Performa *Clustering Controller* pada Arsitektur *Software Defined Network*

Mokhamad Aguk Nur Anggraini[1], I Made Suartana[2]

[1,2]Jurusan Teknik Informatika/Teknik Informatika, Universitas Negeri Surabaya

[1]mokhamad.18081@mhs.unesa.ac.id

[2]imadesuartana@unesa.ac.id

***Abstrak*— *Software Defined Network* (SDN) saat ini telah menjadi sebuah paradigma baru dalam teknologi jaringan karena kemampuan manajemen jaringannya yang secara terpusat serta arsitektur yang berbasis *software* dan *programmable*. Dalam implementasinya, SDN memisahkan antara *control plane* dan *data plane, control plane* dilakukan oleh *controller* dan *data plane* dilakukan oleh *switch. Controller* menjadi pusat kontrol dari sebuah jaringan SDN, dengan begitu beban semua kontrol jaringan berada pada *controller*. Semakin besar jaringan yang ditangani oleh suatu *controller* mengakibatkan semakin besar beban *controller* tersebut, yang berakibat juga pada performa jaringan, sehingga perlu adanya solusi untuk dapat mengurangi beban jaringan yang ditangani *controller* agar performa jaringan tetap terjaga bahkan menjadi lebih baik. Pada penelitian ini dilakukan *clustering-controller* dengan menggunakan 3 *controller*, 1 sebagai *master controller* dan 2 sebagai *slave controller*, dengan begitu beban jaringan dapat dibagi pada 3 *controller*. Uji coba pada penelitian ini menggunakan Opendaylight sebagai *controller*. Dari hasil pengujian *end-to-end* QoS yang telah dilakukan menggunakan parameter *throughput, delay, jitter* dan *packet poss* menunjukkan bahwa jaringan SDN dengan *clustering controller* lebih baik dari pada jaringan SDN yang menggunakan *single-controller* dan *multi-controller* tanpa *clustering*. Pengujian dilakukan dengan variasi ukuran paket UDP sebesar 100Mb-15Gb. Rata-rata keseluruhan hasil pengujian parameter *throughput, delay, jitter* dan *packet loss* dari *clustering-controller* berturut-turut yaitu 12762,14Kbps, 16,954ms, 3,142ms, dan 0,08%. Sedangkan pada jaringan dengan *multi-controller* tanpa *clustering* yaitu 12327,80Kbps, 205,828ms, 16,968ms dan 2,21%, dan hasil pada jaringan dengan *single-controller* yaitu 12331,93Kbps, 207,087ms, 15,691ms dan 2,19%.**

Kata Kunci*— *Software Defined Network (SDN), Controller, Clustering-Controller, OpenDayLight.

I. PENDAHULUAN

Otomatisasi teknologi saat ini telah mengalami perkembangan yang sangat pesat, era industri 4.0 juga penuh akan syarat otomatisasi teknologi. Tidak dapat dipungkiri, hampir semua lini pekerjaan telah terdampak akan berkembangnya otomatisasi teknologi, sehingga semua pekerjaan menjadi berbasis *software* dan teknologi terpusat. Otomatisasi teknologi bertujuan untuk lebih memudahkan pekerjaan konvensional yang dulunya harus selalu bertatap muka dengan semua alat penunjang pekerjaan, dengan otomatisasi teknologi hal tersebut tidak menjadi hal utama lagi. Dengan menggunakan *software,* kontrol terhadap peralatan penunjang bisa dilakukan secara terpusat tanpa harus berinteraksi langsung dengan peralatan tersebut. Salah satu bidang yang telah terdampak dari otomatisasi teknologi yaitu bidang teknologi jaringan atau biasa disebut *network engineering.* Jaringan konvensional sudah mulai teragantikan dengan teknologi jaringan terbaru saat ini. Teknologi jaringan yang dulunya mengandalkan perangkat fisik sebagai perangkat utama untuk mengontrol jalannya sistem jaringan, sekarang sebagian besar telah digantikan dengan fungsi perangkat lunak sehingga tidak hanya mengandalkan perangkat fisik dan dapat dikontrol secara terpusat. Teknologi tersebut bernama *Software Defined Network* (SDN).

Software Defined Network (SDN) saat ini menjadi sebuah paradigma baru dalam teknologi jaringan karena kemampuan kecerdasan jaringannya yang secara terpusat [1]. SDN merupakan sebuah konsep evolusi baru dalam arsitektur jaringan yang sistem kerjanya memisahkan antara *control plane* dan *data plane.* Sedangkan konsep pada jaringan konvensional, *control plane* dan *data plane* tergabung dalam satu perangkat [2]. Berbeda dengan konsep yang ada pada SDN, *control plane* dipisahkan dari *data plane* sehingga mengadopsi sistem kontrol secara terpusat. Pemisahan *control plane* dan *data plane* yang dimaksud yaitu pemisahan *router* dan *switch* yang ada pada *data plane* yang biasanya berfungsi untuk *forwarding* paket data. Fungsi *data plane* dilakukan oleh *software* tertentu yang disebut sebagai *controller,* yang secara terpusat mengontrol sebuah jaringan. *Data plane* hanya meneruskan lalu lintas jaringan berdasarkan instruksi dari *control plane. Control plane* menjadi pusat kontrol dari SDN dengan mengamati *forwarding data* dari *data plane* serta aspek lain yang dibutuhkan dari sebuah fungsi jaringan. Karena *controller* berfungsi sebagai pusat kontrol jaringan SDN, maka *controller* menjadi salah satu faktor utama dalam baik atau buruknya performa sebuah jaringan SDN. Menjaga performa sebuah jaringan merupakan hal yang sangat penting, karena dengan performa jaringan yang baik maka sistem yang dibangun akan berjalan dengan baik dan lancar. Selain itu, apabila *controller* mengalami kendala dan bahkan sampai *down* atau gagal fungsi maka jaringan yang dibawahnya akan terganggu, sehingga sangat penting untuk menjaga stabilitas

