Jurnal Teknologi Informasi dan Ilmu Komputer (JTIIK) DOI: 10.25126/jtiik.202296715
Vol. 9, No. 6, Desember 2022, hlm. 1315-1322 p-ISSN: 2355-7699
Akreditasi KEMENRISTEKDIKTI, No. 36/E/KPT/2019 e-ISSN: 2528-6579

PENERAPAN DYNAMIC FLOW REMOVAL UNTUK MENCEGAH FLOW TABLE OVERFLOW PADA SOFTWARE-DEFINED NETWORKING

Achmad Basuki[*1], Kasyful Amron[2], Primantara Hari Trisnawan[3]

[1,2,3]Universitas Brawijaya, Malang
Email: [1]abazh@ub.ac.id, [2]kasyful@ub.ac.id, [3]prima@ub.ac.id
[*]Penulis Korespondensi

(Naskah masuk: 24 November 2022, diterima untuk diterbitkan: 19 Desember 2022)

Abstrak

Software-Defined Networking (SDN) memungkinkan penerusan paket data secara terprogram dalam sebuah jaringan dengan mendefinisikan rincian *flow* dalam *flow table* setiap switch jaringan. Namun, kapasitas *flow table* adalah sumber daya yang terbatas, sehingga memerlukan pengelolaan yang cermat untuk hal ini. Artikel ilmiah ini membahas pengelolaan *flow table* dengan metode penghapusan rincian flow secara dinamis (*dynamic flow removal*) untuk mencegah terjadinya *flow table overflow* pada SDN. *Dynamic flow removal* yang dimaksud adalah dengan melakukan pemantauan *flow expiry* dan secara selektif melakukan penghapusan rincian *flow* yang sudah tidak lagi aktif sehingga dapat mengurangi jumlah okupansi rincian *flow* pada *flow table*. Penghapusan rincian *flow* secara selektif akan dipicu setiap kali kapasitas *flow table* hampir penuh. Implementasi *dynamic flow removal* dilakukan dengan studi kasus aplikasi server *load-balancing* berbasis *round-robin* pada SDN dengan framework Ryu, Mininet, dan modifikasi kapasitas *flow table* pada OpenvSwitch. Hasil pengujian menunjukkan bahwa penerapan metode yang diusulkan mampu mencegah terjadinya *flow table overflow* dengan 100% rincian *flow* aktif dapat menempati *flow table* tanpa menyebabkan kegagalan komunikasi *client-server*.

Kata kunci: *Software-Defined Network, OpenFlow, Flow Table Overflow, Flow Removal, Server Load-balancing*

AVOIDING FLOW TABLE OVERFLOW USING DYNAMIC FLOW REMOVAL ON SOFTWARE-DEFINED NETWORKING

Abstract

Software-Defined Networking (SDN) enables programmable packet forwarding by defining flow rules in the flow table of each network switch. However, the flow table capacity is a limited resource that requires careful management. This paper discusses the implementation of dynamic flow removal in managing flow tables in an OpenFlow-based SDN switch to prevent flow table overflow. Dynamic flow removal is realized by monitoring flow expiry and selectively removing flow rules that are no longer active to reduce the number of flow rules in the flow table. Selective removal of flow rules will be triggered whenever the flow table capacity is almost full. Dynamic flow removal was implemented using a case study of a round-robin-based load-balancing server application on SDN with Ryu framework, Mininet, and a modified flow table space in OpenvSwitch. The evaluation results indicate that the proposed method can prevent flow table overflow while maintaining 100% of active flow rules in the flow table without compromising client-server communication.

Keywords: *software defined network, OpenFlow, flow table overflow, flow removal, server load balancing*

1. PENDAHULUAN

Software-Defined Network (SDN) telah dikembangkan sejak tahun 2008 untuk membuat jaringan yang inovatif dan efisien untuk menyelesaikan masalah-masalah dalam jaringan tradisional (McKeown dkk., 2008). Arsitektur SDN memisahkan *control-plane* dan *data-plane* yang memungkinkan inovasi baru dilakukan dalam pengelolaan infrastruktur jaringan yang berbasiskan pemrograman *flow table* melalui protokol OpenFlow (ONF, 2012). OpenFlow telah banyak diterapkan pada berbagai data-plane switch jaringan, sehingga memungkinkan pengembang aplikasi infrastruktur jaringan melakukan pemrograman flow (*flow-based*) pada *control-plane* melalui *controller* SDN.

Selain memungkinkan inovasi lebih cepat dilakukan pada SDN, tetapi juga memunculkan masalah baru tentang keterbatasan kapasitas *flow table* dari perangkat keras switch OpenFlow yang menyimpan rincian *flow* (Adam Zarek, 2012).

Keterbatasan yang dimaksud dikarenakan *flow table* umumnya menempati ruang memory terbatas dengan kemampuan waktu pencarian dan pembacaan sangat cepat pada komponen yang dikenal dengan *Ternary Content Addressable Memory* (TCAM). Penggunaan TCAM dapat memberikan keuntungan kecepatan penerusan paket data setidaknya 40 kali lebih cepat dibandingkan tanpa penggunaan TCAM (D. Moon dkk., 2010). Namun demikian, TCAM adalah komponen mahal pada perangkat keras *data-plane* switch (Cheng, Tao, dkk, 2018) dan kapasitasnya sangat terbatas pada kisaran ribuan rincian *flow* saja (Li, Rui, dkk, 2019).

