Analisis Mekanisme Active Queue Management (AQM) Berbasis Controlled Delay (CoDel) terhadap Bufferbloat pada Koneksi Asymmetric Digital Subscriber Line (ADSL)

Mechanism Analysis of Active Queue Management (AQM) Based on Controlled Delay (CoDel) against Bufferbloat on Asymmetric Digital Subscriber Line (ADSL) Connection

Ginanjar Rahmansyah[1] , Tri Brotoharsono, ST.,MT [2], Gandeva Bayu Satrya, ST.,MT [3]
[1,2,3] Fakultas Informatika , Universitas Telkom, Bandung
[1] ginanjar@students.telkomuniversity.ac.id, [2] tbh@telkomuniversity.ac.id, [3] gbs@telkomuniversity.ac.id

Abstrak

Bufferbloat adalah kondisi ketika buffer dengan ukuran besar cenderung selalu penuh, menyebabkan antrian panjang di dalam buffer, bila terjadi secara terus menerus dapat menyebabkan delay transmisi menjadi tinggi. Active Queue Management (AQM) merupakan salah satu mekanisme untuk mengatasi bufferbloat. Dalam Tugas Akhir ini, dilakukan implementasi dan pengujian AQM Controlled Delay atau CoDel untuk penanganan bufferbloat, kemudian membandingkannya dengan Drop Tail. CoDel akan diuji terhadap berbagai kondisi bottleneck dan intensitas traffic. Parameter keluaran yang ditinjau adalah latency, throughput dan packet loss. CoDel akan diterapkan pada PC Router berbasis Linux yang terhubung ke Internet menggunakan koneksi ADSL. Hasil pengujian menunjukkan bahwa pada parameter CoDel menunjukkan bahwa nilai target 1 ms lebih baik dari parameter yang direkomendasikan CoDel yang bernilai 5 ms dengan mendapatkan delay yang lebih rendah. Pada perbandingan AQM, CoDel jauh mengungguli Drop Tail bila ditinjau dari kemampuanya dalam menjaga latency yang rendah. Namun, apabila ditinjau dari parameter throughput dan packet loss-nya, CoDel lebih buruk daripada Drop Tail.
Kata Kunci: *bufferbloat, AQM, CoDel, delay, ADSL*

Abstract
Bufferbloat is a persistently full large size buffer problem that lead long queue in the buffer, if this occurs continuously, it could make transmission delay quite high. Active Queue Management (AQM) is one of the mechanism to solve bufferbloat. In this final project, Codel AQM will be implemented and tested to see its effectivity on handling bufferbloat, then it will be compared with Drop Tail. CoDel performance will be tested against various bottleneck conditions and traffic intensities. CoDel's output parameters measured are latency, throughput, and packet loss. CoDel will be implemented on Linux-based PC Router connecting to Internet using ADSL connection. The implementation result on CoDel parameter testing shows that target value with 1 ms is better in keeping low latency than the one recommended by CoDel which value is 5 ms. in comparison of AQM, CoDel outperforms Drop Tail in keeping low latency. But in terms of throughput and packet loss, CoDel is worse than Drop Tail.
Key words: bufferbloat, AQM, Codel, delay, ADSL

1. Pendahuluan

Beberapa tahun terakhir Internet mengalami kinerja yang buruk. Tingginya permintaan *bandwidth* tidak disertai dengan kualitas *latency* jaringan yang baik. Paket data pada protokol *transport* seperti TCP/IP dirancang untuk mengalir secepat mungkin sehingga membuat *buffer* dalam ukuran apapun menjadi cepat penuh. *Buffer* diperlukan untuk menampung arus paket yang deras. Namun, fungsi *buffer* saat kondisi penuh menjadi tidak berguna. Banyak antrian paket pada *buffer* menyebabkan waktu penanganan paket menjadi sangat lama. Jika hal tersebut terjadi, maka dapat menyebabkan *latency* menjadi tinggi. Fenomena ini dikenal dengan istilah *bufferbloat*. Fenomena *bufferbloat* dapat terjadi di mana pun, termasuk pada koneksi *Asymmetric Digital Subscriber Line* atau ADSL. ADSL adalah teknologi pengiriman data berkecepatan tinggi melalui kabel telepon dengan perbandingan *download* dan *upload* yang tidak sama atau asimetris. Teknologi ini adalah yang paling banyak digunakan oleh masyarakat.

