SURVEI MEKANISME CONGESTION KONTROL PADA TRANSMISSION CONTROL PROTOCOL DI SOFTWARE DEFINED NETWORK

Faishal H. Saputra[1)] dan Royyana M. Ijtihadie[2)]
[1, 2)] Departemen Informatika, Institut Teknologi Sepuluh Nopember
Jl. Teknik Kimia, Gedung Teknik Informatika, Kampus ITS Sukolilo Surabaya, 60111
e-mail: ical@its.ac.id[1)], roy@its.ac.id[2)]

ABSTRAK

Sebuah paradigma baru jaringan, Software Defined Network (SDN) dikembangkan membagi integrasi vertikal dalam perangkat jaringan, memisahkan logika kontrol dari infrastruktur jaringan sehingga memungkinkan untuk mengubah keadaan dan kondisi jaringan dari pengontrol yang dapat diprogram secara terpusat. Sebagian besar komunikasi one-to-many pada SDN diimplementasikan melalui beberapa unicast seperti TCP, yang tidak efisien. Hal itu menghasilkan banyak trafik direplikasi, yang dapat berakibat menurunkan kinerja aplikasi karena beberapa permasalahan seperti congestion, redundancy dan collision. Permasalahan congestion terjadi ketika sebuah network mempunyai beban yang banyak dan mengakibatkan performa menurun karena jumlah pengiriman melebihi kapasitas router yang ada. Salah satu solusi penanganan congesti dengan mereduksi ukuran TCP receive window. Tujuan utama dari pembuatan makalah ini adalah merangkum beberapa mekanisme kontrol kemacetan menggunakan TCP yang telah diakukan oleh para peneliti untuk menangani permasalahan kemacetan pada jaringan.

***Kata Kunci**: Kontrol kemacetan, receive window, SDN, TCP.*

ABSTRACT

Software Defined Network (SDN), a new paradigm network developed for dividing the vertical integration in the network system and separating the control logic of network infrastructure making it possible to change the state and condition of network centrally. Most one-to-many communications are implemented via multiple unicast, such as TCP, which is not efficient. It generates a lot of traffic replicated, which may result in lower performance applications due to several problems such as congestion, redundancy and collision. Congestion problem occurs when a network consists a lot of load and result in decreased performance. Congestion problems can be addressed and reduced by adjusting the size of the TCP receive window. The main objective of this survey paper is to summarize previous TCP congestion control mechanisms that has been propose by researchers to deal with congestion problems.

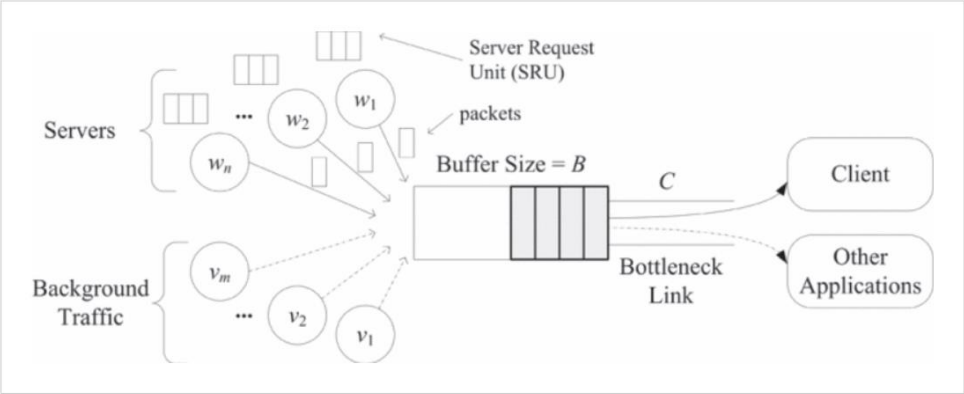
***Keyword**: Congestion control, receive window, SDN, TCP.*

I. PENDAHULUAN

SOFTWARE Defined Networking (SDN) adalah arsitektur jaringan yang memisahkan fungsi kontrol jaringan dan aliran data dari perangkat keras yang mendasari, dimana menggunakan openflow sebagai standar protokol pada SDN [1]. Teknik ini memungkinkan administrator jaringan dan pengembang aplikasi untuk secara dinamis mengelola dan mengubah parameter jaringan sesuai dengan permintaan [2]. karakteristik unik dari SDN telah membuat pilihan yang tepat untuk jaringan pusat data khususnya cocok untuk jaringan *cloud computing* [3].

Dalam rangka untuk merancang mekanisme transmisi yang efisien untuk pusat data, analisa karakteristik lalu lintas pusat data sangat penting [4], [5]. Aliran dalam jaringan pusat data cenderung memiliki arus data yang besar mengakibatkan *throughput* dan *delay* yang tinggi sehingga perlu adanya kontrol agar *delay* menjadi rendah [6]. Karakteristik SDN menggunakan kontroler terpusat rawan terjadinya masalah seperti *TCP incast. TCP incast* adalah pola komunikasi *one-to-many* biasanya ditemukan pada pusat data yang menerapkan penyimpanan terdistribusi dan komputasi framework seperti *Hadoop, MapReduce, HDFS, Cassandra*, dll. Pada Gambar 1 terdapat scenario TCP incast dimana ada banyak server (*w*) mengirimkan atau mentransmit data fragment (*Server Request Unit*) menuju *single* klien (C) sehingga terjadi komunikasi *many-to-one* yang rawan akan *bottleneck link* karena proses *buffer size* (B) yang penuh, dengan menggunakan mekanisme antrian *drop tail* apabila terdapat *server* baru yang akan mengirimkan data menuju klien namun antrian *buffer* habis maka paket tersebut akan di *drop* hingga *buffer size* mempunyai ruang cukup untuk menerima paket *request* baru sehingga perlu adanya metode yang dapat menangani masalah ini.

Solusi sebelumnya, fokus pada mengurangi waktu menunggu pemulihan *packet loss* [9], atau mengendalikan *buffer switch* untuk menghindari meluap dengan menggunakan ECN (*Explisit Congestion Notification*) dan dimodifikasi TCP pada sisi pengirim dan penerima [7]. Beberapa survei telah dilakukan berkaitan dengan TCP di



Gambar 1. Scenario TCP incast, dimana n server mentransmit data (SRU) menuju single client secara bersamaan.

