Analisa Performansi Penanganan Kegagalan Link Pada Layer 2 pada Jaringan Sofware-Defined Network OpenFlow

Performance Analysis Link Failure Recovery on Layer 2 of Sofware-Defined Netwok OpenFlow

Satria Akbar M[1], Tri Broto Harsono, S.T.,M.T.[2], Sofia Naning H, S.T.,M.T.[3]

[1,2,3]Prodi S1 Teknik Informatika, Fakultas Teknik, Universitas Telkom

[1]satriamugitama@gmail.com, [2]tri.brotoharsono@gmail.com,[3]snh@telkomuniversity.ac.id

Abtrak

***Software Defined Networking* (SDN) adalah sebuah paradigma yang merubah cara merancang, mengatur dan mengontrol jaringan. Inti dari SDN adalah membuat jaringan yang dapat diprogram. Salah satu protokol jaringan yang dapat mendukung SDN yaitu OpenFlow. OpenFlow didesain dan dikembangkan oleh Standford University yang memisahkan antara perangkat kontrol (*control plane*) dan perangkat penerus/penyalur paket data *(data plane)*. Pada pengembangan OpenFlow muncul permasalahan utama yaitu *reliability*. Dalam penelitian ini dibuat sebuah algoritma prototipe penanganan kegagalan link dan menguji kinerjanya dengan membandingkan waktu pergantian jalur dan overhead dengan metode-metode yang sudah ada pada OpenFlow.**

Pada penelitian ini algoritma prototipe penanganan kegagalan link memerlukan waktu rata-rata 59 mili detik dengan overhead 10.1%. Ini menunjukan masih belum memenuhi standar carrier grade sebesar 50 mili detik. Tetapi algoritma prototipe memiliki kinerja lebih baik dibanding dengan metode yang sudah ada pada OpenFlow.

Kata kunci : SDN, OpenFlow, kegagalan link

Abstract

Software Defined Networking (SDN) is a paradigm that is changing the way design, organize and control the network. The essence of SDN is to create a network that can be programmed. One of the network protocol that can support SDN is OpenFlow. OpenFlow is designed and developed by Stanford University which separates control plane and data plane. Reliability is the Major issue to deploy OpenFlow in this network. In this study will be made a prototype algorithm handling link failures and test its performance by comparing the switchover time and the overhead to the methods that already exist on OpenFlow.

In this research, prototype algorithms link failure handling requires an average time of 59 milliseconds with overhead 10.1%. It showed prototype algorithms still not meet the standards of carrier grade by 50 milliseconds. But the prototype algorithm has better performance compared with existing methods on OpenFlow

K ey Wo rd : SDN, O penFlo w, Lin k F a ilure

1. Pendahuluan

SDN (*Software Defined Networking*) merupakan pendekatan dalam jaringan komputer dimana kontrol jaringan (*control-plane*) berfungsi untuk menangani penentuan *forwarding traffic* terpisah dari *data-plane*. SDN menawarkan kemudahan dalam mengatur perangkat jaringan dengan hanya mengatur sofware *control-plane*. Software *control plane* pada SDN berfungsi mengatur banyak *data-plane* pada jaringan seperti router, switch maupun perangkat middlebox lainya melalui API (*Application Programming Interface*) [1]. Salah satu API yang sekarang digunakan adalah OpenFlow yang dikembangkan oleh Standford Univeristy. Terdapat beberapa kontroler yang dapat menjalankan OpenFlow yaitu POX yang berbasis python dan NOX berbasis C. POX adalah kontroler yang cocok digunakan untuk penelitian.

Dalam jaringan konvensional untuk mendukung *High availibility* pada jaringan diperlukan sebuah topologi yang redundan. Pada layer 2 jaringan redudansi topologi akan mengakibatkan masalah *broadcast strom/frame-looping* akibat *broadcast* kesetiap port yang dilakukan oleh switch. Masalah tersebut dapat diselesakan dengan menggunakan STP (*Spanning Tree Protocol*) 802.1 yang akan mencegah frame-loop dengan melakukan block pada link/jalur yang sebagai backup. Akan tetapi permasalahan muncul saat terjadi kegagalan link utama terjadi. Pada STP memerlukan lebih dari waktu 30 detik untuk melakukan pembentukan link baru [2]. Sehingga menyebabkan tidak terpenuhi standar "*Carrier grade"* yaitu melakukan perbaikan(*recovery*) kegagalan link tidak lebih dari 50 milidetik pada jaringan ethernet [3]. Pada pengembangan STP yaitu RSTP mampu melakukan perbaikan link kurang dari 50 milidetik.