Solusi untuk mengatasi masalah tersebut adalah dengan menerapkan teknik *Active Queue Management* (AQM) yang diterapkan untuk memantau dan mengelola *queue*. Sayangnya AQM tidak banyak diterapkan pada *router* atau perangkat jaringan lainnya [2]. CoDel atau *Controlled Delay* dipilih dan diyakini dapat mengatasi *bufferbloat* tersebut [7]. CoDel memiliki nilai parameter standar yang diklaim cocok diterapkan pada berbagai jenis *traffic* dan kondisi *bottleneck.* Namun pada *bandwidth* rendah yaitu di bawah 5 Mbps, nilai parameter standar tersebut belum teruji dengan baik [3].

Permasalahan yang dijadikan objek penelitian pada tugas akhir ini adalah apakah parameter *default* yang direkomendasikan [1] dengan nilai *target* 5 ms dapat secara efektif menjaga *latency* agar tetap rendah pada berbagai kondisi *bottleneck* dan intensitas *traffic*? Bila didapat *latency* yang rendah, bagaimana pengaruhnya terhadap parameter *throughput* dan *packet loss*? Apakah CoDel lebih baik dari Drop Tail dalam menjaga *latency* tetap rendah pada berbagai kondisi *bottleneck* dan *traffic*?

Tujuan yang ingin dicapai pada Tugas Akhir ini adalah mendapatkan nilai parameter masukkan CoDel (*target*) yang tepat sebagai acuan untuk mendapatkan latency yang rendah pada berbagai kondisi bottleneck dan intensitas traffic, dan membandingkan kinerja CoDel dan Drop Tail dengan mengamati parameter latency, throughput, dan packet loss. Pengujian dilakukan terhadap berbagai kondisi *bottleneck* dan intensitas *traffic.*

Batasan masalahnya adalah lab pengujian dilakukan pada lingkungan jaringan rumah yang terhubung ke Internet, bandwidth maksimal untuk download 2.4 Mbps dan upload 0.6 Mbps, tipe jaringan yang digunakan adalah jaringan rumah berbasis kabel, menggunakan IP versi 4, *bottleneck* antara modem ADSL dan *router* menggunakan *Hierarchy Token Bucket* (HTB) yang disesuaikan dengan *bandwidth* saat pengujian. Kondisi awal di antara keduanya menggunakan koneksi LAN 100 Mbps, buffering pada modem ADSL diabaikan karena memerlukan penelitian tersendiri [11], termasuk pada sepanjang path yang dilewati menuju tujuan (Server), pengukuran hanya dilakukan dari sisi Client.

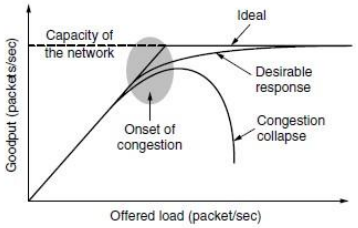
2. Dasar Teori

2.1 *Bufferbloat*

Bufferbloat adalah fenomena yang terjadi saat *buffer* pada suatu jaringan berada dalam kondisi penuh dan cenderung tetap penuh,

menyebabkan *latency* menjadi tinggi. Istilah yang baru-baru ini diperkenalkan Jim Gettys [2] untuk memberi kesadaran akan bahayanya masalah ini. Bila dilihat sejarahnya, sebenarnya masalah ini telah terjadi sejak Internet pertama kali dikembangkan. Dari sisi *user*, masalah yang sangat terasa adalah pada aplikasi seperti *browsing*, *video conference*, *voice call*, *online game* dan aplikasi lainya menjadi sangat lambat. Lambat di sini maksudnya adalah interaksi *user* dan aplikasi tidak lagi berjalan secara *real-time*, terjadi delay, atau ada beberapa gambar, suara, *frame* yang hilang.

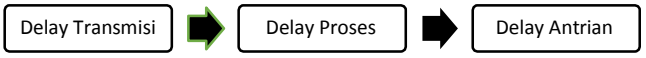
Kongesti merupakan masalah lama di Internet, muncul dengan berbagai bentuk dan gejala menyebabkan masalah besar. *Buffer* sangat penting dalam menangani paket di jaringan, namun ukuran *buffer* yang terlalu besar dan tidak adanya pengaturan di dalamnya membuat masalah ini menjadi semakin parah pada beberapa dekade terakhir. Keadaan ini dapat menyebabkan menurunnya kinerja jaringan seperti yang terlihat pada gambar berikut [18].