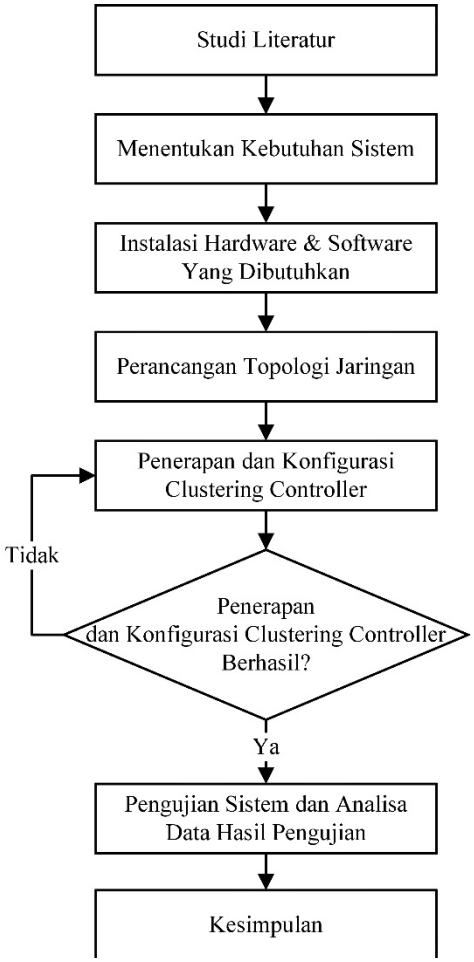
dan meningkatkan performa jaringan dengan cara mengoptimalkan kinerja dari sebuah *controller.*

Pada penelitian [3], yaitu melakukan perbandingan performa dari beberapa *controller(*Floodlight, Maestro, Ryu, POX, dan ONOS*)* dengan tujuan untuk mengetahui *controller* dengan performa terbaik. Pada penelitian tersebut hanya menerapkan konfigurasi *controller* secara *default* dan tanpa adanya modifikasi konfigurasi tertentu pada *controller* yang diterapkan. Lalu penelitian [4] meneliti bagaimana cara untuk mengurangi beban kerja dari sebuah *controller.* Hasil yang didapatkan bahwa untuk meringankan beban kerja pada *controller* yaitu dengan cara mengoptimalkan kinerja *controller* yang dapat dilakukan yaitu menerapkan fitur yang ada pada *controller* yang salah satunya dengan cara membangun arsitektur *multi-controller*. Pada penelitian [5] meneliti tentang *high availability controller* yang dilakukan dengan cara *multi-controller* dengan bantuan *software* Heartbeat dan DRBD pada *controller* POX dan Opendaylight. Tujuan dari penelitian tersebut yaitu untuk menjaga ketersediaan jaringan serta memaksimalkan waktu *backup* dari *controller* dengan pengujian yang memperhatikan parameter *failover* dan *failback* dengan kesimpulan bahwa perbedaan hasil parameter *failover* dan *failback* tergantung pada jenis *controller* yang digunakan, semakin banyak fitur yang dijalankan sebuah *controller*, semakin lama waktu *downtime* yang dihasilkan. Hal tersebut terjadi karena dibutuhkan waktu tambahan untuk menjalankan fitur-fitur *controller* yang digunakan. Lalu pada penelitian [6] melakukan penelitian performa *high availability* pada *clustering-controller* dengan membandingkan OSCP (Opendaylight SDN *Controller Platform) clustering* dengan Opendaylight menggunakan Heartbeat-DRBD yang berfokus pada ketersediaan jaringan serta memaksimalkan waktu *backup* dari *controller* pada parameter *failover* dan *failback* serta parameter *delay, throughput* & *packet loss* pada kondisi *failover* dan *failback* dan hasil akhirnya yaitu lebih baik pada OSCP (Opendaylight SDN *Controller Platform*) *clustering.* Lalu pada penelitian [1] melakukan penelitian dengan judul *Distributed Controller Clustering in Software Defined Network.* Penerapan *clustering-controller* pada penelitian ini yaitu *multi-controller* berjumlah 3 *controller* dengan pengujian yang dilakukan yaitu perbandingan performa dari *clustering-controller* dengan *multi-controller* tanpa *clustering.* Parameter yang diujikan yaitu *delay* dan *packet loss* dengan menggunakan variasi ukuran paket dan jumlah *switch. Controller* yang digunakan yaitu *controller* ONOS dan *controller* HP Virtual Application Network. Hasil dari penelitian ini yaitu performa *clustering-controller* lebih baik daripada *controller* tanpa *clustering* [1].

Pada penelitian ini menggunakan *clustering-controller* dengan *multi-controller* yang terdiri dari 3 *controller,* 1 *controller* sebagai *master* dan 2 *controller* sebagai *slave.* Penelitian ini menggunakan *controller* Opendaylight untuk membangun sistem *clustering-controller* dengan tujuan untuk menjaga stabilitas dan meningkatkan performa jaringan. Performa jaringan diukur berdasarkan uji *end-to-end* QoS menggunakan parameter *throughput, delay, jitter* dan *packet loss.* Pengujian performa jaringan dilakukan menggunakan 3 skenario topologi jaringan yaitu topologi SDN menggunakan *clustering-controller,* topologi *single-controller,* dan topologi *multi-controller* tanpa *clustering*. Pengujian performa dilakukan menggunakan *software* D-ITG pada parameter *throughput, delay, jitter* dan *packet loss* dengan variasi ukuran paket pada setiap skenario topologi jaringan.

