Nama : Ninda Dwiyana

NIM : 425 17 014

menyatakan dengan sebenar-benarnya bahwa segala pernyataan dalam skripsi ini yang berjudul **Implementasi Jaringan *Wifi* Pada *Software-Defined Networking (SDN)* dengan menggunakan ONOS *Controller* Dan Open vSwitch** merupakan gagasan dan hasil karya saya sendiri dengan arahan komisi pembimbing, dan belum pernah diajukan dalam bentuk apapun pada perguruan tinggi dan instansi manapun.

Semua data dan informasi yang digunakan telah dinyatakan secara jelas dan dapat diperiksa kebenarannya. Sumber informasi yang berasal atau dikutip dari karya yang diterbitkan dari penulis lain telah disebutkan dalam naskah dan dicantumkan dalam skripsi ini.

Jika pernyataan saya tersebut di atas tidak benar, saya siap menanggung risiko yang ditetapkan oleh Politeknik Negeri Ujung Pandang.

Makassar, 15 September 2021

Ninda Dwiyana

NIM. 425 170 14

IMPLEMENTASI JARINGAN *WIFI* PADA *SOFTWARE-DEFINED NETWORKING (SDN)* DENGAN MENGGUNAKAN ONOS *CONTROLLER* DAN OPEN VSWITCH

RINGKASAN

Software-Defined Networking atau SDN adalah konsep pendekatan baru untuk mendesain, membangun dan mengelola jaringan komputer dengan memisahkan *control plane* dan *data plane*. Pada jaringan tradisional atau non-SDN, kedua fungsi tersebut berada dalam satu perangkat yang sama. Dalam arsitekturnya, SDN menggunakan arsitektur jaringan terpusat yang memisahkan *logical* dari perangkat jaringan ke sebuah entitas yang disebut dengan *controller*. *Controller* ini bersifat *programmable*. Kelemahan dari penerapan SDN adalah metodenya yang masih menggunakan media transmisi kabel. Banyaknya pengguna pada jaringan yang kompleks, pengeluaran biaya kabel serta keterbatasan port pada perangkat membuat penggunaan kabel tidak efektif dan efisien, sehingga dalam perkembangan SDN diperkenalkan sebuah konsep yang dikenal dengan nama *Software-Defined Wireless Networking* (SDWN) yang mengatasi keterbatasan SDN yang menggunakan kabel dalam penerapannya. Penelitian ini menganalisis kinerja dari implementasi jaringan *Wi-Fi* dengan konsep Software-defined Networking dengan menggunakan arsitektur *distributed controller*. Pengujian dilakukan dengan menguji *leader election* pada *distributed controller*, mengukur *throughput*, *packet loss*, dan *latency* menggunakan *video streaming* dan software *Apache Jmeter*. Hasil dari penelitian ini menunjukkan bahwa *leader election* pada *distributed controller* dapat bekerja dengan baik dan jaringan *Wi-Fi* dengan SDN menunjukkan nilai indeks QoS secara keseluruhan (QoS terhadap jumlah *user* dan QoS terhadap ukuran *bandwidth*) sebesar 3.97 dengan kategori "sangat memuaskan" menurut ketentuan yang telah ditetapkan oleh TIPHON.

Kata kunci : *Software-Defined Networking, Distributed Controller, Software-Defined Wireless Networking,* ONOS, Open vSwitch.

BAB I PENDAHULUAN

1.1 Latar Belakang

Software-Defined Networking (SDN) merupakan pendekatan inovatif untuk merancang, membangun dan mengelola jaringan yang memisahkan pengontrolan jaringan (*control plane*) dan proses *forwarding* (*data plane*) sehingga menawarkan banyak manfaat dalam hal fleksibilitas dan kemampuan pengontrolan jaringan (Mulyana, 2015). Pada jaringan tradisional atau non-SDN, kedua fungsi tersebut berada dalam satu perangkat yang sama. *Control Plane* adalah fungsi pengaturan atau kontrol pada suatu perangkat jaringan, sedangkan *data plane* adalah fungsi dari pengiriman paket-paket informasi (Flowgrammable, 2016). Dalam arsitekturnya, SDN menggunakan arsitektur jaringan terpusat yang memisahkan *logical* dari perangkat jaringan ke sebuah entitas yang disebut dengan *controller*. *Controller* ini bersifat *programmable*. Kelemahan dari penerapan SDN adalah metodenya yang masih menggunakan media transmisi kabel. Banyaknya pengguna pada jaringan yang kompleks, pengeluaran biaya kabel serta keterbatasan port pada perangkat membuat penggunaan kabel tidak efektif dan efisien, sehingga dalam perkembangan SDN diperkenalkan sebuah konsep yang dikenal dengan nama *Software-Defined Wireless Networking* (SDWN) yang mengatasi keterbatasan SDN yang menggunakan kabel dalam penerapannya.

Dengan adanya kontrol terpusat pada SDWN memberikan kemudahan pada administrator jaringan untuk mengelola, melakukan manajemen trafik dalam jaringan nirkabel, dan mampu memberikan solusi yang menjadi keterbatasan

jaringan tradisional untuk permasalahan-permasalahan jaringan yang sekarang seperti sulitnya mengintegrasikan teknologi baru karena alasan perbedaan perangkat atau platform, kinerja yang buruk karena ada beberapa operasi yang berlebihan pada *protocol layer* dan sulitnya menyediakan layanan-layanan baru (Mininet, 2014).

Pada penelitian sebelumnya "Perancangan dan Implementasi Jaringan *Wi-Fi* berbasis OpenFlow" (Fauzi, 2016) sudah menggunakan jaringan Wi-Fi berbasis OpenFlow Protokol dengan memanfaatkan perangkat fisik yaitu PC yang digunakan sebagai *controller* dan *virtual switch*, juga penggunaan *wireless router* yang di-*set* sebagai *access point.* Namun hanya terbatas pada *single controller* atau *centralized controller* dan pemanfaatan perangkat fisik PC dan 1 *host*.

Oleh karena itu pada penelitian ini akan dibahas mengenai implementasi jaringan nirkabel yaitu *Wi-Fi* pada SDN menggunakan beberapa *ONOS* sebagai *controller* yang berada dalam cluster atau *distributed controller* dan Open vSwitch dan diinstalasi pada *virtual machine*(*VM*), sebagai solusi untuk manajemen dan pengelolaan jaringan yang lebih mudah, meningkatkan kinerja serta fleksibilitas jaringan, dan menguji performansi Open vSwitch pada *VM* dalam lingkup jaringan nirkabel.

1.2 Rumusan Masalah

Berdasarkan latar belakang yang disajikan, rumusan masalah yang menjadi fokus penelitian adalah :

1. Bagaimana implementasi jaringan Wi-Fi pada *Software-Defined networking*?

2. Bagaimana analisis kinerja jaringan Wi-Fi pada *Software-Defined networking*?

1.3 Ruang Lingkup Penelitian

Ruang Lingkup penelitian ini adalah:

1. Penerapan konsep SDN dengan menghubungkan ke jaringan di laboratorium Jaringan Komputer.
2. Optimasi penggunaan *controller* yaitu dengan memanfaatkan *clustering*.
3. Menganalisis kinerja SDN melalui jaringan *wireless* dengan mengukur beberapa parameter QOS yaitu *throughput*, packet loss, dan latency.

1.4 Tujuan Penelitian

Berdasarkan latar belakang yang di uraikan, maka tujuan dari penelitian ini adalah sebagai berikut:

1. Implementasi jaringan *Wi-Fi* dengan menggunakan konsep SDN.
2. Menganalisis kinerja jaringan *Wi-Fi* dengan konsep SDN.

1.5 Manfaat Penelitian

Manfaat dari penelitian ini yaitu :

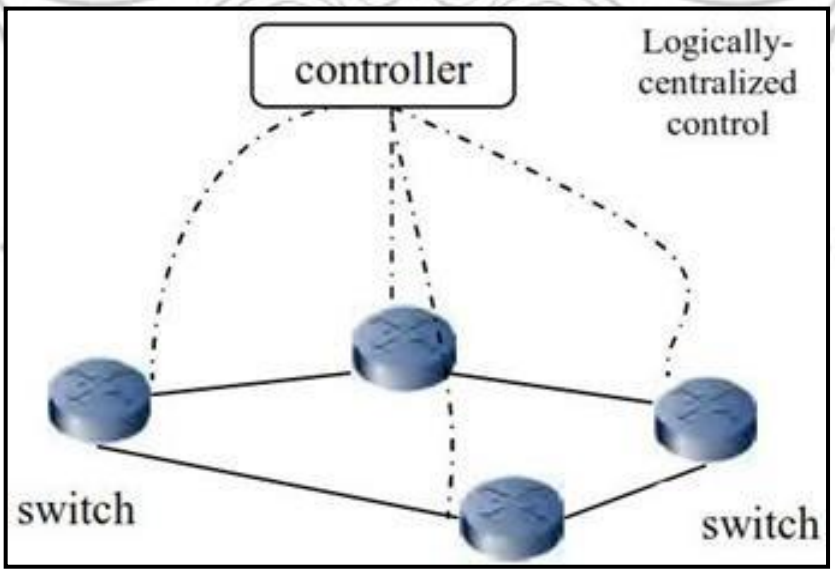
1. Memperdalam pengetahuan mengenai konsep jaringan *Wi-Fi* pada SDN dengan menerapkan *distributed controller.*

2. Meningkatkan kinerja jaringan *Wi-Fi* dengan SDN.

BAB II TINJAUAN PUSTAKA

2.1 Software-Defined Networking

Software Defined Networking (SDN) merupakan pendekatan arsitektur jaringan komputer yang memisahkan *control plane* dari sebuah perangkat jaringan komputer (*switch* atau *router*) dengan *data plane* perangkat jaringan komputer tersebut. Pemisahan *data-plane* dan *control-plane* ini memungkinkan untuk memprogram perangkat tersebut sesuai dengan yang diinginkan secara terpusat (SDN *Controller*), sehingga hal ini memungkinkan untuk mengontrol, memonitor, dan mengatur sebuah jaringan komputer dari sebuah titik (*node*) terpusat tersebut (Satasiya & Raviya Rupal, 2016). Arsitektur *Logical-SDN* dengan control terpusat ditunjukkan pada Gambar. 2.1.



Gambar 2. 1 *Logically-centralized control*
Sumber : Satasiya & Raviya Rupal, 2016

Pada teknologi jaringan konvensional, sistem pembuat keputusan ke mana arus data dikirimkan dibuat menyatu dengan perangkat kerasnya. Contoh fungsi

dan penerapan dari control plane dan data plane pada jaringan tradisional dapat dilihat pada tabel 2.1

Tabel 2. 1 Fungsi *Control Plane* dan *Data Plane*

Control Plane	**Data Plane**
a. Data konfigurasi	a. Forwarding Information Base (FIB)
b. Protokol routing Unicast	b. Access List
c. Protokol routing multicast	c. Quality of Service
d. Routing Information Base (RIB)	d. Interface

Sumber : (Marschke dkk., 2015)

Namun di dalam teknologi SDN memiliki dua karakteristik: pertama, SDN memisahkan antara *control plane* dari *data plane*, kedua, SDN menggabungkan *control plane* setiap perangkat menjadi sebuah controller yang berbasiskan *programmeable software*, sehingga sebuah *controller* tersebut dapat mengontrol banyak perangkat dalam sebuah *data plane*. Di dalam SDN sebuah jaringan tersentralisasi dalam sebuah *controller* yang berbasiskan *software* dapat memelihara jaringan secara keseluruhan, sehingga dapat mempermudah dalam mendesain dan mengoperasikan jaringan karena hanya melalui sebuah *logical point* (Agarwal, 2013).

SDN adalah cara untuk merancang dan mengelola jaringan. SDN memiliki kontrol terpisah dan logika data, sehingga dapat mudah untuk mengelola arsitektur jaringan dalam skenario berdasarkan *virtual*. Keputusan kontrol dinamis untuk menangani lalu lintas memungkinkan *switch* tanpa menimbulkan biaya *overhead* pemeliharaan. SDN merupakan arsitektur futuristik yang kuat, mudah untuk mengelola, lebih murah dan fleksibel (Satasiya & Raviya Rupal, 2016).

SDN memerlukan beberapa metode agar *control-plane* dapat berkomunikasi dengan *data-plane*. Salah satu mekanisme tersebut adalah *OpenFlow* yang sering

disetarakan dengan SDN. *OpenFlow* adalah sebuah protokol yang memungkinkan pengaturan penyaluran dan pengiriman paket ketika melalui sebuah *switch*. Dalam sebuah jaringan konvensional, setiap *switch* hanya berfungsi meneruskan paket yang lewat ke *port* yang sesuai tanpa dapat membedakan tipe protokol data yang dikirimkan misalnya *elastic* atau *inelastic traffic*. *OpenFlow* tidak hanya dapat melakukan *flow forwarding* berbasis *network layer* tetapi juga dapat dilakukan pengaturan pergerakan paket secara terpusat mulai dari *layer* 2 sampai *layer* 7 *forwarding* (*flow granuality*), sehingga aliran paket di jaringan dapat diprogram secara independen. Hal ini dapat dilakukan dengan membuat algoritma dan *forwarding rules*-nya pada *controller* kemudian aturan tersebut didistribusikan ke *switch* yang ada di jaringan. Terdapat beberapa *OpenFlow controller* yang dapat digunakan seperti *NOX (C base)*, POX *(phyton base)*, dan *ONOS (java base)* (Azodolmolky, 2013).

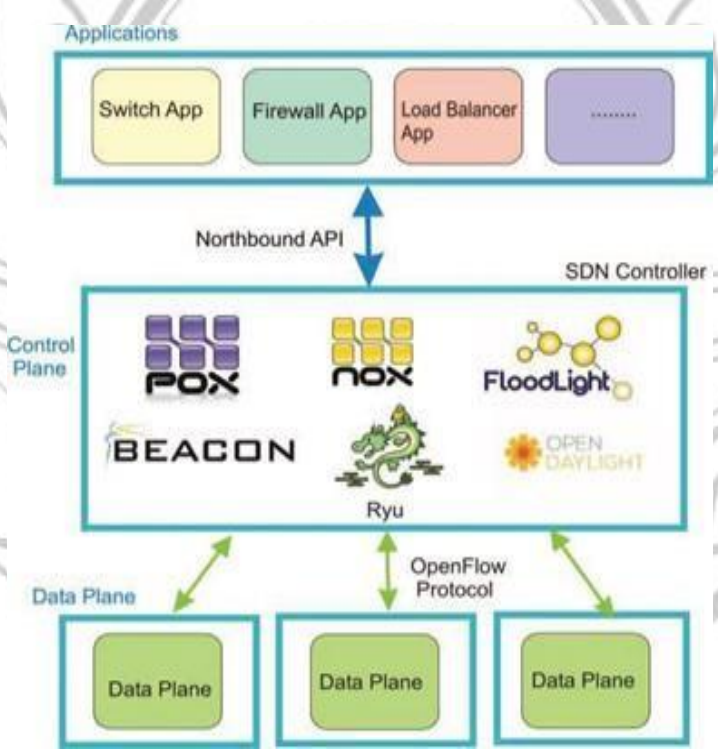
2.1.1 Arsitektur SDN

Data Plane, Control Plane dan aplikasi SDN adalah tiga komponen arsitektur SDN. *Data Plane* terdiri dari *switch* fisik atau virtual berbasis *OpenFlow*. *Decoupling* telah memungkinkan *control plane* untuk berkembang secara mandiri sehingga siklus evolusi singkat. Hal ini menyebabkan pengembangan berbagai teknologi kontrol. Siklus evolusi *forwarding plane* ditujukan untuk pengiriman paket cepat. SDN memisahkan *forwarding* dan *control plane. Control plane* terpisahkan disebut SDN *controller* (Shalimov dkk., 2013)

Northbound interface memungkinkan aplikasi untuk berinteraksi dengan *controller*. *Northbound interface* digunakan untuk membuat *Firewall*, *Load Balancer*, aplikasi NAT. SDN pengontrol berinteraksi dengan *switch OpenFlow* menggunakan *southbound interface*. *OpenFlow* adalah *southbound interface* paling populer. Komponen utama dari SDN adalah *southbound interface controller*, *northbound interface*, *forwarding* perangkat dan manajemen aplikasi. SDN memungkinkan pemanfaatan yang lebih baik dari sumber daya jaringan (Kaur dkk., 2014).Arsitektur SDN terdiri dari tiga layer yaitu sebagai berikut, seperti tampak pada Gambar 2.2 :

1) *Application Plane*: berada pada lapisan teratas, berupa aplikasi yang dapat secara langsung dan eksplisit mendefinisikan *network requirement* dan *network behavior* yang diinginkan. *Layer* ini berkomunikasi dengan *Control Layer* melalui *NorthBound Interface* (NBI).
2) *Controller Plane*: yaitu entitas kontrol yang memiliki tugas yaitu mentranslasikan *network requirement* yang telah didefinisikan oleh *Application Layer* menjadi instruksi-instruksi yang sesuai untuk *Infrastructure Layer*, dan memberikan *abstract view* yang dibutuhkan bagi *Application Layer* (*abstract view* meliputi informasi statistik dan *event* yang terjadi di jaringan).
3) *Data Plane*: terdiri dari elemen jaringan yang dapat menerima instruksi dari *Control Layer*. *Interface* antara *Controller Plane* dan *Data Plane* disebut *SouthBound Interface* (SBI), atau *Control-To-Data-Plane Interface* (CDPI).