Gambar 2-1 Kongesti Jaringan

Kongesti terjadi ketika sebuah paket yang datang lebih cepat dari jumlah paket yang dapat ditransmisikan, menyebabkan adanya paket yang di-*drop*. Keadaan ini semakin buruk karena pengirim akan terus mengirim ulang paket sampai pengirim mendapatkan *ack*. Pada *traffic* yang padat besar, kemungkinan akan terjadi *congestion collapse* [18].

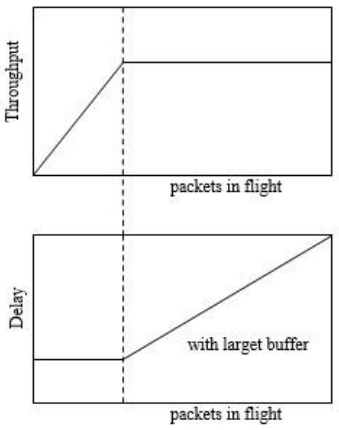
Delay transmisi adalah waktu yang diperlukan untuk mengirim paket pada link komunikasi, *delay* proses adalah waktu yang diperlukan untuk memroses paket, dan *delay* antrian adalah waktu tunggu suatu paket untuk diproses atau ditransmisikan, semua proses tersebut disebut *latency*. Proses yang dialami paket dalam sekali pengiriman adalah seperti yang ditunjukkan pada gambar berikut.



Gambar 2-2 *Delay* Transmisi, Proses, dan Antrian [5]

Dalam sekali pengiriman, paket juga harus melewati banyak titik atau *hop*. Selain itu harus juga melewati berbagai jenis ukuran *bandwidth*. Aliran paket dari pipa yang besar (*bandwidth* besar) menuju pipa kecil (*bandwidth* kecil) disebut *bottleneck*. Paket tidak dapat sampai di tujuan lebih cepat dari waktu yang diperlukannya untuk mengirim pada kondisi *bottleneck*. Tanpa adanya kontrol pada jaringan, *delay* akan menjadi lebih parah. Tempat ideal diletakkanya *buffer* adalah pada titik *bottleneck* tersebut untuk menahan *burst packet* atau aliran sejumlah paket yang sangat deras dalam waktu bersamaan. Selain itu *buffer* juga perlu diletakan pada kondisi *fast-to- slow transition*, dan perubahan dinamis *bandwidth*. Hubungan antara *throughput* dan *delay* adalah seperti yang digambarkan pada gambar 2-3. *Throughput* adalah banyaknya paket yang diterima dalam sekali transmisi.

Dari gambar tersebut dapat dilihat bahwa saat pengiriman paket, pada awalnya *throughput* naik secara konstan, dan *delay* berada dalam kondisi stabil. Namun saat paket bertemu *bottleneck*, *throughput* akan berhenti naik dan *delay* akan terus menaik. Pada *bottleneck* yang terdapat *buffer* yang besar, kondisinya akan semakin parah. *Delay* akan terus naik sampai pada titik *buffer* penuh. Setidaknya lebih dari beberapa detik, cukup untuk membuat pengguna aplikasi *real-time* frustasi.



Gambar 2-3 *Throughput* dan *Delay*

Membuat ukuran *buffer* menjadi kecil pun bukan lah solusi, justru hanya akan membuat masalah lain [2[15]. Tanpa adanya *buffer*, artinya paket tidak memiliki tempat untuk menunggu transmisi. Menyebabkan banyaknya paket yang didrop dan tentunya mengurangi *throughput*, walaupun setiap paket tetap akan memiliki *delay* yang konstan.