II. METODOLOGI PENELITIAN

Berikut ini merupakan alur penelitian dari *clustering-controller* pada arsitektur *Software Defined Network* (SDN).



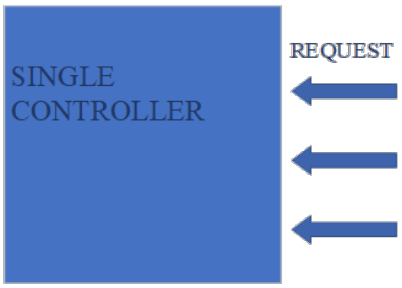
Gbr. 1 Alur penelitian

Tahapan dari alur penelitian seperti yang digambarkan pada Gbr. 1 yaitu studi literatur terkait penelitian dengan topik yang sama. Identifikasi masalah yang terkait performa dari *clustering-controller.* Permasalahan pada penelitian ini yaitu bagaimana performa jaringan dengan penerapan *clustering-controller*. Selanjutnya menentukan kebutuhan perangkat keras dan perangkat lunak yang akan digunakan pada penelitian ini. Perangkat keras yang dibutuhkan yaitu empat *virtual* komputer (tiga digunakan sebagai *controller* dan satu sebagai *resource*

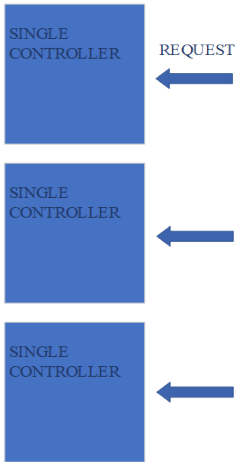
untuk menjalankan simulasi jaringan). Perangkat lunak yang dibutuhkan antara lain *controller* Opendaylight, *emulator* Mininet sebagai *software* emulasi topologi jaringan, sistem operasi Ubuntu, *Distributed Internet Traffic Generator* atau biasa disebut D-ITG sebagai *software* untuk menguji performa jaringan. Tahap selanjutnya melakukan instalasi *hardware* dan *software* yang telah disiapkan dengan baik dan benar. Setelah itu menentukan rancangan topologi jaringan yang akan digunakan. Setelah perancangan topologi, selanjutnya yaitu penerapan dan konfigurasi dari *clustering-controller* pada 3 *controller*. Setelah berhasil konfigurasi *clustering-controller* selanjutnya yaitu pengujian sistem dan analisis data dari hasil pengujian sistem yang telah dilakukan. Pengujian performa dilakukan dengan cara *end-to-end* QoS dengan parameter uji meliputi *throughput, delay, jitter* dan *packet loss.* Dari pengujian dan analisis data tersebut maka akan dapat dijadikan kesimpulan akhir dari penelitian yang telah dikerjakan.

A. *Arsitektur Sistem*

Pada arsitektur *single-controller,* pusat kontrol dari jaringan hanya terpusat pada 1 *controller,* begitu juga dengan *multi-controller* tanpa *clustering.* Berikut ilustrasi dari *single-controller* pada Gbr. 2 dan *multi-controller* tanpa *clustering* pada Gbr. 3.

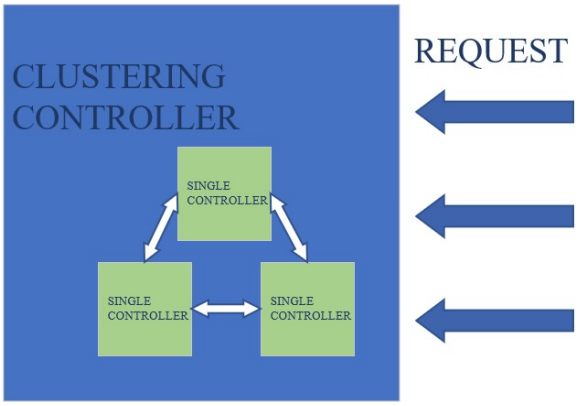


Gbr. 2 Ilustrasi *single-controller*



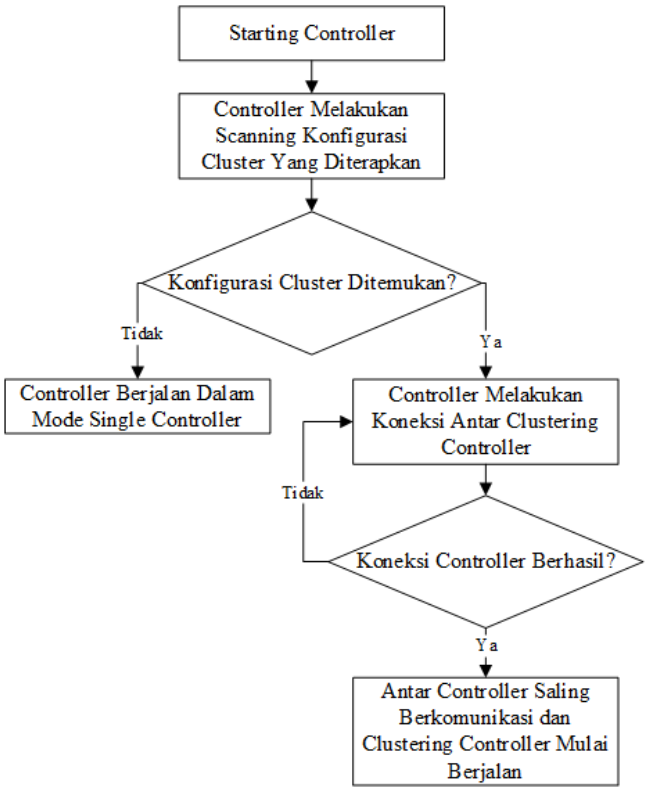
Gbr. 3 Ilustrasi *multi-controller* tanpa *clustering*