Control Plane merupakan otak dari jaringan SDN, dapat dijalankan secara terpisah dari *data plane*. Sedangkan *data plane* merupakan perangkat-perangkat keras jaringan yang terprogram secara khusus dan dikendalikan penuh oleh *Control Plane*. Pada SDN tersedia *open interface* yang memungkinkan sebuah entitas *software* atau aplikasi untuk mengendalikan konektivitas dari sumber daya jaringan, mengendalikan aliran *traffic*, serta melakukan inspeksi atau memodifikasi *traffic* tersebut (Lara dkk., 2013) dapat dilihat pada gambar 2.2.



Gambar 2. 2 Arsitektur SDN

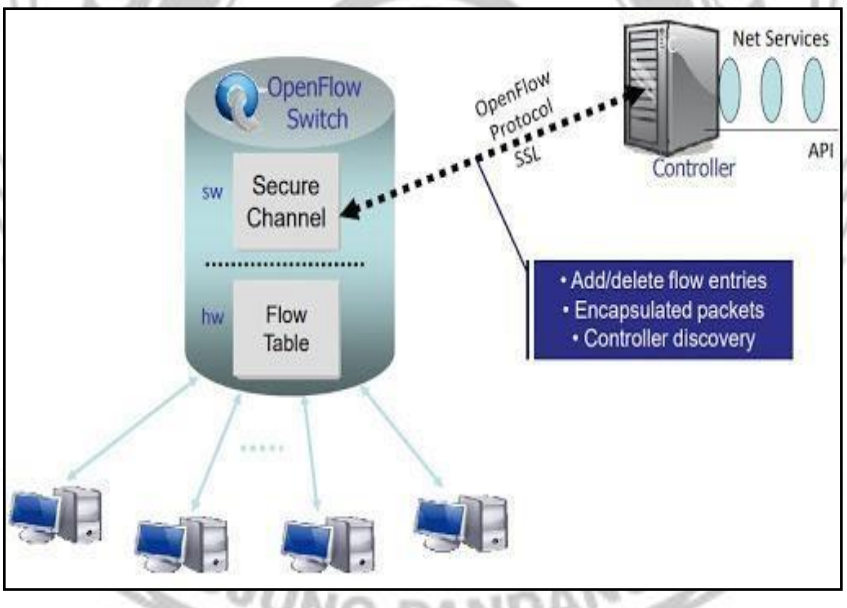
Sumber : (Kaur dkk., 2014)

2.2 OpenFlow *Protocol*

OpenFlow adalah protokol yang memungkinkan server memberitahukan switch jaringan tempat mengirim paket. Dalam jaringan konvensional, setiap switch

memiliki perangkat lunak berpemilik yang memberitahukan apa yang harus dilakukan (Rouse, 2012).

Openflow mendefinisikan infrastruktur *flow-based forwarding* dan *Application Programmatic Interface* (API) standart yang memungkinkan *controller* untuk mengarahkan fungsi dari *switch* melalui saluran yang aman (*secure channel)*(Patel dkk., 2013). Fungsi OpenFlow sebagai penghubung antara *controller* yaitu termasuk dalam *control plane* dengan *data plane* melewati *secure channel* lalu ke *flow table* dan diteruskan ke *user*. Hal ini dilihat pada Gambar 2.3.



Gambar 2. 3 *OpenFlow Protocol*

Sumber : (Rouse, 2012)

2.3 Open vSwitch (OvS)

Open vSwitch (OvS) adalah *multilayer virtual swicth* dilisensikan dibawah *open source* lisensi Apache 2.0. Hal ini dirancang untuk memungkinkan otomatisasi jaringan besar melalui ekstensi program,

sementara masih mendukung antarmuka manajemen standar dan protokol. Selain itu, Open vSwitch dirancang untuk mendukung distribusi di beberapa server fisik mirip dengan VMware's vNetwork atau vswitch or Cisco's Nexus 1000V (Graf, 2013). OvS dirancang untuk menjadi fleksibel, portabel dan berada dalam *hypervisor* atau manajemen domain, untuk menyediakan konektivitas antara mesin *virtual* dan *interface* fisik.

Hal ini dapat beroperasi sebagai dasar L2 *switch* dalam konfigurasi *standalone*, mendukung *VLAN*, *SPAN*, *RSPAN*, *ACL*, kebijakan QoS, *port bonding*, *trunking*, GRE dan IPsec, *tunneling*, dan per-VM *traffic policing*. Arus visibilitas dengan NetFlow dan sFlow juga disediakan. Untuk mendukung integrasi ke lingkungan *virtual*, OVS mengeluarkan *interface* untuk memanipulasi *forwarding state* dan mengelola konfigurasi pada saat *run-time*, yang memungkinkan spesifikasi tentang bagaimana paket ditangani berdasarkan L2, L3, dan *header* L4 (Sans & Gamess, 2013).

2.4 ONOS *Controller*

ONOS atau *Open Network Operation System* adalah sebuah sistem operasi (OS) yang dirancang untuk membantu penyedia layanan jaringan membangun jaringan berbasis *carrier-grade* yang dirancang untuk skalabilitas, ketersediaan dan kinerja tinggi. Meskipun dirancang khusus untuk memenuhi kebutuhan penyedia layanan, ONOS juga dapat bertindak sebagai pesawat kontrol SDN untuk jaringan area lokal (LAN) dan jaringan pusat data (Rouse, 2017).

2.5 *Quality of Service* (QoS) Jaringan

Quality of Service (QoS), sebagaimana dijelaskan dalam rekomendasi CCITT E.800 adalah efek kolektif dari kinerja layanan yang menentukan derajat kepuasan seorang pengguna terhadap suatu layanan. *Quality of Service* (QoS): "*the collective effect of service performance which determines the degree of satisfaction of a user of the service*". *International Telecommunication Union* (ITU 1998, X.641).

QoS didesain untuk membantu *end user (client)* menjadi lebih produktif dengan memastikan bahwa user mendapatkan performansi yang handal dari aplikasi-aplikasi berbasis jaringan. QoS mengacu pada kemampuan jaringan untuk menyediakan layanan yang lebih baik pada trafik jaringan tertentu melalui teknologi yang berbeda-beda. QoS merupakan suatu tantangan yang besar dalam jaringan berbasis IP dan internet secara keseluruhan. Tujuan dari QoS adalah untuk memenuhi kebutuhan-kebutuhan layanan yang berbeda, yang menggunakan infrastruktur yang sama. QoS menawarkan kemampuan untuk mendefinisikan atribut-atribut layanan yang disediakan, baik secara kualitatif maupun kuantitatif (Flannagan dkk., 2003)

Fungsi-fungsi QoS dijelaskan sebagai berikut (Flannagan dkk., 2003):

1) Pengkelasan paket untuk menyediakan pelayanan yang berbeda-beda untuk kelas paket yang berbeda-beda.
2) Penanganan kongesti untuk memenuhi dan menangani kebutuhan layanan yang berbeda- beda.

3) Pengendalian lalu lintas paket untuk membatasi dan mengendalikan pengiriman paket- paket data.

4) Pensinyalan untuk mengendalikan fungsi-fungsi perangkat yang mendukung komunikasi di dalam jaringan IP.

Tabel QoS seperti di bawah ini:

Tabel 2. 2 Indeks parameter QoS

Nilai	Persentase (%)	Indeks
3.8-4	95-100	Sangat memuaskan
3-3.79	75-94.75	Memuaskan
2-2.99	50-74.75	Kurang memuaskan
1-1.99	20-49.75	Buruk

Untuk mengukur nilai QoS secara keseluruhan dalam penelitian ini digunakan nilai rata-rata indeks ketiga parameter yang diukur :

$$QoS = \frac{\text{Throughput + Latency + Packet Loss}}{\text{Jumlah parameter}} \quad (2.1)$$

Ada beberapa parameter QoS yang dapat digunakan untuk mengukur kinerja jaringan antara lain :

2.5.1 *Throughput*

Throughput adalah jumlah data jaringan yang mengirim atau menerima data, atau jumlah data yang diproses dalam satu ruang waktu yang ditentukan. Ini memiliki sebagai unit dasar dari tindakan bit per detik (bit /s atau bps) (Gouveia & Magedanz, 2011). *Throughput* adalah laju data aktual per satuan waktu. *Throughput* bisa disebut sebagai *Bandwidth* dalam kondisi yang sebenarnya. *Bandwidth* lebih bersifat tetap, sementara *Throughput* sifatnya dinamis tergantung trafik yang

sedang terjadi. *Throughput* mempunyai satuan Bps (Bits per second) (Flannagan dkk., 2003). Detail kategori throughput ditunjukkan pada Tabel 2.3.

Tabel 2. 3 Kategori *Throughput*

Kategori *Throughput*	*Throughput*	Indeks
Sangat Baik	>2.1 mbps	4
Baik	1,200 kbps – 2.1 mbps	3
Cukup	700 – 1,200 kbps	2
Kurang Baik	338 – 700 kbps	1
Buruk	0 – 338 kbps	0

Sumber: (Tiphon, 2002)

Untuk mengukur nilai *Throughput* digunakan :

$$Throughput = \frac{\text{Ukuran data yang diterima}}{\text{Waktu pengiriman data}} \quad (2.2)$$

2.5.2 Packet Loss

Packet Loss merupakan suatu parameter yang menggambarkan suatu kondisi yang menunjukkan jumlah total paket yang hilang, dapat terjadi karena *collision* dan *congestion* pada jaringan. Pada Tabel 2.4 ditunjukkan nilai indeks dan kategori *Packet Loss*.

Tabel 2. 4 Kategori Packet Loss

Kategori *Packet Loss*	***Packet Loss (%)***	**Indeks**
Sangat Baik	0% - 2%	4
Baik	3% - 14%	3
Sedang	15% - 24%	2
Buruk	>25 %	1

Sumber : (Tiphon, 2002)

Untuk mengukur nilai *Packet Loss* digunakan :

$$Packet\ Loss = \frac{(\text{Paket data dikirim-Paket data diterima}) \times 100\%}{\text{Paket data yang dikirim}} \quad (2.3)$$

2.5.3 Latency

Latency adalah waktu yang dibutuhkan paket untuk mencapai tujuan, karena adanya antrian, atau mengambil rute yang lain untuk menghindari kemacetan. *Latency* adalah waktu antara ketika perangkat menerima sebuah *frame* dan ketika *frame* yang diteruskan keluar dari *port* tujuan, *serialization latency* adalah waktu yang dibutuhkan untuk benar-benar mengirimkan paket atau *frame* , dan *end to-end* (Gouveia & Magedanz, 2011). Detail kategori latency ditunjukkan pada tabel 2.5.

Tabel 2. 5 Kategori dari Latency

Kategori Latency	Besar latency (ms)	Indeks
Sangat Baik	< 150 ms	4
Baik	150 ms s/d 300	3
Sedang	300 ms s/d 450	2
Buruk	> 450 ms	1

Sumber : (Tiphon, 2002)

Latency dapat dicari dengan membagi antara panjang paket (L, *packet length* (bit/s)) dibagi dengan link bandwith (R, link bandwith (bit/s)).

Untuk mengukur nilai *Latency* digunakan :

$$\text{Latency} = \frac{\text{Packet length}}{\text{Link bandwidth}} \quad \text{..... (2.4)}$$

Atau

$$\text{Rata-rata Latency} = \frac{\text{Total Latency}}{\text{Total paket yang diterima}} \quad \text{..... (2.5)}$$

2.6 Algoritma Konsensus RAFT

Untuk mempermudah pengertiannya maka dapat dibagi menjadi dua topik yang berbeda yaitu algoritma konsensus serta RAFT.

2.6.1 Algoritma Konsensus

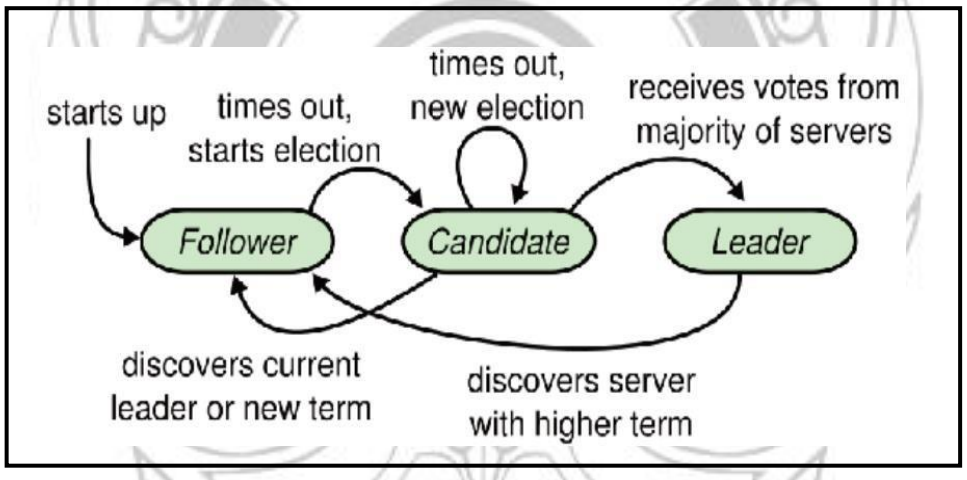
Problem utama dari *distributed computing* adalah menjaga *realibility* dari keseluruhan sistem dan mengurangi kesalahan proses. Untuk melakukan hal itu maka keseluruhan agent/node harus melakukan suatu persetujuan tertentu pada saat sistem berjalan. Contoh dari penggunaan konsensus adalah persetujuan atas identitas *leader, state machine replication, dan atomic broadcast*. (Dibaji & Ishii, 2015)

Dengan algoritma konsensus maka beberapa node tetap akan dapat berjalan dengan baik jika salah satu atau beberapa node mengalami kerusakan sistem, oleh karena itu penentuan identitas sebuah node menjadi sangat penting pada algoritma consensus. Beberapa *protocol consensus* yaitu *Phase King Algorithm*, *Lockstep protocol*, *Raft Consensus Algorithm*.

2.6.2 RAFT Consensus Algorithm

RAFT adalah sebuah *protocol* algoritma *consensus* yang mudah dimengerti. Pada dasarnya RAFT sama dengan algoritma *Paxos* dalam segi *fault-tolerance* dan performa, namun strukturnya berbeda dengan *Paxos* (Wang et al., 2019). Hal ini membuat RAFT lebih mudah dimengerti dan memudahkan untuk membangun implementasinya. Proses pemilihan *leader* dalam algoritma RAFT ditandai dengan perubahan status node sebagai *follower* berjalan hingga penentuan *leader* serta *follower*. Pada posisi awal, semua node mempunyai status sebagai *follower* dengan nilai variabel *timeout* masing-masing, ketika mencapai *timeout* maka status *follower* menjadi *candidate* dan mengoleksi *vote* dari semua node, jika ada beberapa node yang mempunyai jumlah *vote* yang sama maka proses sebelumnya

diulangi kembali dengan *random timeout* yang berbeda hingga terpilih satu node yang mempunyai *vote* tertinggi. Setelah itu node dengan *vote* tertinggi mengganti statusnya menjadi *leader* dan *node* lainnya sebagai *follower* (Ongaro et al., 2014). Proses pemilihan leader dalam algoritma RAFT ditunjukkan pada gambar 2.4. *Leader* ini bertanggung jawab untuk menerima dan permintaan *follower* dan mengelola replikasi log ke *server* lain. Dalam aliran data atau *heartbeat*-nya, hanya satu arah yaitu dari *leader* ke node lain. Terdapat tiga status node dalam algoritma RAFT yaitu, *follower, candidate,* dan *leader*.



