ANALISIS PERFORMANSI *GATEWAY BALANCING* PADA *WIRELESS SOFTWARE DEFINED NETWORK* (WSDN)

PERFORMANCE ANALYSIS OF GATEWAY BALANCING ON WIRELESS SOFTWARE DEFINED NETWORK (WSDN)

Febri Surya Hadi[1], Sofia Naning Hertiana S.T., M.T.[2], Ridha Muldina Negara S.T., M.T.[3]

Prodi S1 Teknik Telekomunikasi, Fakultas Teknik Elektro, Universitas Telkom

[1]febri.surhadi@gmail.com, [2]sofiananing16@gmail.com, [3]ridhanegara@gmail.com

Abstrak

Saat ini sedang berkembang sebuah paradigma baru dalam jaringan komputer, yaitu *Software Defined Networking (SDN)*. SDN adalah sebuah arsitektur jaringan yang dengan kondisi *network control* terpisah dari *forwarding plane* dan dapat diprogram secara langsung, sehingga mampu menyediakan jaringan yang fleksibel, *programable*, *vendor-agnostic*, *cost efficient*. Namun, SDN yang tersedia saat ini kebanyakan masih digunakan dalam jaringan kabel dan bukan nirkabel.

Pada penelitian ini dilakukan simulasi yang mengintegrasikan SDN dengan *wireless network*. Pengintegrasian tersebut dilakukan dengan membuat suatu *Wireless Mesh Router* (WMR) yang di dalamnya terdapat OpenFlow *switch* untuk menghubungkan WMR dengan SDN *controller*. Pada *wireless* SDN, terdapat fungsi *gateway balancing* yang akan dieksekusi untuk menangani laju *traffic* sehingga akan didapatkan nilai *bandwith* yang lebih baik dibandingkan tanpa menggunakan *gateway balancing*.

Penelitian ini mengaplikasikan *wireless* SDN (WSDN) pada jaringan *wireless* sebagai *routing engine* untuk melakukan *gateway balancing* pada perutingan yang akan melalui *gateway* pada klien. Penelitian ini dilakukan dua skenario pengujian: i) *controller gateway balancing* dan ii) *controller gateway fault handling*. Perangkat lunak yang digunakan dalam penelitian ini antar lain Openvswitch, POX *controller*, OLSR *daemon*, dan *script* Python dan Bash yang tersedia. Simulasi dibuat dengan menggunkan Linux *container* (LXC), CORE, dan NS-3.

Kata kunci : ***SDN, WSDN, OpenFlow, Gateway balancing***

Abstract

In this time, a new paradigm has been developed in computer networks, it is called Software Defined Networking (SDN). SDN is a network architecture which separates the network controller from the forwarding plane and can be programmed directly, so it provides flexible, programmable, vendor-agnostic, cost efficient networks. However, the SDN that is currently available at this time is mostly still used in wired networks and not wireless.

In this project, I simulated the SDN that integrates within the wireless network. The integration is done by creating a Wireless Mesh Router (WMR) in which it is using the OpenFlow switch to connect the WMR with SDN controller. Gateway balancing function in SDN controller is excecuted for handling the traffic flows so we get better bandwidth values than without the gateway balancing activated.

This project applicates the wireless SDN (WSDN) as routing engine that do the gateway balancing in the traffic that is going through gateways on clients. This project has tested three scenarios: i) controller gateway balancing and ii) controller gateway fault handling. The softwares are used in this simulation are Openvswitch, POX controller, OLSR daemon, and Python and Bash scripts provided. The tools are used for the simulation are Linux containers (LXC), CORE, and NS-3.

Keywords: SDN, WSDN, OpenFlow, Gateway balancing

1. Pendahuluan

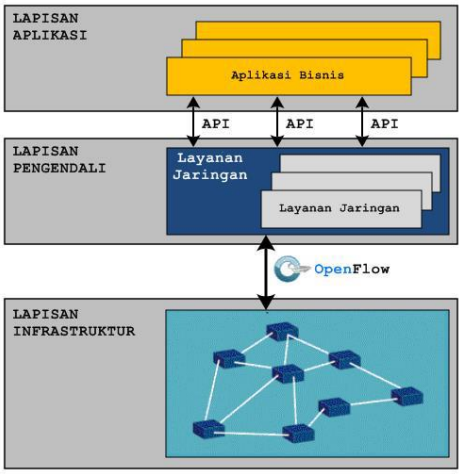
Perkembangan teknologi pada jaringan komunikasi nirkabel menuntut kita untuk terus berinovasi mengembangkankan teknologi terbaru. *Software Defined Network* (SDN) merupakan salah satu dari paradigma baru dalam dunia jaringan. Pada jaringan konvensional non-SDN, perangkat pengontrol (*control plane*) dan perangkat penerusan paket data (*forwarding plane*) bersatu dalam satu perangkat jaringan seperti *switch* atau *router*. Sedangkan, pada jaringan SDN memisahkan secara fisik antara penerus paket data yang berada pada perangkat *switch* atau *router* dengan *controller* yang dapat digunakan untuk mengontrol beberapa perangkat seperti merubah perutingan secara otomatis [6]. Pemasangan SDN pada jaringan *wireless* telah berhasil diterapkan pada penelitian sebelumnya [7]. *Software Defined Networking* dengan *OpenFlow* yang diterapkan pada jaringan *wireless* mengatur perutingan pada jaringan untuk melakukan *gateway balancing*, yaitu suatu sistem yang mengatur perutingan sehingga setiap paket yang ada akan melewati semua *gateway* yang tersedia secara merata sehingga akan memaksimalkan penggunaan *bandwidth* pada jaringan *wireless* dengan menggunakan SDN dan hasilnya akan didapatkan nilai *throughput* yang lebih baik [14].

