Implementasi POX pada Perangkat Lunak Software-Defined Networking *Controller* untuk *Data Center* Berbasis *Container*

Dhanar Prayoga, Royyana Muslim Ijtihadie, dan Muchammad Husni
Departemen Teknik Informatika, Fakultas Teknologi dan Informasi Institut Teknologi Sepuluh Nopember
e-mail: roy@if.its.ac.id

***Abstrak*—Software-Defined Networking(SDN) adalah arsitektur jaringan yang memisahkan antara *control plane* dan *data plane* sebuah jaringan, sehingga memungkinkan pengendalian jaringan dari sebuah SDN *Controller*. SDN Memungkinkan perubahan bentuk jaringan dengan konfigurasi minimal oleh *administrator* jaringan. Hal ini membuat arsitektur SDN sangat cocok untuk digunakan pada jaringan yang selalu berubah – ubah, salah satunya adalah *data center virtual* berbasis *software container*. *Software Container* adalah sebuah perangkat lunak yang mampu menggantikan *virtual host* pada sebuah *virtual data center*. Pada makalah ini akan dibuat sebuah SDN *Controller* yang dapat digunakan pada jaringan fisik maupun *virtual*, dengan orientasi sebuah data center berbasis *software container*. SDN *Controller* akan dibangun menggunakan POX, sebuah platform pengembangan SDN Controller. Hasil uji coba menunjukkan bahwa sistem yang dibangun mampu diimplementasikan pada jaringan *virtual* maupun jaringan fisik, serta mampu menangani jaringan yang memiliki *looping*.**

***Kata Kunci*—Software-Defined Networking, *Software Container*, *Data Center*, POX.**

I. PENDAHULUAN

PERKEMBANGAN teknologi dewasa ini menunjukkan tren yang mengarah pada virtualisasi. Virtualisasi yang terjadi tidak hanya pada virtualisasi perangkat, namun juga virtualisasi jaringan (NV)[1] dan virtualisasi fungsi jaringan (NFV)[2] Virtualisasi memiliki banyak keuntungan dari berbagai sisi. Keuntungan, terutama dalam sisi ekonomi, membuat pasar memilih untuk memvirtualisasikan perangkat mereka ketimbang membeli dan memelihara perangkat secara tersendiri. Virtualisasi juga memiliki banyak permasalahan. Salah satu permasalahan yang dihadapi adalah permasalahan konfigurasi jaringan yang harus dilakukan setiap kali melakukan *deploy* perangkat *virtual*. Konfigurasi jaringan yang selalu berubah ini tidak memungkinkan network administrator untuk merubah konfigurasi jaringan setiap ada perangkat virtual baru, sehingga dapat disimpulkan dibutuhkan sebuah solusi yang mampu menjawab permasalahan ini.

Software-Defined Networking (SDN)[3] dapat menyelesaikan permasalahan yang dialami pada *virtual data center*. SDN adalah sebuah arsitektur jaringan yang memisahkan *data plane* dan *control plane*. Hal ini memungkinkan *Network Administrator* melakukan konfigurasi jaringan secara otomatis dan hanya mengakses sebuah SDN *Controller*, sehingga memudahkan *Network Administrator* dalam mengelola jaringan.

POX[4] adalah sebuah *platform* berbasis bahasa pemrograman Python yang digunakan untuk mengembangkan perangkat lunak SDN *Controller*. POX memiliki beberapa komponen yang dapat digunakan ulang untuk membuat SDN *Controller* sesuai dengan kebutuhan pengguna. Hal ini memungkinkan untuk pembuatan perangkat lunak SDN *Controller* yang cocok untuk sebuah jaringan spesifik, contohnya *virtual data center* berbasis *Software Container*.

Makalah ini menawarkan sebuah solusi pembangunan *SDN* Controller menggunakan teknologi POX yang akan berjalan diatas jaringan *virtual* berbasis Mininet[5]. Penulis berharap perangkat lunak *SDN Controller* dapat menyelesaikan permasalahan konfigurasi jaringan yang dihadapi. Sehingga dapat membantu *Network Administrator* untuk melakukan konfigurasi jaringan yang optimal guna tercapainya optimasi virtualisasi perangkat.