Pada POX memiliki mekanisme untuk melakukan *failover* yaitu *L2 Learning Switch* dan *L2 Multi* dengan menggunakan modul *discovery* yang memanfaatkan paket LLDP (*Link Layer Discovery Protocol*) yang dikirimkan secara berkala. Pada standar OpenFlow 1.0 terdapat pesan yang dikirim secara *asynchron* antara switch dan kontroler yang dapat mendeteksi adanya perubahan port. Dengan memanfaatkan pesan *port-status* pada algoritma prototipe penanganan kegagalan link diharapkan dapat mempercepat waktu penanganan kegagalan link.

Dalam tugas akhir ini akan disimulasikan jaringan SDN OpenFlow dengan menggunakan emulator Mininet untuk mengetahui performansi mekanisme penganganan kegagalan link. Parameter yang dianalis adalah waktu recovery, dan overhead. Hasil akhir dari tugas akhir ini diharapkan dapat mengetahui performansi dari metode L2 *Learing Switch , l2 Multi* dan algoritma prototipe dalam melakukan penanganan kegagalan link.

2. Dasar Teori /Material dan Metodologi/perancangan

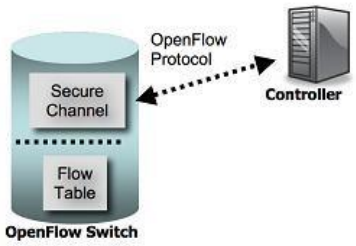
2.1 SDN (Software Defined Network)

Software Defined Network (SDN) adalah sebuah 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.

2.2 OpenFlow 1.0

OpenFlow dianggap sebagai salah satu standar *software-defined networking* (SDN) pertama. OpenFlow sesungguhnya adalah sebuah protokol komuniskasi yang didefinisikan untuk membolehkan kontroler SDN/OpenFlow untuk berinteraksi dengan *forwarding plane* dan melakukan penyesuaian pada jaringan. Jadi OpenFlow lebih baik untuk beradaptasi dalam perubahan syarat-syarat bisnis (*business reuirements*) [6].

Kontroler SDN adalah "otak" dari jaringan. Kontroler adalah pusat kendali strategis yang menyalurkan informasi kepada perangkat switch/router "pada bagian bawah" (melalui *southbound APIs*) dan pada layer aplikasi dan *business logic* "pada bagian atas" (melalui *northbound APIs*). Sekarang beberapa organisasi sedang mengembangkan jaringan SDN, sedangkan kontroler SDN dikerjakan oleh federasi yang menangani domain kontroler SDN. Pengembangan kontroler SDN menggunakan aplikasi umum, seperti OpenFlow dan *Open Virtual Switch Database* (OVSDB) [6].



Gambar 2-1OpenFlow 1.0[3]

Standar OpenFlow berbasis pada fakta bahwa modern switch/router memiliki FIB (*Forwarding Information Bas)* yang diimplementasikan dengan menggunakan TCAM (*Ternary Content Addressable Memory*). FIB dapat diprogram dengan menggunakan protokol OpenFlow dengan cara "*adding*/*deleting*" (menambah/mengurangi) entri pada Flow-tabel.

