OPEN ACCESS
Indo-JC
Indonesian Journal of Computing
ISSN 2460-9056
socj.telkomuniversity.ac.id/indojc

Ind. Journal on Computing
Vol. 1, Issue. 1, March 2016. pp. 95-106
doi:10.21108/indojc.2016.1.1.20

Perancangan Simulasi Jaringan Virtual Berbasis Software-Define Networking

Izzatul Ummah[#1], Desianto Abdillah[#2]

 # Department of Computational Science, Telkom University
Jl. Telekomunikasi Terusan Buah Batu Bandung 40257 Indonesia

[1] izzatulummah@telkomuniversity.ac.id
[2] desianto@grid.telkomuniversity.ac.id

Abstract

Software-Define Networking (SDN) is a new approach in designing, building and managing computer network by separating control plane and data plane. The main concept of SDN is to centralize the network regulation/administration in control plane. OpenFlow, the main protocol in SDN, is a defined standard communication interface between control and forwarding layer. In this research, we simulated a virtual network based on SDN. The simulation of virtual network SDN is carried out by using Mininet. Mininet is a tool based on ligh-weight virtualization application that could create a realistic virtual network, running real kernel, switch, and application code. We evaluated the performance of our SDN simulation network, using several scenarios using topology of 2-switches, 4-switches, 8-switches, and 16-switches. The result showed that our simulation had achieved good performance in delay, jitter, and throughput. As the number of switches grows, the delay and jitter also grows but still conform with the standard recommendation of ITU-T, while the throughput is stable either for TCP and UDP ports.

Keywords: SDN, OpenFlow, Mininet, delay, jitter, throughput

Abstrak

Software-Define Networking (SDN) adalah sebuah pendekatan baru untuk merancang, membangun dan mengelola jaringan komputer dengan memisahkan *control plane* dan *data plane*. Konsep utama pada SDN adalah sentralisasi jaringan, di mana semua pengaturan berada pada *control plane*. Protokol yang paling menonjol pada SDN yaitu OpenFlow. OpenFlow adalah protokol/standar komunikasi antarmuka yang berada antara *control* dan *forwarding* layer. Pada penelitian ini dibangun sebuah simulasi jaringan virtual berbasis SDN, menggunakan tool/aplikasi Mininet. Mininet yaitu aplikasi yang berbasis *light-weight virtualization* yang dapat menciptakan jaringan virtual yang realistis, mampu menjalankan *real kernel*, serta *switch* dan kode aplikasi. Hasil penelitian menunjukkan bahwa simulasi jaringan virtual SDN telah bekerja dengan baik, berdasarkan hasil pengujian kinerja jaringan yang meliputi delay, jitter dan throughput, yang dilakukan dengan beberapa skenario yaitu topologi 2-switch, 4-switch, 8-switch dan 16-switch. Dengan jumlah switch yang meningkat, nilai delay dan jitter juga mengalami peningkatan namun masih memenuhi standard rekomendasi ITU-T, sedangkan untuk nilai throughput cenderung stabil baik untuk port TCP maupun UDP.

Kata Kunci: SDN, OpenFlow, Mininet, delay, jitter, throughput

Received on February 2016. Accepted on March 2016

I. PENDAHULUAN

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* [1]. 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*.

Pada jaringan konvensional yang non-SDN, perangkat keras jaringan (*router* dan *switch*) serta perangkat lunak jaringan (*firmware*) selama ini berada di bawah kendali vendor-vendor perusahaan yang memproduksi perangkat tersebut. *Programmer* tidak bisa mengujicobakan program mereka langsung ke *device* tersebut, tapi harus menggunakan simulator, di mana kode program simulasi sangat berbeda dengan kode program pada *real network environment*. Dengan menggunakan SDN dan OpenFlow, maka *firmware* dapat di-*switch* secara *remote* dan diakses dari jarak jauh. OpenFlow dapat mengakses dan memanipulasi secara langsung *forwarding plane* (*data plane*) dari perangkat-perangkat jaringan tersebut, baik secara fisik maupun virtual.

Saat ini belum semua perangkat jaringan mendukung protokol OpenFlow. Oleh karena itu untuk melakukan simulasi dengan protokol ini diperlukan simulator SDN yang mendukung protokol OpenFLow. Pada penelitian ini simulasi jaringan SDN dilakukan dengan menggunakan simulator Mininet.

II. DASAR TEORI

Pada bagian ini akan diuraikan terlebih dahulu mengenai SDN, OpenFlow, serta Mininet.