SDN seperti Yife lu [8] mengusulkan SDN *Explicit-Deadline-aware* TCP (SED) untuk mengatasi masalah *packet-loss timeout*, Navin Kukreja dkk [9] mengusulkan peningkatan pemanfaatan sumber daya *throughput* dan ketahanan jaringan terhadap kemungkinan kegagalan *link* menggunakan *multipath* TCP (MPTCP), Zhi Wang dan Chuan Wu [10] mengusulkan responsif *multipath* TCP yang dapat memperkerjakan pengontrol terpusat untuk melakukan perhitungan rute *intelligent subflow* dan memonitor setiap *server* secara aktif menyesuaikan jumlah *subflow*, namun secara spesifik belum ada survei yang fokus membahas tentang mekanisme kontrol kemacetan pada TCP di lingkungan SDN.

Tujuan survei ini antara lain untuk membandingkan mekanisme yang digunakan dalam menangani masalah *congestion* pada *Transmission Control Protocol* (TCP) di lingkungan SDN. TCP menjadi fokus pada survei karena protokol ini umumnya terjadi ketika penerima meminta data dari beberapa pengirim secara simultan. Melalui survei ini kami mendiskusikan dan membandingkan beberapa metode yang merupakan pengembangan lebih lanjut dari skema kontrol kemacetan.

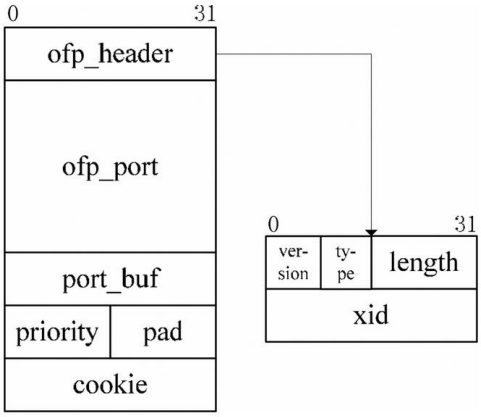
Pembahasan selanjutnya pada makalah ini dibagi menjadi beberapa bagian sebagai berikut. Bagian 2 membahas mengenai mekanisme mengontrol kemacetan pada jaringan SDN. Metode-metode baru yang telah dikembangkan oleh para peneliti untuk menangani congestion problem disajikan dalam Bagian 3. Pada Bagian 4 kami melakukan pembandingan dan pembahasan metode-metode yang telah diulas pada Bagian 3. Bagian 5 merupakan sesi penutup yang berisi kesimpulan dari survei ini dan usulan metode penanganan *congestion* yang dapat dikembangkan lebih lanjut.

Kami berharap survei ini dapat memberi penyegaran informasi mengenai berbagai mekanisme yang dapat dilakukan dalam menangani masalah kemacetan dalam pengembangan *software-defined network (SDN)*. Survei ini juga diharapkan dapat memberi kontribusi dalam pengembangan TCP lainnya dalam SDN yang lebih efektif, efisien dan komprehensif.

II. CONGESTION PROBLEM

Congestion problem terjadi ketika sebuah jaringan mempunyai beban yang banyak dan mengakibatkan performansi menurun atau lambat dengan kata lain jumlah pengiriman data melebihi kapasitas *hardware* yang ada [11]. kontrol kemacetan berkaitan dengan mengalokasikan sumber daya dalam jaringan sehingga jaringan dapat beroperasi pada tingkat kinerja yang sesuai ketika permintaan melebihi kapasitas sumber jaringan. Sumber daya ini termasuk *bandwidth*, *buffer* (memori) dan kapasitas pengolahan pada node *intermediate* [12]. Tanpa mekanisme kontrol *congesti* yang tepat, *throughput* jaringan dapat berkurang jauh karena beban trafik yang besar. Menurut Nurlinaamik [13] *congestion* pada lalu lintas jaringan disebabkan oleh:

a. *Terlalu banyak host dalam sebuah broadcast domain*, *host* adalah alat yang terhubung ke jaringan dan dapat menerima dan mengirimkan informasi dari alat ke alat lainnya dalam jaringan tersebut. *Broadcast* domain adalah kumpulan dari alat-alat pada sebuah segmen jaringan yang menerima paket *broadcast* yang dikirim oleh alat lain dalam segmen jaringan tersebut.
b. *Broadcast Storm*, terjadi karena semua alat mengirimkan paket *broadcast* ke seluruh alat lain melalui jaringan. Semakin banyak *host* maka semakin besar *broadcast storm*.
c. *Multicasting*, jika dalam satu jaringan terdapat banyak komputer di mana setiap komputer mengakses beberapa halaman situs web bervolume tinggi dalam satu waktu yang sama, maka besar kemungkinan akan terjadi kemacetan bandwith. Jalur yang kecil akan membuat lalu lintas jaringan padat jika dilewati oleh banyak data dalam satu periode.
d. *Data Collision*, yaitu suatu kondisi network dimana sebuah alat mengirimkan paket data ke sebuah segmen jaringan yang kemudian memaksa semua alat lain yang ada di segmen jaringan tersebut untuk



Gambar 2. OFP_header [1]

memperhatikan paketnya. Pada saat yang bersamaan alat yang berbeda mencoba untuk mengirimkan paket yang lain, yang mengakibatkan tabrakan *(collision),* paket yang dikirim menjadi rusak akibatnya semua alat harus melakukan pengiriman ulang paket.

e. *Bandwith yang kecil*, media jaringan dengan bandwith kecil tidak seimbang dengan banyaknya lalu lintas data yang terjadi sehingga dapat mengakibatkan *overload*.

III. MEKANISME UNTUK MENANGANI *CONGESTION*

Beberapa mekanisme *congestion control* yang dibahas dalam makalah ini sebagai berikut:

A. SDN-based TCP Congestion Control (SDTCP)

Mekanisme yang diusulkan oleh Yifei Lu dkk [1] berawal dari pola komunikasi *one-to-many* pada jaringan pusat data sehingga rawan terjadinya *congestion*. Yifei Lu mengusulkan mekanisme kontroler yang berperan untuk memilih *long-lived flow (elephant-flow)* dan mengurangi TCP *receive window* ACK apabila mendapatkan informasi pesan *congestion* dari *openflow switch* untuk mengurangi transmisi dari pengirim. *Long-lived flow* adalah sebuah arus yang membawa trafik besar sehingga memicu terjadinya *congestion* seperti *file transfer*, *video streaming* [14].