FlowTabel adalah sebuah abstraksi dari FIB. Dalam jaringan OpenFlow semua logic dipusatkan pada satu sistem yang disebut kontroler OpenFlow. Kontroler memliliki tugas untuk mengelola switch OpenFlow. Jadi switch OpenFlow terdiri dari FlowTabel yang berfungsi melakukan pencarian dan paket *forwarding*, dan *Secure Channel* untuk berkomunikasi dengan eksternal kontroler. [1]

2.3 POX

POX adalah sebuah platform open source yang digunakan untuk aplikasi kontroler *software-defined networking* (SDN) yang berbasis python. Dengan menggunakan POX memungkinkan untuk pengembangan dan pembuatan prototipe secara cepat, sehingga POX menjadi lebih banyak digunakan dibanding NOX. [7]
POX memiliki beberapa keistimewaan yaitu:

- Interface OpenFlow yang "Pythonic".
- Menyediakan contoh kode yang dapat digunakan ulang(Reuseable).
- "Runs anywhere" dapat dipasang dengan PyPy runtime.
- Khusus ditargetkan untuk Linux, Mac OS, dan Windows.
- Topologi discovery.
- Mendukung GUI dan perangkat visualisasi seperti NOX.

Kinerja lebih baik dibanding dengan aplikasi pada NOX yang ditulis dengan bahasa Python.

Komponen pada POX:

2.3.1 L2-Learning

L2-learning merupakan salah satu komponen yang terdapat di POX. Komponen ini bertugas untuk membuat *switch* Openflow bekerja sebagai L2 *learning switch*, yaitu dengan memetakan alamat MAC dan *port* dari *switch* Openflow ke kontroler. Alamat pemetaan berasal dari pemantauan paket yang masuk pada switch kemudian dicatat alamat MAC dari paket tersebut pada kontroler. [1]

Pengiriman paket bergantung pada pencocokan alamat tujuan pada paket masuk (*packet in*) dengan alamat yang ada pada tabel MAC (berisi alamat tujuan dan port switch) yang ada pada kontroler. Ketika pada saat pencocokan terdapat alamat yang cocok antara alamat tujuan paket dengan daftar MAC, maka kontroler menambah FlowEntri pada switch OpenFlow sehingga switch mampu meneruskan paket melalui port yang tepat. Akan tetapi, saat tidak ditemukan kecocokan maka alamat tujuan paket akan disimpan pada daftar MAC dan paket akan di-*broadcast* kesemua port. Paket yang memiliki tujuan broadcast dan multicast akan di-broadcast secara langsung tanpa melakukan penyimpanan pada tabel MAC.

Pada Switch OpenFlow mendukung fitur penuntupan *port-flood* yaitu dengan melarang port tersebut untuk mengirim paket *broadcast*. Kontroler POX teradapat modul Spanning Tree yang dapat menjalankan fitur tersebut, sehingga dapat menghilangkan *loop* pada topologi redundan yang mengakibatkan *broadcast-storm/frame-looping.*

L2 learning tidak mengimplementasikan *aging timer logic*. *Aging timer logic* [1] adalah sebuah mekanisme saat switch belajar sebuah alamat asal, switch mencatat waktu (*timestamp*) pada entri. Setiap waktu switch melihat *frame* dari *source*(asal), dan switch memutakhirkan *timestamp*. Ketika saat switch tidak menerima *frame* dari *source* selama batas waktu kadaluarsa *aging timer*, maka switch akan menghapus entri dari tabel MAC.

Dikarenakan tidak adanya *aging timer logic,L2 learning* tidak dapat menghapus entri dari tabel MAC-nya jika terjadi kegagalan link. Tetapi terdapat sebuah cara untuk memutakhirkan entri saat paket diterima dari beberapa port lain. Ini memungkinkan *L2 learning* untuk mem-broadcast. Ada dua kasus paket di-broadcast. Pertama saat alamat tujuan tidak terdapat pada tabel MAC. Ke-dua saat alamat tujuan adalah broadcast atau multicast. Disaat tidak adanya *aging timer logic,* kasus pertama tidak mungkin terjadi karena entri sudah terdapat pada tabel MAC, tetapi kasus ke-dua dapat terjadi.