Masalah ini telah terjadi sejak lama, solusi dalam mengatasi masalah ini pun telah ada sejak dulu [2]. Algoritma *slow-start* dan *congestion-avoidance* dikembangkan pada protokol TCP dan banyak diterapkan di seluruh Internet sejak tahun 1990 sebagai adopsi dari *world wide web*. Tujuannya adalah untuk menjaga *delay* tetap rendah, memaksimalkan *throughput*, dan meminimalkan *packet loss*. Algoritma *slow-start* melakukan tebakan seberapa cepat TCP beroperasi, ketika *packet loss* pertama terdeteksi, TCP akan diminta untuk mengurangi rata-rata pengiriman paketnya, setelah itu masuk ke fase *congestion-avoidance*. Perlu diingat juga bahwa *packet loss* sangat penting keberadaanya bagi TCP dalam menangani kongesti, sebagai penanda bagi TC untuk memperlambat laju pengiriman paket.

Ukuran yang benar untuk *buffer* bukanlah perkara mudah [2]. Adapun rekomendasi mengenai ukuran *buffer* adalah menggunakan BDP atau *bandwidth-delay-product*, dimana *bandwidth* adalah rata-rata link di *bottleneck* dan *delay* adalah RTT atau *round-trip-time* antara pengirim dan tujuan. Idealnya *buffer* dengan BDP dapat menahan derasnya aliran paket pada *bottleneck*. Link *bandwidth* saat ini memiliki RTT yang bervariasi. Hal tersebut membuat pengaturan *buffer* statis menjadi tidak mungkin untuk kebanyakan link. Diharapkan ada algoritma *robust* yang dapat mengontrol *delay* tanpa melihat ukuran buffernya. *Bufferbloat* dapat terjadi dalam berbagai jenis link *bottleneck*, pada *router*, switch, host, GSM, 3G, *cable modem*, koneksi broadband, DSL, ADSL, DSLAM, dan sebagainya [2].

2.2 *Active Queue Management*

Anjuran untuk menggunakan AQM telah lama direkomendasikan [1]. Bila berbicara mengenai AQM maka hal tersebut tidak lepas dari kongesti. Kongesti adalah kondisi ketika agregat kebutuhan *bandwidth* melebihi kapasitas link yang tersedia. Akibatnya adalah banyaknya *packet loss*, utilisasi link yang rendah (*throughput* rendah), *delay queue* yang tinggi, dan *congestion collapse*.

Dalam mengatasi kongesti ada dua cara yang dapat dilakukan yaitu *congestion control* dan *congestion avoidance* [13]. *Congestion control* berperan saat jaringan *overloaded*, sedangkan *congestion avoidance* aktif sebelum jaringan menjadi *overloaded*. AQM merupakan satu cara *congestion* control pada *closed-loop* atau packet *switched network* yang eksplisit. AQM berperan penting dalam mengatur *buffer* dan mengurangi *latency*. Secara umum berfungsi dalam mengontrol *queue* agar tidak full atau tumbuh terlalu besar dengan memonitor *queue* paket dengan menandai (*marking*) atau mendropnya.

Pada kenyataanya, AQM tidak banyak digunakan pada *router*, dan benar-benar tidak diterapkan pada banyak perangkat [2]. Lebih jauh lagi, murahnya harga memori dan keinginan untuk menghindari *packet loss* demi memperbesar *throughput* telah menjadikan *buffer* semakin besar tak terkontrol, memasangnya pada banyak host, *router*, switch, dan semua perangkat jaringan lainnya telah menjadikan Internet saat ini ditutupi awan gelap yang kini terkenal dengan istilah *bufferbloat*.

Selagi mengembangkan algoritma *congestion-avoidance*, RED atau *Random Early Detection* [14] diciptakan untuk menangani masalah *large buffering* ini. Kemudian RED ini dikenal sebagai *Active Queue Management* (AQM). Namun RED ini sulit dikonfigurasikan karena memiliki banyak parameter. CoDel atau *Controlling Delay* oleh Kathie Nichols dan Van Jacobson [7] memiliki banyak kelebihan dalam mengatasi *bufferbloat*, terutama dalam menangani *delay*/*latency*. CoDel banyak berinteraksi saat *dequeue stage*, yaitu saat paket meninggalkan *queue* atau saat paket ditransmisikan. CoDel menandai setiap paket yang masuk ke *queue* dengan timestamp.