Penggunaan TCAM pada *data-plane* switch harus dikendalikan karena jika TCAM penuh, maka rincian *flow* lain yang datang kemudian dapat diabaikan. Sementara itu, *controller* SDN akan selalu menuliskan *flow* berdasarkan mekanisme "match-and-action" pada *flow table*. Apabila penulisan jumlah *flow* tidak terkendali, maka ruang TCAM akan segera penuh dan selanjutnya menimbulkan masalah operasional dalam SDN yang dikenal dengan *flow table overflow* seperti yang dinyatakan oleh (Zarek, A., 2012), (N. Kang, dkk., 2015), (Zehua, G. dkk, 2018), dan (Li, Rui, dkk, 2019).

Data-plane switch OpenFlow meneruskan paket data secepat-cepatnya berdasarkan rincian *flow* yang tersimpan pada *flow table*. Ketika paket data yang datang tidak sesuai rincian *flow* yang tertulis pada *flow table*, data-plane akan meminta rincian *flow* baru dari control-plane (saat kondisi ini penerusan paket data akan lebih lambat). Sesaat kemudian, control-plane akan menuliskan *flow* baru ke *flow table* pada data-plane switch. Namun demikian, bila *flow table* sedang penuh, maka control-plane tidak lagi dapat memprogram data-plane yang dapat berakibat kegagalan meneruskan paket data (*packet data loss*) sesuai rincian *flow* baru. Kondisi *flow table overflow* pada data-plane dapat menyebabkan beban *controller* meningkat dan menyebabkan penurunan kinerja jaringan SDN seperti dijelaskan dalam (Zehua, G dkk, 2018), (Cheng, Tao, dkk, 2018).

Kemungkinan terjadinya permasalahan *flow table overflow* pada data-plane switch OpenFlow dapat dihindari dengan mekanisme penghapusan rincian *flow* yang tidak lagi berguna. Spesifikasi OpenFlow Switch versi 1.3.0 (ONF, 2012) mendefinisikan mekanisme penghapusan *flow* (*flow removal*) dalam dua cara: 1) atas permintaan *controller*, dan 2) *flow expiry* berdasarkan nilai tidak nol parameter *idle timeout* dan *hard timeout*. Secara spesifik, switch OpenFlow harus secara independen menghapus rincian *flow* ketika salah satu *timeout* terlampaui.

Eksplorasi penggunaan mekanisme *flow expiry* telah diulas oleh (Adam Zarek, 2012) yang mendapati bahwa penggunaan *flow expiry* hanya efektif untuk kasus-kasus tertentu, dimana penggunaan ulang *flow* tidak sangat sering terjadi. Namun demikian, bila perpanjangan *timeout* dilakukan, maka akan terdapat sejumlah rincian flow dengan masa tidak aktif mengokupansi *flow table* yang dapat berujung pada terjadinya *flow table overflow*. Adam Zarek menyimpulkan bahwa kombinasi kedua mekanisme *flow removal* adalah cara terbaik dalam manajemen *flow table*. Pada eksperimen yang ia lakukan dengan menggunakan algoritma FIFO dan algoritma *random replacement* dalam menentukan rincian *flow* yang akan dihapus menunjukkan hasil yang menjanjikan walaupun belum sepenuhnya dapat menanggulangi ledakan jumlah rincian *flow* yang mungkin terjadi.

Pada jaringan yang sangat sibuk, jumlah rincian *flow* dalam *flow table* dapat bertambah secara eksponensial. Temuan oleh (T.Benson dkk., 2010) menunjukkan pada jaringan data center dengan 2000 server terdapat jumlah rincian *flow* aktif pada kisaran 1000 – 5000 dalam 90% waktu. Sementara itu, keberadaan produk komersial perangkat keras switch OpenFlow hanya dapat menampung rincian *flow* antara 1000 – 4000 pada *flow table* yang menggunakan TCAM (*hardware flow table*) dan selebihnya akan menggunakan *software flow table* menggunakan RAM (Rygielski, Piotr, dkk., 2017). Produk switch OpenFlow umumnya telah menggunakan metode hibrid antara *hardware flow table* dan *software flow tabl*e untuk penerusan paket data. Apabila *hardware flow table* penuh, maka *software flow table* akan digunakan dan berdampak penurunan kinerja menjadi kurang dari 10% dari kapasitas maksimal seperti hasil evaluasi oleh (Rygielski, Piotr, dkk., 2017).

Berdasarkan fakta produk switch OpenFlow tersebut, beragam penelitian atas keterbatasan *hardware flow table* dapat dikategorikan menjadi dua metode pengelolaan *flow table*: **1) penghapusan rincian flow secara dinamis berdasarkan *idle timeout* dan *hard timeout*** (***dynamic flow removal***) seperti pada (Adam Zarek, 2012), (H. Zhu, dkk., 2015), (L. Zhang, dkk., 2015), (He, Cheng-Hun, dkk., 2018), dan (H. Yang, dkk., 2018) dan **2) reduksi algoritmik rincian *flow* melalui kompresi atau *caching*** (***algorithmic flow reduction***) seperti pada (C.R. Meiners. dkk., 2012), (N. Kang, dkk., 2015), (Zehua,G. dkk., 2018), (Cheng, Tao, dkk., 2018), dan (Li, Rui, dkk., 2019).