Ketika paket data dikirim pada jaringan, ARP(*Address Resolution Protocol*) juga berjalan secara paralel jika entri static permanent ARP pada node tidak ada. Paket *ARP_request* menggunakan alamat *broadcast*, sehingga memungkinkan *L2 learning* untuk membentuk jalur baru. ARP *request* dengan alamat *broadcast* dikirim saat "node reachable time" pada entri ARP kadaluarsa. Ini berarti pembentukan jalur baru pada *L2 learning* bergantung pada waktu pengiriman ARP *request* dengan alamat *broadcast*. Pada saat terjadi kegagalan link, flow entri yang salah akan terus ditambahkan pada switch OpenFlow jika host tidak mengirimkan ARP *request.*

Host yang memiliki entri permanen pada tabel ARP tidak akan mengirim ARP *request.* Ketiadaan paket yang mengirim dengan alamat *broadcast* atau *multicast* mengakibatkan *L2 learning* tidak mampu menangani kegagalan link.

2.3.2 L2 Multi

L2 Multi adalah sebuah metode penentuan jalur terpendek antar *end host.* Ada 4 mekanisme untuk membangun jalur terpendek antar host. Mekanisme itu adalah *discovery, topology, authentication* dan *routing*.

Mekanisme yang dilakukan oleh L2 multi yaitu: Yang pertama adalah mekanisme Dicovery yaitu menggunakan pesan "*packet in*" dan "*packet out*" untuk menjalankan *discovery* protokol pada OpenFlow switch. Mekanisme ini mengirim "*packet out*" untuk mengirim LLDP(Link Layer Discovery Protocol) pada seluruh OpenFlow switch. Ketika switch menerima "*packet out*", paket LLDP segera dirikim ke output port. Ketika switch menerima LLDP packet, switch mengirim juga "*packet in*" kepada POX untuk memberitahu jalur terdeteksi.

Ke-dua adalah mekanisme topologi yaitu berguna untuk menyimpan jalur(link antar switch) dari switch yang terbentuk hasil dari *discovery*. Selanjutnya adalah mekanisme *authentication* berfungsi membentuk data host/user terkoneksi dengan salah satu switch (host dengan switch).

Terakhir, mekanisme *routing* yaitu berfungsi untuk menentukan jalur terpendek yaitu dengan menggunakan algoritma *Floyd-Warshall(*default algoritma pencarian pada POX*)* antara host yang sudah terdaftar. Sebuah paket akan diteruskan kontroler dikarenakan tidak ada flow tabel yang cocok. Jika mekanisme *authentication* tidak mengetahui alamat paket sumber maka akan alamat host akan didaftarkan pada tabel host dengan switch. Jika tujuan diketahui maka mekanisme *routing* akan menghitung jarak terpendek, dan menambah *flow* tabel pada masing-masing switch OpenFlow. Akan tetapi, jika tidak diketahui atau tidak terdapat pada tabel host-> switch maka paket akan di-*broadcast*.

Mekanisme pendeteksian kegagalan link pada L2 Multi sangat bergantung pada "*LinkEvent*" modul discovery dan waktu interval *timeout* dalam melakukan pergantian perubahan topologi atau konvergensi. Modul *discovery* mengirim paket LLDP secara berkala jika dalam waktu tertentu kontroler tidak menerima paket LLDP maka kontroler mendeteksi link putus.

Penghapusan seluruh flow entri pada switch akan mengakibatkan switch mengirim paket kepada kontroler dikarenakan tidak adanya kecocokan flowentri dengan paket yang baru masuk. Kejadian ini akan memunculkan *event Packet In* pada kontroler dan kontroler akan membentuk jalur baru sesuai mekanisme pengiriman data pada L2 Multi dengan data jalur switch yang valid.

2.4 Algoritma hasil modifikasi

POX memiliki beberapa metode dalam pengiriman packet pada layer 2, yaitu hub, L2 learning, dan L2 multi. L2 multi merupakan metode yang tercepat dalam melakukan penangganan kegagalan link. Tetapi, L2 Multi bergantung pada modul *discovery* yang menggunakan paket LLDP yang dikirimkan secara berkala.