Skema SDTCP memodifikasi *Openflow* 1.3 khususnya pada *ofp_header* untuk membuat pesan OFPT_BUF_ALARM dan OFPT_BUF_NORMAL yang digunakan untuk memicu aksi kontrol pada kontroler yang dapat dilihat pada Gambar 2. SDTCP menggunakan pesan OFPT_BUF_ALARM untuk memicu pemberitahuan kemacetan antrian di *Openflow*-switch (OF-switch) dalam protokol *Openflow*, setelah menerima pesan pemberitahuan yang berisi antrian *long-lived flow* dengan ukuran melebihi *threshold* kemudian *OF-switch* mengidentifikasi antrian dan informasi *port* kemudian diteruskan ke kontroler melalui protokol *Openflow.*

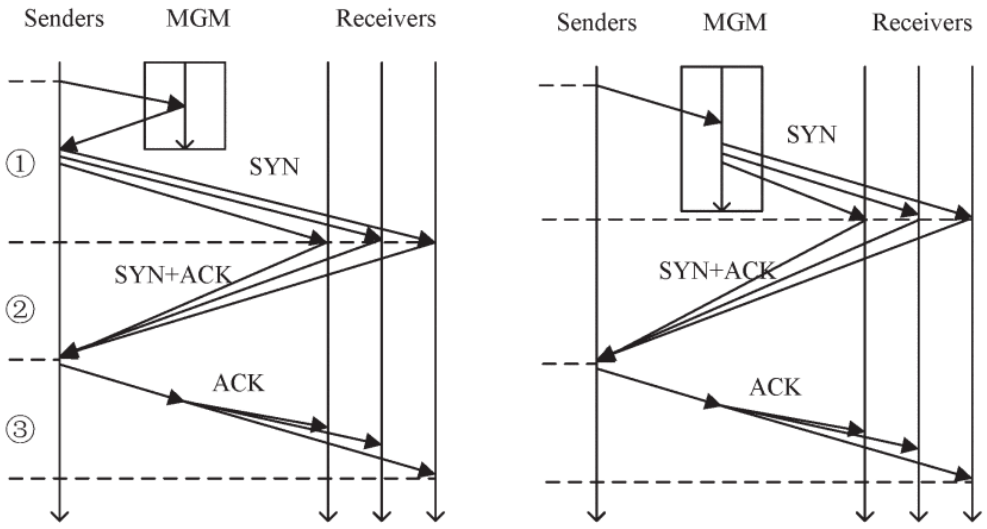
Ketika menerima pesan pemberitahuan kemacetan, kontroler memilih arus *long-live* dan memodifikasi perintah untuk *OF-switch* yang akan bereaksi dengan mengurangi *receive window* paket ACK. Selanjutnya, pengirim akan mengurangi ukuran *receive window* yang akan menghasilkan pengurangan tingkat transmisi. Jika buffer lebih kecil dari *threshold*, maka sebuah mekanisme pemberitahuan pemulihan pesan OFPT_BUF _NORMAL diteruskan ke kontroler yang bereaksi untuk mengecek informasi dan memeriksa database *congestion flow*. Dengan perintah ini, mendorong perintah pemulihan paket ACK tidak akan diubah dan tingkat transmisi arus *long-live flow* akan kembali.

Keunggulan metode ini adalah penggunaan informasi seperti pesan OFPT_BUF_ALARM dan OFPT_BUF_NORMAL yang memicu aksi kontrol apabila mendeteksi *congestion* pada *long-lived flow* dan menurut uji evaluasi hanya perlu membutuhkan $\frac{1}{2}$RTT (*Round-Trip Time*) untuk memulihkan status ALARM menjadi NORMAL.

Kelemahan metode ini hanya fokus pada *long-lived flow,* sementara *short-lived flow* dengan antrian yang sangat besar juga dapat memicu terjadinya *congestion.* Kemudian, pada bagian evaluasi peneliti membandingkan *buffer* size selama 14 detik dengan metode SDTCP, DCTCP dan TCP, Hasilnya presentase pada aspek kontrol *buffer* SDTCP masih dibawah DCTCP.

B. Congestion-Aware and Robust multicast TCP (MCTCP)

Mekanisme yang diusulkan Tingwei Zhu dkk [15] berawal dari masalah tidak efisien dan ketahanan yang rendah



(a) Skema out-band (b) Skema in-band

Gambar 3. MCTCP *session establishment* [15]

pada komunikasi *multicast* di lingkungan SDN. *Multicast* menawarkan cara yang elegan dalam membangun kelompok komunikasi antara *user* dengan menggunakan konsep kelompok *multicast*. Klien dapat bergabung dan meninggalkan kelompok untuk mengirim atau menerima data dari anggota group lainnya. Selain itu mekanisme *multicast* memastikan bahwa strategi yang efisien untuk menyampaikan paket data ke semua anggota secara bersamaan sehingga dapat mengurangi terjadinya *congestion* [16]. Tingwei Zhu mengajukan mekanisme MCTCP untuk mengelola kelompok *multicast* secara terpusat dan secara reaktif mengaktifkan arus *multicast* yang mempunyai link aktif dan utilisasi rendah dengan memperluas TCP sebagai protokol *host-side* dan mengelola kelompok *multicast* di SDN-kontroler.

MCTCP terdiri dari dua modul, yaitu HSP (*Host-Side Protocol*) dan MGM (*Multicast Group Manager*). HSP adalah protokol untuk pengirim, di mana pengirim (*sender*) mendefinisikan transmisi dan penerima (*receiver*) tidak perlu mengetahui alamat *multicast*. MGM terletak di SDN kontroler mengelola *session multicast* dan *Multicast Spanning Trees* (MSTS). *Spanning Tree Protocol* (STP) adalah protokol layer 2 yang diaplikasikan pada *bridge* dan *switch* [17]. Spesifikasi STP adalah IEEE 802.1D dengan tujuan utama yaitu memastikan tidak terjadi *looping* pada lalu lintas jaringan *redundant* seperti komunikasi *multicast*. Dengan menjaga topologi jaringan global dan memantau status *link* secara *real-time*, MGM dapat menyesuaikan MSTS pada masalah seperti kemacetan link atau kegagalan.