Pada arsitektur *multi-controller* tanpa *clustering* masing-masing *controller* tidak saling terhubung dan tidak ada proses bertukar data, sehingga *request* data dari *data plane* hanya di tangani oleh masing-masing *controller.* Sedangkan pada arsitektur *clustering-controller, controller* saling terhubung dan saling bertukar data satu sama lain.



Gbr. 4 Ilustrasi *clustering-controller*

Pada Gbr. 4 merupakan ilustrasi dari *clustering-controller.* Pada arsitektur *clustering-controller,* masing-masing *controller* terhubung dan saling bertukar data untuk menangani beban jaringan, dengan 1 *controller* sebagai *master* dan 2 *controller* sebagai *slave.* Pada penelitian ini, jumlah *controller* yang digunakan dalam *clustering* sebanyak 3 *controller.* Alasan penggunaan 3 *controller* karena pada Opendaylight *clustering-controller* minimal harus menggunakan 3 *controller*, apabila kurang dari 3 *controller* fitur *clustering* tetap dapat berjalan namun apabila salah satu *controller* mengalami *down* maka *controller* tidak akan berjalan [7].



Gbr. 5 Cara kerja *clustering-controller*

Pada Gbr. 5 merupakan cara kerja dari *clustering-controller.* Sebelum *controller* dijalankan, konfigurasi *clustering-controller* harus sudah diterapkan pada masing-masing *controller.* Sehingga saat masing-masing *controller* mulai berjalan, *controller* akan melakukan *scanning* dari konfigurasi *clustering*. Apabila konfigurasi *clustering* ditemukan maka *controller* akan mengeksekusi konfigurasi tersebut dengan melakukan koneksi pada masing-masing *controller* yang tergabung dalam *cluster.* Apabila masing-masing *controller* telah berhasil melakukan koneksi maka *clustering-controller* akan berjalan. Namun apabila konfigurasi *clustering* tidak ditemukan, maka *controller* akan berjalan pada mode *single-controller.*

Pada Opendaylight, *clustering-controller* berjalan melalui arsitektur Akka. Dalam penerapannya, Akka berfungsi sebagai penghubung komunikasi data antar *controller* berdasarkan konfigurasi pada *script akka.conf* pada Opendaylight.

```
 GNU nano 2.5.3              File: akka.conf


odl-cluster-data {
  akka {
    remote {
      netty.tcp {
        hostname = "192.168.43.160"
        port = 2550
      }
    }

    cluster {
      seed-nodes = ["akka.tcp://opendaylight-cluster-data@192.168.43.160:2550",
                    "akka.tcp://opendaylight-cluster-data@192.168.43.45:2550",
                    "akka.tcp://opendaylight-cluster-data@192.168.43.2:2550"]

      roles = [
        "member-1"
      ]
```

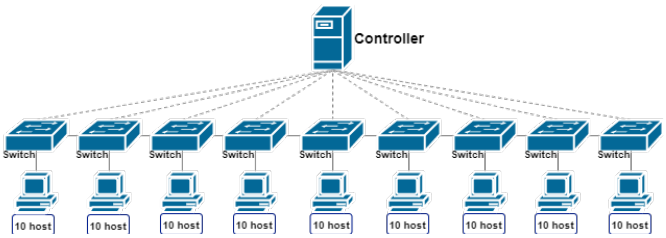
Gbr. 6 *Script akka.conf*

Pada Gbr. 6 merupakan contoh *script* dari *akka.conf*. Penjelasan dari *script* tersebut yaitu *hostname* merupakan *ip address* dari *controller* yang dikonfigurasi, *port 2550* merupakan *port clustering* pada Opendaylight, lalu *seed-nodes* yaitu *domain* yang menghubungkan antar *cluster node* yang berisi *ip address* dari masing-masing *controller,* serta *roles* merupakan nama atau *alias* dari *controller*. *Script* tersebut diterapkan di semua *cluster controller* dengan memperhatikan dan menyesuaikan *ip address* tiap *controller* pada bagian *hostname* serta *roles* sebagai nama atau *alias* harus berbeda pada tiap *controller.* Dengan begitu masing-masing *controller* Opendaylight dapat saling terhubung dan saling bertukar data. Sedangkan pada *controller* tanpa *clustering,* tidak terdapat konfigurasi Akka yang diterapkan sehingga *controller* hanya bertanggung jawab atas beban jaringan sendiri, tidak ada *controller* yang saling berkomunikasi dan bertukar data.