2.3 Codel: AQM *for Low Latency*

CoDel adalah AQM baru yang dikembangkan untuk mengatasi masalah *bufferbloat* (seperti pada *router*) dan menggantikan algoritma RED karena dinilai sulit dikonfigurasi karena perlu dilakukan *tuning* dengan tingkat ketelitian tinggi [7]. CoDel bersifat adaptif "*no-knobs*" karena dikembangkan dengan tujuan agar dapat dikonfigurasikan dengan mudah (*parameterless*) atau tidak banyak parameter yang perlu diatur, tidak berdasar pada *queue size*, dapat menangani *good queue* dan *bad queue* secara berbeda, mengontrol *delay*, menjaga *delay*/*latency* rendah saat terjadi *burst traffic*, dapat beradaptasi dengan link dinamis tanpa berdampak buruk pada *throughput*, dan cukup sederhana untuk diimplementasikan pada *router* dengan skala kecil ataupun besar.

2.3.1 *Queue* pada Codel

Cara kerja CoDel adalah dengan membedakan dua tipe *queue*, yaitu *good queue* dan *bad queue*. *Good queue* adalah *queue* (tanpa *bufferbloat*) menyerap *burst traffic* pada *bottleneck*, seperti antara LAN dengan kecepatan tinggi dan Internet dengan koneksi lebih lambat, atau antara koneksi kabel dan nirkabel. *Delay* cenderung rendah. *Bad queue* adalah *queue* (terjadi *bufferbloat*) yang terisi penuh bersamaan saat transmisi paket, jadi *queue* tidak pernah kosong. *Delay* cenderung eksesif atau tinggi [7][15]

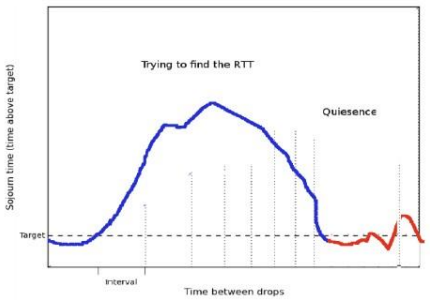
Cara CoDel membedakan dua jenis *queue* tersebut adalah dengan melihat minimum *queue* pada *interval*. Bila *queue* tersebut tidak di-*drop* pada *threshold* tertentu saat *queue* tersebut mengalir untuk bereaksi terhadap *congestion signal* (secara kasar satu RTT), menyebabkan paket harus di-*drop* untuk memicu *signal congestion* dan menurunkan aliran paket. Sebagaimana *queue* yang telah melebihi *threshold*, paket akan di-*drop* dengan rata-rata menaik secara linear

2.3.2 Algoritma Codel

```
On arrival of every packet:
if current_queue_size < queue_limit then
    Enqueue the packet
    Attach a timestamp in packet header
end
else
    Drop the Packet
end
On departure of every packet:
dequeue_time = timestamp for dequeue time
soujourn_time = dequeue_time – enqueue_time
if inside the dropping state then
    if soujourn_time < target or
    current_queue_size < MTU then
        Do not drop packets
        Come out of dropping state
    end
    else
        while dequeue_time ≥ next_drop_time do
            Drop the packet
            count = count + 1
            next_drop_time + = interval /√count
        end
    end
end
else if outside dropping state and first packet is
being dropped then
    Enter the dropping state
end
```

Gambar 2-4 Algoritma Codel [10]

Cara kerja CoDel adalah dengan menambahkan timestamp pada setiap paket yang datang (*enqueue)*. Ketika paket sampai pada *queue head*, waktu selama berada di *queue* dihitung. Perhitunganya sederhana karena hanya menghitung satu value saja, jadi akan relatif cepat. Jika waktu yang dihabiskan oleh paket antara *queue* lebih lama dari *threshold* yang telah didefinisikan, CoDel mengatur timer untuk men-*drop* paket saat *dequeue*. Proses drop paket ini hanya dilakukan ketika *delay* antrian di *time window* melebihi nilai *threshold*, dan *queue* menyimpan setidaknya satu byte MTU atau *Maximum Transfer Unit* [15].



Gambar 2-5 *Codel Drop Scheduler*

Penjelasan dari ilustrasi pada gambar 5 [6] adalah sebagai berikut:

- Codel berasumsi bahwa *standing queue* dari target dapat diterima (seperti yang telah ditentukan *local minimum*), dan tidak akan dilakukan drop ketika paket yang masuk kurang dari byte MTU di dalam *buffer*.
- Untuk menjamin nilai minimum tidak "basi", Codel harus sudah mengalami *sliding window*.
- Ketika *sojourn time* melebihi target atau setidaknya interval, sebuah paket harus didrop dan *control law* digunakan untuk men-*set next drop time*.
- *Next drop time* adalah akar dari sejumlah paket yang didrop sejak *dropping state* telah dimasuki (untuk mendapatkan *throughput* linear).
- Saat *sojourn time* kurang dari target, *controller* berhenti melakukan *drop* paket.