Ada dua skema alternatif yang diusulkan pada *host-side protocol* yaitu *out-band* dan *in-band* yang dapat dilihat pada Gambar 3. Untuk skema *out-band* permintaan dikirim ke MGM sebelum membangun koneksi *three-way handshake*. Setelah menghitung MST, MGM memberitahu pengirim untuk memulai koneksi *three-way handshake*. Sementara itu pada skema *in-band*, paket SYN digunakan kembali untuk meminta perhitungan MST kemudian MGM mengirimkan SYN ke semua penerima. Sehingga skema *in-band* tidak memiliki *extra time overhead* yang membuat tekanan *load* berada pada SDN kontroler. Skema *in-band* cocok untuk topologi jaringan yang memiliki anggota kecil dan memiliki *delay* yang sensitif.

Keunggulan metode ini adalah memperluas penggunaan TCP sebagai *host-side-protokol* yang bertugas untuk mengatur grup *multicast* pada kontroler SDN dan skema pembangunan jaringan *three-way handshake* seperti *out-band* yang cocok untuk topologi yang besar. Selain itu penggunaan algoritma MPH dan Djikstra dalam penghitungan routing sangat baik karena mencari *path* dengan *cost* minimal. Kelemahan MCTCP adalah diperlukan waktu untuk membangun komunikasi jaringan *multicast* dan *route discovery* HSP yang tidak dijelaskan pada penelitian ini.

C. Scalable Congestion Control Protocol (SCCP)

Jaehyung Hwang dkk [18] mengusulkan SCCP dengan tujuan untuk menghindari *switch buffer overflow* dan mengurangi *delay* antrian bahkan dalam situasi di mana sejumlah besar arus *bursty*. Untuk mencapai tujuan ini, Jaehyung memperpanjang spesifikasi *Openflow* [19] pada *switch* SDN agar secara akurat menilai informasi jumlah arus TCP yang melintasi setiap *port switch*. Informasi ini digunakan untuk menghitung *fair-share* masing-masing *flow*, sehingga utilisasi *link* totalnya tidak melebihi *bandwidth delay product* (BDP). Informasi ini dikirim ke masing-masing pengirim TCP dengan memperbarui *advertisement window* pada TCP *header*.

Switch pada pusat data umumnya terdiri dari beberapa *input* dan *output port*, di mana masing-masing *port* dibagi oleh beberapa arus. kemacetan jaringan di pusat data yang umumnya sering terjadi pada *output-port* [18]. Sehingga

skema kontrol kemacetan dilakukan pada setiap *output-port switch* dengan detail sebagai berikut:

1) Untuk *outgoing* paket : (a) Melacak arus TCP (N_i)
2) Untuk *incoming* paket: (a) Hitung *fair-share* ($fair\_share_i$). (b) merubah nilai *advertisement window* pada TCP *header (awnd)* dengan nilai $fair\_share_i$, jika TCP *header (awnd)* > $fair\_share_i$.

Pada proses (1)-(a) N_i adalah jumlah arus yang melintasi *port switch.* Untuk melacak arus ini, *switch* menggunakan inisiasi arus dan terminasi paket seperti TCP SYN / FIN. Pada proses (2) - (a) menghitung *fair-share* dengan membagi BDP dari port oleh *Ni*. Pada *Bandwidth Delay Product, bandwidth* adalah kapasitas *link port* dan *delay* adalah *common RTT* dari semua arus. Kami membahas formula untuk mendefinisikan *common RTT* sebagai berikut.

$$fair\_share_i = \frac{link\_capacity_i \text{ x } common\_RTT}{N_i} \quad (1)$$

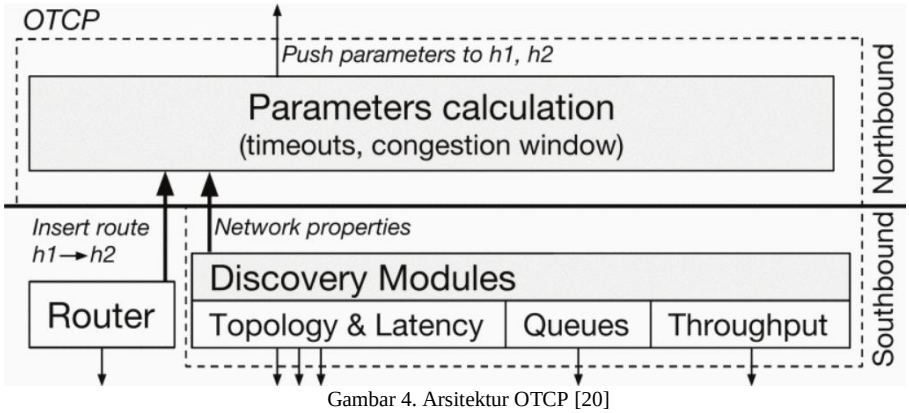
Untuk memberikan informasi *fair-share* ke *end-host server, advertisement window* pada TCP *header* digantikan oleh *fair-share* jika nilai TCP *header* yang ada lebih besar daripada nilai *fair-share* yang baru. Dengan demikian, setiap server mengetahui *port fair-share* yang *bottleneck* pada arus *end-to-end*. Untuk mengukur *fair-share*, pendekatan dengan mengukur masing-masing arus RTT dan menjaga informasi per-arus dalam *switch*, tapi ini menimbulkan *overhead* yang signifikan. Untuk menghindari hal ini, peneliti mendefinisikan *common RTT* untuk setiap arus. Dengan asumsi bahwa RTT pada umumnya kurang dari 250µs di jaringan data center [7] dengan mengambil nilai yang cukup besar untuk *common RTT*, seperti 300µs atau 400µs sehingga RTT selalu lebih rendah dari *common RTT*.

Keuntungan metode SCCP adalah mudah diaplikasikan di lingkungan SDN karena SCCP tidak memerlukan modifikasi perangkat keras seperti switch pada jaringan data SDN karena SCCP mematuhi prinsip-prinsip spesifikasi *openflow*. Kedua, SCCP tidak memerlukan perubahan ke *end-host* dan sisi aplikasi. Kelemahan metode ini hanya bisa diaplikasikan pada versi *openflow* 1.5 keatas, karena harus mendukung TCP *flags* untuk mendeteksi paket TCP SYN/FIN *flags* dan memodifikasi *advertisement window* pada TCP *header*.