II. STUDI LITERATUR

A. POX

POX adalah sebuah *platform* pengembangan perangkat lunak SDN *Controller* berbasis bahasa pemrograman Python. POX memiliki beberapa komponen yang dapat digunakan dan digabung untuk membentuk SDN *Controller* sesuai kebutuhan pengguna.

B. Containernet

Containernet adalah modifikasi dari Mininet[5] yang mampu menggunakan *Software Container* sebagai *host*. Containernet menambahkan beberapa fungsionalitas dari Mininet yang terkait dengan *software container*, seperti pengaturan jaringan *container*, alokasi sumber daya *container*, penggantian topologi *on-the-fly*, kontrol trafik, dan lain lain. Containernet merupakan hasil penelitian Manuel Peuster, Holger Karl, dam Steven van Rossem yang berjudul "MeDICINE: Rapid Prototyping of Production-Ready Network Services in Multi-PoP Environments." Yang muncul di konferensi IEEE tentang *Network Function Virtualization and Software Defined Network* (NFV-SDN), 2016[6].

C. FLASK

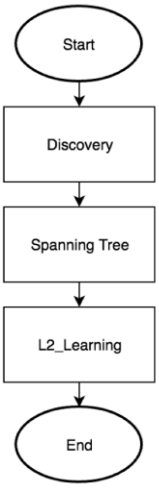
FLASK[7] adalah sebuah kerangka kerja mikro aplikasi web

yang dibangun dari Jinja2, Werkzeug, dan dilisensi dengan lisensi BSD[18]. Flask dibangun diatas bahasa pemrograman Python.

III. DESAIN UMUM SISTEM

Arsitektur sistem dapat dilihat pada *Gambar 2*. *Gambar 2* menunjukkan arsitektur sistem yang dibangun. Sistem yang dibangun ada didalam kotak "sistem", sedangkan entitas diluar "sistem" adalah entitas yang akan saling berinteraksi dengan sistem, namun tidak dibuat dalam makalah ini. SDN *Controller* yang dibangun akan mengatur fungsi routing pada jaringan fisik (MikroTik) maupun jaringan *virtual* (*virtual data center)*. Selain itu akan dibangun Web Server API yang dapat digunakan membangun website untuk membuat jaringan virtual berbasis Containernet[6] untuk membangun *virtual data center* berbasis *software container*. Web Sever API akan dibangun dengan bahasa pemrograman Python menggunakan kerangka kerja FLASK.

Desain SDN *Controller* dapat dilihat pada *Gambar 1*. SDN *Controller* dibangun dari komponen yang disediakan oleh platform POX. Platform yang digunakan ada 3, yaitu *Discovery*, *Spanning_Tree*, dan *L2_Learning*. Komponen *Discovery* digunakan untuk terus - menerus mendeteksi bentuk jaringan, sehingga SDN *Controller* dapat mengetahui bentuk jaringan beserta perubahannya. Hasil dari komponen *Discovery* akan digunakan oleh komponen *Spanning_Tree*, yang akan membentuk sebuah *spanning tree* dari jaringan dengan cara menutup port yang tidak dibutuhkan. Komponen ini digunakan untuk mengatasi *looping* pada jaringan, sebuah kasus yang sangat mungkin terjadi. Komponen terakhir yaitu komponen *L2_Learning*, yaitu komponen *forwarding*. Komponen ini dipilih karena jaringan yang selalu berubah membutuhkan algoritma forwarding yang memiliki *timeout*.



Gambar 1. Desain SDN *Controller*

Proses implementasi sistem pada jaringan *virtual* dilakukan untuk merubah bentuk perintah jaringan yang sinkron menjadi asinkron. Hal ini dicapai dengan mengimplementasikan *task scheduler* yaitu Celery[8]. Dalam mengimplementasikan Celery, dibutuhkan sebuah *message queue* seperti Redis[9].

Proses implementasi sistem pada jaringan fisik membutuhkan aktivasi fitur OpenFlow[10] pada *switch*. Hal ini dapat dilakukan dengan pemasangan perangkat lunak RouterOS ataupun OpenWRT pada *switch*.