B. *Perancangan Topologi*

Penelitian ini menggunakan tiga jenis topologi jaringan yang dibangun menggunakan Mininet yaitu topologi jaringan menggunakan *single-controller, multi-controller* tanpa *clustering,* dan *multi-controller* dengan *clustering* atau *clustering-controller.* Berikut rancangan topologi jaringan yang digunakan:

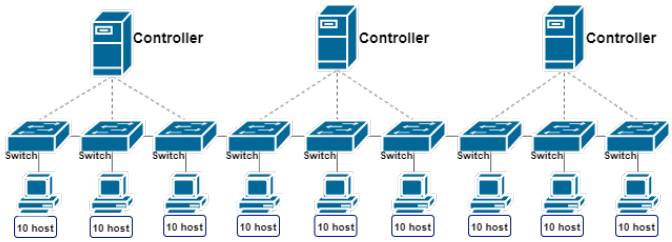
1) Topologi *Single-Controller:*



Gbr. 7 Topologi *single-controller*

Pada Gbr. 7 merupakan topologi yang dibangun menggunakan *single-controller.* Terdiri dari 9 *switch,* semua *switch* terhubung pada 1 *controller* dan masing-masing *switch* saling terhububng dengan membawahi 10 *host* pada tiap *switch.*

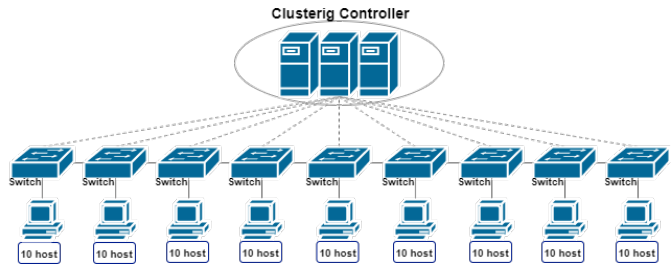
2) Topologi *Multi-Controller* Tanpa *Clustering:*



Gbr. 8 Topologi *multi-controller* tanpa *clustering*

Pada Gbr. 8 merupakan topologi yang dibangun menggunakan *multi-controller* tanpa *clustering.* Terdiri dari 3 *controller,* masing-masing *controller* membawahi 3 *switch* dengan total 9 *switch* dan semua *switch* saling terhubung dan membawahi 10 *host*.

3) Topologi *Clustering-Controller:*



Gbr. 9 Topologi *clustering-controller*

Pada Gbr. 9 merupakan topologi yang dibangun menggunakan *clustering-controller.* Terdiri dari 3 *controller,* 1 sebagai *master* dan 2 sebagai *slave.* Semua *switch* terhubung pada *master controller* dengan total 9 *switch* dan masing-masing *switch* saling terhubung dan membawahi 10 *host* pada tiap *switch.*

C. *Skenario Pengujian*

Pengujian performa jaringan dilakukan pada 3 topologi yang sudah dirancang. Pengujian akan dilakukan dengan cara *end-to-end* QoS menggunakan *software Distributed Internet Traffic Generator* (D-ITG) berdasarkan parameter *throughput, delay, jitter dan packet loss.* Berikut penjelasan dari parameter pengujian yang digunakan:

1) *Throughput*: *Throughput* merupakan nilai kecepatan transfer data dengan satuan *bit per second* (bps) yang dihitung berdasarkan total packet yang sukses diterima pada interval waktu tertentu lalu dibagi dengan interval waktu tersebut [8]. Berikut rumus perhitungan *trhoughput* yang ditunjukkan pada persamaan (1).

$$Throughput = \frac{\text{Paket data diterima}}{\text{Waktu pengiriman}} \quad (1)$$

2) *Delay*: *Delay* atau juga disebut *latency* merupakan waktu jeda yang dibutuhkan data untuk sampai ke tujuan dari suatu *node* asal ke *node* tujuan yang dipengaruhi oleh jarak, media fisik, dan *congestion* [8]. Berikut rumus perhitungan *delay* yang ditunjukkan pada persamaan (2).

$$Delay = \frac{\text{Total } delay}{\text{Total paket yang diterima}} \quad (2)$$

3) *Jitter*: *Jitter* merupakan jumlah variasi *delay* yang disebabkan oleh variasi-variasi panjang antrian paket dalam waktu pengolahan paket dan waktu penghimpunan ulang paket-paket diakhir perjalanan *jitter* [8]. Berikut rumus perhitungan *jitter* yang ditunjukkan pada persamaan (3).

$$Jitter = \frac{\text{Total variasi } delay}{\text{Total paket yang diterima}} \quad (3)$$

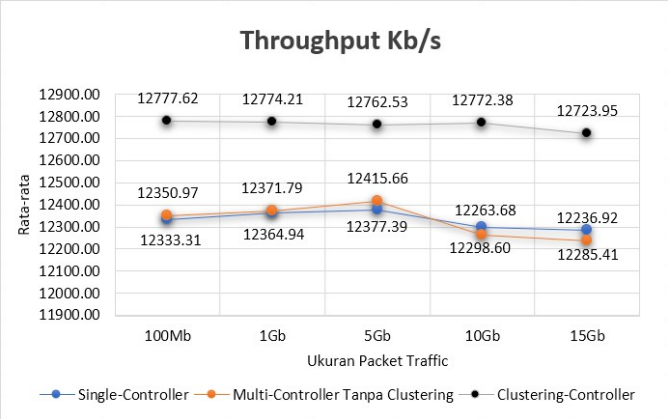
4) *Packet Loss*: *Packet loss* merupakan total keseluruhan paket yang hilang atau tidak sampai ke *node* tujuan yang disebabkan adanya *collison* dan *congestion* pada jaringan [8]. Berikut rumus perhitungan *packet loss* yang ditunjukkan pada persamaan (4).

$$PL = \left(\frac{\text{data yg dikirim} - \text{data yg diterima}}{\text{paket data yang dikirim}}\right) \times 100\% \quad (4)$$