Gambar 2. 4 Tahap Leader Election Algoritma RAFT

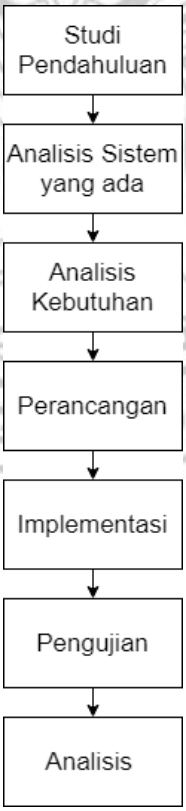
BAB III METODOLOGI PENELITIAN

3.1 Tempat dan Waktu

Penelitian ini dilaksanakan di Laboratorium Jaringan Komputer Program Studi Teknik Komputer dan Jaringan, Jurusan Teknik Elektro Politeknik Negeri Ujung Pandang. Jl. Perintis Kemerdekaan Km.10 Kota Makassar, Sulawesi Selatan. Penelitian ini dilakukan mulai bulan Agustus 2020 sampai dengan Agustus 2021.

3.2 Prosedur Penelitian

Tahapan prosedur penelitian diperlukan agar penelitian dapat terstruktur sehingga hasil yang diperoleh sesuai dengan tujuan penelitian. Gambar 3.1 merupakan gambaran alur proses penelitian.



Gambar 3. 1 Alur Prosedur Penelitian

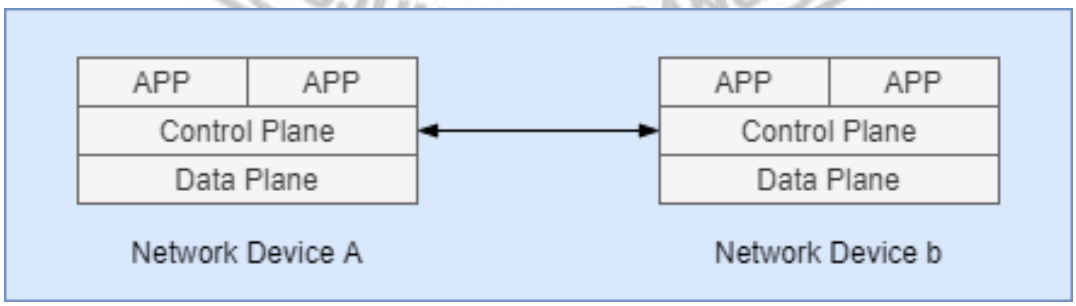
Berikut ini merupakan uraian dari setiap tahapan proses penelitian yang akan dilakukan :

3.2.1 Studi Pendahuluan

Studi pendahuluan dilakukan dengan menggunakan *Library Research* yang merupakan cara mengumpulkan data dari beberapa buku, jurnal, skripsi, tesis maupun literatur lainnya yang dapat dijadikan acuan pembahasan dalam masalah ini. Penelitian ini memiliki keterkaitan pada sumber-sumber data *online* atau internet ataupun hasil dari penelitian sebelumnya sebagai bahan referensi bagi peneliti selanjutnya.

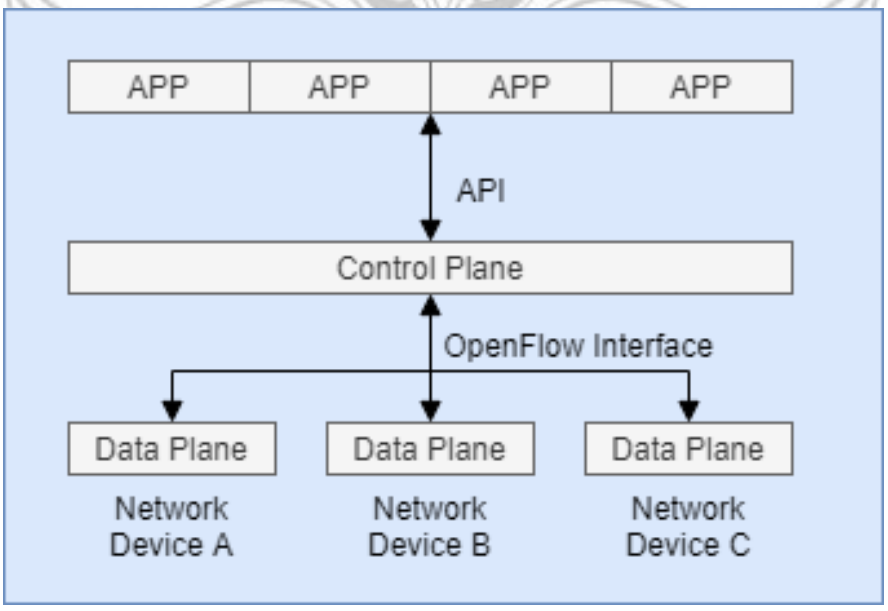
3.2.2 Analisis Sistem yang Ada

Pada tahap ini dilakukan proses penguraian dan pengidentifikasian terhadap sistem yang ada untuk membantu dalam proses penerapan *Software-Defined Networking*. Satu perangkat jaringan menggabungkan *control plane* dan *data plane* sehingga tidak efisien dan efektif dalam melakukan konfigurasi perangkat karena perlu dilakukan satu per satu, hal ini ditunjukkan pada gambar 3.2.



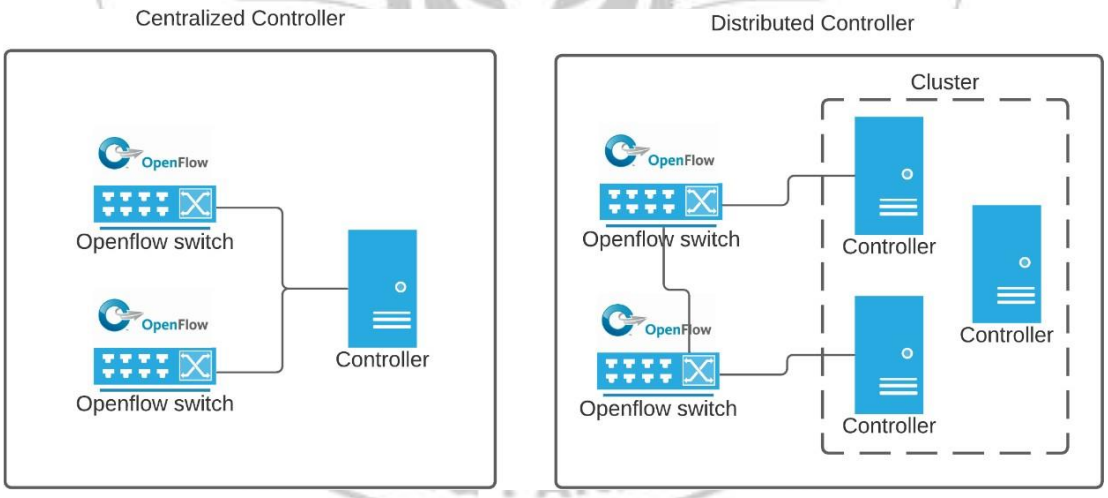
Gambar 3. 2 Kondisi sistem saat ini

Untuk memudahkan dalam manajemen perangkat dan jaringan secara efektif dapat ditangani dengan menggunakan *Software-Defined Networking.* yang memungkinkan untuk mengontrol jaringan lebih mudah, memungkinkan operator untuk menentukan tugas yang lebih kompleks yang melibatkan pengintegrasian banyak fungsi jaringan terpisah (misalnya, keamanan, kontrol sumber daya, prioritas) ke dalam kerangka kerja kontrol tunggal, yang memungkinkan operator jaringan untuk membuat kebijakan yang lebih canggih, dan membuat konfigurasi jaringan lebih mudah untuk mengkonfigurasi, mengelola, memecahkan masalah, dan men-*debug.* Pemanfaatan teknologi *Software-Defined Networking* mampu memisah *control plane* dan *forwarding plane* dalam suatu perangkat yang berbeda seperti yang ditunjukkan pada Gambar 3.3



Gambar 3. 3 Pemanfaatan teknologi *Software-Defined Networking*

Model arsitektur SDN yang digunakan berbeda dengan penelitian sebelumnya yang menggunakan model *centralized controller*, pada penelitian ini digunakan *distributed controller*. Pada *centralized controller*, satu *controller* ditunjuk sebagai pengontrol dan bertanggungjawab untuk mengelola eksekusi komponen lainnya. Keunggulan dari model *distributed controller* adalah dapat diterapkan *multi-instance controller* yang memiliki kemampuan untuk mengatasi masalah kegagalan yang terjadi pada satu *controller* dan untuk meningkatkan skalabilitas, dengan metode ini *controller* lain dapat menggantikan *controller* yang sedang *down* atau tidak aktif. Perbedaan antara model *centralized controller* dan *distributed controller* dapat dilihat pada gambar 3.4.



Gambar 3. 4 Perbandingan *Distributed Controller* dan *Centralized Controller*

3.2.3 Analisis Kebutuhan

Berdasarkan analisis sistem yang ada di Lab. Jaringan Komputer, maka dirincikan kebutuhan pada penelitian yang dilakukan meliputi kebutuhan fungsional, kebutuhan non-fungsional, dan kebutuhan penunjang penelitian.

1) Kebutuhan fungsional

Berikut adalah kondisi sistem yang dibutuhkan :

a) Perangkat yang disediakan mampu menciptakan lingkungan virtual yang memenuhi syarat ketersediaan dan fleksibilitas. Ketika terjadi perubahan *resource*, aplikasi tetap pada kondisi normal.

b) *Virtual machine* dapat menggunakan *resource* yang disediakan oleh *controller* dan Open vSwitch

2) Kebutuhan perangkat

Kebutuhan perangkat yang digunakan pada penelitian dibagi menjadi dua jenis yaitu perangkat keras dan perangkat lunak.

Tabel 3. 1 Kebutuhan Perangkat Keras

No	Perangkat keras	Spesifikasi	Jumlah	Deskripsi
1.	*Personal Computer* (PC)	a. *Processor* Intel core i5-4460 b. CPU 3.20 GHz c. RAM 16 GB d. HDD 100 GB	1	Digunakan sebagai *server* proxmox
2	*Access Point* Unifi ubiquiti	a. Standard 802.11 ac b. Kecepatan sampai 867 Mb/s	2	Digunakan sebagai *Access Point*
3	Switch D-Link DES-1016D		1	Digunakan sebagai *switch* penghubung ke perangkat *server*

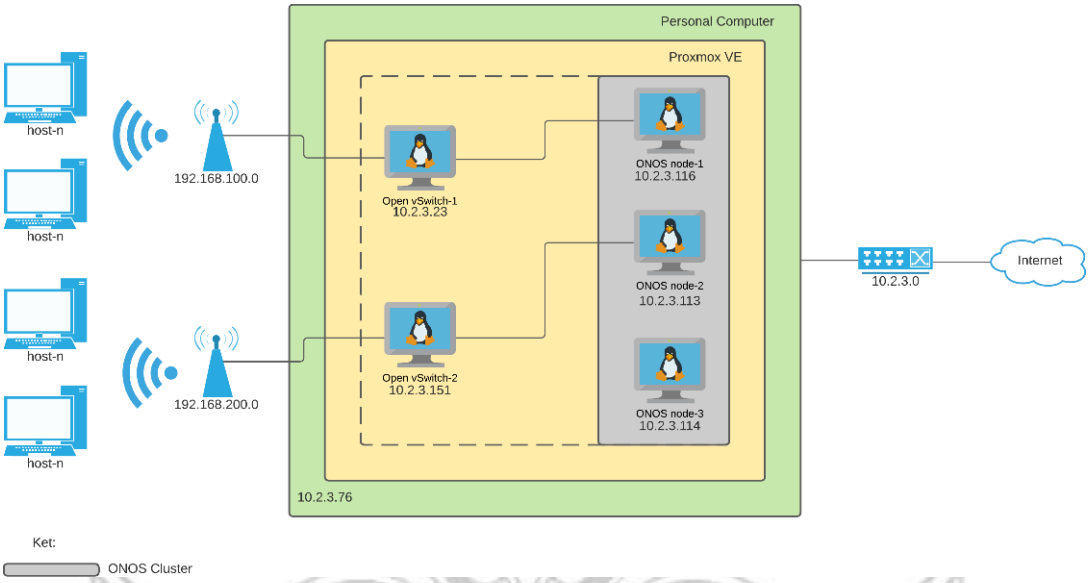
Tabel 3. 2 Kebutuhan Perangkat Lunak

No	Perangkat Lunak	Deskripsi
1	Proxmox	Versi 6.1
2	Ubuntu 16.04 LTS	Sistem operasi yang digunakan pada perangkat lunak lainnya.
3	ONOS *Controller*	Perangkat lunak ini berfungsi digunakan sebagai *controller* yang mengatur dan memberikan keputusan terhadap apa yang akan dilakukan oleh *switch*. Versi yang digunakan yaitu 1.13
4	Open vSwitch	Perangkat lunak ini berfungsi sebagai *switch* OpenFlow. *Switch* OpenFlow ini menggunakan OpenVSwitch versi 2.11.
5	Youtube	Platform ini berfungsi untuk mengambil video live streaming yang digunakan sebagai data pengujian.
6	Apache Jmeter	Perangkat lunak ini berfungsi sebagai *tools* untuk *load testing*. *Software* ini digunakan sebagai alat ukur kualitas jaringan *Software-Defined Networking* yang telah dibangun.

3.2.4 Perancangan

Pada tahap ini, kegiatan yang dilakukan adalah melakukan perancangan sistem. Penelitian ini merancang dan menerapkan konsep *Software-defined Networking* (SDN) dengan model *distributed* pada jaringan *Wi-Fi* menggunakan *protocol* OpenFlow. SDN tidak hanya dapat melakukan *flow forwarding* tetapi juga dapat melakukan pengaturan pergerakan paket secara distribusi. Hal ini yang diterapkan pada jaringan *Wi-Fi*, sehingga pergerakan paket dapat diatur secara independen dalam satu buah *controller* yang bertugas untuk mengatur *traffic*. Perancangan ini dilakukan dengan menggunakan perangkat keras yang ada di

laboratorium Jaringan Komputer. Gambaran topologi yang digunakan dalam membangun jaringan Wi-Fi pada SDN dapat dilihat pada gambar 3.5.



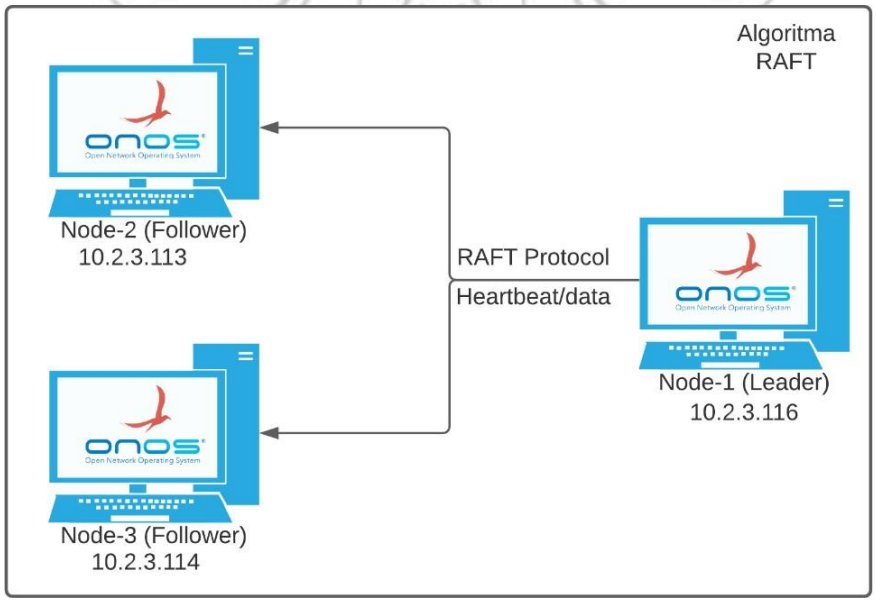
Gambar 3. 5 Rancangan topologi

Perancangan dilakukan dengan menggunakan beberapa perangkat yang berada dalam Lab Jaringan Komputer. Dalam membuat jaringan *Wi-Fi* dengan konsep SDN, dibutuhkan 2 perangkat *Access Point* dan satu buah PC dijadikan sebagai server dengan proxmox VE sebagai hypervisor yang mana di dalamnya terdapat 5 *virtual machine (VM)*, yaitu 2 mesin *virtual* untuk open vSwitch dan 3 untuk *controller* yang berada dalam satu *cluster*. Setiap perangkat pada topologi memiliki *IP address* tertentu, dan pengalamatan tersebut dibutuhkan agar setiap *interface* dapat saling berkomunikasi.