Desain SDN *Controller* dapat dilihat pada *Gambar 1*. SDN *Controller* dibangun dari komponen yang disediakan oleh platform POX. Platform yang digunakan ada 3, yaitu *Discovery*, *Spanning_Tree*, dan *L2_Learning*. Komponen *Discovery* digunakan untuk terus - menerus mendeteksi bentuk jaringan, sehingga SDN *Controller* dapat mengetahui bentuk jaringan beserta perubahannya. Hasil dari komponen *Discovery* akan digunakan oleh komponen *Spanning_Tree*, yang akan membentuk sebuah *spanning tree* dari jaringan dengan cara menutup port yang tidak dibutuhkan. Komponen ini digunakan untuk mengatasi *looping* pada jaringan, sebuah kasus yang sangat mungkin terjadi. Komponen terakhir yaitu komponen *L2_Learning*, yaitu komponen *forwarding*. Komponen ini dipilih karena jaringan yang selalu berubah membutuhkan algoritma forwarding yang memiliki *timeout*.

IV. UJI COBA DAN PEMBAHASAN

A. Skenario Uji Coba

Pengujian sistem bertujuan untuk mengetahui performa dari sistem yang dibangun. Pengujian dilakukan dengan melakukan uji coba kecepatan jaringan, baik pada jaringan fisik maupun jaringan *virtual*.

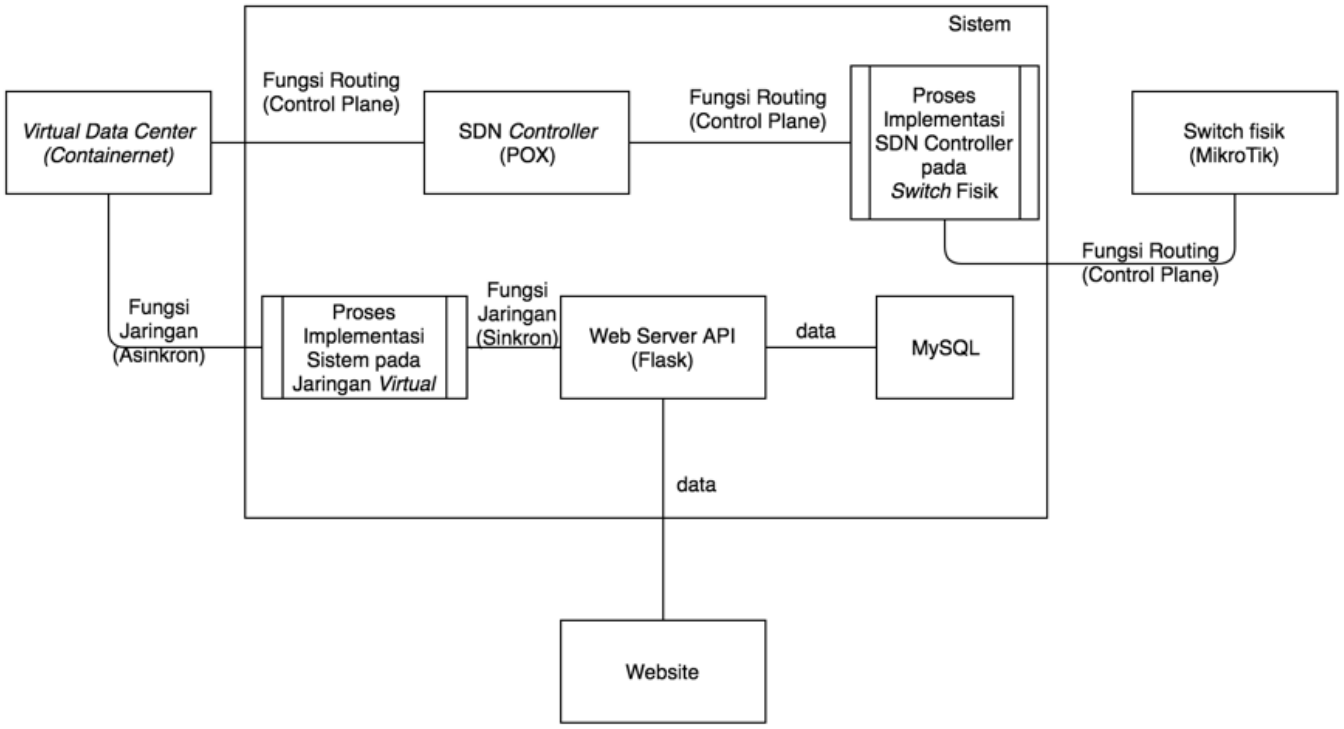
Tabel 1 merupakan skenario uji coba yang dilakukan. Skenario uji coba adalah perbedaan bentuk jaringan virtual maupun fisik, jumlah *switch* yang ada pada jaringan, maupun bentuk topologi jaringan terhadap kecepatan jaringan. Pengujian dilakukan dengan menggunakan kakas iperf[11]. Pada jaringan *virtual*, iperf server dan *client* dijalankan pada Ubuntu pada sebuah *Container*.