Pada penelitian ini dilakukan implementasi teknis pengelolaan *flow table* dengan metode *dynamic flow removal* untuk menghindari terjadinya *flow table overflow* pada SDN. Kontribusi penelitian ini adalah sebuah referensi penerapan teknis pengelolaan *flow table* pada lingkungan SDN berbasis switch OpenFlow (Open vSwitch) dengan *controller* berbasis framework RYU (Ryu SDN). Realisasi lingkungan penelitian menggunakan emulator SDN (Mininet) dengan konfigurasi pembatasan kapasitas *flow table* pada OpenFlow Virtual Switch (OpenvSwitch). Sementara itu, untuk memicu terjadinya *flow table overflow* menggunakan studi kasus server *load-balancing* pada *controller* SDN dengan jumlah *request* jamak dari sisi klien.

2. STUDI LITERATUR

Penggunaan TCAM sebagai *hardware flow table* sangat signifikan meningkatkan kinerja data-plane switch OpenFlow dibandingkan dengan *software flow table* yang menggunakan RAM seperti dikonfirmasi oleh (D. Moon dkk., 2010) dan (Rygielski, Piotr, dkk., 2017). Faktanya mayoritas produk switch OpenFlow menggunakan keduanya secara hibrid dikarenakan keterbatasan kapasitas ruang penyimpanan pada *hardware flow table* (Li, Rui, dkk., 2019). Dengan demikian pengelolaan *flow table* menjadi sangat krucial pada perangkat keras switch OpenFlow dalam menjaga kinerja jaringan SDN.

Pengelolaan *flow table* dengan metode penghapusan rincian *flow* secara dinamis (*dynamic flow removal*) sangat sederhana dan hanya berbasiskan pengaturan nilai *idle timeout* dan *hard timeout* serta *flow modification* pada setiap rincian *flow* karena tidak terlalu memerlukan proses yang rumit pada sisi *controller* SDN. Adam Zarek dalam (Adam Zarek, 2012) telah mengulas secara komprehensif penggunaan *flow expiry* berdasarkan nilai *flow timeout* dan *flow modification* untuk secara proaktif menghapus rincian *flow*. Pada penggunaan *flow expiry*, jika nilai *timeout* yang diberikan terlalu besar, maka okupansi rincian *flow* pada flow table akan terlalu lama dan penambahan rincian *flow* baru akan cepat mencapai batasan kapasitas *hardware flow table* yang dapat memicu *flow table overflow*. Sementara itu, *flow modification* yang secara proaktif melakukan penghapusan rincian *flow* yang tidak lagi diperlukan atau pun mempertahankan rincian *flow* yang masih aktif sangat membantu menghindari terjadinya *flow table overflow*. Namun demikian, penggunaan *flow modification* yang terlalu sering sangat membebani *controller* SDN dan dapat berdampak pada berkurangnya kinerja penerusan paket data.

Secara prinsip, pengelolaan *flow table* dengan metode *dynamic flow removal* merupakan teknik yang mengkombinasikan antara *flow expiry* dan *flow modification*. Pada penelitian (Adam Zarek, 2012) mengusulkan algoritma penjadwalan FIFO untuk *flow expiry* dan algoritma penggantian (*random replacement*) rincian *flow* secara acak melalui *flow modification*. Penelitian sejenis pada (L. Zhang, dkk., 2015) lebih jauh melakukan eksplorasi berdasarkan pola trafik jaringan variasi *short - long* dan *small - large flow* dengan memperhatikan durasi *flow* yang aktif, tipe *flow*, dan rasio utilisasi *flow table* untuk kemudian menentukan *fixed* dan *dynamic timeout*. Sementara itu, pada penelitian (H. Zhu, dkk., 2015) hanya fokus pada penentuan *dynamic timeout*.

Penelitian pada (H. Yang, dkk., 2018) melakukan eksplorasi teknik pembelajaran mesin dalam penghapusan rincian *flow* secara proaktif melalui *flow modification* tanpa memperhatikan *flow timeouts*. Penerapan teknik pembelajaran mesin didasarkan pada statistik rincian *flow* yang dihapus dan melakukan prediksi rincian *flow* dengan durasi aktif paling pendek. Teknik dalam penelitian ini secara prinsip hanya fokus pada menemukan cara paling efisien untuk mengurangi beban proses pada *controller* SDN dengan mempertimbangkan rincian *flow* berdasarkan algoritma penjadwalan FIFO dan melakukan penghapusan rincian *flow* secara acak (*random deletion*) untuk mencegah terjadinya *flow table overflow*. Meski demikian, perilaku penghapusan acak dapat mengakibatkan kesalahan penghapusan rincian *flow* yang masih aktif yang berujung pada peningkatan beban proses controller SDN dan penurunan kinerja penerusan paket data.