Tabel 3. 3 *IP Address*

Perangkat	Interface	IP Address	Ket.
Server Proxmox	vmbr1 (enp2s0)	10.2.3.76/24	
	vmbr2 (enx00e04c534458)		
	vmbr0 (enp3s0)		
ONOS node-1	vmbr1 (ens18)	10.2.3.116/24	
	vmbr0 (ens19)	192.168.100.101/24	
ONOS node-2	vmbr1 (ens18)	10.2.3.114/24	
	vmbr2 (ens19)	192.168.200/24	
ONOS node-3	vmbr1 (ens18)	10.2.3.113/24	
VM ovs-1	vmbr1 (ens18)	10.2.3.23/24	
	br0 (ens19)	192.168.100.1/24	AP 1
VM ovs-2	vmbr1 (ens18)	10.2.3.151/24	
	br1 (ens19)	192.168.200.1/24	AP 2
Host 1		192.168.100.126/24	
Host 2		192.168.100.124/24	
Host 3		192.168.200.106/24	
Host 4		192.168.200.105/24	

Adapun cluster ONOS *controller* dapat dilihat pada gambar 3.6.



Gambar 3. 6 Cluster ONOS controller

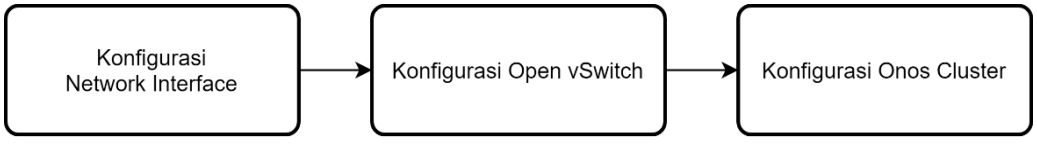
Dalam cluster ONOS, terdapat algoritma RAFT yang mengatur pemilihan *leader*. Pada penelitian ini, ketika semua node dijalankan secara bersamaan, awalnya berstatus sebagai *follower*, namun node-1 dengan alamat IP 10.2.3.116 terpilih menjadi *leader*, hal ini dikarenakan node-1 memiliki waktu *timeout* tercepat sehingga dapat menjadi *leader* dalam *cluster*.

3.2.5 Implementasi

Penelitian ini bersifat testbed yang diterapkan pada jaringan internet yang ada di Laboratorium Jaringan Komputer. Hal ini dilakukan untuk melihat kinerja dari Open vSwitch dan ONOS *Controller* pada implementasi SDN pada jaringan *Wi-Fi*. Berdasarkan topologi yang telah dibuat, kebutuhan pada penelitian ini di antaranya instalasi perangkat lunak dan aplikasi yang dibutuhkan dalam proses penelitian ini serta dilakukan pemasangan perangkat jaringan yang digunakan.

Gambar 3. 7 Tahap instalasi perangkat lunak dan aplikasi

Setelah proses instalasi, dilakukan konfigurasi untuk menyesuaikan dengan skenario dan kebutuhan dalam penelitian ini.



Gambar 3. 8 Tahap konfigurasi sistem

Berdasarkan Gambar 3.7 dan 3.8 tahap instalasi dan konfigurasi sistem dijelaskan langkah-langkah di bawah ini.

1) Instalasi

   a) Instalasi Proxmox VE

      Prosmox VE di-*install* untuk membuat mesin virtual. Di dalam Proxmox VE, dibuat 5 mesin virtual, masing-masing untuk Open vSwitch dan ONOS Controller.

   b) Instalasi Ubuntu 16.04 LTS dan Open vSwitch

      Open vSwitch di-*install* pada sistem operasi Linux. Linux yang digunakan adalah Ubuntu 16.04 LTS. Open vSwitch dapat menggunakan segala program yang berjalan pada Linux. Proses ini dilakukan sebanyak 2 kali, karena diperlukan 2 Open vSwitch. Pada penelitian ini, switch OpenFlow yang digunakan adalah PC yang di-*install* menggunakan Open vSwitch versi 2.11.1. Penggunaan software ini mampu menjalankan fungsi-fungsi *switch* OpenFlow yang berbentuk fisik. Perbedaan mendasar antara switch OpenFlow *software-based* dan *switch* OpenFlow *hardware-based* adalah skala penggunaan terhadap perangkat yang terhubung. Switch OpenFlow dengan *hardware-based* dapat menghubungkan dengan banyak perangkat jaringan dalam satu perangkat switch. Switch OpenFlow dalam bentuk software dipilih sebagai pengujian untuk melihat QoS yang dihasilkan switch tersebut. Penggunaan switch jenis ini bertujuan memudahkan dalam mengimplemtasikan protokol OpenFlow tanpa harus menggunakan *switch* OpenFlow *hardware-based.*

c) Instalasi Ubuntu 16.04 LTS dan ONOS *Controller*

ONOS yang berfungsi sebagai *controller* juga diinstall pada sistem operasi linux. Linux yang digunakan adalah Ubuntu 16.04 LTS. ONOS di-*install* agar menjadi *control plane* pada sistem dan mampu berkomunikasi dengan Open vSwitch agar *protocol* OpenFlow dapat digunakan pada S*oftware-defined* N*etworking*. ONOS yang digunakan adalah ONOS dengan versi 1.13 atau disebut dengan *nightingale*.

2) Konfigurasi

a) Konfigurasi *Network Interface*

Konfigurasi *network interface* dilakukan untuk menghubungkan seluruh *interface* yang ada pada jaringan agar dapat saling berkomunikasi. *IP Address* yang digunakan terlihat seperti pada tabel 3.3

b) Konfigurasi Open vSwitch

Konfigurasi Open vSwitch dilakukan untuk menghubungkan Open vSwitch dengan jaringan *Access Point* yang dibuat agar *client* dapat menggunakan jaringan tersebut dan saling berkomunikasi. Pada konfigurasi ini diperlukan *server DHCP*, yang diinstall dan dikonfigurasi pada mesin, melakukan setting NAT melalui perintah *iptables* dan mengaktifkan *IP forwarding*. Konfigurasi yang dilakukan untuk membuat switch OpenFlow adalah dengan membuat *bridge* dengan nama br0. Dalam *bridge* ini terdapat eth0, eth1, dan eth2. Ethernet dibuat dalam satu bridge untuk memudahkan dalam pengalamatan IP *address*. *Bridge* tersebut diberi IP address secara *static* yaitu 192.168.100.101/24.

c) Konfigurasi *ONOS Cluster*

Konfigurasi *ONOS* Cluster dilakukan agar ketiga ONOS *controller* yang di-*install* pada masing-masing mesin virtual berada dalam satu cluster.

3.2.6 Pengujian

Pada tahap ini dilakukan pengujian dari sistem yang telah dibangun. Pengujian dilakukan untuk mengukur apakah suatu sistem dapat berjalan dengan baik dan tepat atau perlu diperbaiki atas kesalahan yang terjadi pada sistem. Dari hasil pengujian sistem dilakukan analisis terhadap data yang didapatkan sehingga diperoleh informasi yang dapat dievaluasi agar sesuai dengan tujuan yang ingin dicapai. Beberapa tahap pengujian yang dilakukan yaitu :

1) Pengujian perangkat jaringan

   Pengujian dilakukan dengan memastikan perangkat jaringan berjalan sesuai standar atau tidak. Pada pengujian ini dilakukan pengujian untuk ONOS *cluster* dalam melakukan pemilihan leader dengan mematikan node lain secara berkala.

2) Pengujian Video Streaming

   Pengujian dilakukan dengan mengakses *video streaming* menggunakan protocol *HTTP Live Streaming* (HLS). Pengujian ini dilakukan untuk mengukur *throughput*, *packet loss*, dan *latency*. Pengujian *video streaming* dilakukan dengan menggunakan *software* Apache Jmeter dengan mengatur *user*, resolusi video, dan lama waktu *video streaming*.

a) Spesifikasi pengujian QoS terhadap jumlah user

Jumlah user	Durasi video	Resolusi video	Max bandwidth
10-100	15 menit	1280x720	65 bps

b) Spesifikasi pengujian QoS terhadap ukuran *bandwidth*

Ukuran bandwidth	Jumlah user	Durasi video	Resolusi video
10-60 mbps	50	15 menit	1280x720

3.2.7 Analisis

Pada tahap ini dilakukan analisis terhadap kinerja hasil pengujian, selanjutnya dijadikan bahan dokumen produk akhir dalam penelitian ini. Masalah yang ditemukan dalam proses penelitian turut dijadikan sebagai data hasil penelitian untuk memudahkan pembaca dalam mengembangkan penelitian lebih lanjut :

1) Analisis Kinerja *Throughput*

Throughput diukur dengan mengamati jumlah total kedatangan paket yang sukses, yang diamati pada tujuan selama interval waktu tertentu dibagi oleh durasi *interval* waktu tersebut. Tabel 3.4 menunjukkan langkah-langkah identifikasi *throughput*. Hasil pengamatan ini divalidasi dengan persamaan *throughput* sesuai Persamaan 2.2.

Tabel 3. 4 Identifikasi *throughput* menggunakan Apache Jmeter

Langkah	**Proses**
Langkah 1	Menjalankan aplikasi Jmeter
Langkah 2	Membuat thread pada Test Plan, Test Plan > Add > Thread (Users) > Thread Group

(Lanjutan Tabel 3.4 pada halaman berikut)

Langkah	Proses
Langkah 3	Mengatur jumlah user, ramp-up period, loop count
Langkah 4	Setelah menambah dan mengatur thread group, tambah sampler, Klik kanan pada thread Group > Add > Sampler > bzm – Streaming sampler
Langkah 5	Mengatur ukuran bandwidth, resolusi video, lama video yang diakses
Langkah 6	Klik kanan pada "bzm – Streaming sampler", pilih add > listener > summary report
Langkah 7	Amati dan catat hasil pada tabel *throughput*

2) Analisis Kinerja Packet Loss

Packet Loss diukur dengan mengamati jumlah paket yang terkirim dan jumlah paket yang diterima. Tabel 3.5 menunjukkan langkah-langkah identifikasi *packet loss* di Apache Jmeter. Hasil pengamatan ini divalidasi dengan persamaan *packet loss* sesuai Persamaan 2.3.

Tabel 3. 5 Identifikasi paket loss menggunakan Apache Jmeter

Langkah	Proses
Langkah 1	Menjalankan aplikasi Jmeter
Langkah 2	Membuat thread pada Test Plan, Test Plan > Add > Thread (Users) > Thread Group
Langkah 3	Mengatur jumlah user, ramp-up period, loop count

(Lanjutan tabel 3.5 pada halaman berikut)

Langkah	Proses
Langkah 4	Setelah menambah dan mengatur thread group, tambah sampler, Klik kanan pada thread Group > Add > Sampler > bzm – Streaming sampler
Langkah 5	Mengatur ukuran bandwidth, resolusi video, lama video yang diakses
Langkah 6	Klik kanan pada "bzm – Streaming sampler", pilih add > listener > summary report
Langkah 7	Amati dan catat hasil pada tabel *packet loss*

3) Analisis Kinerja *latency*

Latency diukur dengan mengamati waktu yang dibutuhkan data untuk menempuh jarak dari asal ke tujuan. Tabel 3.6 menunjukkan langkah-langkah identifikasi *latency* di Apache Jmeter. Hasil pengamatan ini divalidasi dengan persamaan packet loss sesuai Persamaan 2.4.

Tabel 3. 6 Identifikasi latency menggunakan Apache Jmeter

Langkah	Proses
Langkah 1	Menjalankan aplikasi Jmeter
Langkah 2	Membuat thread pada Test Plan, Test Plan > Add > Thread (Users) > Thread Group
Langkah 3	Mengatur jumlah user, ramp-up period, loop count

(Lanjutan Tabel 3.6 pada halaman berikut)

Langkah	Proses
Langkah 4	Tambah sampler, Klik kanan pada thread Group >Add > Sampler > bzm – Streaming sampler
Langkah 5	Mengatur ukuran bandwidth, resolusi video, lama video yang diakses
Langkah 6	Klik kanan pada "bzm – Streaming sampler", pilih add > listener > view results in table
Langkah 7	Amati dan catat hasil pada tabel *latency*

BAB IV HASIL DAN PEMBAHASAN

Pada bab ini dibahas hasil yang diperoleh dari implementasi sistem dan hasil dari serangkaian proses pengambilan data yang sudah dibahas sebelumnya. Penelitian ini berfokus untuk membahas bagaimana kinerja jaringan *Wi-Fi* pada *Software-Defined Networking* dengan menggunakan ONOS *Controller* dan Open vSwitch. Pengujian dilakukan untuk mengetahui bahwa konfigurasi yang dilakukan dapat berjalan dengan baik berdasarkan gambaran tahapan pengujian sistem pada Bab III.

4.1 Hasil Pengujian Perangkat Jaringan

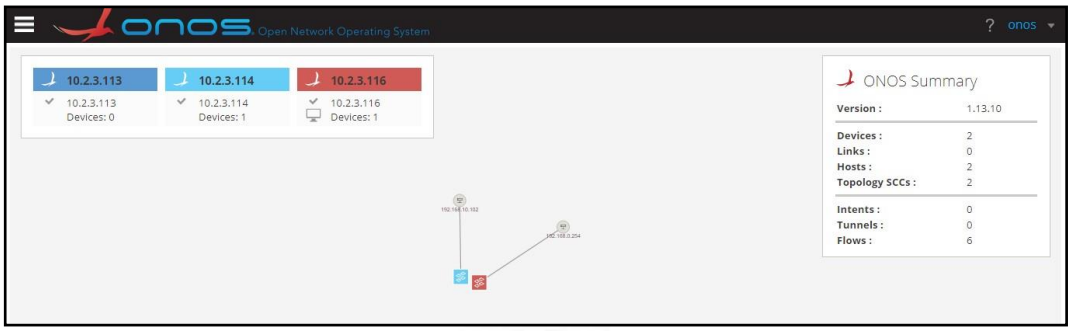
Penerapan Jaringan *Wi-Fi* pada *Software-Defined Networking* dilakukan melalui protokol OpenFlow dengan *Cluster* ONOS *controller* dan Open Switch.

4.1.1 ONOS Clustering

ONOS *clustering* diterapkan dalam jaringan *Wi-Fi* dengan konsep *Software-Defined Networking* melalui penginstallan ONOS *controller* versi 1.13. Untuk mengaktifkan ONOS *cluster*, dilakukan dengan mengakses onos-form-cluster yang terdapat pada direktori bin dan menjalankan *command* :

onos-form-cluster [nodeip1] [nodeip2] [nodeip3]

Setelah itu, alamat IP ONOS cluster dikonfigurasi dalam bentuk file json yang terdapat pada direktori /opt/ONOS-1.13.10/config.



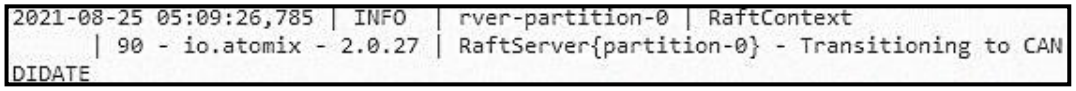
Gambar 4. 1 Tampilan web interface ONOS

Tampilan web *interface* ONOS *controller* dapat dilihat pada Gambar 4.1. Dalam mengcluster ONOS *controller* dibutuhkan file onos-cluster dan cluster.json yang berisi alamat IP ONOS yang di-*cluster*, yang dapat dilihat pada lampiran 8.