Pada penelitian ini dilakukan skenario uji performansi *wireless* SDN (WSDN) dengan fungsi *gateway balancing*, yaitu pembagian *traffic* data secara merata pada tiap *gateway*, lalu membandingkannya dengan WSDN yang tanpa menggunakan *gateway balancing*. Selain itu pada penelitian ini juga dilakukan uji SDN *controller* dalam melakukan *self-healing* menangani *gateway failure*, yaitu bilamana ada salah satu *gateway* yang mati, maka SDN *controller* akan menginisiasikan perubahan perutingan. Metode ini dilakukan dengan sebuah simulasi pada jaringan *wireless* dengan SDN *controller* yang terhubung dengan *Openflow switch* pada *Wireless Mesh Router* (WMR). Parameter yang menjadi acuan dalam menetukan QoS pada WSDN ini antara lain *throughput, delay, jitter*, dan *packet loss*. Pengaplikasian SDN pada jaringan *wireless* diharapkan mampu meningkatkan efisiensi dan mendapatkan QoS yang lebih baik.

2. Dasar Teori

2.1 *Software Defined Network* (SDN)

Software Defined Network (SDN) merupakan salah satu dari paradigma baru dalam dunia jaringan. Pada jaringan konvensional perangkat pengontrol (*control plane*) dan perangkat penerusan paket data (*forwarding plane*) bersatu dalam satu perangkat jaringan seperti *switch* atau *router*. Sementara itu pada jaringan SDN memisahkan secara fisik antara penerus paket data yang berada pada perangkat *switch* atau *router* dengan pengontrol yang dapat digunakan untuk mengontrol beberapa perangkat sekaligus [6].



Gambar 2.1 Arsitektur *Software Defined Network* [2]

Jaringan SDN menciptakan arsitektur jaringan yang dinamis (dynamic), mudah diatur (*managable*), biaya yang efektif (*cost-effective*), dan mudah beradaptasi (*adaptable*). Dengan memisahkan antara pengontrol dan penerus paket data, otomatis pengontrol SDN dapat diprogram (*programable*) dan infrastruktur jaringan diabstraksi untuk pengaplikasian dan memberi layanan - layanan jaringan.

2.2 SDN yang dikembangkan pada jaringan *wireless*

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* atau *router*. Dalam sebuah jaringan, setiap *router* 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 [1].

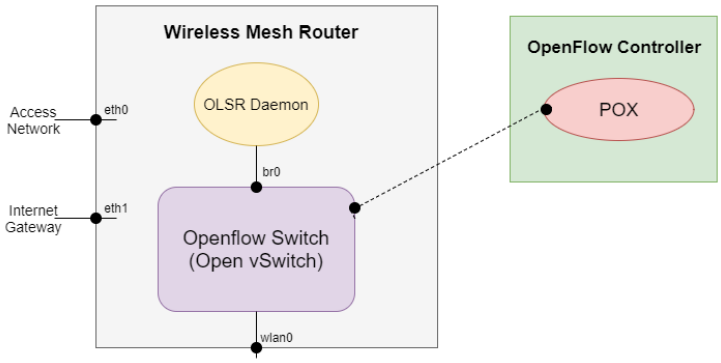
Pengembangan *wireless* SDN berbasis OpenFlow menemui beberapa masalah, seperti pengaturan dari kekuatan kerangka kontrol yang memungkinkan switch untuk berkomunikasi dengan *controller*, dan untuk menghadapi keadaan darurat di mana *controller* tidak tersedia, misalnya karena partisi jaringan atau kegagalan. Bahkan, VLAN tidak bisa digunakan untuk mendukung strategi pengendalian out-of-band dan mekanisme perutingan *wireless* biasanya tidak didasarkan pada mekanisme *layer* 2 (algoritma *spanning tree* atau *auto learning*), tetapi pada protokol perutingan *layer* 3 seperti OLSR [14].

Sebuah solusi untuk membuat *wireless network* OpenFlow dengan pengandali *out-of-band* diusulkan pada *paper* [7], menggunakan SSID yang berbeda untuk kontrol dan data jaringan. Pada paper tersebut, penggunaan praktis dari OpenFlow di skenario *wireless network* adalah bergantung pada kemampuan *driver wireless* untuk mendukung beberapa SSID. Namun, di dalam proposal ini, saya menggnakan arsitektur yang menggunakan satu SSID, *inband-control strategy*, dan mendukung *controller failure*. Kita menggunakan kontroler OpenFlow untuk merekayasa perutingan pada data menggunakan OLSR untuk mengatur *control-rules* yang digunakan oleh lalu lintas kontrol OpenFlow. Selain itu, OLSR juga digunakan untuk mendorong *emergency-rules*

pada *router* dengan mengatur rute lalu lintas data dalam kondisi darurat, yang mana kontroler OpenFlow gagal atau tidak bisa diakses.

Suatu *wireless* SDN terdiri dari *Wireless Mesh Router* (WMR) yang menyediakan konektivitas untuk satu set jaringan akses (baik menawarkan antarmuka kabel atau nirkabel ke terminal pengguna). Suatu *subset* dari WMR beroperasi sebagai *gateway* dan memberikan konektivitas layanan internet. Konfigurasi ini adalah ciri khas dari *Wireless Community Network* saat ini. Pada pendekatan SDN ini, dipasangkan OpenFlow *controller* yang terhubung ke WMR melalui koneksi kabel maupun nirkabel.