A. *Software-Defined Networking*

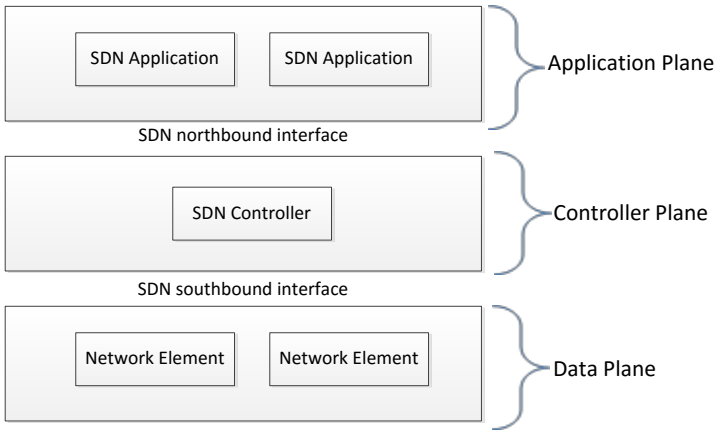
Software-Defined Networking (SDN) adalah sebuah paradigma arsitektur baru dalam bidang jaringan komputer, yang memiliki karakteristik dinamis, *manageable*, *cost-effective*, dan *adaptable*, sehingga sangat ideal untuk kebutuhan aplikasi saat ini yang bersifat dinamis dan *high-bandwidth*. Arsitektur ini memisahkan antara *network control* dan *fungsi forwarding*, sehingga *network control* tersebut menjadi *directly programmable* (dapat diprogram secara langsung), sedangkan infrastruktur yang mendasarinya dapat diabstraksikan untuk layer aplikasi dan *network services*.

Arsitektur SDN terdiri dari tiga layer yaitu sebagai berikut, seperti tampak pada Gambar 1 [1]:

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).



Gambar 1. Arsitektur SDN

Controller 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.

Keunggulan SDN antara lain:

1) *Directly programmable*: Dapat diprogram secara langsung karena *control plane* terpisah dari *forwarding plane* (*data plane*).

2) *Agile*: Pemisahan antara *control plane* dengan *forwarding plane* menyebabkan administrator jaringan dapat secara dinamis menyesuaikan *flow traffic network* sesuai kebutuhan organisasi/institusi.

3) *Centrally managed*: Network intelligence berada terpusat di SDN controller, dan mengelola satu gambaran umum (*global view*) yang utuh mengenai jaringan, sehingga tampak bagi pengguna hanya terdiri atas satu logical switch saja.

4) *Programmatically configured*: SDN memungkinkan network administrator untuk mengelola sumberdaya jaringan dengan sangat cepat, melalui program SDN yang terotomatisasi dan dapat ditulis sendiri karena program tersebut tidak bergantung pada *proprietary software* atau *device*.

5) *Open standards-based and vendor-neutral:* Format instruksi controller plan didefinisikan menggunakan open standard dan tidak tergantung pada *vendor-specific device* atau protokol tertentu, sehingga dapat diimplementasikan pada jaringan apapun tanpa terikat oleh vendor, dan pada akhirnya akan mempermudah inovasi di bidang jaringan.

Dalam [2] dijelaskan bahwa salah satu kelebihan SDN bila dibandingkan teknologi yang sudah ada adalah sifatnya yang *realistic* namun *light-weight*. Simulator jaringan yang telah ada sebelumnya (misalnya ns2 dan Opnet) tidak menyediakan lingkungan yang *realistic*, artinya kode yang dijalankan pada simulator tersebut jauh berbeda dari kode yang dijalankan di *real network*. Sementara itu, simulasi jaringan menggunakan VM (*virtual machine*) dapat memberikan lingkungan yang *realistic*, namun VM bersifat *heavy-weight* sehingga cukup berat untuk dijalankan di satu single node. Dalam aspek ini SDN menjadi solusi yang sangat menarik. Keunggulan

lain SDN adalah dalam hal *live migration*. Pada penelitian lainnya [3], diperoleh hasil bahwa live migration pada SDN dapat dilakukan melalui mekanisme yang sangat efisien. Sedangkan pada [4], dibahas aspek intelektual historis yang melatarbelakangi perkembangan *programmable network*.

Untuk mempromosikan perkembangan SDN, telah dibentuk sebuah organisasi yaitu Open Networking Foundation (ONF). Gugus-gugus kerja di ONF telah menghasilkan berbagai rancangan konsep, framework, arsitektur, dan standar terkait SDN.