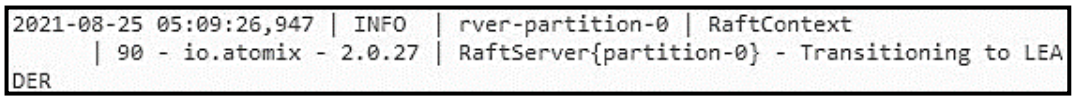
Dalam algoritma RAFT, semua node dalam *cluster* ketika diaktfikan memulai statusnya sebagai *follower*. Hal ini dapat dilihat pada log ONOS node, seperti yang ditunjukkan pada gambar 4.2 dimana node berstatus sebagai *follower*. Setiap node memiliki *timeout* masing-masing, node yang memiliki *timeout* tercepat akan menjadi *candidate* seperti yang ditunjukkan pada gambar 4.3. *Candidate* kemudian mengirim request vote pada node lain, jika node lain menjawab vote tersebut, maka node yang menjadi *candidate* akan terpilih menjadi leader seperti yang ditunjukkan pada gambar 4.4 dimana node berstatus sebagai *leader*. Jika *leader down* atau dalam keadaaan tidak aktif, maka akan dilakukan pemilihan *leader* kembali sesuai dengan informasi log replikasi yang paling lengkap.

```
2021-08-25 05:09:24,046 | INFO  | rver-partition-0 | RaftContext
      | 90 - io.atomix - 2.0.27 | RaftServer{partition-0} - Transitioning to FOL
LOWER
```

Gambar 4. 2 Node status follower



Gambar 4. 3 Node status candidate



Gambar 4. 4 Node status leader

Dalam penelitian ini khususnya dengan menggunakan ONOS *controller*, dicoba untuk mengamati dan menguji *leader election* algoritma konsensus RAFT dengan cara mematikan node *leader* dan melihat apakah *leader* baru akan terpilih. Pengujian ini dilakukan untuk mengetahui apakah sistem dapat melakukan pemilihan *leader* dengan baik dan tidak ada konflik. Konflik akan terjadi jika terdapat lebih dari satu *leader* pada satu sistem. Hasil pengujian *leader election* dapat dilihat pada Tabel 4.1.

Tabel 4. 1 Hasil uji *leader election*

No	Aksi	Status		
		ONOS node-1	ONOS node-2	ONOS node-3
1	Start up	leader	follower	follower
2	node-1 down	down	leader	follower
3	node-1 up	follower	leader	follower
4	node-2 down	leader	down	follower
5	Node-2 up	leader	follower	follower
6	Node-1 down	down	follower	leader
7	Node-3 down	down	candidate	down
8	Node-3 up	down	follower	leader
9	Node-2 down	down	down	candidate
10	Node-1 up, node-2 up	follower	follower	leader

Berdasarkan Tabel 4.1, proses pengujian dilakukan secara berurutan dengan melakukan aksi secara acak, berawal dari sistem start up dimana node-1 berlaku sebagai *leader* hingga aksi terakhir dimana posisi *leader* berpindah pada node-3. Dapat dilihat dari hasil pengujian bahwa setiap aksi yang dilakukan tidak terdapat lebih dari satu *leader*, hal ini telah sesuai dengan algoritma konsensus RAFT yang dalam satu sistemnya tidak bisa mempunyai dua leader.

4.1.2 Komunikasi dengan *protocol* OpenFlow

Setelah mengcluster ONOS, maka dapat diakses web interface ONOS dan mengaktifkan aplikasi *Openflow Provider Suite* yang merupakan rangkaian penyedia basis OpenFlow dari sisi *controller*. Setelah itu, Open vSwitch yang telah diinstall sudah dapat dihubungkan dengan salah satu member ONOS *controller*. ONOS *controller* yang telah di-*install* ditambahkan ke dalam network dengan cara menghubungkannya dengan Open vSwitch yang mengaktifkan *protocol OpenFlow* dari sisi *switch*. Perintah yang digunakan untuk menghubungkan *controller* dengan Open vSwitch:

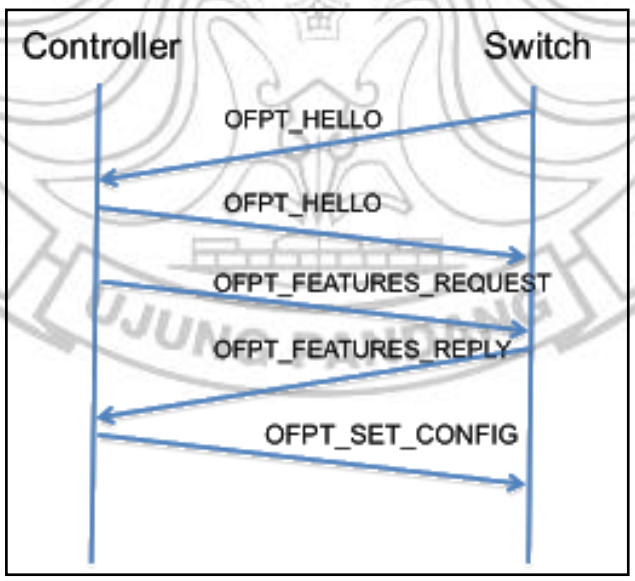
```
Ovs-vsctl set-controller br0 tcp:10.2.3.125:6633
```

Port 6633 adalah port protokol *OpenFlow* berada. Dengan aktifnya *protocol* ini, perangkat sudah bisa saling mengenali dan *controller* secara otomatis berkenalan dengan semua perangkat atau kondisi ini disebut sebagai *device provisioning system*. Kemudian *control plane* dan *data plane* terpisah secara otomatis oleh protocol OpenFlow yang memudahkan aktivitas konfigurasi dan pengaturan karena hanya perlu mengakses *controller*. Perangkat Open vSwitch

yang terhubung dengan *controller* dapat dilihat pada *web interface* seperti yang ditunjukkan pada Gambar 4.5.

Gambar 4. 5 Tampilan device ovs pada web interface ONOS

Interface yang menghubungkan setiap *OpenFlow Logical Switch* ke *OpenFlow controller* disebut OpenFlow channel. Melalui *interface* ini *controller* dapat saling berkomunikasi dengan *switch*. Komunikasi yang terjadi melibatkan protocol SDN yaitu *protocol* OpenFlow yang alur komunikasinya ditunjukkan pada Gambar 4.6.



Gambar 4. 6 Komunikasi Controller-Switch

Pertama, dengan *OpenFlow protocol* yang terdapat pada *switch* akan mengirimkan pesan *hello* kepada *controller*, yang juga akan dibalas pesan *hello* oleh *controller* seperti yang ditunjukkan pada Gambar 4.7. *Onos-of-worker* adalah *worker protocol openflow* yang terdapat pada ONOS, *worker* ini mengenali *switch OpenFlow* yang terhubung dengan *controller* dan mengirim balasan pesan Hello kepada *OpenFlow Switch* dengan alamat IP 10.2.3.151.

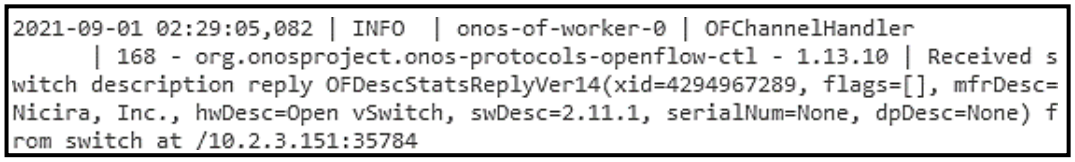
```
2021-09-01 02:29:05,022 | INFO  | onos-of-worker-0 | OFChannelHandler
      | 168 - org.onosproject.onos-protocols-openflow-ctl - 1.13.10 | Sending OF
 14 Hello to /10.2.3.151:35784
```

Gambar 4. 7 Controller mengirimkan pesan Hello

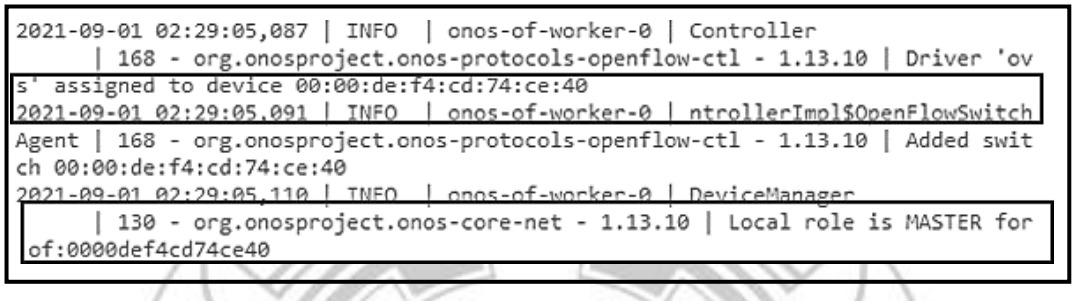
Selanjutnya *onos-of-worker controller* akan me-request *feature* atau *Feature Request* yang kemudian dibalas oleh *switch* openFlow sehingga *controller* menerima detail *feature* yang berupa *device id* Open vSwitch yang ditunjukkan pada Gambar 4.8. Setelah itu, *controller* menerima deksripsi *switch* berupa vendor *switch* yaitu Nicira, Inc. , jenis *OpenFlow switch* yang merupakan *Open vSwitch* dan versi switch yaitu 2.11.1. Setelah deskripsi switch diterima *controller* seperti yang ditunjukkan pada Gambar 4.9, *onos-of-worker* akan mengenali *switch* tersebut. Awalnya controller mengenali open vSwitch sebagai 'driver ovs', setelah mengenali *switch*, *switch* kemudian mendapatkan *id device* yaitu 00:00:de:f4:cd:74:ce:40 yang kemudian dijadikan sebagai role master dengan nama perangkat of:0000def4cd74ce40 seperti yang ditunjukkan pada Gambar 4.10.

```
2021-09-01 02:29:05,073 | INFO  | onos-of-worker-0 | OFChannelHandler
      | 168 - org.onosproject.onos-protocols-openflow-ctl - 1.13.10 | Received m
eter features from [/10.2.3.151:35784 DPID[00:00:de:f4:cd:74:ce:40]] with max me
ters: 0
```

Gambar 4. 8 Controller menerima *features*



Gambar 4. 9 Controller menerima deskripsi *switch*



Gambar 4. 10 Komunikasi ovs dan ONOS pada log ONOS

4.2 Hasil Pengujian *Video Streaming*

Pengujian *Video Streaming* bertujuan untuk mengetahui QoS Jaringan dengan mengukur tiga parameter yaitu *throughput*, *packet loss*, dan *latency*. Dalam pengujian ini digunakan *software* Apache Jmeter dengan 10 kali perulangan dan dilakukan skenario pengukuran QoS dengan jumlah *user* dan terhadap ukuran *bandwidth*.

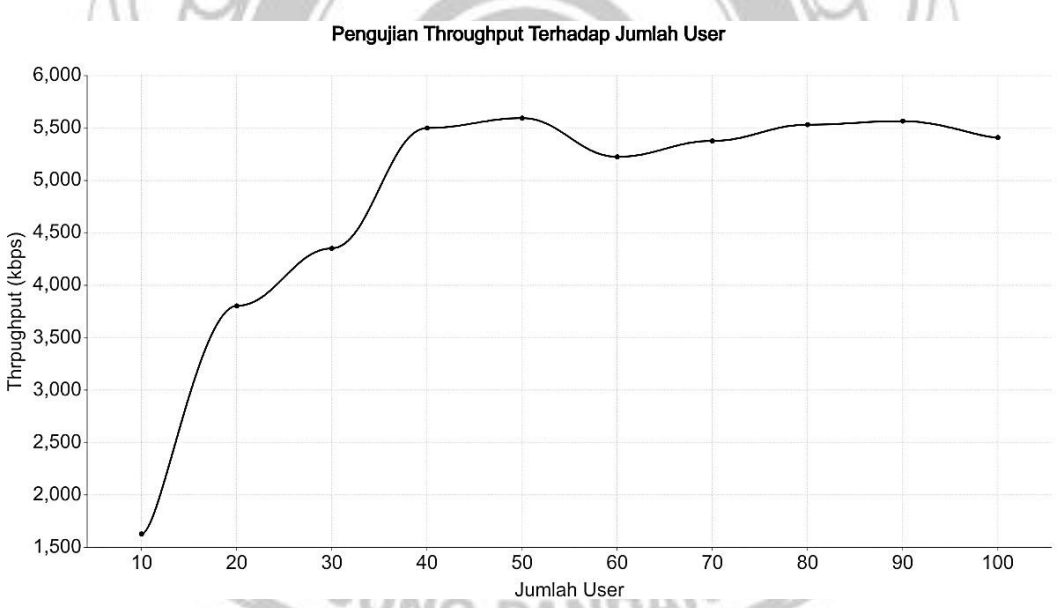
4.2.1 Hasil Pengujian *Throughput*

Pengujian *Throughput* bertujuan untuk mengetahui kemampuan kapasitas jaringan yang sebenarnya dan berapa lama waktu yang diperlukan untuk mentransfer data. Dalam pengujian ini digunakan *software* Apache Jmeter untuk menghitung hasil *throughput*, terdapat dua hasil pengujian yang dilakukan yaitu dengan menguji nilai *throughput* terhadap jumlah user dan ukuran bandwidth. Pengujian masing-masing dilakukan dengan 10 kali perulangan. Hasil pengujian

throughput menggunakan Persamaan 2.2 secara spesifik dapat dilihat pada Tabel 4.2 dan Tabel 4.3 serta grafik hasil pengujian dapat dilihat pada Gambar 4.11 dan Gambar 4.12.

a) Hasil pengujian *throughput* terhadap jumlah user

Hasil pengujian *throughput* terhadap jumlah *user* dapat dilihat pada Gambar 4.11 dimana dilakukan pengujian akses *video streaming* dengan jumlah *user* 10 hingga 100 user dengan interval 10, maksimum *bandwidth* 65 mbps dengan waktu akses video selama 15 menit yang memiliki resolusi 720p .



Gambar 4. 11 Grafik hasil pengujian *throughput* terhadap jumlah *user*

Pada Gambar 4.11, dapat dilihat grafik *throughput.* Secara umum, nilai *throughput* tidak akan melebihi nilai *bandwidth* dan semakin banyak jumlah *user*, maka nilai *throughput* semakin menurun. Pada penelitian ini, nilai *throughput* tidak melebihi nilai *bandwidth* dan semakin banyak jumlah *user*, maka semakin rendah pula nilai *throughput*. Nilai *throughput* cenderung meningkat seiring dengan

bertambahnya jumah *user* sampai dengan 50 *user*, namun setelah itu ketika jumlah *user* sebanyak 60 hingga 100, nilai *throughput* cenderung menurun dibanding dengan nilai *throughput* ketika terdapat 50 *user*. Hal tersebut sesuai dengan penelitian sebelumnya oleh Azhar,dkk (2016) dengan menggunakan jaringan *Wi-Fi* tanpa implementasi *Software-defined Networking* yang berpendapat bahwa semakin besar jumlah *user* maka semakin kecil nilai *throughput*. Semakin meningkatnya nilai *throughput* seiring dengan peningkatan jumlah *user* seperti yang terjadi ketika terdapat 10 hingga 50 *user* yang mengakses *video streaming* dikarenakan jumlah paket dari jenis trafik yang dikirimkan semakin banyak dan kapasitas *bandwidth* masih memadai. Lamanya waktu akses *video streaming* juga menentukan kualitas *throughput*, hal ini disebabkan kepadatan lalu lintas pada jaringan yang digunakan bervariasi tergantung pada waktu penggunaan jaringan. Grafik menunjukkan *throughput* cenderung stabil hingga jumlah user sebanyak 50 *user*, namun pada saat terdapat 60 *user* yang mengakses *video streaming* secara bersamaan, nilai *throughput* tampak menurun. Hal ini mengindikasikan bahwa penggunaan bandwidth secara maksimal dan stabil dapat digunakan hingga 50 *user*. Detail hasil pengujian *throughput* dapat dilihat pada Tabel 4.2.