Arsitektur WMR dalam penelitian ini ditunjukkan pada Gambar 2.2. Hal ini termasuk: satu antarmuka nirkabel milik jaringan *wireless* (wlan0); antarmuka kabel opsional menuju *Access Network* klien (eth0); antarmuka kabel opsional digunakan sebagai pintu gerbang (*gateway*) ke internet (eth1); antarmuka *virtual* br0, yang merupakan jembatan perangkat lunak menggunakan OpenFlow logika switching, misalnya Openvswitch [11]. Suatu generik nyata dari WMR *node* memiliki antarmuka nirkabel atau kabel tambahan terhadap *Access Network* klien dan antarmuka nirkabel tambahan dapat dijembatani untuk br0 jika *multi-channel* WMR digunakan.



Gambar 2.2 Interaksi antara OpenFlow dengan WMR

Interface br0 memiliki alamat IP milik *control-subnet*, wlan0 tidak memiliki alamat IP, eth0 memiliki alamat *subnet Network Access* dan eth1 ke *subnet* terhubung ke Internet. OLSR terhubung ke br0, dan br0 digunakan sebagai tujuan untuk setiap paket yang dihasilkan oleh *node* dan diarahkan menuju WMR. Perlu dicatat bahwa dalam arsitektur ini memiliki dua tingkatan yang berbeda dari kontroler dalam menyiapkan aturan OpenFlow: pengontrol dengan distribusi lokal untuk mengatur *control-rules* yang merupakan pasangan OLSR dan OpenFlow, dan *remote* pengontrol terpusat untuk lalu lintas data.

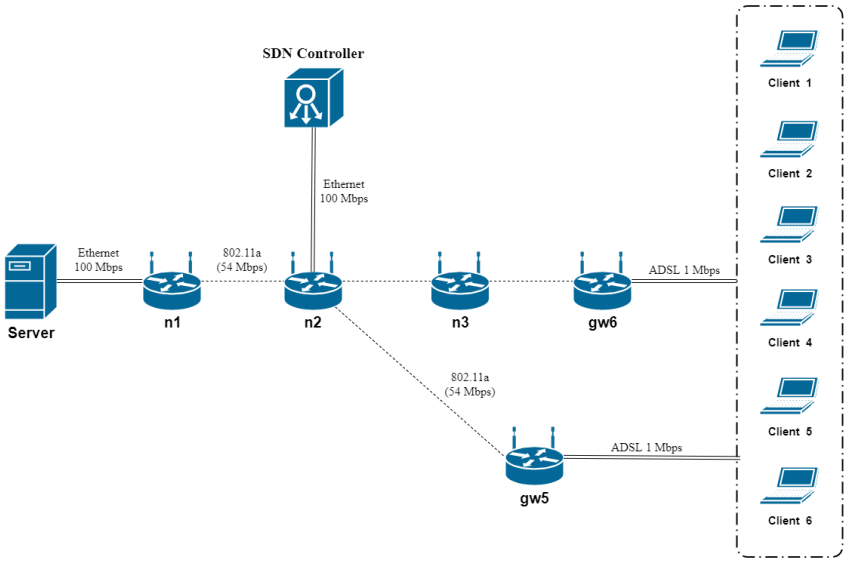
2.3 *Quality of Service* (QoS)

Quality of Service (QoS) adalah efek kolektif dari kinerja layanan yang menentukan derajat kepuasan seorang pengguna terhadapt sebuah layanan. Ada beberapa parameter yang menjadi acuan untuk QoS di antaranya yaitu *delay*, *throughput*, *jitter*, dan *packet loss* [5].

3. Perancangan Sistem

3.1 Model Sistem Simulasi

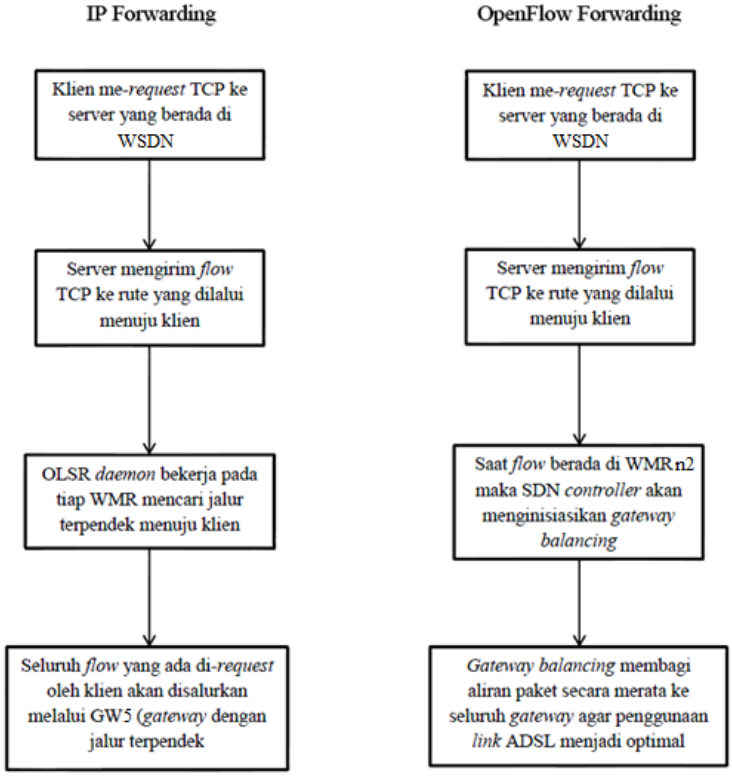
Sistem ini melakukan simulasi pada sebuah jaringan *Wireless Software Defined Network* (WSDN) menggunakan Linux *container* (LXC), CORE *daemon* [8], Openvswitch [11], olrsd [9], dan NS-3 [12]. LXC bertugas untuk menjalankan layanan *virtual IP routing* yang dapat diimplementasikan melalui OpenFlow API dalam sebuah *framework* yang bersifat *open source*. Protokol *routing default* yang digunakan untuk *wireless data plane* adalah OLSR yang akan dilakukan dengan *routing engine* olsrd. Kemudian pembuatan topologi pada *wireless data plane* menggunakan NS-3.