Penelitian lain pada (He, Cheng-Hun, dkk., 2018) mengimplementasikan mekanisme *flow expiry* dengan nilai timeout 0 (rincian *flow* tidak pernah *timeout*) dengan melakukan penghapusan rincian *flow* melalui *flow modification* secara reaktif ketika mendeteksi *packet header flag* FIN dan RST pada protokol TCP. Deteksi packet header flag FIN dan RST dengan mengasumsikan telah selesainya sesi komunikasi pada TCP di layer transport. Sekalipun mekanisme ini dapat mencegah *flow table overflow* dan juga mengurangi beban *controller* SDN, tetapi solusi ini tidak mengendalikan rincian *flow* selain TCP.

Pendekatan lain untuk menghindari *flow table overflow* adalah *algorithmic flow reduction* seperti yang dilakukan pada penelitian (C.R. Meiners. dkk., 2012), (N. Kang, dkk., 2015), (Zehua,G. dkk., 2018), (Cheng, Tao, dkk., 2018), dan (Li, Rui, dkk., 2019). Penelitian tersebut pada dasarnya menggunakan mekanisme reduksi jumlah rincian *flow* yang akan ditempatkan pada *hardware flow table* dengan kombinasi yang diletakkan pada *software flow table*, baik dengan mekanisme *caching* atau pun agregasi dengan mempertimbangkan *wildcard flow*. Namun demikian, teknik ini memerlukan modifikasi pada perangkat keras data-plane switch agar dapat menempatkan program di ruang RAM yang sangat spesifik untuk setiap vendor produk switch OpenFlow.

Pada penelitian ini melakukan eksplorasi implementasi teknis pengelolaan *flow table* menggunakan pendekatan *dynamic flow removal* karena tidak memerlukan modifikasi pada sisi data-plane switch. Penerapan *flow expiry* berdasarkan *idle timeout* dan *hard timeout* dipilih yang secara natural akan otomatis menghapus rincian *flow* yang sudah kadaluwarsa durasi aktifnya. Sementara itu, penghapusan rincian *flow* secara selektif diberlakukan apabila okupansi *flow table* mendekati batas kapasitasnya berdasarkan urutan durasi aktif setiap *flow* melalui *flow modification* dari *controller* SDN.

3. PENGELOLAAN FLOW TABLE

Pada penelitian ini, kami menerapkan pengelolaan *flow table* secara dinamis atau *dynamic flow removal* untuk mencegah terjadinya *flow table*

overflow pada studi kasus sistem server *load-balancing* di SDN. Penelitian ini menggunakan dua mekanisme *flow removal*. Mekanisme pertama adalah *flow expiry* dengan menetapkan nilai ke *idle timeouts*. Sedangkan mekanisme yang kedua adalah mekanisme *flow modification* yang biasa disebut dengan metode penghapusan rincian *flow* secara proaktif.

Realisasi metode yang diusulkan dengan membuat aplikasi server *load-balancing* dengan algoritma round-robin sederhana pada *controller* SDN dengan framework RYU (Ryu SDN). Pemilihan *idle timeout* yang diberikan adalah 60 detik. Sedangkan, metode *flow modification* akan mengirimkan pesan kontrol eksplisit ke *controller* SDN dengan terlebih dahulu memeriksa kondisi rincian *flow*. *Controller* SDN akan memeriksa rincian *flow* setiap 5 detik dan apabila ditemukan rincian *flow* yang tidak lagi memiliki aktivitas, maka *controller* akan mengirimkan pesan penghapusan ke switch untuk rincian *flow* tersebut beserta parameter *cookie* (identitas spesifik setiap *flow*).

```
Class Round Robin Server_Load_Balancing{

        Application of round_robin_server_load_balancing{
        ...
        set idle_timeouts 60 second for all new_flow_entry
        while defined new flow entry in flow table
        ...
        }

        max_flow_entries = read maximum capacity of flow entries

        Monitoring_Flow_Entries()
                Send request traffic monitoring to OpenFlow switch
                Read Reply from OpenFlow switch
                Fetch data number of flow entries

        num_of_flow = call function Monitoring_Flow_Entries
        FOR i in num_of_flow:
                IF NOT new_flow_entry in flow_temp:
                        flow_temp[] = save information
                                      of new_flow_entries
                                      (cookie, duration, packet_count)

                ELIF new_flow_entry in flow_temp:
                        new_flow_entry.index = flow_temp.index(cookie)

                        IF new_flow_entry.packet_count > flow_temp.packet_count:
                                print flow_entry (it means that flow_entry has activity)

                        ELIF new_flow_entry.packet count == flow_temp.packet count:
                        #it means flow_entry has no activity
                                IF num_of_flow > 80% of max_flow_entries:
                                Send message to controller to delete flow with
                                cookie of flow entry that has no activity (FLOW MODIFICATION)

                                ELSE :
                                        Flow entry will be deleted by idle_timeouts (FLOW EXPIRY)
```

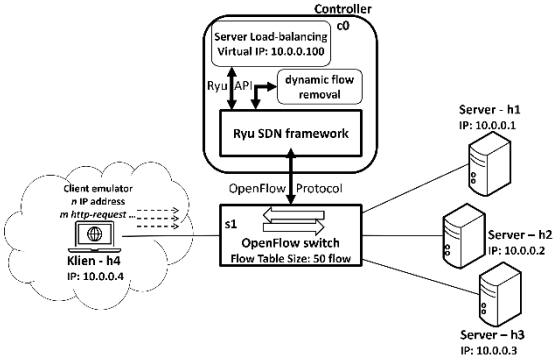
Gambar 1. Pseudocode Pengelolaan Flow Table secara Hibrid