B. OpenFlow

OpenFlow adalah protokol paling utama pada SDN. Posisinya berada di antara *controller* dan *forwarding (data plane)*. OpenFlow memungkinkan pengaturan *routing* dan pengiriman paket ketika melalui sebuah switch. Dalam sebuah jaringan, setiap switch hanya berfungsi meneruskan paket yang melalui suatu port tanpa mampu membedakan tipe protokol data yang dikirimkan. OpenFlow memungkinkan untuk mengakses dan memanipulasi *forwarding plane* secara langsung dari perangkat-perangkat jaringan seperti switch dan router baik secara fisik maupun virtual.

Nick McKeown et.al memberikan pembahasan yang rinci tentang latarbelakang OpenFlow [5]. Ide dasarnya cukup sederhana, yaitu mengeksploitasi fakta bahwa switch dan router Ethernet yang paling modern mengandung *flow table* yang dijalankan pada tingkat *line-rate* untuk mengimplementasikan firewall, NAT, QoS, dan juga untuk mengumpulkan data statistik. *Flow table* bisa berbeda implementasinya tergantung vendor dari perangkat keras tersebut. Akan tetapi terdapat beberapa *common set of function* (kumpulan fungsi atau *primitive* yang serupa) di antara berbagai switch dan router *vendor-based* tersebut. OpenFlow dalam hal ini menyediakan *open protocol* untuk memprogram *flow-table* pada berbagai switch dan router tersebut. Di lain pihak, vendor pun tidak dirugikan karena tidak perlu membuka mekanisme kerja internal dari produk switch atau router buatan masing-masing vendor.

Untuk bisa menggunakan OpenFlow diperlukan controller SDN yang mendukung jalannya protokol OpenFlow. POX controller adalah salah satu controller SDN yang mendukung protokol OpenFlow. POX adalah controller yang berbasis bahasa Python, dan dapat dijalankan pada sistem operasi Windows, MacOS dan Linux. POX masih dalam tahap pengembangan, dan merupakan perkembangan dari controller terdahulu yaitu NOX (berbasis bahasa C). Dalam jaringan SDN, controller harus diaktifkan dahulu sebelum menjalankan jaringan itu sendiri.

 # ./pox.py samples.pretty_log forwarding.l2_learning

Perintah di atas adalah perintah untuk meng-*invoke* atau mengaktifkan POX. **./pox.py** digunakan untuk menjalankan program utama controller POX, **semples.pretty_log** digunakan untuk melihat setiap log dalam perubahan komunikasi. Sedangkan **forwarding.l2_learning** digunakan untuk mengoperasikan OpenFlow switch. Detail spesifikasi teknis OpenFlow dapat dilihat di [6].

C. Mininet

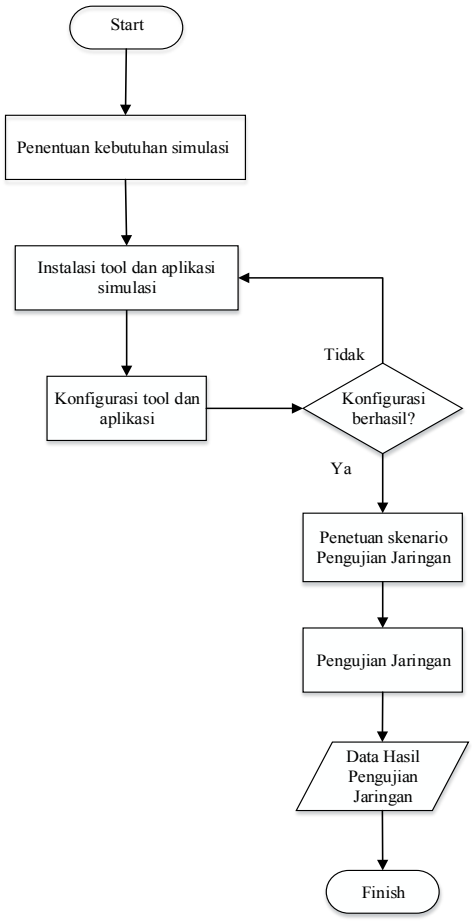
Mininet adalah sebuah emulator untuk membuat prototype jaringan berskala besar secara cepat pada sumberdaya yang terbatas (seperti pada *single* komputer atau laptop maupun Virtual Machine) [7]. Mininet diciptakan dengan tujuan untuk mendukung riset di bidang SDN dan OpenFlow. Emulator Mininet memungkinkan kita untuk menjalankan sebuah kode secara interaktif di atas laptop atau di atas virtual hardware, tanpa harus memodifikasi kode tersebut. Artinya kode simulasi sama persis dengan kode pada *real network environment*.