Pengujian *throughput, delay, jitter* dan *packet loss* dilakukan dengan cara mengirim sejumlah paket dari *host* pada *switch* pertama menuju *host* pada *switch* terakhir dan dilakukan pada semua jenis rancangan topologi jaringan. Pengiriman paket dilakukan menggunakan *software* D-ITG dengan mengirim paket UDP dengan jumah 25 paket dengan variasi ukuran paket mulai dari 100Mb, 1Gb, 5Gb, 10Gb, dan 15Gb. Paket dikirimkan selama durasi 10 detik dan pengujian dilakukan 10 kali di setiap variasi ukuran paket, lalu diambil rata-rata dari 10 kali pengujian tiap variasi data tersebut. Pada akhirnya akan diambil rata-rata keseluruan dari semua variasi ukuran paket yang diujikan dan selanjutnya akan dibandingkan dari hasil pengujian dengan skenario SDN *single-controller, multi-controller* tanpa *clustering* dan *clustering-controller.*

III. HASIL DAN PEMBAHASAN

A. *Hasil Pengujian Throughput*



Gbr. 10 Grafik hasil pengujian *throughput*

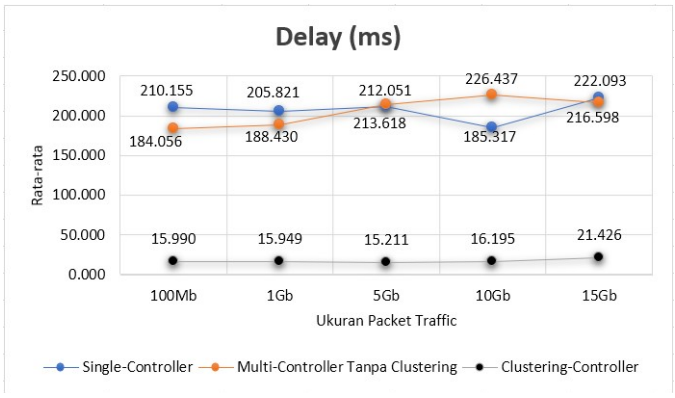
Pada Gbr. 10 merupakam grafik perbandingan hasil pengujian dari parameter *throughput.* Untuk parameter *throughput,* arsitektur *clustering-controller* lebih baik daripada penggunaan *single-controler* dan *multi-controller* tanpa *clustering.* Data hasil pengujian *throughput* lebih jelas dapat dilihat pada Tabel I.

TABEL I
DATA HASIL PENGUJIAN *THROUGHPUT*

Throughput (Kb/s)			
Data	Single-Controller	Multi-Controller tanpa Clustering	Clustering-Controller
100Mb	12333,31	12350,97	12777,62
1Gb	12364,94	12371,79	12774,21
5Gb	12377,39	12415,66	12762,53
10Gb	12298,60	12263,68	12772,38
15Gb	12285,41	12236,92	12723,95
Rata – rata	12331,93	12327,80	12762,14

Pengujian *throughput* dilakukan dengan 10 kali pengujian pada setiap variasi ukuran paket 100Mb, 1Gb, 5Gb, 10Gb, dan 15Gb. Dalam pengujian performa jaringan pada parameter *trhoughput,* semakin besar nilai *throughput* yang didapat maka semakin baik performa dari jaringan tersebut. Berdasarkan pada Tabel I dapat dilihat bahwa performa *clustering-controller* mendapatkan hasil tertinggi, dibandingkan pada penggunaan *single-controller* dan *multi-controller* tanpa *clustering* hasil yang didapat hampir sama dengan performa dibawah *clustering-controller.* Nilai *throughput* yang dihasilkan pada penggunaan *clustering-controller* juga cukup stabil, sedangkan pada penggunaan *single-controller* dan *multi-controller* tanpa *clustering* nilai *throughput* yang dihasilkan tidak stabil. Rata-rata dari semua pengujian *throughput* tersebut yaitu *single-controller* sebesar 12331,93Kb/s, lalu *multi-controller* tanpa *clustering* sebesar 12327,80Kb/s, sedangkan *clustering-controller* sebesar 12762,14Kb/s.

B. *Hasil Pengujian Delay*



Gbr. 11 Grafik hasil pengujian *delay*

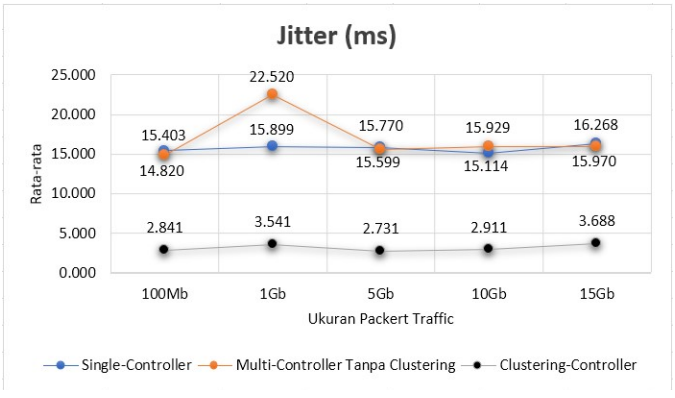
Pada Gbr. 11 merupakan grafik hasil pengujian dari parameter *delay*. Terjadi penurunan waktu *delay* yang cukup jauh pada topologi *clustering-controller*, hal tersebut menunjukkan bahwa hasil dari parameter *delay,* arsitektur *clustering-controller* lebih baik daripada penggunaan *single-controler* dan *multi-controller* tanpa *clustering*. Data hasil pengujian *delay* lebih jelas dapat dilihat pada Tabel II.