Pseudocode seperti tampak pada Gambar 1 menunjukkan mekanisme keseluruhan *dynamic flow removal* yang merupakan mekanisme hibrid antara *flow expiry* dan *flow modification*. Pada inisiasi awal, membaca kapasitas *flow table* (*max_flow_entries*) dari switch OpenFlow dan selanjutnya memanggil fungsi monitoring *flow* (*num_of_flow*) dengan mencatat nomor identitas, durasi dan jumlah paket pada setiap *flow*. Apabila fungsi server *load-balancing* terpanggil, maka setiap *flow* akan diberikan *idle_timeout* 60 detik sebagai *flow expiry*.

Selanjutnya, mekanisme penghapusan rincian *flow* melalui pesan *flow modification* mengacu pada fungsi monitoring *flow* setiap 5 detik terhadap rincian *flow* yang masih aktif (*flow_entry.packet_count* > *flow_temp.packet_count*). Ketika kondisi *flow table* < 80% *max_flow_entries* (kapasitas *flow table*), maka *flow expiry* berdasarkan nilai *idle timeout* 60 detik akan dieksekusi. Ketika kondisi *flow table* > 80% kapasitas *flow table*, maka penghapusan rincian *flow* secara proaktif dengan metode *flow modification* akan dieksekusi. Penghapusan rincian *flow* dilakukan berdasarkan urutan durasi dan jumlah paket untuk setiap flow yang direkam (*flow_temp.index*) dan menghindari penghapusan aktivitas *flow* yang masih aktif.

4. IMPLEMENTASI

Penerapan mekanisme *dynamic flow removal* untuk mencegah terjadinya *flow table overflow* dilakukan pada studi kasus server *load-balancing*. Penerapan aplikasi server *load-balancing* pada SDN mengadopsi pendekatan pada (Al-Najjar, A., dkk, 2016) dengan lingkungan emulator SDN menggunakan Mininet (Bob Lantz dkk., 2010), (Mininet) dengan Controller SDN berbasiskan framework Ryu (Tomonori, F., 2013), (Ryu SDN). Lingkungan eksperimen pengelolaan *flow table* pada SDN dengan mekanisme *dynamic flow removal* bersama aplikasi server *load balancing* direalisasikan dalam Mininet dengan ilustrasi topologi jaringan beserta komponennya seperti tampak pada Gambar 2. Lingkungan eksperimen tersebut terdiri atas 3 host (*h1*, *h2*, *h3*) sebagai server, 1 OpenFlow switch (*s1*) dengan kapasitas *flow table* terbatas pada 50 *flow*, 1 SDN *controller* (*c0*) dengan Ryu framework dan menjalankan aplikasi server *load-balancing* dengan *virtual* IP 10.0.0.100 dan *dynamic flow removal*, dan tentunya 1 host (*h4*) sebagai emulator klien yang secara acak berganti *n* IP address dan membuat sejumlah *m http-request*.



Gambar 2. Lingkungan SDN Emulator - Mininet

Spesifikasi dan konfigurasi lingkungan eksperimen yang digunakan dalam penelitian ini adalah seperti ditunjukkan pada Tabel 1. Keseluruhan program dan petunjuk penggunannya dapat diakses di *https://github.com/abazh/flowtable_slb.git*. Program dalam bahasa pemrograman Python yang terdapat pada repositori tersebut terdiri atas 3 program utama: 1) program server *load balancing* yang menerapkan mekanisme *dynamic flow removal* dan dieksekusi pada *controller* Ryu melalui perintah *ryu-manager*

flowtable_lb.py, 2) program untuk aktivasi lingkungan Mininet melalui perintah *sudo python net_topo.py* yang akan mengaktifkan topologi jaringan dan web server pada 3 *hosts* serta konfigurasi batas maksimum *flow table* 50 *flow*, dan 3) program untuk emulasi klien untuk membangkitkan 100 *http-request* dari 20 IP acak melalui perintah di konsol terminal mininet *h4 python client_emulation.py*. Pada Open vSwitch nilai *default* batas maksimum pada setiap *flow table* adalah 1.000.000 *flow* dan untuk eksperimen ini perlu mengubah maksimum jumlah *flow* menjadi 50 untuk memicu kejadian *flow table overflow*. Tampilan ketika mengeksekusi eksperimen adalah seperti tampak pada Gambar 3 untuk kondisi awal dan Gambar 4 ketika terjadi penghapusan rincian *flow* dikarenakan jumlah rincian flow mendekati kapasitas maksimum *flow table*.