Mininet adalah solusi yang dianggap paling unggul dalam hal kemudahan penggunaan, performansi, akurasi, dan skalabilitas. Ia mampu menyediakan lingkungan yang realistis dan nyaman (convenience) dengan harga yang murah (*low cost*). Kita dapat menggunakan alternatif lain seperti hardware *test-bed* untuk simulasi jaringan, yang mana dapat berjalan cukup kencang dan akurat, namun harganya mahal dan harus di-*shared* dengan pengguna lain. Begitu pula, kita dapat menggunakan simulator yang harganya murah, namun seringkali kode simulasi akan harus dimodifikasi lagi bila akan dijalankan di *real network environment*.

Mininet menggunakan pendekatan *light-weight virtualization* menggunakan fitur virtualisasi level OS mencakup proses-proses dan namespace jaringan, sehingga memungkinkan dilakukannya simulasi jaringan dengan skala sangat besar (hingga ratusan node). Mininet dapat menciptakan jaringan virtual yang realistis, menjalankan real kernel, switch dan kode aplikasi, pada single machine (baik berupa physical machine, virtual machine, atau cloud). Mininet sangat berguna untuk pengembangan riset, pengajaran, serta penelitian.

III. ANALISIS DAN PERANCANGAN SISTEM

Secara umum, langkah-langkah yang dilakukan dalam penelitian ini dapat dilihat pada Gambar 2.



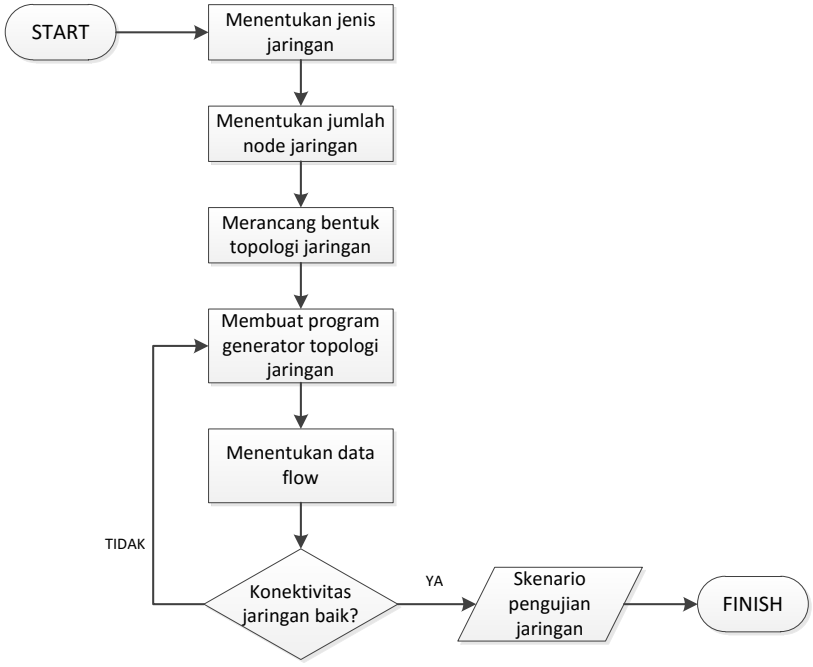
Gambar 2. Diagram alur perancangan sistem

Penjelasan langkah-langkahnya yaitu sebagai berikut:

1) Penentuan kebutuhan simulasi. Sesuai dengan konsep utama SDN yang memisahkan antara *control plane* dan *data plane*, maka host untuk menginstal controller SDN (POX) dipisahkan dengan host untuk menginstal Mininet.
2) Instalasi tool/aplikasi untuk simulasi. Instal Mininet dengan versi 2.2.1, instal paket protokol OpenFlow dengan versi 1.0.0, dan kemudian install controller POX dengan versi/*branch dart* 0.3.0. Instalasi Mininet dan OpenFlow dilakukan pada host Mininet. Sedangkan controller POX diinstall pada host controllerSDN.
3) Konfigurasi tool/aplikasi. Konfigurasikan server Mininet agar dapat terhubung dengan server controllerSDN. Indeks konfigurasi sudah berjalan dengan baik, yaitu saat server Mininet menjalankan program/suatu topologi jaringan, pada server seharusnya terdeteksi MAC Address dari switch-switch yang dijalankan oleh server Mininet.
4) Pengecekan konfigurasi. Seperti yang sudah dijelaskan di atas, pada server controllerSDN meng*invoke controller* POX. Sedangkan pada pihak server Mininet gunakan perintah *remote controller* yang artinya Mininet mengaktifkan *controller* namun bukan *local controller* Mininet. Apabila konfigurasi konvigurasi berhasil identitas-identias perangkat-perangkat jaringan seperti MAC Address terdeteksi oleh controller POX.
5) Penentuan skenario pengujian jaringan. Akan diuraikan lebih detail di bawah ini.
6) Pengujian jaringan dan pembahasan hasil uji. Akan diuraikan lebih detail di bawah ini.