D. Omniscient TCP Congestion Control (OTCP)

Simon Jouet dkk [20] mengusulkan *omniscient* TCP (OTCP) untuk meningkatkan kinerja jaringan secara keseluruhan dan pemanfaatan infrastruktur modern seperti SDN yang dikelola menggunakan protokol TCP. Melalui *Openflow*, OTCP dapat digunakan *side-by-side* dengan kontroler untuk mengumpulkan metrik jaringan seperti topologi, *latency, throughput, queues* dan ukuran *buffer switch*. Setelah metrik dikumpulkan, OTCP menghitung TCP *retransmission time*, *RTOmin* dan *RTOinit*, serta ukuran maksimum *congestion window* yang cocok dengan *Bandwidth Delay Product* (BDP) antara *end-host*. Kemudian, OTCP mengekspos JSON / REST *northbound interface*, yang mana merupakan koneksi *daemon end-host* untuk menerima parameter *congestion* kontrol ketika koneksi dibangun dalam infrastruktur jaringan. Arsitektur OTCP dapat dilihat pada Gambar 4.

Keunggulan metode ini bahwa OTCP dapat digunakan bersama-sama dengan DCTCP untuk mencegah antrian *build-up* arus *burst flow* dan dapat mengurangi *flow completion time* (FCT) dalam jaringan yang sangat padat. Hal ini menunjukkan bahwa *fine-tuned* parameter kontrol kemacetan secara signifikan dapat meningkatkan kinerja aplikasi yang mengalami TCP *incast* di lingkungan DC. Kelemahan metode ini adalah dengan menyesuaikan nilai *retransmission timeout* dengan nilai BDP maka akan berdampak apabila nilai *retransmission timeout* sangat tinggi pada pembangunan komunikasi *three-way handshake* jika terjadi kegagalan komunikasi maka waktu yang



Gambar 4. Arsitektur OTCP [20]

dibutuhkan lebih lama untuk meretransmisi sehingga rawan terjadi *overhead* atau *redundant* komunikasi.

E. Flow-Aware Congestion Control (FATCP)

Chunghan Lee [21] mengusulkan metode berbasis *switch* yang mengidentifikasi arus dan memberikan kontrol kemacetan menggunakan *Explicit Congestion Notification* (ECN). ECN adalah ekstensi TCP/IP yang dapat memberitahukan apabila terjadi *congestion* kepada *end-host* tanpa *droping* paket [22]. Hal ini diperlukan untuk membangun DCNs jangka panjang yang menyediakan layanan penanganan arus yang berbeda seperti QoS, *path*, dan kontrol kemacetan di DCNs. Dalam metode yang diusulkan terdapat dua komponen konseptual: identifikasi arus target dan modifikasi kontrol kemacetan. Identifikasi arus target menggunakan nilai *Differentiated Services Code Point* (DSCP) yang terletak pada *header* TCP/IP. DSCP digunakan untuk mengidentifikasi arus target dan banyak digunakan di sebagian besar switch komersial. Untuk identifikasi arus, nilai DSCP ditugaskan untuk mendeteksi *elephant flow* dan arus ditentukan ketika port yang ditunjuk dan protokol (TCP) yang digunakan. Jika *port* dan protokol cocok pada pengirim maka proses modifikasi nilai DSCP dilakukan oleh ovs-ofct1 di OVS. Komponen yang kedua yaitu memodifikasi mekanisme *congestion control* menggunakan ECN sebagai berikut:
1) Network switch, pertama menentukan *threshold* (K) antrian pada switch. Jika ukuran rata-rata antrian dibawah K maka data paket ditandai sebagai *ECN-capable transport* (ECT). Namun, data paket akan ditandai *congestion experienced* (CE) jika ditemuka ukuran rata-rata antrian lebih dari K.
2)End Nodes, paket dengan tanda CE dikirim dari *sender* menuju *receiver* kemudian ECN echo (ECE) *flag* memberitahukan kemacetan dalam serangkaian ACK yang diatur dan dikirimkan ke pengirim. Kemudian pengirim akan mengurangi *size congestion window* setengah dan menandai *windows reduced* (CWR) *flag*.

Pada makalah ini peneliti telah melakukan evaluasi dengan beberapa jumlah *elephant flow* untuk menentukan ECN threshold dapat dilihat pada Tabel 1, begitu juga dengan formula yang digunakan dimana *N* adalah jumlah *elephant flows (long-lived flows)*, *R* adalah *Round-Trip Time* (RTT) pada arus TCP dan *C* adalah *bottleneck link capacity*.

$$ECN\ threshold\ (K) \geq \frac{R.C}{\sqrt{N}} \quad (2)$$

Keuntungan metode ini adalah walaupun fokus pada TCP di DCNs namun tidak membutuhkan mekanisme TCP baru dan memodifikasi pada *header* TCP sehingga memungkinkan untuk menjaga kompatibilitas *data center* di masa depan. Kelemahan meode ini adalah hanya dapat digunakan menggunakan perangkat *switch* ECN yang tidak terlalu banyak di dunia industri sekarang dan pada makalah referensi tidak dijelaskan maupun tidak diuji apakah *switch* EDN mendukung dengan arsitektur SDN yang mempunyai beberapa kontroler dengan berbagai bahasa pemrograman.

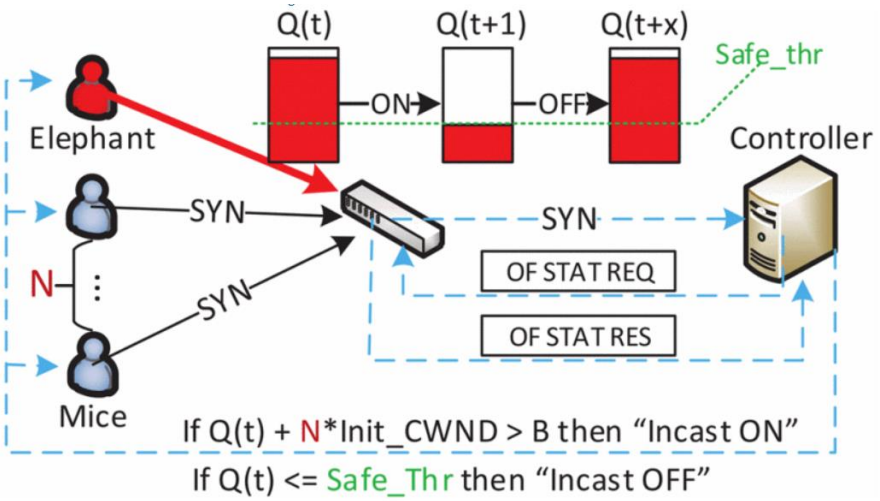
F. SDN Incast Congestion Control (SICC)