Gambar 3.1 Topologi sistem simulasi

Pada Gambar 3.1 menunjukkan konfigurasi topologi dari simulasi WSDN yang digunakan dalam penelitian ini. Simulasi WSDN tersebut dibentuk oleh 5 WMR, dua di antaranya adalah *gateway* dan menyediakan akses internet melalui koneksi ADSL dengan *bandwidth uplink* sebesar 1 Mbit/s. Sebuah *server* publik terhubung ke WMR n1 lewat *ethernet* dan beberapa klien yang terletak di internet me-*request* data dari *server* ini.

3.2 Proses Kerja Sistem



Gambar 3.2 Blok diagram proses kerja sistem

Ada dua sistem proses kerja yaitu *IP forwarding* yang sistemnya hanya *wireless data plane* dengan perutingan OLSR dari olsrd dan *OpenFLow forwarding* yang merupakan sistem perutingan WSDN yang menggunakan fungsi *gateway balancing*. Dalam pengerjaan sistem ini, jika digunakan OLSR IP *routing* biasa di jaringan *wireless*, maka secara teori semua lalu lintas yang dikirim dari *server* ke internet klien mengalir melalui *gateway* terdekat ke *server*. Masing-masing *gateway* mengumumkan ke WMR melalui OLSR rute *default* dan OLSR memasukkan rute *default* ini di IP *routing* tabel WMR dengan menggunakan strategi jalur terpendek sesuai dengan algoritma perutingan OLSR (*shortest path*). Misalnya, pada kasus seperti topologi Gambar 3.1, semua lalu lintas antara server publik dan klien internet mengalir melalui gw5, yang merupakan pintu gerbang paling dekat dengan server.

Menggunakan *gateway balancing* pada WSDN dalam skenario ini memungkinkan untuk melaksanakan operasi *forwarding* pada aliran dasar dan untuk rute arus yang berbeda pada *gateway* yang berbeda dalam rangka untuk lebih mengeksploitasi kapasitas *uplink* yang disediakan oleh semua *gateway*. Pada konsep ini, diterapkan algoritma round robin *Gateway Selection Algorithm* (GSA) untuk kontroler OpenFlow. GSA mendorong aturan dalam tabel aliran WMR, yang bertujuan untuk meruting data tunggal menuju *gateway* yang dipilih. Aturan mengidentifikasi aliran melalui kriteria yang didasarkan pada beberapa IP sumber dan IP tujuan. Tindakan aturan yang dilakukan ada dua: i) untuk mengubah alamat MAC sumber dengan alamat MAC pada node WMR dan alamat MAC tujuan dengan salah satu WMR *next-hop* di jalur menuju *gatewa*y yang dipilih, ii) untuk meneruskan paket pada *interface* nirkabel menuju jalur WMR. *Next-hop* WMR diperhitungkan dengan menggunakan topologi jaringan, secara berkala dipelajari oleh *controller* dengan menghubungi *plugin* OLSR JSONinfo dari WMR yang terhubung (WMR n2 dalam kasus ini).

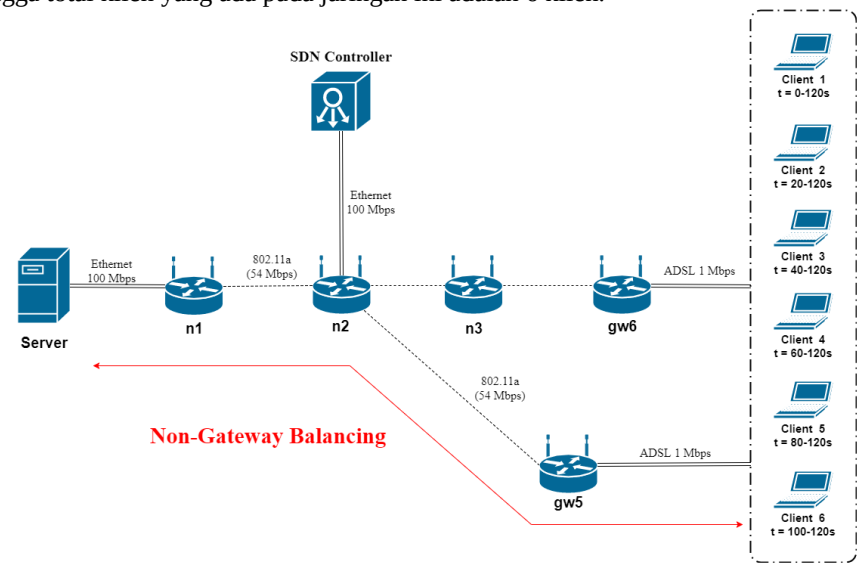
4. Simulasi Dan Analisis

4.1 Skenario

Terdapat dua skenario yang dilakukan, yaitu *gateway balancing* dan *gateway fault handling*. Skenario pertama, *gateway balancing*, dilakukan dengan mengaktifkan fungsi *gateway balancing* pada SDN *controller* untuk membagi jalur *gateway* pada setiap paket yang dialiri oleh sejumlah *clients* sehingga akan didapatkan nilai *bandwidth* yang lebih baik dibandingkan dengan jaringan WSDN tanpa *gateway balancing*. Skenario kedua, *gateway fault handling*, bertujuan untuk menguji kemampuan SDN *controller* untuk *gateway balancing* dalam menangani perubahan jumlah jalur *gateway* dikarenakan salah satu *gateway* yang ada pada jaringan mengalami *down*.