2.3.3 Parameter Codel

CoDel memiliki empat parameter yaitu *limit, target, interval, dan explicit congestion control* (ECN). *Limit* merupakan batasan ukuran antrian yang ditentukan. ECN adalah fitur tambahan yang digunakan untuk menandai paket daripada men-*drop* paket. *Target* adalah *delay* minimum saat di dalam antrian yang dapat diterima, sedangkan *interval* adalah nilai parameter untuk menjamin minimum *delay* tidak terlalu lama

Terkait *target* (5ms) dan *interval* (100ms), nilai parameter tersebut merupakan nilai yang direkomendasikan [19]. Untuk menentukannya pun dilakukan perhitungan matematis, sesuai pada draft jurnal IETF [9] section 3.2. Nilai *target* adalah (5-10%) dari nilai *interval* (100ms).

2.4 ADSL

ADSL atau *Asymmetric Digital Subscriber Line* adalah sebuah teknologi yang memungkinan data kecepatan tinggi dikirim melalui kabel telepon [16]. Standar modem ADSL yang banyak digunakan seperti ITU G.992.1, ITU G.992.2, ITU G.994.1 dan ITU G.995.1. Untuk standar modem ADSL2 seperti ITU G.992.3, dan ITU G.992.4. Dan untuk standar modem di atas keduanya adalah ADSL2+ ITU-T 992.5. ADSL membagi frekuensi dari sambungan yang digunakan dengan asumsi sebagian besar pengguna Internet akan lebih banyak mengambil (*download*) data dari Internet daripada mengirim (*upload*) ke Internet. Oleh karena itu, kecepatan data dari Internet biasa sekitar tiga sampai empat kali kecepatan ke Internet. Karena kecepatan *upload* dan *download* tidak sama, digunakan istilah asimetris.

3. Analisis Perancangan Sistem

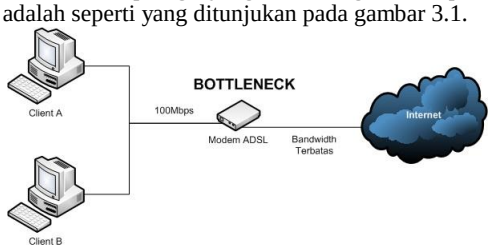
3.1 Gambaran Umum Sistem

Seperti yang sudah dijelaskan pada bab dua, kondisi ideal yang ingin dicapai oleh CoDel adalah untuk menjaga *latency* tetap rendah. Pengujian dan implementasi akan dilakukan pada studi kasus nyata (*router*) untuk mendapatkan hasil yang dapat dirasakan langsung oleh *user*. Parameter inputan dari CoDel yaitu *target* diuji terhadap berbagai kondisi. Pengujian tersebut bertujuan untuk mendapatkan hasil maksimal terhadap masalah yang sudah terdefinisi (akan dijelaskan pada bagian berikutnya). Setelah itu hasil analisis CoDel akan dibandingkan dengan *router* yang tidak menggunakan AQM.

3.2 Perancangan Sistem

Bufferbloat adalah masalah yang perlu diatasi. Dalam Tugas Akhir ini, studi kasus yang digunakan adalah pada jaringan rumahan yang terkoneksi ke Internet dan biasanya terdiri dari beberapa *client*. Umumnya *home network* atau jaringan rumahan terdiri dari beberapa *client* yang terhubung ke modem, kemudian langsung terhubung ke Internet. Terdapat kondisi *bottleneck* (*fast-to-slow*) dari koneksi LAN ke WAN (Internet). Masalah ini menjadi studi kasusnya.