Tabel 1. Spesifikasi dan konfigurasi lingkungan eksperimen

Sistem Operasi	Linux Ubuntu 20.04
Emulator SDN	Mininet
Controller SDN	Ryu version 4.34
Bahasa Pemrograman	Python v3.8.2
Aplikasi SDN	Server Load-balancing (round-robin) – virtual IP 10.0.0.100
Switch SDN	OpenvSwitch v2.13.8
Protocol SDN	OpenFlow v1.3
Jumlah Host	3 Server dan 1 klien
Jumlah *request* klien	100 *http request*
Jumlah klien IP address	20 (acak)
Interval antar *request*	0.5 detik
Kapasitas Flow Table	50 *flow*

```
[redacted]~/flowtable_slb$ ryu-manager flowtable_lb.py
loading app flowtable_lb.py
loading app ryu.controller.ofp_handler
instantiating app flowtable_lb.py of loadbalancer
Initialized new Object instance data
instantiating app ryu.controller.ofp_handler of OFPHandler
Flow Entry saat ini : 1
  cookie           duration         packets     bytes
________________ _________________ __________  __________
0000000000000000        2                0           0
Flow Entry saat ini : 13
  cookie           duration         packets     bytes
```

```
Web Server running in h2 Web Server running in h3 [1] 2274

************************
*** Starting CLI:
mininet> h4 python client_emulation.py
http-request ke- 1  dari IP:  10.0.0.17  RTT(ms):  1045.585155
http-request ke- 2  dari IP:  10.0.0.11  RTT(ms):  1052.570105
http-request ke- 3  dari IP:  10.0.0.17  RTT(ms):  2.090931
http-request ke- 4  dari IP:  10.0.0.13  RTT(ms):  1066.653013
http-request ke- 5  dari IP:  10.0.0.21  RTT(ms):  1057.313919
http-request ke- 6  dari IP:  10.0.0.20  RTT(ms):  1054.085016
```

Gambar 3. Tampilan kondisi awal dari 2 terminal konsol

```
Flow Entry saat ini : 45
  cookie           duration         packets     bytes
________________ _________________ __________  __________
cf6d4f689c813307       55                18          1398
16fbf53213cc064b       55                17          3561
Flow Table PENUH!!!! Perlu dilakukan penghapusan paksa !!
menghapus flow entry dengan 'total duration' paling lama . . . .0xcf6d4f689c813307
flow dengan cookie 0xcf6d4f689c813307 telah dihapus
7bbe7c55a02d4a0e       52                19          1464
fa0c26f804d62d7b       52                17          3561
Flow Table PENUH!!!! Perlu dilakukan penghapusan paksa !!
menghapus flow entry dengan 'total duration' paling lama . . . .0x7bbe7c55a02d4a0e
```

```
http-request ke- 31  dari IP:  10.0.0.15  RTT(ms):  1054.270983
http-request ke- 32  dari IP:  10.0.0.15  RTT(ms):  1.342058
http-request ke- 33  dari IP:  10.0.0.13  RTT(ms):  1077.039003
http-request ke- 34  dari IP:  10.0.0.15  RTT(ms):  2.794981
http-request ke- 35  dari IP:  10.0.0.19  RTT(ms):  1054.251909
http-request ke- 36  dari IP:  10.0.0.9  RTT(ms):  3.137112
http-request ke- 37  dari IP:  10.0.0.10  RTT(ms):  1065.510988
http-request ke- 38  dari IP:  10.0.0.20  RTT(ms):  2.869129
http-request ke- 39  dari IP:  10.0.0.21  RTT(ms):  1066.493988
http-request ke- 40  dari IP:  10.0.0.22  RTT(ms):  1054.419041
http-request ke- 41  dari IP:  10.0.0.6  RTT(ms):  1057.644129
```

Gambar 4. Tampilan saat terjadi penghapusan rincian *flow*

Tampilan pada Gambar 3 dan Gambar 4 juga menjelaskan cara kerja eksperimen untuk keperluan pengujian berdasarkan spesifikasi pada Tabel 1 dimana controller Ryu dengan aplikasi server *load balancing* dan *dynamic flow removal* memonitor lalu lintas paket data pada switch OpenFlow *s1*. Pada saat klien *h4* mengirimkan *http-request* setiap 0.5 detik ke *virtual* IP 10.0.0.100 dari aplikasi server *load balancing* dan pada *flow table* switch *s1* belum terdapat rincian *flow* yang sesuai, maka switch *s1* akan mengirimkan pesan *packet_in* ke controller *c0* dan menentukan IP server (server *h1*, *h2*, atau *h3*) yang dituju secara *round-robin* yang kemudian menuliskan *flow* baru dengan nilai *idle_timeouts* 60 detik pada *flow table*. Secara detil *flow table* pada switch *s1* akan terus berisikan setiap rincian *flow* dari setiap IP address klien ke server dan juga sebaliknya.

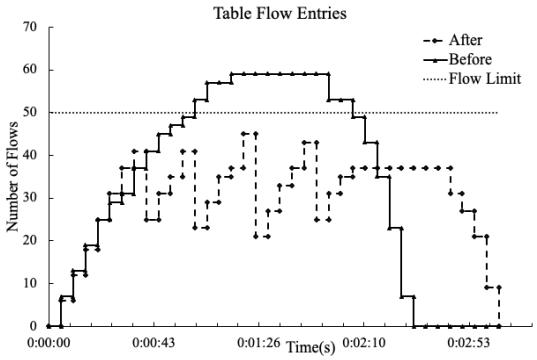
Pada kondisi berikutnya, apabila sudah terdapat rincian *flow* yang sesuai, maka switch *s1* akan langsung meneruskan paket tanpa perlu intervensi controller *c0*. Hal tersebut dapat terlihat pada Gambar 3 ketika *http-request* ke-1 dan ke-3 yang dikirimkan dari IP sama menghasilkan nilai RTT berbeda (1045.58 *ms* dan 2.09 *ms*). Selanjutnya selama berlangsungnya eksperimen ketika jumlah rincian *flow* pada *flow table* mendekati 50 *flow*, maka akan terjadi penghapusan rincian *flow* yang telah berdurasi lama dan tidak lagi terjadi pertambahan jumlah paket (rincian *flow* tidak lagi aktif) seperti ditunjukkan pada Gambar 4. Sepanjang waktu eksperimen, controller *c0* akan secara periodik memonitor kondisi *flow table* setiap 5 detik.