Abdelmoniem dkk [23] mengusulkan sebuah *framework* SDN yang dapat memberikan data statistik mengenai jumlah *flow* dan *queue* (Q) untuk setiap *switch port*. Pada saat kontroler SDN melakukan *forecasting* kemungkinan terjadinya *incast*, kemudian kontroler dapat mengirimkan pesan khusus kepada *end-host* apabila terjadi *incast*. *End-host* memulai menulis ulang *receiver window* (Rwnd) pada ACK yang baru masuk ke 1 MSS (*Maximum Segment Size*). *Sending window* TCP yang terkecil di antara *congestion window* seperti (Cwnd dan Rwnd), pengirim (*elephant connection*) diubah menjadi 1 MSS per RTT^2. Pengubahan semua arus menjadi 1 segmen per RTT memungkinkan *congestion switch* untuk mengurai antrian di bawah ambang kemacetan atau *threshold*. Penulisan ulang Rwnd akan berhenti setelah beberapa RTTs membiarkan *elephant connection* untuk memulihkan tingkat pengiriman sebelumnya. Sehingga kontroler dapat memperkirakan dengan hanya menghitung jumlah SYNs dikurangi jumlah FIN dalam pengecekan secara *interval*.

Pada Gambar 5 menunjukkan interaksi protokol di antara berbagai modul pada kontroler, *switch* dan *end-host* sebagai berikut: 1) Modul pemantauan kontroler bertanggung jawab untuk melacak, menghitung dan menggabungkan informasi (*windows scaling option*) dari SYN/FIN. 2) Modul peringatan kontroler bertanggung jawab untuk memprediksi kejadian *incast* berdasarkan tingkat kedatangan SYN / FIN dan untuk mengirimkan pesan khusus on/off yang dinyalakan dan diarahkan ke alamat VM pengirim yang terlibat. 3) Setelah menerima pesan

TABEL I
JUMLAH ELEPHANT FLOW DAN ECN THRESHOLD [21]

Jumlah Elephant Flow	ECN Threshold (K)
1	250
3	145
5	112
7	95
9	83



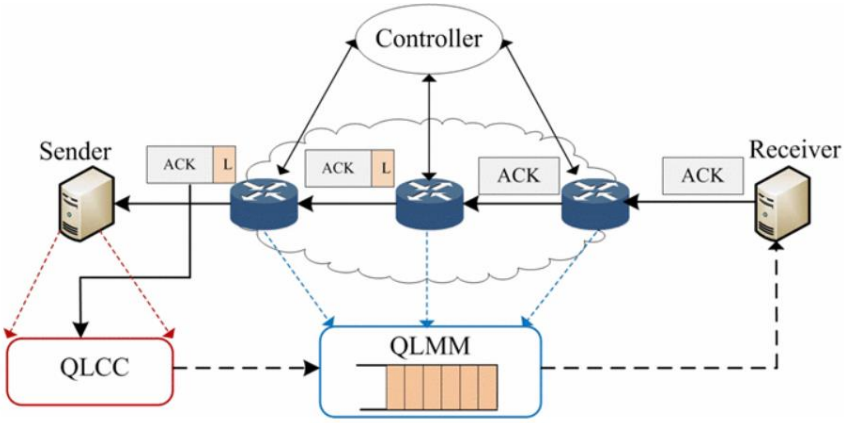
Gambar 5. Interaksi komponen SSIC *framework* [23]

incast ON untuk VM tertentu, modul SICC mulai menghadang dan memodifikasi ACK yang masuk untuk VM tersebut sampai pesan OFF akan diterima (Safe_Thr). 4) *Switch* SDN hanya perlu diprogram dengan aturan *Copy-to-Controller* untuk paket SYN / FIN, kontroler akan menetapkan sebuah aturan pada *switch* untuk meneruskan salinan paket SYN / FIN melalui antarmuka protokol *openflow.*

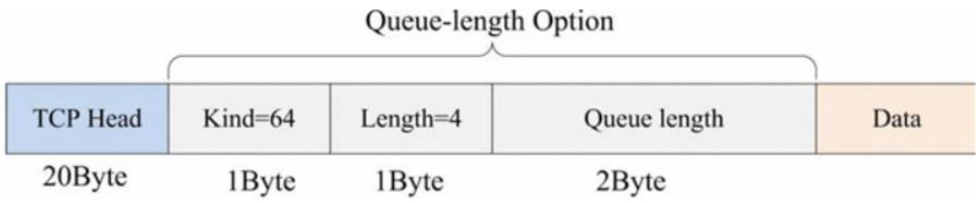
Keuntungan metode ini dapat membantu mengurangi waktu penyelesaian aliran TCP *incast* pada jangka waktu pendek. Kerangka kerja SICC mengandalkan kontroler SDN untuk memantau kedatangan paket SYN / FIN secara teratur membaca penghitungan antrian *switch openflow* untuk menyimpulkan awal periode *incast* sebelum arus mulai mengirim data ke jaringan. SICC ditunjukkan melalui simulasi dan percobaan testbed untuk memperbaiki waktu penyelesaian aliran untuk lalu lintas *incast* tanpa mengganggu *throughput elephant flow*. Kontribusi utama SICC adalah kemampuannya untuk mencapai perbaikan tanpa memodifikasi algoritma TCP atau perangkat keras jaringan untuk memungkinkan potensi penyebaran yang cepat dan benar dalam jaringan data center.

G. Explicit Queue-Length Feedback TCP Congestion Control (EQF)

Yifei Lu dkk [24] mengusulkan EQF, sebuah *explicit queue-length feedback* untuk pengendalian kemacetan TCP di DCN. Tidak seperti pengendalian kemacetan implisit yang digunakan oleh metode sebelumnya (misalnya DCTCP, DC-Vegas, DX, dll.), EQF memanfaatkan sinyal kemacetan eksplisit, yang merupakan antrian panjang pada *congestion switch* yang dapat menunjukkan tingkat kemacetan dan penyesuaian ukuran *congestion windows*



Gambar 6. Arsitektur EQF [24]



Gambar 7. Format Paket ACK-QL [24]

yang berupa *flow* menurut *congestion signal*. Penelitian ini mengukur panjang antrian dalam jaringan berdasarkan teknologi *epochal* dari *Software-Defined Networking* (SDN). Mereka melakukan evaluasi menggunakan simulasi ns-3 untuk membandingkan EQF dengan solusi sebelumnya seperti TCP NewReno, TCP Vegas, dan DCTCP. Hasil percobaan menunjukkan bahwa EQF efektif dalam meningkatkan kinerja DCN, seperti mengurangi waktu transmisi, meningkatkan jumlah *flow* secara simultan, dan mengurangi TCP *incast* di DCN (Data Center Network).