Spesifikasi untuk server controllerSDN yang digunakan yaitu 2 processor @ 4 core (total 8 cores), memory RAM 8 GB, harddisk 30 GB, OS Linux Ubuntu 14.04.3 LTS. Adapun spesifikasi untuk server Mininet yaitu 4 processor @ 4 core (total 16 cores), memory RAM 32 GB, harddisk 50 GB, dan OS Linux Ubuntu 14.04.3 LTS.

Jaringan virtual SDN yang sudah dibangun selanjutnya akan diuji performansinya dengan skenario topologi jumlah switch yang berbeda-beda. Langkah-langkah yang dilakukan adalah sebagai berikut (lihat Gambar 3).



Gambar 3. Diagram alur skenario pengujian sistem

Berikut ini adalah penjelasan dari langkah-langkah yang dilakukan pada Gambar 3.

1) Menentukan jenis jaringan. Topologi jaringan yang digunakan dalam simulasi jaringan ini adalah topologi *full mesh*. Topologi full mesh adalah topologi jaringan di mana semua node/perangkat jaringan saling terhubung. Node dalam jaringan berupa komputer/host, switch dan controller. Pada penelitian ini, yang saling terhubung adalah antar switch dan controller, sedangkan host/komputer cukup dihubungkan ke switch masing-masing. Topologi *full mesh* cukup kompleks untuk diukur kinerjanya. Kapasitas channel komunikasi juga terjamin, karena memiliki derajat keterhubungan yang tinggi. Topologi *full mesh* relatif lebih mudah dikelola dan di-troubleshooting, serta memiliki kecepatan *sharing file* lebih tinggi karena terdapat jalur masing-masing untuk berkomunikasi.

2) Menentukan jumlah node. Jumlah controller yang digunakan dalam penelitian ini tetap satu, dengan jumlah switch dan host yang diperbanyak pada setiap skenario pengujian. Jumlah node disesuaikan dengan topologi *full mesh*.

3) Menentukan bentuk topologi. Dalam perancangan bentuk topologi *tools* yang digunakan adalah Visual Network Description (VND). Visual Network Description (VND) adalah sebuah *graphical user interface* untuk perancangan skenario jaringan SDN. VND mendukung perancangan topologi jaringan dengan berbagai topologi dan jumlah switch/node.

4) Membuat program untuk men-*generate* jaringan sesuai topologi yang diinginkan. Program dibuat dalam bahasa Python pada controller SDN. Program Python diimplementasikan sesuai rancangan topologi yang sudah dibuat menggunakan VND. Program tersebut juga berisi konfigurasi routing statis. Routing statis merupakan metode routing yang penjalurannya dilakukan secara manual dengan mengisi setiap entri dalam *forwarding table*.

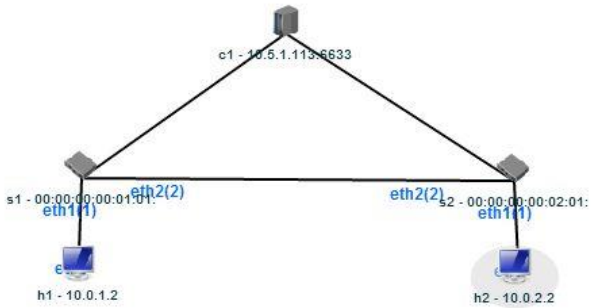
5) Menentukan *data flow*. Data flow merupakan *rule* di dalam jaringan yang menentukan pergerakan paket data yang akan dikirim. Data flow harus sesuai dengan routing statis yang dibuat, agar saat pengiriman paket data tidak terjadi kesalahan dan konektivitas jaringan tetap terjaga.

6) Mengecek konektivitas jaringan. Semua host di cek konektivitasnya agar dapat terbentuk komunikasi. Untuk mengecek konektivitas jaringan digunakan perintah pingall. Apabila terdapat sebagian host yang tidak dapat dijangkau/dihubungi, cek kembali program pembangun jaringannya, routing statis dan data flownya.

Skenario percobaan yang dibuat adalah topologi 2 switch, 4 switch, 8 switch dan topologi 16 switch dengan masing-masing maksimal bandwidth 1 GB/sec. Tujuan dari pengujian scalability ini yaitu agar dapat diketahui pola perilaku jaringan SDN dan bagaimana tingkat kemampuan SDN dalam menangani skala jaringan yang semakin lama semakin besar dan kompleks.