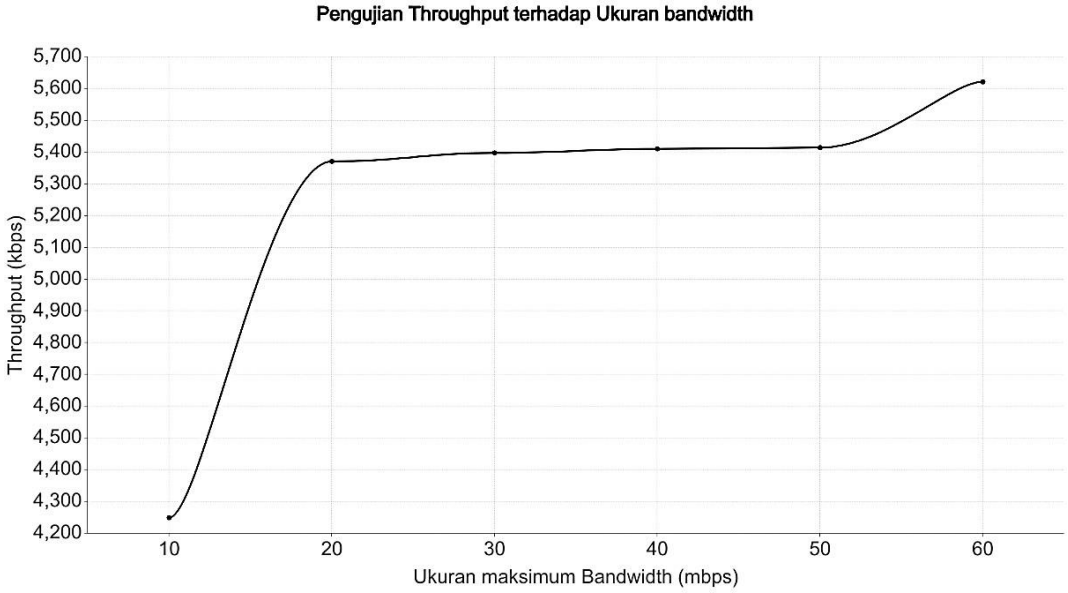
Tabel 4. 2 Hasil pengujian *throughput* (kbps) terhadap jumlah user

User	*Throughput* (kbps)
10	1,629.93
20	3,806.20
30	4,355.21
40	5,502.84
50	5,598.61

60	5,228.34
70	5,380.68
80	5,535.14
90	5,570.01
100	5,412.98

Berdasarkan Tabel 4.2, nilai *throughput* tertinggi sebesar 5,598.61 kbps yang diperoleh ketika terdapat 50 *user* mengakses *video streaming*. Nilai terendah yaitu 1,629.93 kbps yang diperoleh ketika terdapat 10 *user* yang mengakses video streaming. Pengukuran nilai *throughput* dengan jumlah 10 *user* termasuk dalam kategori "baik" dengan indeks 3 dalam standar dan ketentuan TIPHON yang terdapat pada tabel 2.2, dan dengan jumlah *user* 20 hingga 100 *user* termasuk dalam kategori "sangat baik" dengan indeks 4 berdasarkan standar dan ketentuan TIPHON dimana nilai *throughput* melebihi 2.1 mbps.

b) Hasil pengujian *throughput* terhadap ukuran *bandwidth*



Gambar 4. 12 Grafik hasil pengujian *throughput* terhadap *bandwidth*

Pada Gambar 4.12 dapat dilihat hasil pengujian *throughput* terhadap ukuran *bandwidth* dengan *user* tetap sebanyak 50 *user*. Secara umum, semakin tinggi ukuran *bandwidth* yang diberikan maka semakin tinggi pula nilai *throughput*. Hal tersebut sesuai dengan penelitian ini dimana hasil pengujian menunjukkan bahwa nilai *throughput* semakin tinggi seiring dengan meningkatnya ukuran maksimum *bandwidth*. Nilai tertinggi *throughput* diperoleh ketika terdapat 50 *user* yang mengakses *video streaming* selama 15 menit dengan resolusi video 720p, nilai *throughput*-nya sebesar 5,621.82 kbps. Detail hasil pengujian *throughput* dapat dilihat pada Tabel 4.3.

Tabel 4. 3 Hasil pengujian *throughput* (kbps) terhadap ukuran bandwidth

Ukuran bandwidth (mbps)	*Throughput* (kbps)
10	4,250.38
20	5,371.20
30	5,398.09
40	5,410.71
50	5,414.58
60	5,621.82

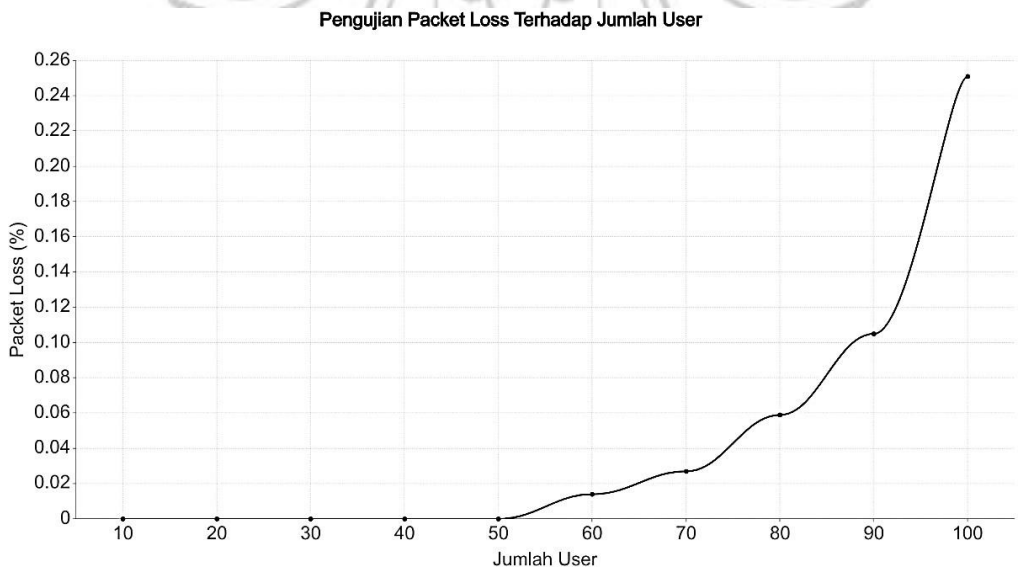
Berdasarkan tabel 4.3, nilai *throughput* saat terdapat 50 *user* dengan ukuran *bandwidth* maksimum 10 mbps sebesar 4,250.38 kbps. Saat diberikan maksimum *bandwidth* 20 mbps, nilai *throughput* meningkat menjadi 5,371.20. Ketika diberikan maksimum *bandwidth* 30 mbps, nilai *throughput* meningkat menjadi 5398,09 kbps. Ketika diberikan maksimum *bandwidth* 40 mbps, nilai *throughput* sebesar 5,414.58 kbps. Nilai tertinggi terjadi saat diberikan maksimum *bandwidth* 60 mbps dengan jumlah *user* yang mengakses *video streaming* sebanyak 50 *user*

yaitu sebesar 5,621.82 kbps. Nilai terendah diperoleh ketika ukuran maksimum *bandwidth* 10 mbps dengan 50 *user* yaitu 4,250.38 kbps. Pengukuran nilai *throughput* dengan maksimum *bandwidth* sebesar 10 hingga 60 mbps termasuk dalam kategori "sangat baik" dengan indeks 4 menurut standar dan ketentuan TIPHON.

4.2.2 Hasil Pengujian *Packet Loss*

Pengujian *packet loss* bertujuan untuk mengetahui persentase paket tidak sampai ke tujuan yang terjadi pada jaringan. Terdapat dua hasil pengujian yang dilakukan yaitu mengukur nilai packet loss terhadap jumlah user dan ukuran bandwidth. Hasil pengujian *packet loss* menggunakan persamaan 2.3 secara spesifik dapat dilihat pada Tabel 4.4 dan Tabel 4.5 serta grafik hasil pengujian dapat dilihat pada Gambar 4.13 dan Gambar 4.14.

a) Hasil pengujian *packet loss* terhadap jumlah *user*



Gambar 4. 13 Grafik hasil pengujian *packet loss* terhadap jumlah *user*

Hasil pengujian *packet loss* terhadap jumlah *user* dapat dilihat pada Gambar 4.13. Secara umum, semakin banyak jumlah *user* yang mengakses *video streaming* maka semakin tinggi pula *packet loss* yang terjadi. Pada grafik terlihat bahwa peningkatan jumlah *user* mempengaruhi nilai *packet loss* dimana perbandingan *packet loss* terhadap banyaknya jumlah *client* menghasilkan data yang tidak stabil. Hal ini sesuai dengan penelitian sebelumnya (Azhar dkk, 2016) yang berpendapat bahwa persentase *packet loss* berdasarkan jumlah *client*, terlihat bahwa semakin banyak jumlah *client*, maka persentase *packet loss* yang diperoleh menurun. Hal ini dikarenakan *packet loss* terjadi ketika kapasitas *buffer* penuh sebagai akibat dari antrian dan keterlambatan yang disebabkan oleh kepadatan lalu lintas data di jaringan, sehingga data baru yang dikirim tidak akan diterima. Pada pengujian ini, tidak terjadi *packet loss* ketika terdapat 10 hingga 50 *user* yang mengakses *video streaming*. *Packet loss* terjadi ketika jumlah *user* melebihi 50 yaitu ketika jumlah user mencapai 60 hingga 100, sehingga dapat disimpulkan bahwa penggunaan maksimal jaringan dapat dilakukan hingga 50 *user* yang merupakan batas dimana tidak terjadi *packet loss*. Detail hasil pengujian *packet loss* dapat dilihat pada Tabel 4.4.

Tabel 4. 4 Hasil pengujian p*acket* loss(%) terhadap jumlah *user*

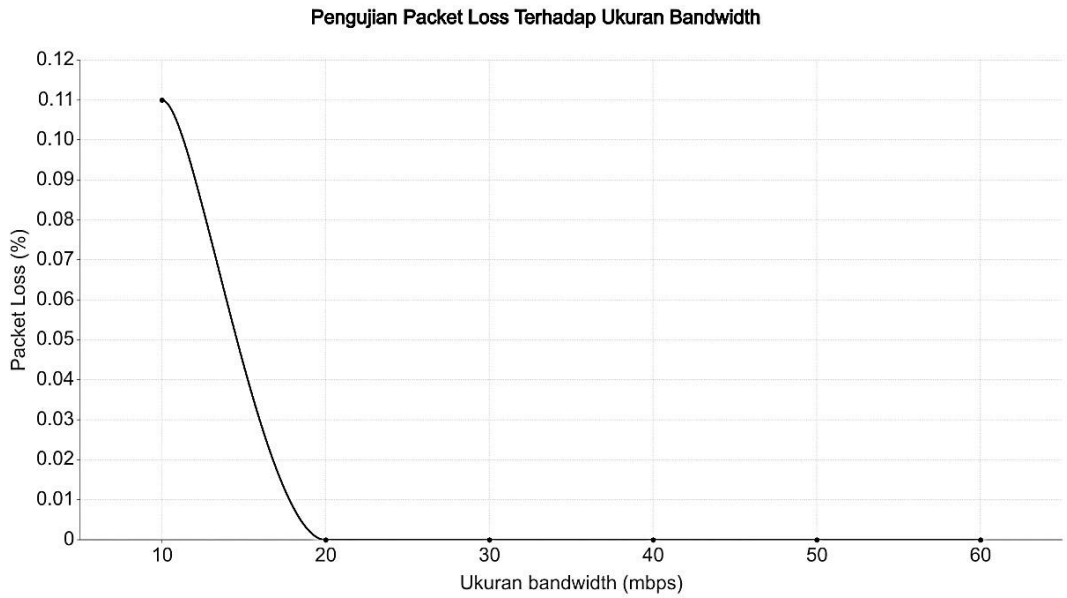
Jumlah user	Packet Loss (%)	Jumlah user gagal
10	0.00	0
20	0.00	0
30	0.00	0
40	0.00	0
50	0.00	0

(Lanjutan Tabel 4.4 pada halaman berikut)

60	0.014	0
70	0.027	0
80	0.059	2
90	0.105	2
100	0.251	3

Berdasarkan tabel 4.4, ketika *video streaming* diakses sebanyak 60 *user* terdapat 0.014% *packet loss* yang terjadi. Ketika terdapat 70 *user* yang mengakses *video streaming*, terjadi 0.027% *packet loss* dan dengan 80 user terjadi 0.059% *packet loss*. Nilai *packet loss* tertinggi yaitu 0.251% diperoleh ketika terdapat 100 *user* yang mengakses *video streaming*. Berdasarkan standar dan ketentuan TIPHON, nilai *packet loss* terhadap jumlah *user* yang dimulai dengan 10 hingga 100 *user*, berada dalam kategori "sangat baik" dan memiliki nilai indeks 4 dengan nilai *packet loss* kurang dari 2%. Selain itu, selama pengukuran *packet loss* didapatkan fakta bahwa dengan jumlah user 80 dan 90 *user*, terdapat 2 *user* yang tidak dapat menyelesaikan *video streaming*, dengan jumlah 100 *user* terdapat 3 *user* yang tidak dapat menyelesaikan *video streaming*. Hal ini dikarenakan keterbatasan *bandwidth* dan beban trafik yang sangat padat serta penggunaan *protocol HTTP Live streaming* (HLS) yang berbasis TCP membuat beberapa user tidak mampu menyelesaikan video streaming karena pengiriman data yang tetap harus melibatkan koneksi internet. Dalam hal ini kapasitas *bandwidth* sudah penuh hingga menyebabkan server *overload*.

b) Hasil pengujian *packet loss* terhadap ukuran *bandwidth*



Gambar 4. 14 Grafik hasil pengujian *packet loss* terhadap *bandwidth*

Secara umum, semakin tinggi penggunaan *bandwidth*, maka semakin tinggi pula nilai *packet loss*. Hal tersebut sesuai dengan penelitian sebelumnya oleh Fauzi (2016), yang berpendapat bahwa penggunaan *bandwidth* yang tinggi menyebabkan data yang dikirim juga semakin banyak, sehingga semakin besar peluang *packet loss* terjadi. Yuvandra, dkk (2013) berpendapat biasanya hal yang menjadi penyebab adanya *packet loss*, pada saat *request* ataupun *receive* data adalah kegagalan pada jaringan, kepadatan trafik di jaringan, karena kesalahan *hardware*, dan keterbatasan *bandwidth* pada jaringan internet saat melakukan *transmisi* data. Pada Gambar 4.14 yang menunjukkan grafik *packet loss* terhadap *bandwidth* sesuai dengan penelitian sebelumnya. Ketika terlalu banyak jumlah *user* dengan kapasitas *bandwidth* yang terbatas atau kurang, maka terjadi banyak *packet loss* seperti yang

terjadi pada penelitian ini khususnya ketika terdapat 50 user yang mengakses video streaming dengan *bandwidth* 10 mbps, terjadi 0.11% *packet loss*.

Tabel 4. 5 Hasil pengujian *packet loss*(%) terhadap ukuran *bandwidth*

Ukuran bandwidth (mbps)	Packet Loss (%)	Jumlah user gagal
10	0.11	10
20	0.00	0
30	0.00	0
40	0.00	0
50	0.00	0
60	0.00	0

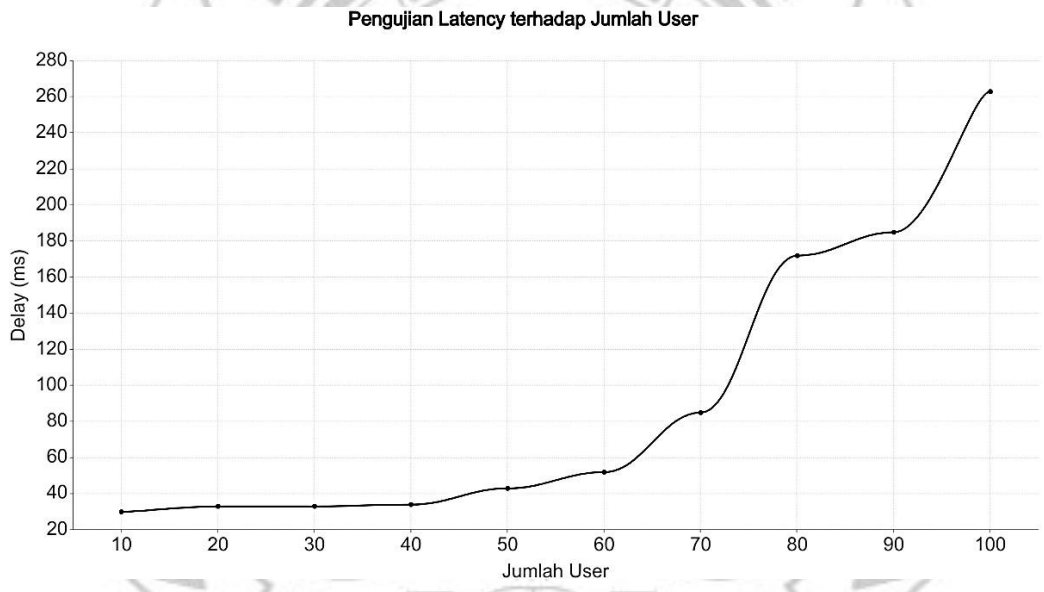
Berdasarkan Tabel 4.5, terdapat nilai *packet loss* tertinggi yaitu 0,11% dengan ukuran maksimum *bandwidth* 10 mbps. Berdasarkan standar TIPHON, nilai tersebut masuk dalam kategori "sangat baik" dimana nilai *packet loss* tidak melebihi 2%. Penggunaan ukuran maksimum *bandwidth* sebesar 10 mbps dengan 50 *user* yang mengakses *video streaming* secara bersamaan, terdapat 10 *user* yang tidak mampu menyelesaikan untuk menonton video selama 15 menit dengan resolusi 720p. Ketika ukuran maksimum *bandwidth* dinaikkan menjadi 20 mbps hingga 60 mbps, tidak terdapat *packet loss* yang terjadi maupun *user* yang gagal menyelesaikan akses *video streaming*. Hal ini mengindikasikan bahwa dengan ukuran maksimum *bandwidth* 20 mbps mampu digunakan oleh 50 *user* untuk menonton *video streaming* dengan resolusi 720p.