5. HASIL DAN EVALUASI

Eksperimen yang telah diulas pada bagian Implementasi memperlihatkan bahwa keberhasilan penerapan *dynamic flow removal* bersama aplikasi server *load balancing* dengan framework Ryu. Data berupa jumlah *http-request* beserta nilai RTT *http-reply* dan juga kondisi okupansi *flow table* dikumpulkan selama eksperimen berlangsung. Terdapat 100 *http-request* dan 100 nilai RTT *http-reply* dari 20 IP acak selama rentang waktu kurang lebih 2.5 menit. Evaluasi dilakukan terhadap data-data terkumpul tersebut.

Pada Gambar 5 merepresentasikan grafik perbandingan eksperimen sebelum dan sesudah penerapan *dynamic flow removal* berdasarkan data jumlah *rincian flow*. Grafik tersebut merepresentasikan garis lurus (“before”) untuk kondisi sebelum dan garis putus-putus (“after”) untuk kondisi setelah penerapan mekanisme *dynamic flow removal* serta garis titik-titik sebagai indikasi batas maksimum kapasitas *flow table* dari switch, yakni 50 *flow*. Terlihat pada Gambar 5, bahwa ketika belum diterapkan *dynamic flow removal* dan hanya berdasarkan *flow expiry* dari *idle_timeouts* 60 detik, maka jumlah rincian *flow* yang harus ditulis ke *flow table* dapat melebihi batasnya (gagal dituliskan) dan dapat mengakibatkan kegagalan komunikasi klien dan server. Sementara itu setelah diterapkan *dynamic flow removal*, semua rincian *flow* mengokupansi *flow table* tanpa melampaui batas maksimum kapasitasnya

dan menjaga komunikasi klien dan server tetap berlangsung. Dengan demikian, penerapan *dynamic flow removal* dapat menjaga okupansi *flow table* tidak penuh dan mencegah terjadinya *flow table overflow*.

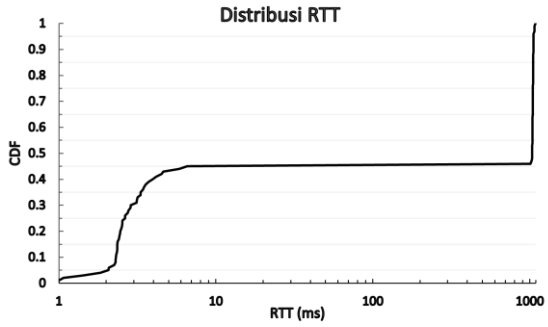


Gambar 5. Representasi keterisian *flow table*

Table 2. Perbandingan sebelum dan sesudah penerapan *dynamic flow removal*

Jumlah Flow	Sebelum	Sesudah
Sukses	42%	100%
Gagal	58%	0%

Tabel 2 menunjukkan perbandingan jumlah rincian *flow* sukses dan gagal menempati *flow table* sebelum dan sesudah penerapan *dynamic flow removal*. Terdapat 42% rincian *flow* sukses menempati *flow table* dan 58% rincian *flow* gagal menempati *flow table* saat hanya menerapkan mekanisme *flow expiry*, sementara itu ketika diterapkan *dynamic flow removal* 100% rincian *flow* dapat menempati *flow table* tanpa ada kegagalan. Jumlah rincian *flow* selama eksperimen selalu berada di bawah batas maksimum *flow table*.



Gambar 6. Distribusi nilai RTT

Pada Gambar 6 menunjukkan evaluasi dalam grafik CDF untuk melihat distribusi nilai RTT berdasarkan 100 *http-request* dan *http-reply* yang dikirim dari 20 klien IP acak. Terdapat 45% nilai RTT berada di bawah 10 *ms* yang menunjukkan terjadinya penggunaan kembali rincian *flow* tanpa intervensi controller dan 55% nilai RTT di atas 1000 *ms* yang menunjukkan adanya penambahan rincian *flow* oleh controller. Dengan demikian, penerapan *dynamic flow removal* tidak sepenuhnya memicu pembuatan *flow* baru dari controller SDN ke switch OpenFlow, sehingga penerusan paket dapat lebih cepat atau sangat baik untuk kinerja komunikasi klien – server pada aplikasi server *load balancing*.