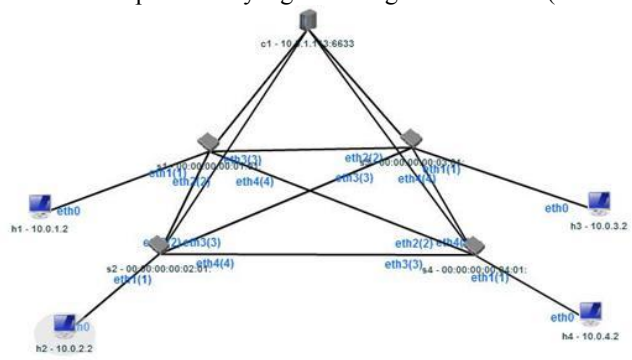
Berikut ini adalah beberapa skenario pengujian yang dilakukan dalam penelitian ini. Pada seluruh scenario, digunakan IP address private dengan subnet 10.0.0.0/16 untuk switch dan host, sedangkan untuk node controller digunakan IP private 10.5.1.113.

1) Topologi 2 switch: pada topologi ini digunakan 1 controller dan 2 switch, dan pada setiap switch terdapat 1 host yang dihubungkan ke switch (Gambar 4).



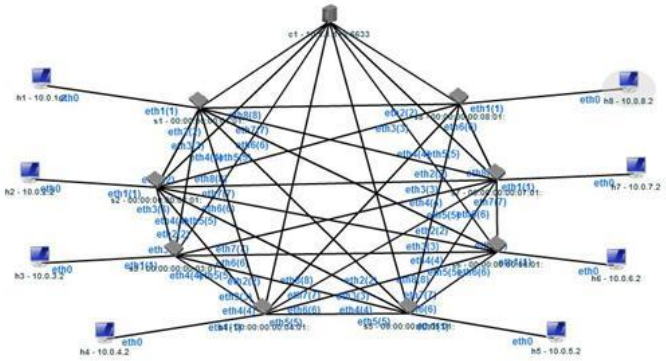
Gambar 4. Topologi 2-switch

2) Topologi 4 switch: pada topologi ini digunakan 1 controller dan 4 switch yang saling terhubung (graf lengkap), dan pada setiap switch terdapat 1 host yang dihubungkan ke switch (Gambar 5).



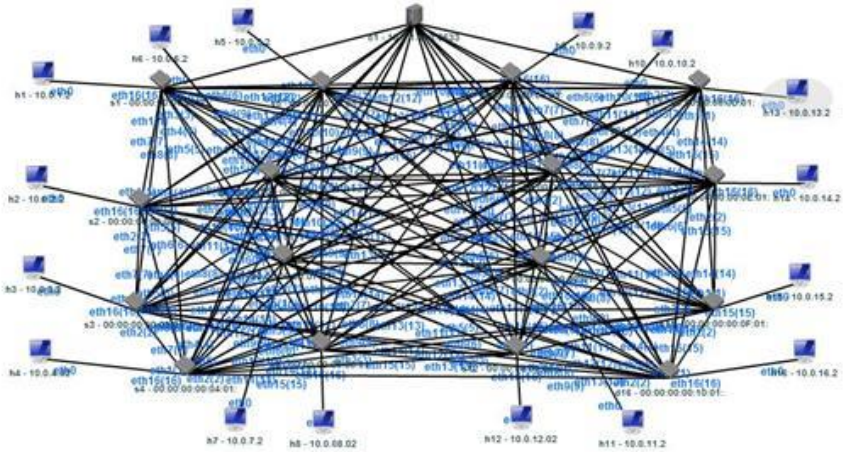
Gambar 5. Topologi 4-switch

3) Topologi 8 switch: pada topologi ini digunakan 1 controller dan 8 switch yang saling terhubung (graf lengkap), dan pada setiap switch terdapat 1 host yang dihubungkan ke switch (Gambar 6).



Gambar 6. Topologi 8-switch

4) Topologi 16 switch: pada topologi ini digunakan 1 controller dan 16 switch yang saling terhubung (graf lengkap), dan pada setiap switch terdapat 1 host yang dihubungkan ke switch (Gambar 5).



Gambar 7. Topologi 16-switch

IV. PEMBAHASAN HASIL PENGUJIAN

Pengujian dilakukan untuk mengetahui perilaku jaringan virtual SDN serta performansinya, diukur dari delay, jitter, dan throughput. Tools/perintah yang digunakan untuk ini yaitu **ping** dan **iperf**. Perintah **ping** digunakan untuk mencari nilai delay dan jitter, sedangkan perintah **iperf** digunakan untuk mengukur throughput bandwidth throuhput yang diukur melalui port TCP dan port UDP.