4.1.1 *Gateway Balancing*

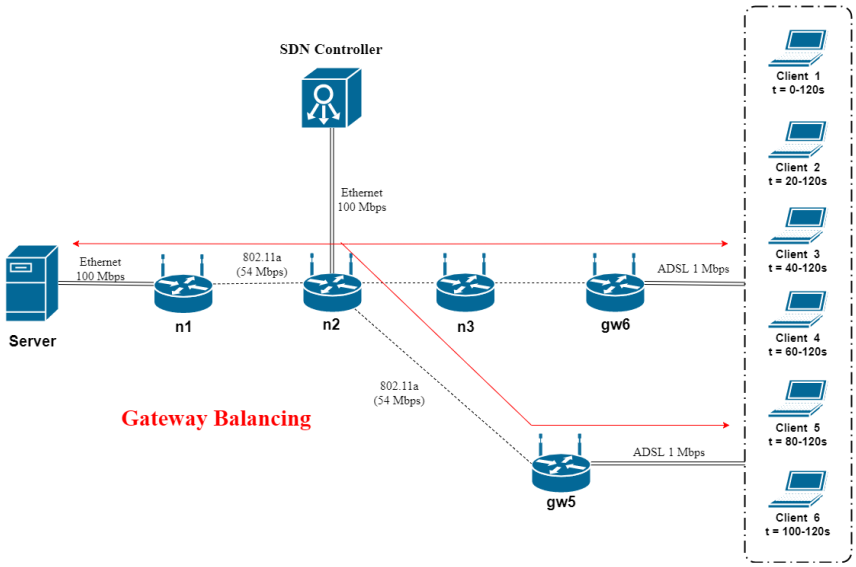
Percobaan pertama dilakukan dengan mengalirkan *flow* paket TCP dengan jumlah klien yang berbeda tiap 20 detik ke dalam jaringan WSDN tanpa mengaktifkan fungsi *gateway balancing* pada SDN *controller* yang mana jaringan tersebut hanya menggunakan perutingan OLSR dengan *routing engin*e olsrd. Jumlah klien yang masuk ke dalam jaringan *wireless* untuk melakukan pertukaran data dengan *server* dimulai dari awal sejumlah satu klien, kemudian akan bertambah satu klien baru setiap 20 detik. Lamanya waktu pengambilan data adalah 120 detik, sehingga total klien yang ada pada jaringan ini adalah 6 klien.



Gambar 4.1 Perutingan WSDN tanpa *gateway balancing*

Pada Gambar 4.1 diperlihatkan aliran *flow* paket TCP dari *server* ke *clients* melalui rute yang telah ditentukan oleh OLSR. Perutingan OLSR dengan *wireless mesh router* yang saling terhubung menggunakan *link* 802.11a akan merutekan jarak *router* yang terpendek, yaitu melalui gw5.

Pada skenario berikutnya, dilakukan penyaluran *flow* paket TCP pada jaringan WSDN yang sama dengan skenario sebelumnya, namun kali ini dilakukan dengan mengaktifkan fungsi *gateway balancing* pada SDN *controller*. Algoritma *gateway balancing* membagi jalur untuk *flow* paket agar setiap jalur dan *gateway* yang ada dalam jaringan dapat digunakan, sehingga dapat memaksimalkan penggunaan *bandwidth* terbaiknya. Percobaan ini dapat dilihat pada Gambar 4.2 dimana SDN *controller* terhubung ke jaringan *wireless* untuk mengaktifkan *gateway balancing* sehingga aliran paket TCP dapat tersalurkan secara merata ke kedua *gateway* yang dihubungkan oleh dua jalur ADSL dengan *clients*.

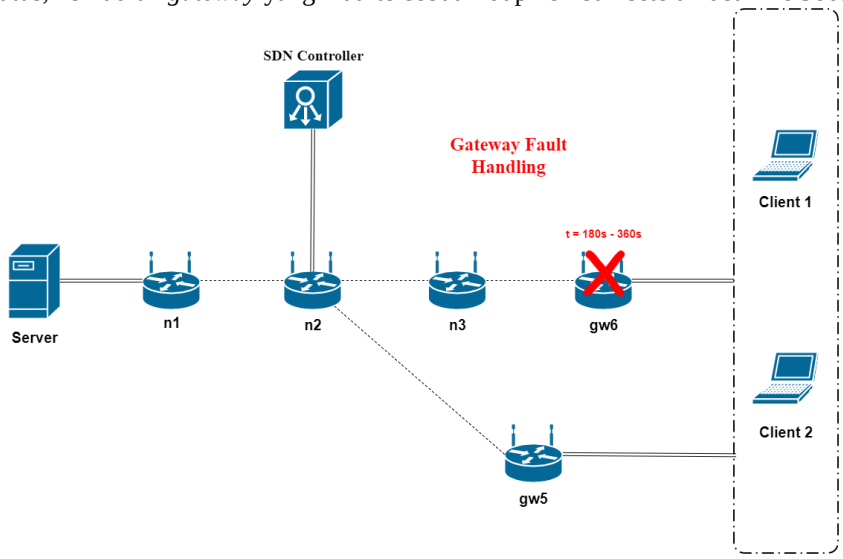


Gambar 4.2 Perutingan WSDN menggunakan *gateway balancing*

Hasil percobaan dari kedua skenario tersebut bertujuan untuk memperlihatkan metode yang mana dari kedua skenario yang menghasilkan performa terbaik.