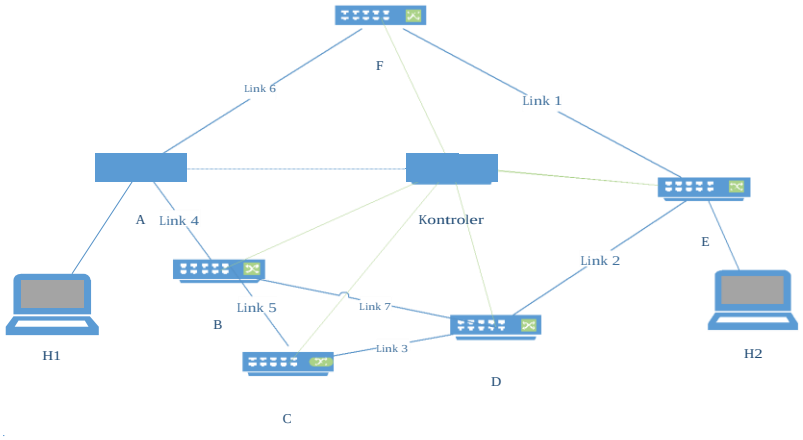
Pengiriman paket LLDP mempengaruhi kecepatan pendeteksian kegagalan link pada L2 Multi. Semakin cepat interval waktu pengiriman paket LLDP maka semakin baik penanganan kegagalan link tetapi menambah besarnya *overhead* pada jaringan.

Pada Openflow 1.0 terdapat mekanisme pendeteksian perubahan status port yang dapat dimanfaatkan untuk mendeteksi putusnya link tanpa menunggu pengiriman paket LLDP secara berkala. Pada penelitian ini dilakukan modifikasi pada L2 multi dengan menambahakan deteksi status port.

3. Pembahasan

3.1 Skenario Pengujian

Simulasi yang dilakukan dalam penelitian ini menggunakan topologi jaringan seperti pada **Error! Reference source not found.**, yang terdiri dari dua komputer host, 6 switch, dan satu kontroler.



Gambar 3-1 Topologi jaringan

3.2 Skenario 1

Pada skenario satu dilakukan uji overhead pada kontroler dan pada jaringan

Ada dua pengukuran yang dilakukan yaitu overhead pada sisi infrastruktur dan pada kontroler. Pada sisi infrastruktur adalah overhead yang terjadi pada jaringan datapath. Sedangkan pada sisi kontroler adalah overhead yang terjadi antara switch dengan kontroler.

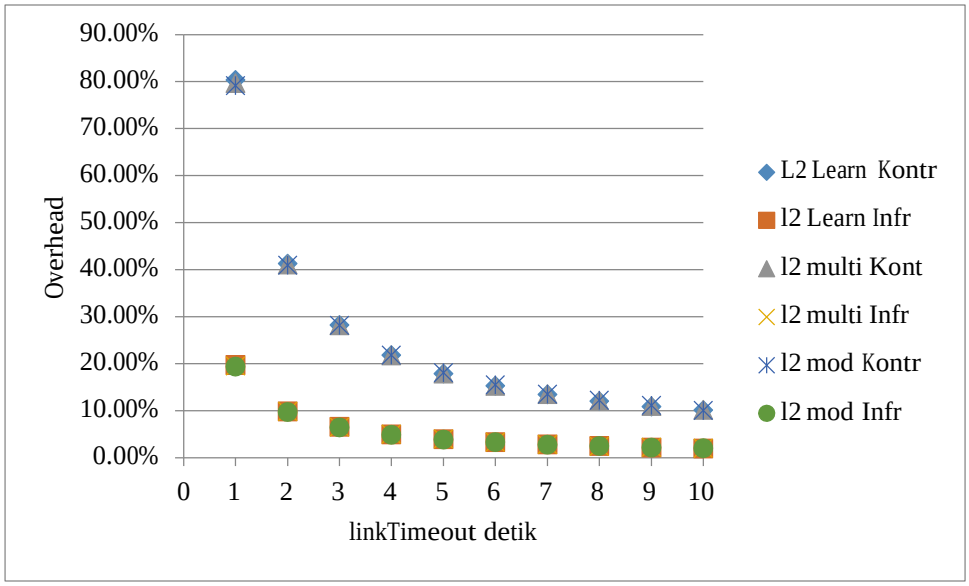
Pada skenario ini masing masing metode diterapkan pada kontroler dengan mengaktifkan modul *Discovery* dan *Spanning Tree*. Pada kondisi normal link-timeout diset pada nilai 10 detik. Pada pengujian tugas akhir ini modul *discovery* akan diset dengan *link-timeout* dari 1 sampai 10 detik. Host 1 akan melakukan ping dengan interval waktu 10 mili detik.