Mekanisme EQF terdiri dari *queue-length monitor module* (QLMM) pada switch untuk mendapatkan panjang antrian yang akurat dan *queue-length congestion control* (QLCC) pada pengirim untuk menyesuaikan *congestion window*. Arsitektur keseluruhan EQF ditunjukkan pada Gambar 6.

Yifei Lu dkk merancang QLMM di *switch* untuk menilai panjang antrian menentukan apakah jaringan mengalami *congestion*. Setelah kemacetan jaringan ditemukan, artinya, panjang antrian lebih besar dari ambang batas empiris (K), QLMM memanipulasi paket TCP ACK arus padat sehingga dapat menambahkan informasi panjang antrian ke paket ACK. Ketika pengirim menerima paket ACK ini, panjang antrian dapat diperoleh dengan menguraikan paket ACK. Setelah itu, pengirim menyesuaikan laju transmisi sesuai algoritma antrian *queue-length congestion control* (QLCC) sehingga dapat meringankan kemacetan jaringan.

QLMM akan memodifikasi paket TCP ACK dari arus padat menjadi yang sesuai *threshold* sehingga dapat menambahkan informasi panjang antrian ke paket ACK. Paket TCP ACK yang dimodifikasi disebut sebagai ACK-QL yang harus memenuhi *port ingress* dari paket ACK. QLMM dapat dicapai dengan menerapkan *openflow* karena menyediakan kemampuan untuk mengimplementasikan *aware forwarding* untuk mengatur arus maju dan mundur sepanjang jalur yang sama dan menyediakan *queue occupancy* untuk setiap *switch port*. Kami memanfaatkan protokol *openflow* yang disesuaikan untuk menambahkan informasi panjang antrian ke paket ACK-QL secara otomatis.

Paket ACK-QL yang baru menggunakan format TCP *option field* untuk memasukkan panjang antrian dalam paket TCP ACK. *Queue length option* ditambahkan ke pesan SYN dan SYN-ACK untuk menegosiasikan kemampuan *queue length announcement*. Header paket ACK-QL ditunjukkan pada Gambar 7. *Queue-length option* terdiri dari beberapa bagian:

- *Kind*: Bagian ini terdiri dari satu byte dan menentukan jenis *option field*. Karena beberapa nomor sudah digunakan, maka nomor 64 digunakan untuk ACK-QL.
- *Length*: *Option length* adalah jumlah oktet dalam *option* yang menghitung *kind*, *length* dan *queue length*. Ukurannya satu byte.
- *Queue length*: Panjang antrian yang macet dalam ukuran 1 KB.

Keuntungan metode ini menawarkan sebuah protokol baru untuk menyediakan layanan transmisi dengan delay rendah di DCN. Bila panjang antrian dari *switch* yang padat lebih besar dari ambang batas, paket ACK-QL yang berisi informasi panjang antrian akan dikirimkan dari *switch* ke pengirim sehingga dapat mengurangi laju pengiriman arus. Mekanisme EQF memanfaatkan panjang antrian yang macet untuk mendapatkan tingkat kemacetan yang akurat. Hasil implementasi demonstrasi percobaan menunjukkan bahwa metode EQF dapat mereduksi masalah seperti TCP *incast* dengan sangat baik.

IV. PEMBAHASAN

Pada makalah ini kami menyajikan pembahasan berupa perbandingan mekanisme *congestion control* dengan karakteristik seperti yang ditampilkan pada Tabel 2. Kolom pertama berisikan nama dari metode yang diajukan untuk mengontrol kemacetan yang dibahas dalam makalah ini. Kolom kedua berisikan lingkungan uji coba yang digunakan oleh masing-masing metode. Kolom ketiga berisi kontroler yang digunakan pada ujicoba namun tidak semua referensi makalah menggunakan kontroler pada uji *testbed*, beberapa makalah hanya fokus *congestion control* menggunakan *forwarding plane*. Kolom keempat berisi topologi yang digunakan saat ujicoba. Kolom kelima berisi versi *openflow* dan kolom keenam berisikan mekanisme yang diajukan membutuhkan keputusan kontroler.

TABEL II
PERBANDINGAN MEKANISME CONGESTION KONTROL PADA SDN

Metode	Lingkungan Uji coba	Kontroler	Topologi	Openflow	Memerlukan keputusan
SDTCP [1]	Mininet	Floodlight	Fat-tree	1.3	Kontroler, switch
MCTCP [15]	HDFS	Ryu	Fat-tree	-	Kontroler
SCCP [18]	NS-3	-	Fat-tree	1.5 [18]	Tidak
OTCP [20]	Mininet HiFi	-	Fat-tree	1.3	Kontroler
FATCP [21]	ECN switch	-	Clos (spine and leaf)	-	Tidak
SSIC	NS-2	-	Fat-tree, single rooted	1.3	Kontroler, switch
EQF	NS-3	-	Fat-tree	-	Switch, end-host

Dari survei yang kami lakukan, kami menemukan beragam metode yang sudah diajukan untuk menangani *congestion problem*. Mayoritas metode-metode tersebut merujuk pada memodifikasi komunikasi TCP dengan mereduksi ACK *receive window* apabila mendeteksi *congestion* pada suatu jaringan [1][15][18][23]. Metode yang diajukan juga berkembang tidak hanya fokus pada mereduksi *flow* yang mengalami *congestion*, namun juga dengan bantuan kontroler pada SDN maka Tingwei Zhu [15] memanfaatkan kontroler untuk dapat berkomunikasi secara *multicast* sehingga data yang dikirim dapat lebih efisien, namun di sisi lain akan membebani beban dari kontroler.