4.1.2 *Gateway Fault Handling*

Skenario *gateway fault handling* menguji kemampuan SDN *controller* dalam menangani terjadi putus fisik salah satu *gateway* pada jaringan yang menyebabkan bergantinya topologi dan perutingan. Agar skenario ini dapat melihat apakah sistem pada Gambar 4.2 dapat menangani kegagalan perutingan akibat putusnya salah satu *gateway*, maka dilakukan percobaan dengan dua *clients* yang mengirimkan *request* paket TCP dalam waktu bersamaan ke *server* pada jaringan WSDN dengan fungsi *gateway balancing* aktif. Percobaan ini diilustrasikan pada Gambar 4.3 dimana saat detik ke 180 sampai 360 salah satu *gateway* gw6 yang terhubung dengan *clients* dalam jaringan putus, kemudian *gateway* yang mati tersebut hidup kembali setelah detik ke 360.



Gambar 4.3 Skenario *gateway fault handling*

4.2 Hasil Pengambilan Data

Pada bagian ini ditunjukkan hasil pengukuran dan analisis hasil pengujian dari skenario *gateway balancing* dan *gateway fault handling*.

4.2.1 Hasil *Gateway Balancing*

Hasil *throughput*

Hasil *throughput* keenam klien dalam skenario *gateway balancing* dapat dilihat pada Tabel 4.1. Setiap data yang ditampilkan merupakan rata-rata dari 30 kali hasil data percobaan yang telah dilakukan.

Tabel 4.1 Hasil rata-rata *throughput* pada keenam klien skenario *gateway balancing*

Jumlah *Clients*	Waktu (detik)	*Throughput* dengan *gateway balancing* (*kbps*)	*Throughput* tanpa *gateway balancing* (*kbps*)
1	0s-20s	953,8	953,2
2	20s-40s	952,8	497,5
3	40s-60s	653,1	327,6
4	60s-80s	488,3	247,1
5	80s-100s	388,1	202,8
6	100s-120s	306,7	155,7

Dari hasil pengukuran yang ditunjukkan pada Tabel 4.1, *client 1* yang masuk sejak detik ke nol dapat menggunakan nilai *throughput* secara maksimal hingga detik ke 40 dengan *gateway balancing* aktif, begitupun *client 2* dari yang masuk dari detk ke 20. Nilai *throughput* yang diterima oleh setiap klien mengalami penurunan seiring dengan bertambahnya jumlah klien yang masuk.

Hasil *delay*

Hasil *delay* dari keenam klien dalam skenario *gateway balancing* dapat dilihat pada Tabel 4.2. Setiap data yang ditampilkan merupakan rata-rata dari 30 kali hasil data percobaan yang telah dilakukan.

Tabel 4.2 Hasil rata-rata *delay* pada keenam klien skenario *gateway balancing*

Jumlah *Clients*	Waktu (detik)	*Delay* WSDN dengan *gateway balancing* (milidetik)	*Delay* WSDN tanpa *gateway balancing* (milidetik)
1	0s-20s	15,59	15,78
2	20s-40s	15,73	24,31
3	40s-60s	22,57	30,29
4	60s-80s	32,76	37,61
5	80s-100s	38,48	50,12
6	100s-120s	51,83	65,03

Dari hasil pengukuran yang ditunjukkan pada Tabel 4.2 nilai rata-rata *delay* terbaik adalah pada saat detik ke nol sampai 40 dengan jaringan yang menggunakan *gateway balancing*. Nilai rata-rata *delay* terburuk ada pada detik ke 100 sampai 120 tanpa *gateway balancing*.

Hasil *jitter*

Hasil *jitter* dari keenam klien dalam skenario *gateway balancing* dapat dilihat pada Tabel 4.3. Setiap data yang ditampilkan merupakan rata-rata dari 30 kali hasil data percobaan yang telah dilakukan.

Tabel 4.3 Hasil rata-rata *jitter* pada keenam klien skenario *gateway balancing*

Jumlah *Clients*	Waktu (detik)	*Jitter* WSDN dengan *gateway balancing* (mikrodetik)	*Jitter* WSDN tanpa *gateway balancing* (mikrodetik)
1	0s-20s	3,22	0,26
2	20s-40s	14,54	0,40
3	40s-60s	57,20	0,71
4	60s-80s	36,03	2,21
5	80s-100s	50,57	197,31
6	100s-120s	196,28	6,14

Dari hasil pengukuran yang ditunjukkan pada Tabel 4.3 nilai rata-rata *jitter* terbaik adalah pada saat detik ke nol sampai 60 tanpa *gateway balancing*. Nilai rata-rata *jitter* terburuk ada pada detik ke 100 sampai 120 dengan *gateway balancing*.

Hasil *packet loss*

Hasil *packet loss* dari keenam klien dalam skenario *gateway balancing* dapat dilihat pada Tabel 4.4. Setiap data yang ditampilkan merupakan rata-rata dari 30 kali hasil data percobaan yang telah dilakukan.