TABEL II
DATA HASIL PENGUJAIN *DELAY*

Delay (ms)			
Data	Single-Controller	Multi-Controller tanpa Clustering	Clustering-Controller
100Mb	210,155	184,056	15,990
1Gb	205,821	188,430	15,949
5Gb	212,051	213,618	15,211
10Gb	185,317	226,437	16,195
15Gb	222,093	216,598	21,426
Rata – rata	207,087	205,828	16,954

Pengujian *delay* dilakukan dengan 10 kali pengujian pada setiap variasi ukuran paket 100Mb, 1Gb, 5Gb, 10Gb, dan 15Gb. Dalam pengujian performa jaringan pada parameter *delay,* semakin redah waktu *delay* yang didapat maka semakin baik performa dari jaringan tersebut. Berdasarkan pada Tabel II dapat dilihat bahwa performa *clustering-controller* mendapatkan hasil *delay* terbaik dibandingkan penggunaan *single-controller* dan *multi-controller* tanpa *clustering* dengan *delay.* Nilai *delay* yang dihasilkan pada penggunaan *clustering-controller* juga cukup stabil, sedangkan pada penggunaan *single-controller* dan *multi-controller* tanpa *clustering* nilai *delay* yang dihasilkan tidak stabil. Rata-rata dari semua pengujian *delay* tersebut yaitu *single-controller* sebesar 207,087ms, lalu *multi-controller* tanpa *clustering* sebesar 205,828ms, sedangkan *clustering-controller* sebesar 16,954ms.

C. *Hasil Pengujian Jitter*



Gbr. 12 Grafik hasil pengujian *jitter*

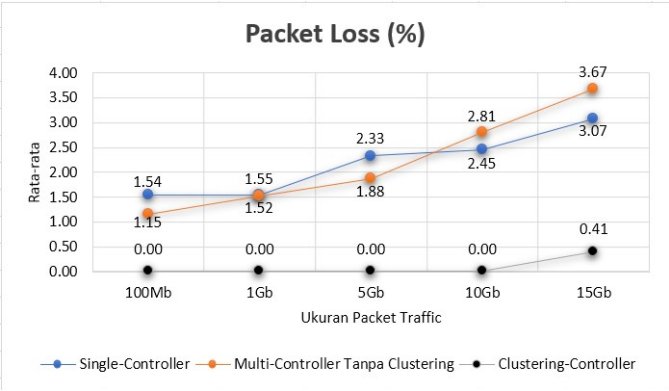
Pada Gbr. 12 merupakan grafik hasil pengujian dari parameter *jitter*. Terjadi penurunan waktu *jitter* yang cukup jauh pada *clustering-controller,* hal tersebut menunjukkan bahwa pada parameter *jitter,* arsitektur *clustering-controller* lebih baik daripada penggunaan *single-controller* dan *multi-controller* tanpa *clustering*. Data hasil pengujian *jitter* lebih jelas dapat dilihat pada Tabel III.

TABEL III
DATA HASIL PENGUJIAN *JITTER*

Jitter (ms)			
Data	Single-Controller	Multi-Controller tanpa Clustering	Clustering-Controller
100Mb	15,403	14,820	2,841
1Gb	15,899	22,520	3,541
5Gb	15,770	15,599	2,731
10Gb	15,114	15,929	2,911
15Gb	16,268	15,970	3,688
Rata – rata	15,691	16,968	3,142

Pengujian *jitter* dilakukan dengan 10 kali pengujian pada setiap variasi ukuran paket 100Mb, 1Gb, 5Gb, 10Gb, dan 15Gb. Dalam pengujian performa jaringan pada parameter *jitter,* semakin rendah waktu *jitter* yang didapat maka semakin baik performa dari jaringan tersebut. Berdasarkan pada Tabel III dapat dilihat bahwa performa pada penggunaan *clustering-controller* mendapatkan hasil *jitter* terbaik dibandingkan penggunaan *single-controller* dan *multi-controller* tanpa *clustering.* Nilai *jitter* yang dihasilkan pada penggunaan *clustering-controller* juga cukup stabil, sedangkan pada penggunaan *single-controller* dan *multi-controller* tanpa *clustering* nilai *jitter* yang dihasilkan tidak stabil. Rata-rata dari semua pengujian *jitter* tersebut yaitu topologi *single-controller* sebesar 15,691ms, lalu *multi-controller* tanpa *clustering* sebesar 16,968ms, sedangkan *clustering-controller* sebesar 3,142ms.

D. *Hasil Pengujian Packet Loss*



Gbr. 13 Grafik hasil pengujian *packet loss*

Pada Gbr. 13 merupakan grafik hasil pengujian dari parameter *packet loss*. Terjadi penurunan waktu *packet loss* yang cukup jauh dan sangat stabil pada *clustering-controller,* hal tersebut menunjukkan bahwa pada parameter *packet loss,* arsitektur *clustering-controller* lebih baik daripada penggunaan *single-controler* dan *multi-controller* tanpa *clustering*. Data hasil pengujian *packet loss* lebih jelas dapat dilihat pada Tabel IV.