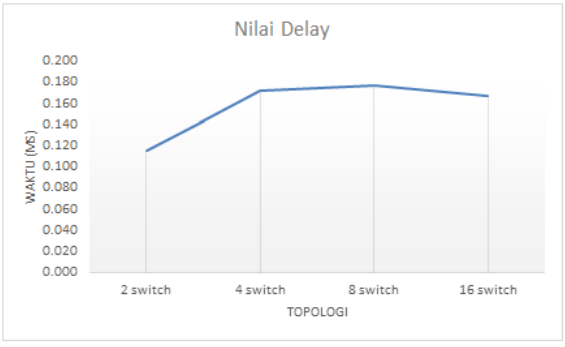
Pada Tabel I ditampilkan hasil pengujian dari seluruh skenario.

TABEL I.
HASIL PENGUJIAN SELURUH SKENARIO

Skenario Topologi	Delay (ms)	Jitter (ms)	Throughput (bits/sec)	
			TCP	UDP
2 switch	0,116	0,015	9,371	9,528
4 switch	0,171	0,111	9,339	9,534
8 switch	0,177	0,098	9,339	9,537
16 switch	0,167	0,154	9,347	9,537

A. *Pengukuran Delay*

Pola keseluruhan nilai delay dari hasil pengukuran cenderung tetap, namun ada beberapa yang mengalami kenaikan dan penurunan. Kenaikan terjadi dari topologi 2-switch ke topologi 4-switch, dan dari topologi 4-switch ke topologi 8-switch. Hal ini terjadi karena pada saat penentuan *data flow*, terdapat penambahan perangkat seperti perangkat jaringan yang dituju dan port-port yang tersedia pada suatu switch pada jaringan tersebut. Hal ini menyebabkan bertambahnya waktu pengecekan alamat fisik dari perangkat jaringan dan port-port yang tersedia. Sedangkan penurunan nilai delay terjadi pada topologi 8-switch ke topologi 16-switch. Hal ini terjadi karena waktu yang dibutuhkan pada saat pengecekan alamat fisik dari perangkat dan port-port yang tersedia cenderung sama (sudah dideskripsikan pada data flow yang sebelumnya). Berdasarkan standard rekomendasi ITU-T (yaitu 100 ms), nilai delay yang didapat masih dalam kategori baik karena memenuhi standard rekomendasi tersebut. Nilai delay paling tinggi yaitu pada topologi 8-switch yaitu 0,177 ms, juga masih berada di bawah 1ms. Untuk lebih jelasnya dapat dilihat di Gambar 8.

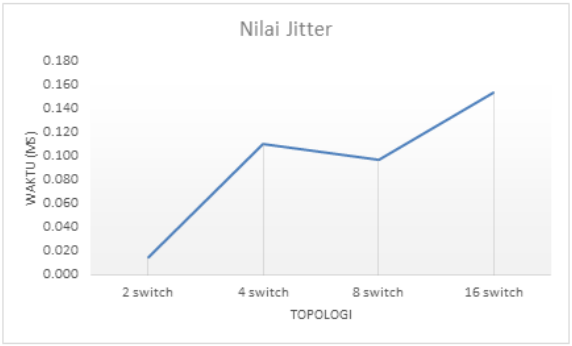


Gambar 8. Hasil pengukuran nilai delay

B. *Pengukuran Jitter*

Pola keseluruhan nilai jitter cenderung meningkat. Dapat dilihat bahwa terjadi peningkatan nilai jitter dari topologi 2-switch ke topologi 4-switch, serta dari topologi 8-switch ke topologi 16-switch. Hal ini menunjukkan