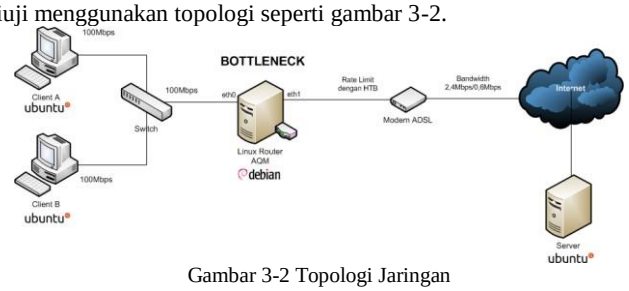
Secara umum topologi yang biasa digunakan pada jaringan rumahan adalah seperti yang ditunjukan pada gambar 3.1.



Gambar 3-1 Topologi Jaringan Rumahan

Pada modem ADSL, selain befungsi sebagai modem, juga berfungsi sebagai *router* dan *access point.* Kebanyakan tidak ada mekanisme pengaturan paket pada modem, lalu lintas paket pun terjadi secara FIFO atau *First In First Out.* Pada *bandwidth* kecil, satu aktifitas torrent saja dapat menyebabkan *browsing* menjadi lambat atau *video conference* menjadi terganggu. Hal tersebut terjadi karena tidak adanya mekanisme pengaturan paket. Sebagai mana yang dijelaskan pada bab 2.3, CoDel dapat menjadi solusi. Kondisi ideal yang ingin dicapai CoDel adalah menjaga *latency* tetap rendah.

Untuk memastikan CoDel dapat berjalan baik, maka perlu dilakukan serangkaian pengujian. Pada tugas akhir ini CoDel akan diuji menggunakan topologi seperti gambar 3-2.



Gambar 3-2 Topologi Jaringan

Pada kondisi sebenarnya antara *Router* dan Modem ADSL memiliki kapasitas link 100 Mbps. Namun agar kondisi *bottleneck* didapatkan hanya pada sisi *Router* saja, maka link antara keduanya akan dibatasi menggunakan *Hierarchy Token Bucket* (HTB) dan disesuaikan dengan kapasitas yang diberikan penyedia jasa internet. Pengukuran parameter uji dilakukan dari sisi *client*.

3.2.1 Spesifikasi Perangkat Keras

Perangkat keras yang digunakan mempunyai spesifikasi sebagai berikut:

1. Satu PC Router, dengan prosesor Intel (R) Pentium (R) 4 CPU 1.60 GHz, RAM 385 MB, 2 buah NIC 100 Mbps dengan masing-masing chipset RTL8139too dan Sundance. PC Router ini berfungsi sebagai router *bottleneck* dengan Codel sebagai *Active Queue Management* dan server netperf.
2. Satu Server, dengan prosesor Intel Core2Duo 2.2 GHz, RAM 1 GB, NIC 100Mbps. Berfungsi sebagai server netperf.
3. Dua Klien, dengan procesor Intel Core i3 2.2 GHz, RAM 4 GB, 1 buah NIC 100Mbps. Berfungsi sebagai netperf *client* untuk melakukan *traffic loader* ke server.
4. Satu Modem Router ADSL2+ TD-W8951ND 150Mbps.
5. Satu Switch D-Link 100Mbps.

3.2.2 Spesifikasi Perangkat Lunak

Perangkat lunak yang digunakan pada implementasi ini adalah sebagai berikut:

1. Sistem Operasi dan Kernel
   Sistem operasi yang digunakan sebagai AQM pada PC Router adalah Linux Debian 7.0 dengan kernel 3.14-1-686-pae 64-bit. Sedangkan sistem operasi yang digunakan pada Server dan Klien adalah Linux Ubuntu 14.04. Pada masing-masing komputer dipasang server dan *client* SSH untuk melakukan *remote host* saat pengujian.
2. AQM
   AQM yang diteliti hanya Codel. Pada Linux (dengan kernel 3.5 atau di atasnya), modul kernel Codel dapat diterapkan menggunakan *tc* atau *traffic control*. Selain itu Codel sendiri memerlukan komponen lain, diantaranya HTB atau *Hierarchy Token Bucket* untuk pengaturan *bandwidth*, ETHTOOL untuk menonaktifkan fitur *offload* agar pengaturan paket dapat sepenuhnya dilakukan di sisi kernel oleh Codel.
3. Perangkat Lunak Pendukung
   a. Netperf-wrapper [13] sebagai *wrapper* netperf dan fping. Tes menggunakan RRUL [14] (*Realtime Response Under Load*) yang diklaim sebagai acuan dalam menguji *bufferbloat*.
   b. Netperf untuk *traffic loader*.
   c. Fping untuk mengukur *latency*.
   d. SSH untuk memudahkan *remote host*.
   e. Siege untuk melakukan menguji HTTP.
   f. Microsoft Excel untuk mengolah file CSV (*Comma Separated Values)*
   g. *Script* yand dibuat penulis untuk memudahkan dalam melakukan pengujian, agar pengujian dapat dilakukan secara otomatis.