B. Hasil Uji Coba dan Pembahasan

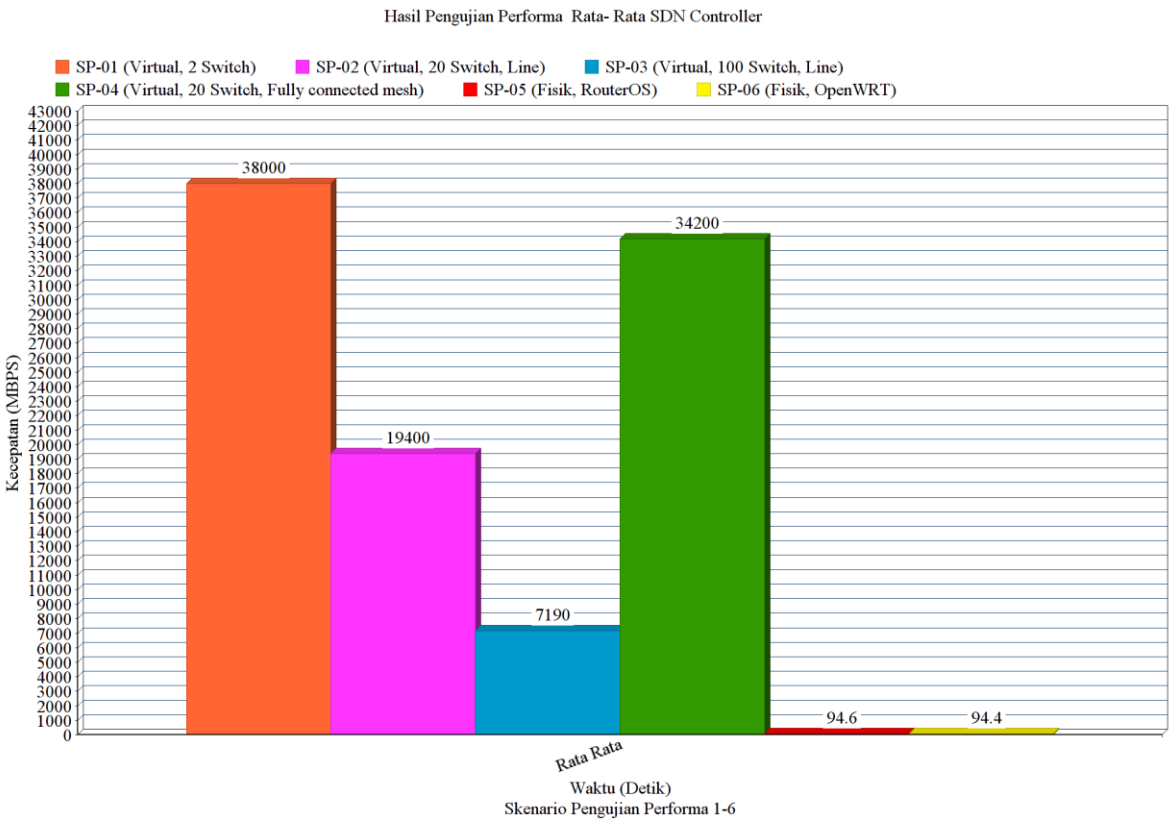
Gambar 3 adalah hasil dari masing masing pengujian. Terlihat bahwa terdapat perbedaan kecepatan antara jaringan *virtual* dan jaringan fisik. Selain itu, kecepatan jaringan juga dipengaruhi oleh banyaknya *hops* yang harus dilewati antara iperf *client* dengan iperf *server*. Keberhasilan skenario pengujian SP-04 menunjukkan bahwa sistem yang dibangun mampu diimplementasikan pada jaringan yang memiliki *looping*.