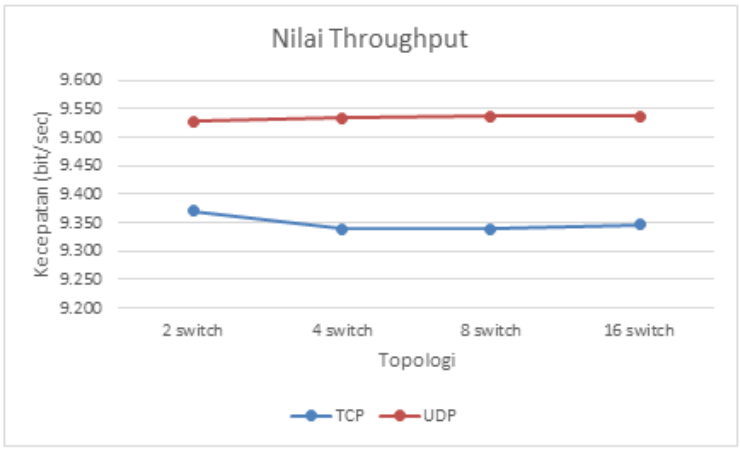
kenaikan jumlah switch berbanding lurus dengan kenaikan nilai jitter. Namun ada penurunan yang terjadi dari topologi 4-switch ke topologi 8-switch, yang diakibatkan karena pengaruh dari waktu komunikasi (nilai delay) itu sendiri. Sedangkan topologi 16-switch mempunyai nilai jitter paling tinggi, hal itu menunjukkan pada topologi tersebut waktu delay yang diperlukan untuk tiap-tiap host sangat bervariatif. Namun nilai-nilai jitter tersebut masih memenuhi standard rekomendasi ITU-T yaitu dibawah 50 ms, bahkan rata-rata dari nilai jitter tersebut masih dibawah 1 ms. Untuk lebih jelasnya dapat dilihat di Gambar 9.



Gambar 9. Hasil pengukuran nilai jitter

C. *Pengukuran Throughput*

Nilai throughput untuk port TCP cenderung stabil, di mana throughput pada topologi 2-switch adalah yang paling tinggi. Hal ini dikarenakan pada topologi 2-switch hanya terdapat dua host dengan salah satunya menjadi client dan salah satunya menjadi server yang menjadikan transfer data lebih cepat. Sedangkan port UDP nilai throughputnya juga cenderung stabil dengan nilainya berkisar di 9,5. Namun dari semua topologi percobaan, port UDP adalah yang paling cepat dengan rata-rata nilai throughputnya 9,534, sedangkan rata-rata nilai throuhput dari port TCP adalah 9,349. Untuk lebih jelasnya dapat dilihat di Gambar 10.



Gambar 10. Hasil pengukuran nilai throughput

V. KESIMPULAN DAN SARAN

A. *Kesimpulan*

1) Topologi jaringan virtual berbasis SDN dapat dirancang dan dapat dijalankan dengan maksimum topologi 16 switch menggunakan sumber daya dengan spesifikasi RAM 32 GB dan CPU 16 Core.

2) Kinerja jaringan virtual berbasis SDN memiliki nilai delay yang tetap meskipun ada beberapa peningkatan sesuai dengan jumlah switch yang semakin meningkat, nilai jitter cenderung meningkat sesuai dengan jumlah switch, namun nilai delay dan jitter masih memenuhi standard rekomendasi ITU-T. Sedangkan untuk nilai throughput cenderung stabil baik untuk port TCP maupun UDP (untuk port UDP lebih tinggi).

3) Bentuk topologi dengan jumlah node yang telah dirancang dapat merepresentasikan bentuk topologi lainya dengan node yang lebih banyak.

B. *Saran*

1) Menambah jumlah node-node jaringan atau menambah skenario pengujian jaringan.

2) Membandingan lebih dari satu jenis topologi, misalkan membandingkan topologi tree atau fat-tree dengan topologi full mesh.

3) Pengukuran atau pengujian jaringan lebih dispesifikasikan kedalam bidang-bidang jaringan, contoh (Quality of Service) QoS, monitoring, traffic engineering dll.

REFERENSI

[1] Open Networking Foundation, “SDN Architecture”, Issue 1 (June 2014), ONF TR-502.
[2] Bob Lantz, Brandon Heller, Nick McKeown. “A Network in a Laptop: Rapid Prototyping for Software-Defined Networks”, ACM 978-1-4503-0409-2/10/10, 2010.
[3] Dushyant Arora, Diego Perez-Botero. “Live Migration of an Entire Software-Defined Network”,
[4] Nick Feamster, Jennifer Rexford, Ellen Zegura. “The Road to SDN An intellectual history of programmable networks”. ACMQueue Volume 11, Issue 12. December 2013.
[5] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, Jonathan Turner. “OpenFlow White Paper: Enabling Innovation in Campus Networks”. March 2008.
http://archive.openflow.org/documents/openflow-wp-latest.pdf
[6] Open Networking Foundation, “OpenFlow Switch Specification” Version 1.5.0 (December 2014), ONF TS-020. https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.0.noipr.pdf
[7] Karamjeet Kaur1, Japinder Singh, Navtej Singh Ghumman; “Mininet as Software Defined NetworkingTesting Platform”. International Conference on Communication, Computing & Systems (ICCCS), 2014.
[8] Eueung Mulyana. “Buku Komunitas SDN-RG”. Bandung. Published by GitBook.