Perhitunggan *overhead* infrastruktur dilakukan pencatatan menggunakan wireshark pada host 1 dan pada switch A. Paket yang akan dicatat adalah paket ping sebagai informasi utama yang dikirim dan paket LLDP sebagai informasi tambahan dalam infrastruktur jaringan. Nilai *overhead* dari infrastruktur didapat dengan rumus:

$$\text{overhead} = \frac{\text{paket lldp}}{\text{paket ping}} \times 100\% \text{ [12].}$$

Perhitungan *overhead* kontroler dilakukan pencatatan pada host 1 dan kontroler. Paket yang dicatat yaitu paket ping informasi utama dan seluruh paket OpenFlow yang melalui kontroler. Nilai dari *overhead* kontroler didapat dengan rumus:

$$\text{overhead kontroler} = \frac{\text{paket openflow}}{\text{paket ping}} \times 100\% \text{ [12].}$$



Gambar 3-2 Grafik perbandingan antar metode

Overhead akan meningkat jika pada saat *link timeout* diperkecil. Ini diakibatkan semakin sedikit waktu untuk mendeteksi kegagalan link pada modul *discovery* maka akan mempercepat interval pengiriman paket LLDP pada jaringan yang mengakibatkan protokol OpenFlow mengirim data dari Switch ke kontroler. Hal ini akan membebani kerja dari kontroler jika terlalu cepat interval pengiriman paket LLDP.

Dari Gambar 3-2 terlihat bahwa tidak ada perbedaan overhead yang besar antara metode yang dicobakan. L*ink-timeout*=1 detik pada metode L2 learning detik *overhead* kontroler sebesar 80.37% dan *overhead* infrastruktur sebesar 19.74%. Pada L2 Multi *overhead* kontroler sebesar 79.68% dan overhead infrastruktur 19.51%. Sedangkan pada L2 modifikasi *overhead* kontroler sebesar 79.17% dan overhead infrastruktur 19.39%.

3.3 Skenario 2

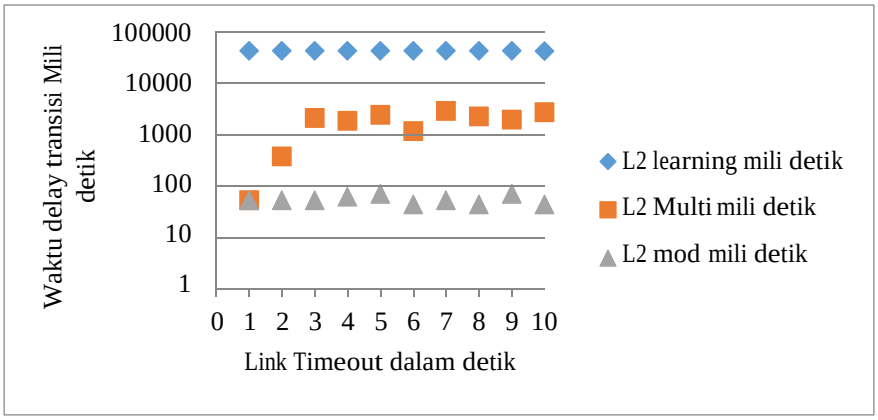
Pada skenario dua dilakukan pengujian waktu kegagalan link pada masing-masing metode.

Waktu perbaikan kegagalan link adalah waktu yang dibutuhkan oleh semua switch yang berada di dalam jaringan untuk kembali pada kedudukan siap (*steady state)* setelah adanya perubahan topologi pada jaringan [10]. Perubahan topologi dipengaruhi oleh terjadinya kegagalan link (*link failure*).