Tabel 1.
Skenario Pengujian Performa

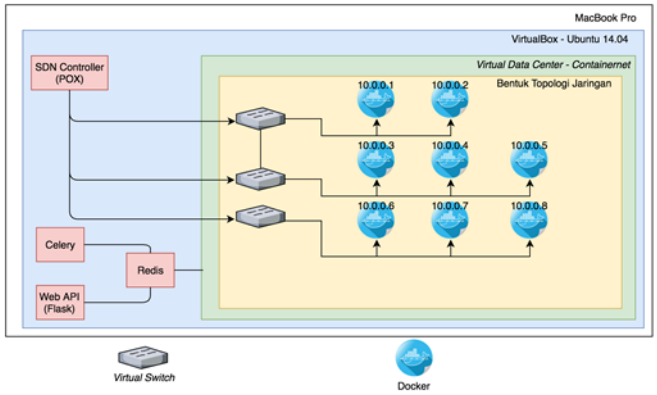
Skenario Pengujian	Bentuk jaringan	Perangkat Lunak *Switch*	Jumlah *Switch*	Topologi Jaringan
SP-01	*Virtual*	Containernet	2	*Line*
SP-02	*Virtual*	Containernet	20	*Line*
SP-03	*Virtual*	Containernet	100	*Line*
SP-04	*Virtual*	Containernet	20	*Fully Connected Mesh*
SP-05	Fisik	RouterOS	1	*Line*
SP-06	Fisik	OpenWRT	1	*Line*



Gambar 2. Desain arsitektur sistem



Gambar 3. Hasil uji coba



Gambar 4. Arsitektur Uji Coba

V. KESIMPULAN/RINGKASAN

Sistem yang dibangun berhasil diimplementasikan pada jaringan fisik maupun jaringan *virtual*. POX dapat digunakan untuk membangun SDN Controller yang diimplementasikan pada jaringan fisik maupun *virtual*. Jaringan *virtual* dapat dibangun menggunakan platform Containernet dan dapat digunakan untuk membangun *virtual data center* berbasis *software container*. Kecepatan jaringan bergantung pada bentuk jaringan yaitu fisik atau *virtual*. Selain itu, kecepatan jaringan juga dipengaruhi oleh banyaknya *hops* yang harus dilewati, sehingga bentuk topologi jaringan sangat mempengaruhi kecepatan jaringan.

DAFTAR PUSTAKA

[1] "What Is a Virtual Network?," *SDxCentral*, 10-May-2016. [Online]. Available: https://www.sdxcentral.com/sdn/network-virtualization/definitions/what-is-a-virtual-network/. [Accessed: 02-May-2017].

[2] "What's Network Functions Virtualization (NFV)?," *SDxCentral*, 25-Aug-2013. [Online]. Available: https://www.sdxcentral.com/nfv/definitions/whats-network-functions-virtualization-nfv/. [Accessed: 14-Jul-2017].

[3] "Software-Defined Networking (SDN) Definition - Open Networking Foundation." [Online]. Available: https://www.opennetworking.org/sdn-resources/sdn-definition. [Accessed: 14-Dec-2016].

[4] "noxrepo/pox," *GitHub*. [Online]. Available: https://github.com/noxrepo/pox. [Accessed: 14-Dec-2016].

[5] "Mininet Overview - Mininet," *Mininet*. [Online]. Available: http://mininet.org/overview/. [Accessed: 14-Dec-2016].

[6] M. Peuster, H. Karl, and S. van Rossem, "MeDICINE: Rapid prototyping of production-ready network services in multi-PoP environments," in *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, 2016, pp. 148–153.

[7] "Welcome | Flask (A Python Microframework)." [Online]. Available: http://flask.pocoo.org/. [Accessed: 03-May-2017].

[8] "Homepage | Celery: Distributed Task Queue." [Online]. Available: http://www.celeryproject.org/. [Accessed: 04-May-2017].

[9] "Redis." [Online]. Available: https://redis.io/. [Accessed: 04-May-2017].

[10] "OpenFlow - Open Networking Foundation." [Online]. Available: https://www.opennetworking.org/sdn-resources/openflow. [Accessed: 14-Jul-2017].

[11] "iPerf - The TCP, UDP and SCTP network bandwidth measurement tool." [Online]. Available: https://iperf.fr/. [Accessed: 30-May-2017].