4.2.3 Hasil Pengujian *Latency*

Pengujian *latency* bertujuan untuk mengetahui rata-rata jeda waktu antar paket data yang terkirim dari *host* satu ke *host* yang lain. Dalam pengujian ini

digunakan *software* Apache Jmeter untuk menghitung hasil *latency*, terdapat dua hasil pengujian yang dilakukan dengan mengukur *latency* terhadap jumlah *user* dan ukuran *bandwidth*. Pengujian masing-masing dilakukan dengan 10 kali perulangan. Hasil pengujian *latency* menggunakan Persamaan 2.4 secara spesifik dapat dilihat pada Tabel 4.6 dan Tabel 4.7 serta grafik hasil pengujian dapat dilihat pada Gambar 4.15 dan Gambar 4.16.

a) Hasil pengujian *latency* terhadap jumlah *user*



Gambar 4. 15 Grafik hasil pengujian *latency* terhadap jumlah *user*

Pada gambar 4.15 dapat dilihat hasil pengujian nilai *latency* terhadap jumlah *user* dengan maksimum *bandwidth* 65 mbps. Secara umum, nilai *latency* dipengaruhi oleh banyaknya jumlah *user*. Hal ini sesuai dengan penelitian sebelumnya oleh Azhar,dkk (2016) yang berpendapat bahwa nilai *latency* berdasarkan jumlah *user* ini terjadi perbandingan yakni semakin banyaknya *user* yang mengakses *video streaming*, maka semakin besar pula waktu *latency* nya.

Sama halnya dengan penelitian ini, pengujian dengan skenario *video streaming* membuktikan bahwa semakin banyak *user* yang mengakses *video streaming*, semakin besar pula waktu *latency* yang terjadi. Detail hasil pengujian *latency* dapat dilihat pada Tabel 4.6.

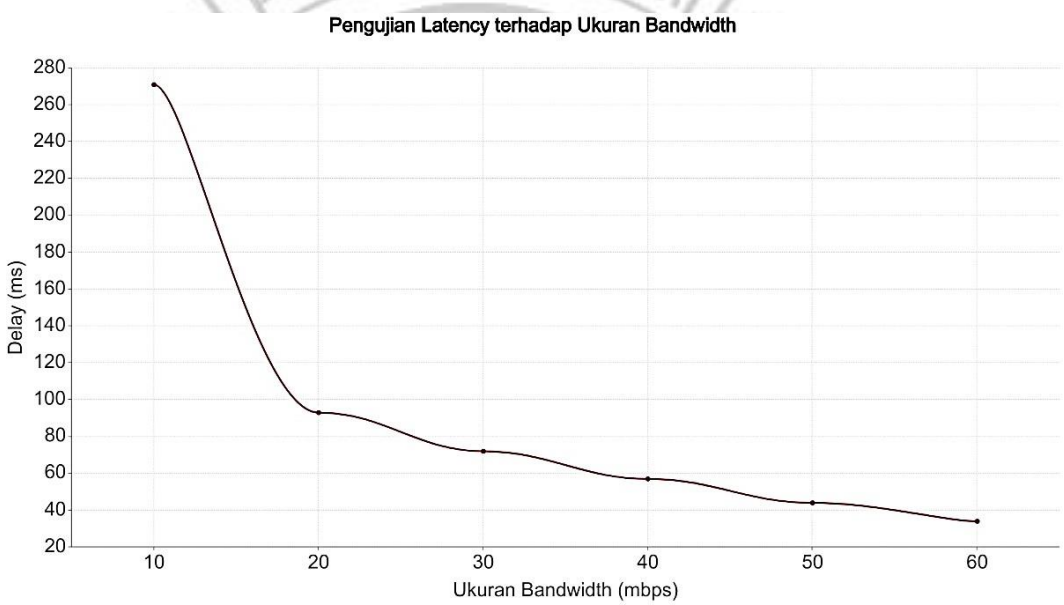
Tabel 4. 6 Hasil pengujian *latency*(ms) terhadap jumlah *user*

Jumlah user	Latency (ms)
10	30
20	33
30	33
40	34
50	43
60	52
70	85
80	172
90	185
100	263

Berdasarkan tabel 4.6, nilai *latency* ketika terdapat 10 *user* yang mengakses video streaming sebesar 30 ms. Ketika terdapat 20 *user*, nilai *latency* sebesar 33 ms. Ketika terdapat 30 *user*, nilai *latency* menjadi 33 ms. Saat terdapat 40 *user*, nilai latency menjadi 34 ms. Ketika diakses oleh 50 *user*, nilai *latency* sebesar 43 ms, dengan 60 user nilai *latency* menjadi 52. Ketika diakses oleh 70 user, nilai *latency* menjadi 85 ms. Ketika diakses oleh 80 user, nilai *latency* sebesar 172 ms, dan dengan 90 *user* nilai *latency* menjadi 185 ms. Nilai *latency* tertinggi didapatkan sebesar 263 ms yang diperoleh ketika terdapat 100 *user* mengakses *video streaming*. Nilai *latency* terendah yaitu 30 ms yang diperoleh ketika hanya terdapat

10 *user* yang mengakses *video streaming*. Menurut standar dan ketentuan yang ditetapkan TIPHON, nilai *latency* 263 ms, 185 ms, dan 172 ms termasuk dalam kategori "baik" dengan indeks 3 dimana nilai *latency* berada pada *range* 150-300 ms. Nilai *latency* 30 ms hingga 85 ms termasuk dalam kategori "sangat baik" dengan indeks 4 dimana nilai *latency* berada pada *range* 0-150 ms.

b) Hasil pengujian *latency* terhadap ukuran *bandwidth*



Gambar 4. 16 Grafik hasil pengujian *latency* terhadap *bandwidth*

Pada gambar 4.16, dapat dilihat hasil pengujian *latency* terhadap ukuran *bandwidth*. Semakin kecil ukuran *bandwidth* yang diberikan maka semakin besar pula *latency* yang akan terjadi. Dalam melakukan pengiriman data, paket yang berukuran besar akan menghabiskan waktu untuk proses membagi-bagi paket tersebut kedalam paket yang lebih kecil, sehingga waktu pengiriman menjadi semakin lambat. Berdasarkan hasil pengujian, Nilai *latency* tertinggi yaitu 271 ms yang terjadi ketika diberikan bandwidth 10 mbps dengan 50 *user* yang mengakses

video. Berdasarkan standar TIPHON nilai tersebut masuk dalam kategori "baik" dengan indeks 3. Ukuran minimal *bandwidth* yang digunakan untuk 50 *user* adalah 20 mbps dengan *latency* rata-rata 93 ms yang termasuk dalam kategori "sangat baik" versi TIPHON. Detail hasil pengujian *latency* dapat dilihat pada Tabel 4.7.

Tabel 4. 7 Hasil Pengujian *Latency*(ms) terhadap ukuran *bandwidth*

Ukuran *bandwidth* (mbps)	*Latency* (ms)
10	271
20	93
30	72
40	57
50	44
60	34

4.3 Hasil Pengujian QoS Jaringan

Pengujian QoS Jaringan bertujuan untuk mengukur nilai QoS secara keseluruhan dengan menggunakan persamaan 2.1. Detail hasil pengujian QoS jaringan dapat dilihat pada Tabel 4.8.

Tabel 4. 8 Nilai parameter QoS

Throughput		Packet Loss		Latency	
Throughput(kbps)	Indeks	Packet Loss(%)	Indeks	Latency(ms)	Indeks
1,629.93	3	0.00	4	30	4
3,806.20	4	0.00	4	33	4
4,355.21	4	0.00	4	33	4
5,502.84	4	0.00	4	34	4
5,598.61	4	0.00	4	43	4
5,228.34	4	0.014	4	52	4
5,380.68	4	0.027	4	85	4
5,535.14	4	0.059	4	172	3
5,570.01	4	0.105	4	185	3
5,412.98	4	0.251	4	263	3

4,250.38	4	0.11	4	271	3
5,371.20	4	0.00	4	93	4
5,398.09	4	0.00	4	72	4
5,410.71	4	0.00	4	57	4
5,414.58	4	0.00	4	44	4
5,621.82	4	0.00	4	34	4
Rata-rata = 4,967.92	**3.93**	**Rata-rata = 0.094**	**4**	**Rata-rata= 93.8**	**4**
QoS = 3.97					

Berdasarkan tabel 4.8, parameter QoS yang diujikan yaitu; *throughput*, *packet loss*, dan *latency* sebagian besar berada pada kategori "sangat baik" dengan *bandwidth* 10-60 mbps dan jumlah *user* sebanyak 10-100. Performa terbaik diperoleh ketika maksimal user yang mengakses video streaming sebanyak 50 user dengan bandwidth 65 mbps. Kategori *Throughput* menunjukkan indeks 4 dengan nilai *throughput* mencapai 5598,61 kbps. Kategori *packet loss* menunjukkan indeks 4 dengan nilai packet loss 0.00%. Kategori *latency* menunjukkan indeks 4 dengan besar latency 43 ms. Jaringan Wi-Fi dengan SDN menunjukkan nilai indeks QoS secara keseluruhan (QoS terhadap jumlah user dan QoS terhadap ukuran bandwidth) sebesar 3.97 dengan kategori "sangat memuaskan" menurut ketentuan yang telah ditetapkan oleh TIPHON.

BAB V KESIMPULAN DAN SARAN

5.1 Kesimpulan

1. Implementasi jaringan *Wi-Fi* dengan konsep *Software-Defined Networking* dapat diterapkan dengan menggunakan Open vSwitch dan ONOS *controller* dan bekerja dengan baik.
2. Berdasarkan pengujian yang telah dilakukan, jaringan *Wi-Fi* dengan *Software-Defined Networking* memiliki kinerja yang baik. Jaringan Wi-Fi dengan SDN menunjukkan nilai indeks QoS secara keseluruhan (QoS terhadap jumlah user dan QoS terhadap ukuran bandwidth) sebesar 3.97 dengan kategori "sangat memuaskan" menurut ketentuan yang telah ditetapkan oleh TIPHON.

5.2 Saran

Penulis menyadari masih terdapat kekurangan dalam penelitian ini. Oleh karena itu penulis mengharapkan agar hasil dari penelitian ini dapat dikembangkan untuk memperbaiki kekurangan yang ada. Adapun saran yang penulis berikan untuk penelitian selanjutnya adalah sebagai berikut.

1. Menerapkan jaringan *Wi-Fi* dengan konsep *Software-Defined Networking* dalam lingkup jaringan yang lebih luas.
2. Melakukan jenis pengujian lain agar kinerja dari jaringan *Wi-Fi* dengan konsep *Software-Defined Networking* dapat lebih terlihat.

DAFTAR PUSTAKA

Agarwal, S. (2013). *Traffic Engineering in SDN.pdf*. 2211–2219.

Azhar, A., Pramono, S., & Supriyanto, E. (2016). *An Analysis of Quality of Service (QoS) In Live Video Streaming Using Evolved HSPA Network Media*. *1*(1), 1–6.

Azodolmolky, S. (2013). *Software Defined Networking with OpenFlow*. 152.

Dibaji, S. M., & Ishii, H. (2015). Consensus of second-order multi-agent systems in the presence of locally bounded faults. *Systems and Control Letters*, *79*, 23–29. https://doi.org/10.1016/j.sysconle.2015.02.005

Fauzi, A. (2016). *Perancangan dan implementasi jaringan wifi berbasiskan protokol openflow*.

Flannagan, M., Froom, R., & Turek, K. (2003). *Cisco Catalyst QoS : Quality of Service in Campus Networks*.

Gouveia, F. C. De, & Magedanz, T. (2011). Quality of Service in Telecommunication M Es Pl C E O – M Es Pl C E O –. *Telecommunication Systems and Technologies*, *II*, 1–8.

Graf, T. (2013). *Underneath OpenStack Quantum: Software Defined Networking with Open vSwitch (PDF)*.

Hunaifi, N., & Fauzan Akbar, R. (2019). Analisis Kinerja Jaringan Berbasiskan Software Definition Network Dengan Protokol Openflow Di Rri Bandung. *Infotronik : Jurnal Teknologi Informasi Dan Elektronika*, *4*(1), 24. https://doi.org/10.32897/infotronik.2019.4.1.172

Kaur, S., Singh, J., & Ghumman, N. S. (2014). *a5B6Be6E0966Acc519D7D577E1F6Ef0C8Bea.Pdf*.

Lara, A., Kolasani, A., & Ramamurthy, B. (2013). *White-Energy-Cultural-Evolution.pdf*. 1–20.

Marschke, D., Doyle, J., & Pete, M. (2015). *Software Defined Networking (SDN): Anatomy of OpenFlow Volume 1*. *Vol 1*. Lulu.com

Mininet. (2014). *Software-defined Networking*. http://mininet.org/.

Ongaro, D., Ousterhout, J., Ongaro, D., & Ousterhout, J. (2014). *In Search of an Understandable Consensus Algorithm In Search of an Understandable Consensus Algorithm*.

Patel, G., Athreya, A. S., & Erukulla, S. (2013). OpenFlow based Dynamic Load Balanced Switching. *COEN 233, Project Report*.

Rouse, M. (2012). *OpenFlow*. www.techtarget.com: http://whatis.techtarget.com/definition/OpenFlow

Sans, F., & Gamess, E. (2013). Analytical performance evaluation of different switch solutions. *Journal of Computer Networks and Communications*, *2013*. https://doi.org/10.1155/2013/953797

Satasiya, D., & Raviya Rupal, D. (2016). Analysis of Software Defined Network firewall (SDF). *Proceedings of the 2016 IEEE International Conference on Wireless Communications, Signal Processing and Networking, WiSPNET 2016*, 228–231. https://doi.org/10.1109/WiSPNET.2016.7566125

Shalimov, A., Zuikov, D., Zimarina, D., Pashkov, V., & Smeliansky, R. (2013). Advanced study of SDN/OpenFlow controllers. *ACM International Conference Proceeding Series*, *January 2014*. https://doi.org/10.1145/2556610.2556621

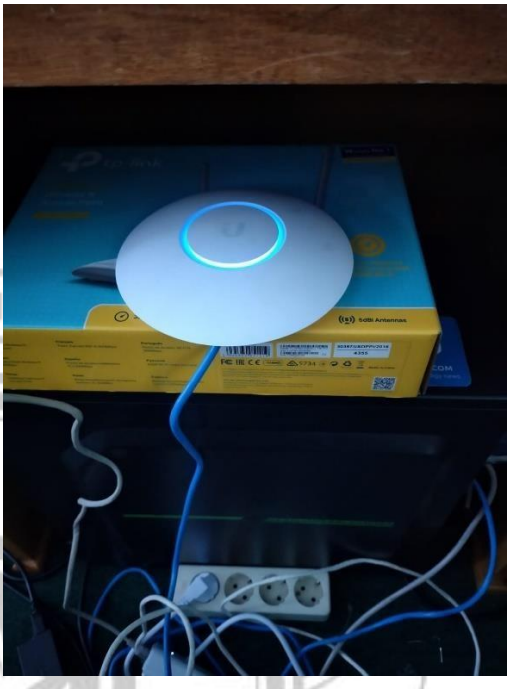
Tiphon. (2002). Telecommunications and Internet Protocol Harmonization Over Networks (TIPHON) Release 3; End-to-end Quality of Service in TIPHON systems; Part 7: Design guide for elements of a TIPHON connection from an end-to-end speech transmission performance point of. *Telecommunications and Internet Protocol Harmonization Over Networks (TIPHON) Release 3; End-to-End Quality of Service in TIPHON Systems; Part 7: Design Guide for Elements of a TIPHON Connection from an End-to-End Speech Transmission Performance Point Of*, *1*, 1–72.