Tabel 4.4 Hasil rata-rata *packet loss* pada keenam klien skenario *gateway balancing*

Waktu (detik)	WSDN dengan *gateway balancing*			WSDN tanpa *gateway balancing*		
	Total paket dikirim	Paket yang diterima	*Packet loss*	Total paket dikirim	Paket yang diterima	*Packet loss*
0-20	1272	1272	0	1275	1275	0%
20-40	1261	1261	0	868	866	0,23%
40-60	927	926	0,11%	613	611	0,33%
60-80	710	709	0,14%	527	525	0,38%
80-100	560	558	0,36%	409	406	0,73%
100-120	405	402	0,74%	311	308	0,96%

Dari hasil pengukuran yang ditunjukkan pada Tabel 4.4 nilai *packet loss* terbaik adalah pada saat detik ke nol sampai 40 dengan *gateway balancing* dan detik ke nol sampai 20 tanpa gateway balancing. Nilai *packet loss* terburuk ada pada detik ke 100 sampai 120 tanpa *gateway balancing*.

4.2.2 Hasil *Gateway Fault Handling*

Hasil *throughput*

Hasil *throughput* dari kedua klien dalam skenario *gateway fault handling* dapat dilihat pada Tabel 4.6. Setiap data yang ditampilkan merupakan rata-rata dari 30 kali hasil data percobaan yang telah dilakukan.

Tabel 4.6 Hasil rata-rata *throughput* pada kedua klien skenario *gateway fault handling*

Jumlah *gateway*	Waktu (detik)	*Throughput* (*kbps*)
2	0-90	952,9
2	90-180	952,8
1 (gw6 *down*)	180-270	390,9
1 (gw6 *down*)	270-360	475,1
2	360-450	909,1
2	450-540	952,7

Dari hasil pengukuran yang ditunjukkan pada Tabel 4.6, *client 1* dan *client 2* yang masuk secara bersamaan mendapatkan nilai *throughput* secara maksimal dengan *gateway balancing* aktif hingga detik ke 180, namun *gateway* gw6 terputus setelah detik ke 180 sehingga hanya mendapatkan nilai *throughput* lebih rendah diakibatkan perubahan perutingan hingga pada detik ke 360 saat *gateway* gw6 hidup kembali dan perutingan dikembalikan menggunakan *gateway balancing*.

Hasil *delay*

Hasil *delay* dari kedua klien dalam skenario *gateway fault handling* dapat dilihat pada Tabel 4.7. Setiap data yang ditampilkan merupakan rata-rata dari 30 kali hasil data percobaan yang telah dilakukan.

Tabel 4.7 Hasil *delay* pada *client 1* dan *client 2* skenario *gateway fault handling*

Jumlah *gateway*	Waktu (detik)	*Delay client 1* (milidetik)	*Delay client 2* (milidetik)	Rata –rata *delay* kedua klien (milidetik)
2	0-90	15,92	15,76	15,84
2	90-180	15,73	15,87	15,80
1 (gw6 *down*)	180-270	54,30	56,86	55,58
1 (gw6 *down*)	270-360	30,78	38,85	34,82
2	360-450	17,83	18,48	18,16
2	450-540	15,89	15,94	15,92

Dari hasil pengukuran yang ditunjukkan pada Tabel 4.7, *client 1* dan *client 2* yang masuk secara bersamaan mendapatkan nilai *delay* rata-rata terbaik dengan *gateway balancing* aktif hingga detik ke 180, namun *gateway* gw6 terputus setelah detik ke 180 sehingga nilai *delay* rata-rata memburuk diakibatkan perubahan perutingan hingga pada detik ke 360 saat *gateway* gw6 hidup kembali dan perutingan dikembalikan menggunakan *gateway balancing*.

Hasil *jitter*

Hasil *jitter* dari kedua klien dalam skenario *gateway fault handling* dapat dilihat pada Tabel 4.8. Setiap data yang ditampilkan merupakan rata-rata dari 30 kali hasil data percobaan yang telah dilakukan.

Tabel 4.8 Hasil *jitter* pada *client 1* dan *client 2* skenario *gateway fault handling*

Jumlah *gateway*	Waktu (detik)	*Jitter Client 1* (mikrodetik)	*Jitter Client 2* (mikrodetik)	Rata-rata *Jitter* kedua klien (mikrodetik)
2	0-90	0,64	0,66	0.65
2	90-180	4,80	5,15	4,98
1 (gw6 *down*)	180-270	37,32	39,38	38,35
1 (gw6 *down*)	270-360	7,92	8,26	8,09
2	360-450	3,94	4,19	4,06
2	450-540	0.79	0.76	0.78

Dari hasil pengukuran yang ditunjukkan pada Tabel 4.8, *client 1* dan *client 2* yang masuk secara bersamaan mendapatkan nilai *jitter* dengan *gateway balancing* aktif saat awal masuk hingga detik ke 90. Pada saat detik ke 180 sampai 270 *gateway* gw6 terputus sehingga nilai *jitter* memburuk diakibatkan fase awal perubahan perutingan hingga pada detik ke 270 dan mulai membaik setelah detik ke 360. Pada saat *gateway* gw6 hidup kembali di detik ke 360, perutingan dikembalikan menggunakan *gateway balancing*.

Hasil *packet loss*

Hasil *packet loss* dari kedua klien dalam skenario *gateway fault handling* dapat dilihat pada Tabel 4.9. Setiap data yang ditampilkan merupakan rata-rata dari 30 kali hasil data percobaan yang telah dilakukan.