Modul discovery sangat berperan dalam pengujian ini karena fungsionalitas pedeteksian perubahan link terdapat pada modul ini. Lamanya waktu yang diperlukan switch untuk kembail pada kedudukan siap dipengaruhi oleh Kecepatan pendeteksian kegagalan link. Pada modul discovery kecepatan pendeteksian dapat diatur menggunkan perintah *link-timeout.*

Topologi yang digunakan seperti pada Gambar 3-1 dengan link 5 akan diputus sehingga hanya terdapat dua jalur, yaitu jalur satu ABDE dan Jalur dua AFE. Jalur dua akan diputus sehingga memaksa kontroler membentuk jalur pada siwtch ABDE(jalur satu). Pada rentang waktu 30 detik jalur dua akan dihubungkan dan 5 detik berikutnya jalur satu diputuskan. Pada kondisi ini akan memaksa kontroler untuk melakukan perubahan jalur.

Pengujian dilakukan dengan melakukan pengiriman paket ICMP Ping dari host 1 menuju ke host 2 dengan interval 10 mili detik. Pada modul *discovery* akan dilakukan perbahan nilai *link-timeout* dari 1 detik hingga 10 detik. Pengukuran dilakukan pada saat terjadinya kegagalan link yaitu pada saat ICMP_req dari host 2(*destination*) tidak diterima oleh host 1(*source*). Pada saat jalur telah diubah oleh kontroler host 1 akan menerima kembali ICMP_req dari host 2. Selisih waktu saat paket ICMP_req diterima (T_1) dengan waktu saat ICMP_req mulai tidak diterima(T_0) adalah waktu yang perbaikan kegagalan link. $T = T_1 - T_0$.



Gambar 3-3 grafik waktu penanganan kegagalan link

Link *timeout* hanya berpengaruh pada *L2 Multi*. Sedangkan *L2 Learning* dan L2 Modifikasi tidak berpengaruh. Hal ini terjadi karena penanganan *L2 learning* bergantung pada Link *timeout* pada modul *discovery* untuk mendeteksi terjadinya kegagalan link. Sedangkan pada *L2 learning* penanganan kegagalan link bergantung dengan ARP *request* yang dilakukan host dan pada *L2* modifikasi menggunakan port status dari switch OpenFlow dalam melakukan penanganan kegagalan link.

Pada Gambar 3-3 menunjukan bahwa L2 multi berpengaruh pada *link-timeout* modul *discovery*. Sedangkan L2 learning dan L2 modifikasi tidak berpengaruh. Pada L2 learning memiliki waktu penanganan kegagalan link rata-rata 48426.4 mili detik(48,4 detik). Sedangkan L2 Mod memiliki waktu penanganan rata-rata 62 milidetik. Sedangkan untuk L2 Multi waktu tercepat dengan link Timeout 1 detik sebesar 60 milidetik dan nilai terbesar dengan link Timeout 10 detik sebersar 3,1 detik.

3.4 Skenario 3

Skenario 3 adalah pengujian dilakukan dengan menambah jumlah switch hingga tiga kali lipat. Analisis dilakukan dengan mengamati pengaruh jumlah switch dengan waktu perbaikan kegagalan link.

Switch akan ditambah pada masing-masing jalur sehingga membentuk sebuah topologi ring. Switch akan ditambahakan antara link AF dan link AB. Pengujian dilakukan dengan melakukan pengiriman paket ICMP Ping dari host 1 menuju ke host 2 dengan interval 10 mili detik. Kemudian dilakukan proses perubahan jalur seperti pada skenario 2.

Metode (dalam mili detik)	Jumlah Swtich										
	6	7	8	9	10	11	12	13	14	15	16
L2 multi	3905	3585	3405	4205	4797.5	3720	4520	3670	4225	3835	3986.8
L2 Mod	62	62.5	52.5	65	50	55	52.5	65	52.5	62.5	67.5

Tabel 3-1 Hubungan jumlah switch dengan waktu penanganan kegagalan link

Pada Tabel 3-1 terlihat penambahan jumlah switch antara enam hingga enam belas switch tidak begitu mempengaruh waktu penangganan kegagalan link. Pada L2 multi diperoleh hasil rata-rata 3986 mili detik, sedangkan L2 modifikasi diperoleh waktu rata-rata 59 mili detik.