Berdasarkan keseluruhan evaluasi didapatkan hasil bahwa ketika *flow table* mendekati kapasitas maksimum, mekanisme penghapusan rincian *flow* dari controller SDN ke switch OpenFlow akan terjadi untuk rincian *flow* yang tidak lagi memiliki aktivitas. Sedangkan ketika kondisi *flow table* normal (kurang dari 80% dari kapasitas maksimum), *flow expiry* berdasarkan *idle_timeout* berkerja secara natural dan rincian *flow* yang mencapai nilai *timeout* akan terhapus secara otomatis. Dengan demikian, hasil evaluasi menunjukkan bahwa dengan menerapkan dua mekanisme *flow removal*, maka jumlah rincian *flow* terbukti tidak pernah melewati batas kapasitas *flow table* dan semua rincian *flow* aktif terfasilitasi dengan baik dibuktikan dengan 100% *http-request* dari emulator klien mendapatkan 100% respon sukses *http-reply* dari server.

6. KESIMPULAN

Penelitian ini menyajikan eksplorasi teknis pengelolaan *flow table* yang secara efektif terbukti dapat mencegah terjadinya *flow table overflow* pada jaringan berbasiskan SDN dengan metode *dynamic flow removal*. Hasil evaluasi berdasarkan eksperimen emulasi 100 *http-request* jamak dari 20 IP acak pada aplikasi server *load-balancing* menunjukkan bahwa kesemuanya mendapatkan respon sukses *http-reply* dan 100% rincian *flow* menempati *flow table* tanpa melewati batas maksimumnya.

Selain itu, berdasarkan analisis distribusi keseluruhan nilai RTT *http-reply* menunjukkan bahwa metode *dynamic flow removal* tidak menyebabkan penurunan kinerja penerusan paket karena 45% nilai RTT bernilai di bawah 10 ms yang mengindikasikan penerusan paket data ditangani langsung oleh switch tanpa intervensi pembuatan rincian *flow* baru dari controller.

DAFTAR PUSTAKA

MCKEOWN dkk. 2008. OpenFlow: Enabling Innovation in Campus Networks. ACM SIGCOMM CCR, 31;38(2):69-74, 2008.

BOB LANTZ., dkk. 2010. A Network in a Laptop: Rapid Prototyping for Software-Defined Networks. Proc. of 9th ACM SIGCOMM Workshop on Hot Topics in Networks. 2010.

D. MOON., dkk. 2010. Bridging the Software/ Hardware Forwarding Divide. Technical Report, University of California at Berkeley, 2010. Dapat diakses di https://people.eecs.berkeley.edu/~istoica/classes/cs268/papers/paper-27.pdf

T.BENSON., dkk., 2010. Network Traffic Characteristics of Data Centers in the Wild. Proc. 10th ACM SIGCOMM IMC, Nov. 2010.

ADAM, ZAREK, 2012. OpenFlow Timeouts Demystified. Master Thesis of Univ. of Toronto, Ontario, Canada, 2012.

C. R. MEINERS dkk, 2012. Bit Weaving: A Non-Prefix Approach to Compressing Packet Classifiers in TCAMs. IEEE/ACM Trans. on Networking, 2012.

TOMONORI, F. 2013. Introduction to RYU SDN Framework." Open Networking Summit (2013): 1-14.

N. KANG, dkk., 2015. Efficient Traffic Splitting on Commodity Switches. ACM CoNEXT, 2015.

H. ZHU, dkk., 2015. Intelligent Timeout Master: Dynamic Timeout for SDN-based Data Centers. Proc. IFIP/IEEE International Symposium on Integrated Network Management (IM), May 2015.

L. ZHANG, dkk., 2015. TimeoutX: An Adaptive Flow Table Management Method in Software Defined Networks. Proc. IEEE GLOBECOM, Dec 2015.

AL-NAJJAR, A., dkk, 2016. Pushing SDN to the end-host, Network Load Balancing using OpenFlow. IEEE PerCom, 2016.

RYGIELSKI, PIOTR, dkk., 2017.Performance Analysis of SDN Switches with Hardware and Software Flow Tables. 10th EAI International Conference on Performance Evaluation Methodologies and Tools. 2017.

H. YANG dkk, 2018. Machine Learning Based Proactive Flow Entry Deletion for OpenFlow. IEEE ICC, 2018, pp. 1-6.

ZEHUA, G. dkk, 2018. Balancing Flow Table Occupancy and Link Utilization in Software-Defined Networks. Future Generation Computer Systems. 12(89):212-223, 2018.

CHENG, TAO, dkk, 2018. An In-switch Rule Caching and Replacement Algorithm in Software Defined Networks. IEEE ICC, 2018.

HE, CHENG-HUN dkk, 2018. A Zero Flow Entry Expiration Timeout P4 Switch. Proc. of the Symposium on SDN Research. 2018.

LI, RUI, dkk, 2019. A Tale of Two (Flow) Tables: Demystifying Rule Caching in OpenFlow Switches. Proc. of the 48th International Conference on Parallel Processing. 2019.

ONF, 2012. OpenFlow Specification v1.3. Open Networking Organization, dapat diakses di https://www.opennetworking.org/wp-content/uploads/.../openflow-spec-v1.3.0.pdf.

Mininet. Mininet: An Instant Virtual Network on your Laptop (or other PC)", dapat diakses di https://mininet.org/

OpenvSwitch. Open Virtual Switch, dapat diakses di https://www.openvswitch.org/

Ryu SDN. RYU: Component-Based Software Defined Networking Framework", dapat diakses di https://ryu-sdn.org/

Halaman ini sengaja dikosongkan