Tabel 4.9 Hasil *packet loss* pada *client 1* dan *client 2* skenario *gateway fault handling*

Waktu (detik)	*Client 1*			*Client 2*			Rata-rata kedua klien		
	Total paket dikirim	Paket diterima	*Packet loss*	Total paket dikirim	Paket diterima	*Packet loss*	Total paket dikirim	Paket diterima	*Packet loss*
0-90	5638	5637	0,02%	5575	5574	0,02%	5606	5605	0,02%
90-180	5645	5645	0	5586	5586	0	5615	5615	0
180-270	1675	1628	2,81%	1634	1585	3%	1654	1606	2,91%
270-360	2794	2794	0	2752	2752	0	2773	2773	0
360-450	5043	5040	0,06%	4967	4963	0,08%	5005	5001	0,07%
450-540	5657	5657	0	5583	5583	0	5620	5620	0

Dari hasil pengukuran yang ditunjukkan pada Tabel 4.9, *client 1* dan *client 2* yang masuk secara bersamaan mendapatkan nilai *packet loos* terbaik dengan *gateway balancing* aktif hingga detik ke 180, namun *gateway* gw6 terputus setelah detik ke 180 sehingga nilai *packet loss* memburuk diakibatkan perubahan perutingan hingga pada detik ke 270 yang merupakan fase awal SDN *controller* melakukan perubahan perutingan. Pada saat detik ke 360 *gateway* gw6 hidup kembali, perutingan dikembalikan menggunakan *gateway balancing*.

5. Kesimpulan

Dari penelitian yang telah dilakukan dapat ditarik beberapa kesimpulan sebagai berikut:

1. SDN *controller* dengan fungsi *gateway balancing* dapat diaplikasikan pada jaringan *wireless*. Hal ini dibuktikan dengan membaiknya nilai *throughput* dan *delay* pada jaringan *wireless* yang diaktifkan SDN *controller* dengan fungsi *gateway balancing*.
2. *Throughput* paket TCP yang diterima dari *server* ke enam klien yang me-*request* paket dalam waktu yang berbeda setiap 20 detik pada jaringan WSDN dengan *gateway balancing* memiliki nilai rata-rata yang lebih baik daripada jaringan tanpa *gateway balancing*. Nilai *throughput* pada WSDN dengan *gateway balancing* dapat lebih baik dengan jumlah klien yang masuk 2 klien sebesar 91,5%, 3 klien: 99,3%, 4 klien: 97,6%, 5 klien: 91,3%, dan 6 klien: 96,9%.
3. *Delay* pada keenam klien yang masuk dalam waktu yang berbeda setiap 20 detik pada jaringan WSDN dengan *gateway balancing* memiliki nilai rata-rata yang lebih baik daripada jaringan tanpa *gateway balancing*. Nilai *delay* pada WSDN dengan *gateway balancing* dapat lebih baik dengan jumlah klien yang masuk 2 klien sebesar 35,3%, 3 klien: 25,5%, 4 klien: 12,9%, 5 klien: 23,2%, 6 klien: 20,3%.
4. SDN *controller* mampu mengubah perutingan secara otomatis pada jaringan SDN apabila saat salah satu *gateway* pada jaringan mengalami *down*.

5. SDN *controller* memerlukan waktu rata-rata 16 detik untuk merubah perutingan dari *gateway balancing* ke *non gateway balancing* dan 2 detik untuk mengubah peruting dari *non gateway balancing* kembali ke *gateway balancing*.

Daftar Pustaka

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," SIGCOMM Comput. Commun. Rev., vol. 38, pp. 69–74, March 2008

[2] Open Networking Foundation, "SDN Architecture", Issue 1 (June 2014), ONF TR-502.Ddd

[3] Clausen, T. and Jacquet P., Oct. 2003. "Optimized Link State Routing Protocol (OLSR)," IETF Network Working Group RFC 3626.

[4] Chandanal, R., Jain, S., Kumar, A., 2012. "A Weighted Approach for MPR Selection in OLSR Protocol" 2nd IEEE International Conference on Parallel, Distributed and Grid Computing

[5] Seitz, N., & NTIA/ITS. (2003). "ITU-T QoS Standards for IP-Based Networks." IEEE Communications Magazine.

[6] T.D. Nadeau and K. Gray. "SDN: Software Defined Networks." O'Reilly, August 2013.

[7] P Dely, A Kassler, N Bayer, "OpenFlow for Wireless Mesh Networks", IEEE International Workshop on Wireless Mesh and Ad Hoc Networks (WiMAN 2011), Hawaii, USA, August 2011

[8] J. Ahrenholz, C. Danilov, T. R. Henderson, J. H. Kim, "CORE: A realtime network emulator", IEEE Military Communications Conference, MILCOM 2008.

[9] A. Tonnesen and T. Lopatic and H. Gredler and B. Petrovitsch and A. Kaplan and S.O. Tuecke, "olsrd Website," URL: http://www.olsr.org/

[10] POX controller website: http://www.noxrepo.org/

[11] Open vSwitch website: http://openvswitch.org/

[12] "How to use Linux Containers to set up virtual networks", http://www.nsnam.org/wiki/index.php/HOWTO_Use_Linux_Containers_to_set_up_virtual_networks

[13] A. Detti, C. Pisa, S. Salsano, N. Blefari-Melazzi, "Wireless Mesh Software Defined Networks (wmSDN)", CNBuB Workshop at IEEE WiMob, France, Lyon, 7 October 2013

[14] "SPlit ARchitecture Carrier-Grade Networks (SPARC)", http://www.fp7-sparc.eu/