4. Kesimpulan

Berdasarkan tujuan serta hasil dari pengujian dan analisis yang telah dilakukan pada pengujian kegagalan link pada SDN Openflow menggunakan metode L2 Learning, L2 multi, dan algorima prototipe L2 Modifikasi diperoleh

kesimpulan bahwa L2 modifikasi melakukan terbaik dalam penanganan kegagalan link dengan waktu tercepat 59 milidetik dengan overhead hanya sebesar 10.1% dengan rincian sebagai berikut:

1. Metode L2 modifikasi memiliki rata-rata lebih cepat melakukan penanganan kegagalan link dibanding L2 Learning dan L2 Multi dengan kecepatan rata-rata 59 milidetik dengan jumlah switch maksimal 16 buah.
2. Metode L2 modifikasi memiliki overhead yang lebih rendah 70,3% dibanding metode L2 multi dengan kecepatan penanganan yang sama sebesar.
3. Penggunaan metode L2 modifikasi memperoleh waktu rata-rata sebesar 59 milidetik, ini masih dibawah standar dari "*Carrier Grade*" sebesar 50 milidetik.

Daftar Pustaka:

[1] S. Sharma, D. Staessens, D. Colle, M. Pickavet and P. Demeester, "Enabling Fast Failure Recovery in OpenFlow Networks," *8th Internasional Workshop on the DRCN*, pp. 164-171, 2011.

[2] M. Pustylnik, M. V. Zafirovic, M. Zafirovic-Vukotic and R. Moore, "Performance of the Rapid Spanning Tree Protocol in Ring Network Topology," 2007. [Online]. Available: http://w3.siemens.com/mcms/industrial-communication/en/rugged-communication/Documents/rstp-in-ring- network-topology-en.pdf. [Accessed 10 Mei 2015].

[3] MEF, "Metro Ethernet Forum," Juni 2013. [Online]. Available: https://www.metroethernetforum.org/carrier-ethernet-services/carrier-ethernet-and-ce-2-0. [Accessed 5 Mei 2015].

[4] ONF, "Software-Defined Networking (SDN) Definition," Open Networking Foundation, [Online]. Available: https://www.opennetworking.org/sdn-resources/sdn-definition. [Accessed 9 Mei 2015].

[5] ONF, "Software-Defined Networking: The New Norm for Networks," 13 April 2012. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf. [Accessed 11 Mei 2015].

[6] sdxcentral, "What is OpenFlow?," [Online]. Available: https://www.sdxcentral.com/resources/sdn/what-is-openflow/. [Accessed 2015 Juni 14].

[7] openflow.org, "OpenFlow Switch Specification, Version 1.0.0," [Online]. Available: http://archive.openflow.org/documents/openflow-spec-v1.0.0.pdf. [Accessed 10 Mei 2015].

[8] M. McCauley, "About POX," [Online]. Available: http://www.noxrepo.org/pox/about-pox/. [Accessed 10 Mei 2015].

[9] sdxcentral, "NOX, POX and Controllers Galore – Murphy McCauley Interview," [Online]. Available: https://www.sdxcentral.com/articles/interview/nox-pox-controllers-murphy-mccauley/2012/09/. [Accessed 20 mei 2015].

[10] M. Team, "Mininet Overview," [Online]. Available: http://mininet.org/overview/. [Accessed 25 Mei 2015].

[11] Minnesota Supercomputer Center, 12 agustus 1994. [Online]. Available: http://www.sonic.net/support/docs/ip-atm.overhead.pdf. [Accessed 25 mei 2015].

[12] A. K.P, "Communication overhead of an OpenFlow wireless mesh network," *Advanced Networks and Telecommuncations Systems (ANTS), 2014 IEEE International Conference on*, pp. 1-6, 2014.

[13] A. Basu and J. G. Riecke, "Stability Issues in OSPF Routing," *SIGCOMM'01*, pp. 225-236, 2001.

[14] N. Feamster, "The Road to SDN: An Intellectual History of Programmable Networks," 2013. [Online]. Available: http://www.cs.princeton.edu/courses/archive/fall13/cos597E/papers/sdnhistory.pdf. [Accessed 10 Mei 2015].