Dari metode yang kami bahas, proses pengujian dilakukan beberapa simulator dan perangkat keras dengan kasus uji yang berbeda. Sehingga tidak dapat dibandingkan hasil uji masing-masing metode. Pengujian dengan simulator dapat dipahami karena keterbatasan perangkat uji yang dapat digunakan untuk melakukan pengujian di dunia nyata. Pengujian pada hardware seperti *Hadoop Distribution File System (HDFS)* dan *switch* ECN menandakan metode yang diusulkan [15][21] sudah dapat diaplikasikan di dunia nyata namun perlu dikaji lebih mendalam mengingat kontroler SDN memiliki beberapa kontroler dengan karakteristik dan bahasa pemrograman yang berbeda-beda.

V. KESIMPULAN

Beberapa mekanisme *congestion control* yang dapat digunakan untuk menangani *congestion problem* pada jaringan SDN telah disajikan pada makalah ini. Penanganan *congestion* secara tepat diperlukan untuk menjamin performa dari *software-defined networking* dengan karakteristik kontrol terpusat. Mekanisme tersebut telah dibandingkan dengan kriteria yang sudah ditunjukan pada Tabel 2. Pengujian terhadap metode yang dibahas pada makalah ini terdiri dari 3 makalah menggunakan simulasi dan 2 makalah menggunakan perangkat keras dengan hasil uji coba yang baik untuk menangani masalah *congestion*. Melalui survei ini penulis mengharapkan dapat memberi kontribusi dalam pengembangan *congestion control* lainnya dalam lingkungan SDN yang lebih efektif, efeisien dan komprehensif.

Penanganan *congestion problem* tidak harus terpaku kepada satu skema saja. Pada beberapa penelitian sebelumnya, telah diajukan penggunaan metode dengan menggunakan lebih dari satu skema, seperti penggunaan *explicit congestion notification* yang sudah dibahas pada paper ini. Untuk penelitian lebih lanjut, dapat menggunakan metode dengan menerapkan skema baru yaitu menggabungkan keunggulan-keunggulan dari skema yang sudah ada. Sebagai contoh penggunaan skema OTCP yang digabungkan dengan DCTCP.

DAFTAR PUSTAKA

[1] Yifei Lu, Shuhong Zhu, "SDN-based TCP congestion control in data center networks", *Computing and Communications Conference (IPCCC), IEEE 34th International Performance,* 2015. hal. 1-7.

[2] Keith Kirpatrick, "Software-Defined Networking, ACM Communication", vol. 56, no. 9, pp. 16-19, 2013.

[3] H. Wu, Z. Feng, C. Guo, Y. Zhang, "ICTCP: Incast Congestion Control for TCP in Data Center Networks", *ACM CoNEXT,* 2010.

[4] T. Benson, A. Akella, D. A. Maltz, "Network traffic characteristic of data centers in the wild".

[5] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, R. Chaiken, "The Nature of data center traffic: measurement & analysis", IMC, 2009.

[6] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, S. Sengupta, "VL2: A Scalable and flexible data center network", SIGCOMM, 2009.

[7] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Paydhe, P. Patel, B. Prabhakar, S. Sengupta, M. Sridharan, "Data center TCP (DCTCP)", SIGCOMM, pp.63-74. 2010.

[8] Yifei Lu, "SED: an SDN-based explicit-deadline-aware TCP for cloud data center networks", *Tsinghua Science and Technology*, 2016, hal. 491-499.

[9] Navin Kukreja, Guido Maier, Rodolfo Alvizu, "SDN based automated testbed foor evaluating multipath TCP", *Communication Workshop (ICC)*, 2016.

[10] Zhi Wang, Chuan Wu, "Responsive multipath TCP in SDN-based datacenters", *Communications (ICC)*, 2015, hal. 5296-5301.

[11] Jean Claude Franchitti, "Data Communication & Networks", Tersedia: http://www.nyu.edu/classes/jcf/g22.2262-001_sp10/slides/session9/NetworkCongestion-Causes-Effects-Controls.pdf.

[12] Raj Jain, "Congestion control in computer networks: issues and trends", *IEEE Network Magazine.* pp 24-30, 1990.

[13] Nurlinaamik, (Juni 2014), congestion [online], Tersedia: http://nurlinaamik.blogspot.co.id/2014/06/congestion-pada-jaringan-data.html.

[14] R. Krishnan dkk, (April 2014), SFC long-lived flow use cases [online], Tersedia: https://tools.ietf.org/html/draft-krishnan-sfc-long-lived-flow-use-cases-00.

[15] Tingwei Zhu, Fang Wang, Yu Hua, Dan Feng, Yong Wan, Qingyu Shi, Yanwen Xie, "MCTCP: Congestion-aware and robust multicast TCP in Software Defined networks", *Quality of Service (IWQoS), 2016 IEEE/ACM 24th International Symposiu,* 2016.

[16] Georgios Rodolakis, Amina Meraihi Naimi, Anis Laouiti, "Multicast Overlay Spanning Tree Protocol for Ad Hoc Networks", WWIC, pp.290-301, 2007.

[17] Cisco System (Agustus 2006), Understanding and configuring spanning tree protocol (STP) on catalyst switches [online], Tersedia: http://www.cisco.com/c/en/us/support/docs/lan-switching/spanning-tree-protocol/5234-5.pdf.

[18] Jaehyung Hwang, Joon Yoo, Sang-Hun Lee, "Scalable congestion control protocol based on SDN in data center networks", *IEEE Global Communications Conference (GLOBECOM),* 2015, hal 1-6.

[19] "OpenFlow Switch Specification version 1.5.0", Tersedia: https://www.opennetworking.org.

[20] Simon Jouet, Colin Perkins, Dimitrios Pezaros, "OTCP: SDN-manage congestion control for data center networks", *Network Operations and Management Symposium (NOMS), 2016 IEEE/IFIP,* 2016.

[21] Chunghan Lee, Yukihiro Nakagawa, Kazuki Hyoudou, "Flow-aware congestion control to improve throughput under TCP incast in datacenter networks, *Computer Software and Applications Conference (COMPSAC),* 2015, hal. 155-162.

[22] S. Bauer, R. Beverly, A. Berger, "Measuring the state of ecn readiness in servers, clients and routers", *SIGCOMM*, pp.171-180, 2011.

[23] Abdelmoniem, A. M., Bensaou, B., Abu, A. J., "SICC: SDN-based incast congestion control for data centers", *Communication* (ICC), 2017, hal 1-6.

[24] Lu, Y., Fan, X., Qian, L, "EQF: An explicit queue-length feedback for TCP congestion control in datacenter networks", *Advanced Cloud and Big Data* (CBD), 2017, hal. 69-74.