TABEL IV
DATA HASIL PENGUJIAN *PACKET LOSS*

Packet Loss (%)			
Data	Single-Controller	Multi-Controller tanpa Clustering	Clustering-Controller
100Mb	1,54	1,15	0,00
1Gb	1,55	1,52	0,00
5Gb	2,33	1,88	0,00
10Gb	2,45	2,81	0,00
15Gb	3,07	3,67	0,41
Rata – rata	2,19	2,21	0,08

Pengujian *packet loss* dilakukan dengan 10 kali pengujian pada setiap variasi ukuran paket 100Mb, 1Gb, 5Gb, 10Gb, dan 15Gb. Dalam pengujian performa jaringan pada parameter *packet loss,* semakin rendah persentase *packet loss* yang didapat maka semakin baik performa dari jaringan tersebut. Berdasarkan pada Tabel IV dapat dilihat bahwa performa pada penggunaan *clustering-controller* mendapatkan persentase nilai *loss* terendah, sedangkan pada penggunaan *single-controller* dan *multi-controller* tanpa *clustering* persentase nilai *loss* yang didapat cukup tinggi. *Packet loss* yang dihasilkan pada penggunaan *clustering-controller* juga stabil, dibandingkan *packet loss* dari penggunaan *single-controller* dan *multi-controller* tanpa *clustering*. Rata-rata dari semua pengujian *packet loss* tersebut yaitu *single-controller* sebesar 2,19%, lalu *multi-controller* tanpa *clustering* sebesar 2,21%, sedangkan *clustering-controller* hanya sebesar 0,08%.

IV. KESIMPULAN

Pada parameter *throughput*, rata-rata pengujian dari *clustering-controller* yaitu sebesar 12762,14Kb/s, sedangkan *single-controller* dan *multi-controller* tanpa *clustering* hanya mendapatkan *throughput* sebesar 12331,93Kb/s dan 12327,80Kb/s. Pada parameter *delay, jitter* dan *packet loss* juga menunjukkan hasil yang lebih baik pada *clustering-controller.* Pada parameter *delay,* rata-rata pengujian dari *clustering-controller* yaitu hanya sebesar 16,954ms, sedangkan pada *single-controller* dan *multi-controller* tanpa *clustering* mendapatkan *delay* sebesar 207,087ms dan 205,828ms. Lalu pada parameter *jitter* rata-rata pengujian dari *clustering-controller* yaitu hanya sebesar 3,142ms, sedangkan pada *single-controller* dan *multi-controller* tanpa *clustering*

mendapatkan *jitter* sebesar 15,691ms dan 16,968ms. Pada parameter *packet loss,* rata-rata pengujian dari *clustering-controller* yaitu hanya sebesar 0,08%, sedangkan pada *single-controller* dan *multi-controller* tanpa *clustering* mendapatkan *packet loss* sebesar 2,19% dan 2,21%.

Dari semua skenario pengujian yang telah dilakukan, maka dapat dijadikan kesimpulan bahwa penerapan teknologi *clustering-controller* pada arsitektur *Software Defined Network* (SDN*)* dapat memberikan performa jaringan yang lebih baik dengan semua parameter pengujian yang telah dilakukan.

UCAPAN TERIMA KASIH

Puji syukur saya panjatkan kehadirat Allah SWT yang telah melimpahkan rahmat-Nya serta memberikan kemudahan dan kelancaran dalam pengerjaan penelitian dan jurnal ini. Serta semua pihak yang senantiasa membantu, memberi saran serta semangat hingga peneltian dan jurnal ini dapat terselesaikan dengan baik.

REFERENSI

[1] A. Abdelaziz, A. T. Fong, A. Gani, U. Garba, S. Khan, A. Akhunzada, H. Talebian & K.-K. R. Choo, "Distributed Controller Clustering in Software," *PLOS ONE,* vol. 12, no. 4, pp. 1-19, 2017.

[2] S. Khan, A. Gani, A. W. A. Wahab, A. Abdelaziz, K. Ko, M. K. Khan & M. Guizani, "Software-Definde Network Forensics: Motivation, Potential Locations, Requirements, and Challenges," *IEEE Network,* vol. 30, no. 6, pp. 6-13, 2016.

[3] M. W. Putra, E. S. Pramukantoro & W. Yahya, "Analisis Perbandingan Performansi Controller Floodlight, Maestro, RYU, POX, dan ONOS Dalam Arsitektur Software Defined Network (SDN)," *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer,* vol. 2, no. 10, pp. 3779-3787, 2018.

[4] C. Caba & J. Soler, "Mitigating the controller performance bottlenecks in software defined network," *Int. J. Communication Networks and Distributed Systems,* vol. 17, no. 3, pp. 275-295, 2016.

[5] M. Purwiadi, W. Yahya & A. Basuki, "High Availbility Controller Software Defined Network Menggunakan Heartbeat dan DRBD," *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer,* vol. 2, no. 8, pp. 2297-2306, 2018.

[6] A. R. D. Nugraha, R. M. Negara & D. D. Sanjoy, "High Availability Performance on OSCP Clustering with Heartbeat-DRBD," *Jurnal Infotel,* vol. 10, no. 3, pp. 149-156, 2018.

[7] Opendaylight Documentation, "Deployment Considerations : Multiple Node Opendaylight Clustering," Opendaylight, - - 2018. [Online]. Tersedia: https://nexus.opendaylight.org /content/sites/site/org.opendaylight.docs/master/userguide/manuals/userguide/bk-user-guide/content/_deployment_consider ations.html. [Diakses pada 19 Juni 2020].

[8] R. Wulandari, "Analisis QoS (Quality of Service) Pada Jaringan Internet (Studi Kasus : UPT Loka Uji Teknik Penambangan Jampang Kulon - LIPI)," *Jurnal Teknik Informatika dan Sistem Informasi,* vol. 2, no. 2, pp. 162-172, 2016.