Wang, Z., Zhao, C., Mu, S., Chen, H., & Li, J. (2019). On the parallels between paxos and raft, and how to port optimizations. *Proceedings of the Annual ACM Symposium on Principles of Distributed Computing*, *Section 3*, 445–454. https://doi.org/10.1145/3293611.3331595

Yuvandra, R., & Zulfin, M. (2013). *Analisis Kinerja Trafik Video Chatting Pada Sistem Client- Client Dengan Aplikasi Wireshark*.

LAMPIRAN

Lampiran 1 Foto perangkat



Lampiran 2 Konfigurasi Network server proxmox

```
auto lo
iface lo inet loopback

auto enp2s0
iface enp2s0 inet manual

auto enp3s0
iface enp3s0 inet manual

auto enx00e04c534458
iface enx00e04c534458 inet manual

auto vmbr1
iface vmbr1 inet static
        address 10.2.3.76/24
        gateway 10.2.3.1
        bridge-ports enp2s0
        bridge-stp off
        bridge-fd 0

auto vmbr0
iface vmbr0 inet manual
        bridge-ports enp3s0
        bridge-stp off
        bridge-fd 0

auto vmbr2
iface vmbr2 inet manual
        bridge-ports enx00e04c534458
        bridge-stp off
        bridge-fd 0
```

Lampiran 3 Konfigurasi Network Interface Open vSwitch

- Konfigurasi Network interface pada Open vSwitch-1

```
ninda@ninda: ~
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

# source /etc/network/interfaces.d/*

# The loopback network interface
auto lo
iface lo inet loopback

allow-hotplug ens18
iface ens18 inet static
        address 10.2.3.23
        netmask 255.255.255.0
        gateway 10.2.3.1

iface ens19 inet manual

allow-hotplug br0
iface br0 inet static
        address 192.168.100.1
        netmask 255.255.255.0
        network 192.168.100.0
        bridge_ports ens19
```

- Konfigurasi Network interface pada Open vSwitch -2

```
root@ovs2: /home/ninda
  GNU nano 2.5.3                                File: /etc/network/interfaces

# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

# source /etc/network/interfaces.d/*

# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
allow-hotplug ens18
iface ens18 inet static
        address 10.2.3.151
        netmask 255.255.255.0
        gateway 10.2.3.1
        network 10.2.3.0

iface ens19 inet manual

allow-hotplug br1
iface br1 inet static
        address 192.168.200.1
        netmask 255.255.255.0
        network 192.168.200.0
        bridge_ports ens19
```

- Konfigurasi IP Forwarding Open vSwitch

```
root@ovs2: /home/ninda
  GNU nano 2.5.3            File: /etc/sysctl.conf


######################################################################3
# Functions previously found in netbase
#

# Uncomment the next two lines to enable Spoof protection (reverse-path filter)
# Turn on Source Address Verification in all interfaces to
# prevent some spoofing attacks
#net.ipv4.conf.default.rp_filter=1
#net.ipv4.conf.all.rp_filter=1

# Uncomment the next line to enable TCP/IP SYN cookies
# See http://lwn.net/Articles/277146/
# Note: This may impact IPv6 TCP sessions too
#net.ipv4.tcp_syncookies=1

# Uncomment the next line to enable packet forwarding for IPv4
net.ipv4.ip_forward=1
```

- Konfigurasi DHCP server pada Open vSwitch-1

1) File "/etc/dhcp/dhcpd.conf"

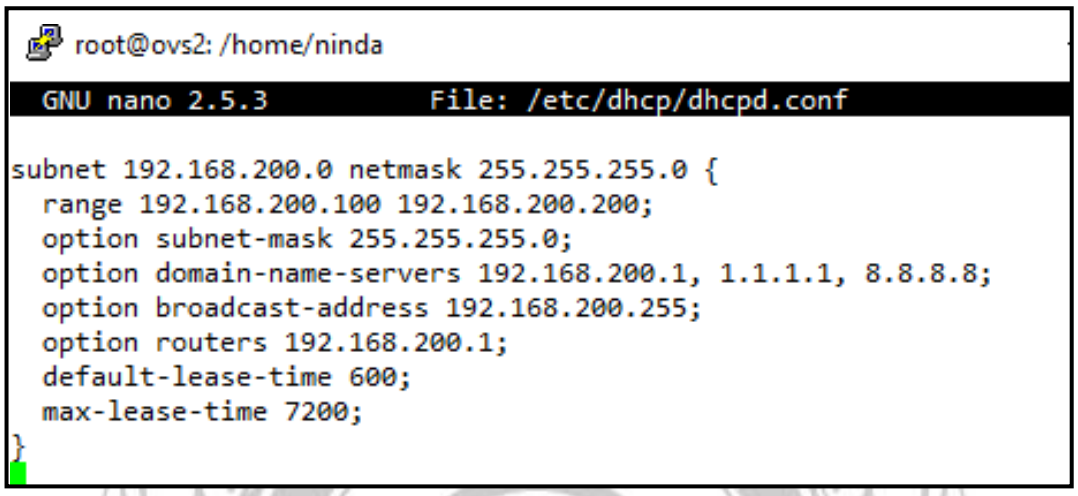
```
  GNU nano 2.5.3                                File: /etc/dhcp/dhcpd.conf

#}

subnet 192.168.100.0 netmask 255.255.255.0 {
  range 192.168.100.100 192.168.100.200;
  option subnet-mask 255.255.255.0;
  option domain-name-servers 192.168.100.1, 1.1.1.1, 8.8.8.8;
  option broadcast-address 192.168.100.255;
  option routers 192.168.100.1;
  default-lease-time 600;
  max-lease-time 7200;
}
```

2) File "/etc/default/isc-dhcp-server"

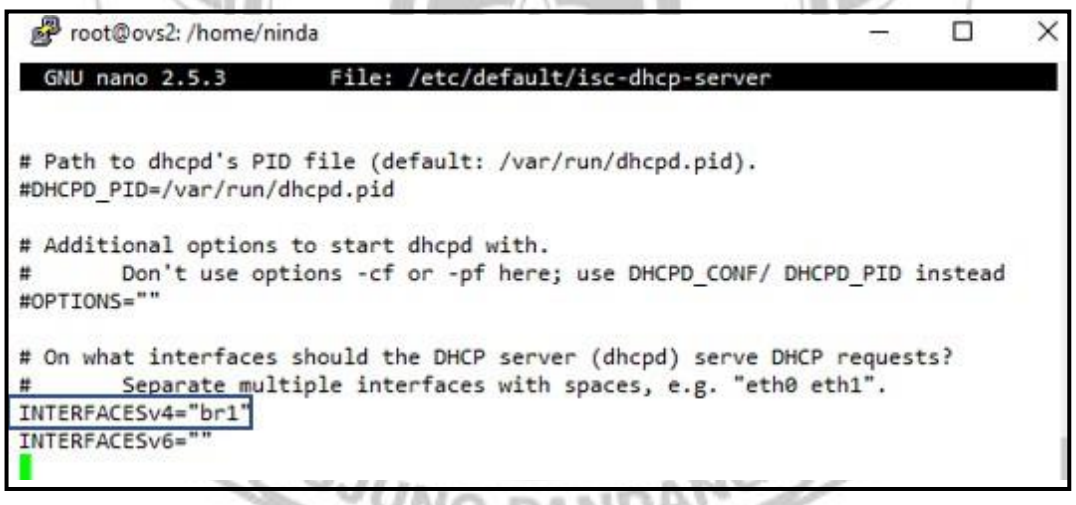
```
# Path to dhcpd's config file (default: /etc/dhcp/dhcpd.conf).
#DHCPD_CONF=/etc/dhcp/dhcpd.conf

# Path to dhcpd's PID file (default: /var/run/dhcpd.pid).
#DHCPD_PID=/var/run/dhcpd.pid

# Additional options to start dhcpd with.
#       Don't use options -cf or -pf here; use DHCPD_CONF/ DHCPD_PID instead
#OPTIONS=""

# On what interfaces should the DHCP server (dhcpd) serve DHCP requests?
#       Separate multiple interfaces with spaces, e.g. "eth0 eth1".
INTERFACESv4="br0"
INTERFACESv6=""
```

- Konfigurasi DHCP server pada Open vSwitch-2

1) File "/etc/dhcp/dhcpd.conf"



2) File "/etc/default/isc-dhcp-server"



Lampiran 4 Konfigurasi NAT

```
iptables -t nat -A POSTROUTING -o br0 -j MASQUERADE
iptables -t nat -L
netfilter-persistent save
netfilter-persistent reload
```

Lampiran 5 Informasi port Open vSwitch

- Menambah bridge dan port

```
root@ninda:/home/ninda# ovs-vsctl set-controller br0 tcp:10.2.3.116:6633

root@ninda:/home/ninda# ovs-vsctl add-br br0

root@ninda:/home/ninda# ovs-vsctl add-port br0 ens19
```

- Informasi port Open vSwitch-1

```
root@ninda:/home/ninda# ovs-vsctl show
79b20e5f-05c7-4a41-91fc-8867ae44f4f9
    Bridge "br0"
        Port "br0"
            Interface "br0"
                type: internal
        Port "ens19"
            Interface "ens19"
    ovs_version: "2.11.1"
```

- Informasi port Open vSwitch-2

```
root@ovs2:/home/ninda# ovs-vsctl show
2a0f19b7-76a9-493e-a33f-1c1f219dc6c2
    Bridge "br1"
        Controller "tcp:10.2.3.114:6633"
        Port "ens19"
            Interface "ens19"
        Port "br1"
            Interface "br1"
                type: internal
    ovs_version: "2.11.1"
```

Lampiran 6 Instalasi Open vSwitch

```
Install prasyarat Open vSwitch
# sudo apt update && sudo apt upgrade -y
# sudo apt install build-essential libssl-dev clang automake autoconf libtool python-pyftpdlib python-six python-dev
python-tftpy graphviz unbound
# wget https://www.openvswitch.org/releases/openvswitch-2.11.1.tar.gz
# tar xzf openvswitch-2.11.1.tar.gz
# cd openvswitch-2.11.1

Konfigurasi Open vSwitch package
# ./configure
Build Open vSwitch
# sudo make
# sudo make check
# sudo make install
Jalankan Open vSwitch kernel
# sudo modprobe openvswitch

Konfigurasi database
# mkdir -p /usr/local/etc/openvswitch
# sudo ovsdb-tool create /usr/local/etc/openvswitch/conf.db vswitchd/vswitch.ovsschema
Konfigurasi ovsdb-server
# mkdir -p /usr/local/var/run/openvswitch
# sudo ovsdb-server --remote=punix:/usr/local/var/run/openvswitch/db.sock --
remote=db:Open_vSwitch,Open_vSwitch,manager_options –private key=db:Open_vSwitch,SSL,private_key --
certificate=db:Open_vSwitch,SSL,certificate --bootstrap-ca-cert=db:Open_vSwitch,SSL,ca_cert --pidfile --detach --
log-file

Inisialisasi database
# ovs-vsctl --no-wait init

Jalankan Open vSwitch daemon.
# ovs-vswitchd --pidfile --detach --log-file

Cek Open vSwitch version.
# ovs-vsctl --version
```

Lampiran 7 Instalasi ONOS controller

```
Tambah user untuk instalasi onos
# sudo adduser sdn --system --group

Install java 8 atau 11
# sudo apt update
# sudo apt install git zip curl unzip python-minimal openjdk-8-jdk -y

Buat folder opt jika belum ada
# sudo mkdir -p /opt && cd /opt

Download ONOS 1.13.10
# sudo wget https://repo1.maven.org/maven2/org/onosproject/onos-releases/1.13.10/onos-1.13.10.tar.gz
# sudo tar xzf onos-1.13.10.tar.gz
# sudo mv onos-1.13.10 onos
# sudo chown -R sdn:sdn onos

Setting startup options
# sudo -u sdn nano /opt/onos/options

Jalankan ONOS dengan user sdn
# export ONOS_USER=sdn

Aktifkan default driver and openflow
# export ONOS_APPS=drivers,openflow,gui2

Install service file
# sudo cp /opt/onos/init/onos.initd /etc/init.d/onos
# sudo cp /opt/onos/init/onos.service /etc/systemd/system/
# sudo systemctl daemon-reload
# sudo systemctl enable onos

Start ONOS
# sudo systemctl start onos
```

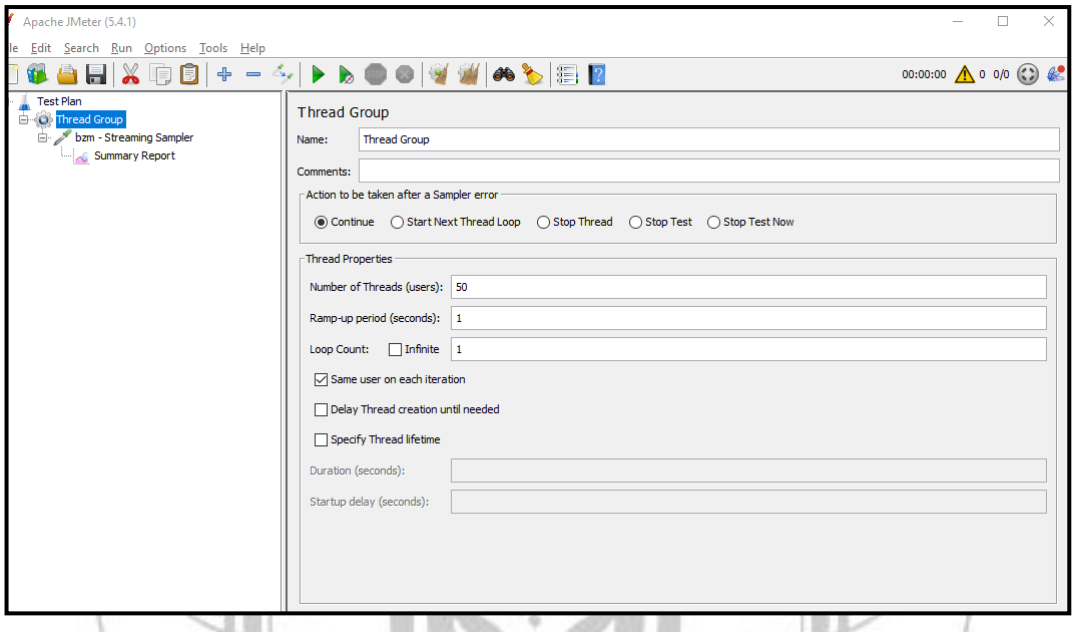
Lampiran 8 File cluster.json

root@onos1: /opt/onos-1.13.10/config

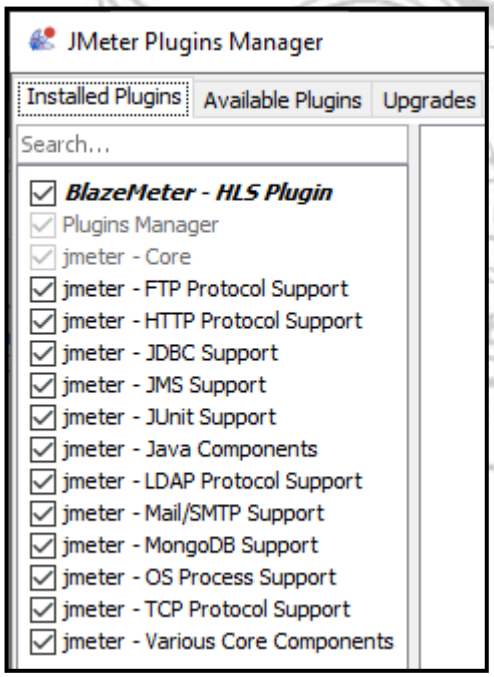

```
{"name":"default","nodes":[{"id":"10.2.3.116","ip":"10.2.3.116","port":9876},{"i
d":"10.2.3.113","ip":"10.2.3.113","port":9876},{"id":"10.2.3.114","ip":"10.2.3.1
14","port":9876}],"partitions":[{"id":3,"members":["10.2.3.116","10.2.3.113","10
.2.3.114"]},{"id":1,"members":["10.2.3.116","10.2.3.113","10.2.3.114"]},{"id":2,
"members":["10.2.3.116","10.2.3.113","10.2.3.114"]}]}
```

Lampiran 9 Penggunaan Software Apache Jmeter

- Konfigurasi Thread Group



- Instalasi Blazemeter Plugin



- Konfigurasi Streaming sampler