3.3 Perancangan Uji

Berdasarkan topologi yang ada pada gambar 3.1, pengujian akan dilakukan dengan memberikan aliran TCP dari node *client* ke *Server*. Secara garis besar klasifikasi aliran *traffic* TCP yang digunakan dapat dibagi menjadi 4 bagian, yaitu skenario *Low Congestion*, skenario *Medium Congestion*, skenario *High Congestion*, dan *Very High Congestion.* Masing-masing skenario akan diberi aliran TCP dengan jumlah aliran (N) setiap arahnya. Artinya jika N = 1, maka 1 TCP flow dari *Client* ke *Server*, dan 1 TCP flow lagi dari *Server* ke *Client*, menjadikan totalnya 2 aliran TCP konkuren.

Parameter input CoDel yang akan diuji pada skenario *traffic* ini adalah nilai *target* dan *interval*. Nilai *Target* merupakan 5% dari nilai *Interval*. Karena keduanya merupakan satu kesatuan, maka untuk selanjutnya parameter yang digunakan hanya *target* saja. Nilai rekomendasi *target* dan *interval* CoDel sebesar 5 ms dan 100 ms, artinya CoDel berusaha untuk menjaga *delay* di dalam *queue* (*sojourn time*) kurang dari 5 ms, dan untuk kondisi terburuknya selama 100 ms.

Pada banyak penelitian [4][5][7][12][15] dengan berbagai skenario pengujian *traffic*, kongesti, dan kondisi lainnya, nilai *target* yang diuji menggunakan default parameter. Sebenarnya hal tersebut bukan tanpa sebab, karena nilai tersebut didapat dari perhitungan matematis, sesuai pada draft jurnal IETF [9]. Selain itu besar nilai tersebut sangat cocok untuk *bandwidth* kurang dari 100 Mbps, namun tidak untuk *bandwidth* kecil yang kurang dari 5 Mbps seperti yang dikemukakan pada project *bufferbloat.net* [3].

Pada penelitian lain [8] parameter ini diuji dengan variasi *target* berbeda (nilai 1, 5, 20, dan 30) dengan *bottleneck* 10 Mbps pada jaringan LAN yang notabene memiliki link 100 Mbps. Hal ini mendorong penulis untuk menguji parameter tersebut dengan variasi nilai *target* dan kondisi *bandwidth* kecil yang dibuat berbeda pula demi mendapatkan *latency* yang rendah.

Untuk selanjutnya variasi parameter *target* yang akan diuji pada skenario ini adalah seperti yang ditunjukan pada tabel berikut:

Target (ms)	*Interval* (ms)
1	25
2	50
5 (default)	100 (default)
7	150
10	200
12	250
15	300
17	350
20	400
22	450
25	500

Tabel 3-1 Variasi *Target* dan *Interval*

Skenario tersebut akan dibandingkan terhadap 3 keadaan (*bottleneck bandwidth*) berbeda. Pengaturan bandwidth pada router akan menggunakan HTB. Variasi keadaan *bottleneck* tersebut didasarkan pada sifat dasar koneksi ADSL yang memiliki kecepatan download yang besar, sementara kecepatan uploadnya kecil. Untuk nilai dari *bandwidth* disesuaikand dengan kapasitas maksimal yang diberikan penyedia jasa layanan Internet.

- *Bottleneck* 1 (up/down): 0.25 Mbps/ 0.50 Mbps
- *Bottleneck* 2 (up/down): 0.40 Mbps/ 1.00 Mbps
- *Bottleneck* 3 (up/down): 0.50 Mbps/ 2.00 Mbps

3.3.1 Skenario 1 – *Low Congestion*

Pada skenario pertama ini akan diberikan *traffic* jaringan dengan *Low Congestion.* Dimana jumlah aliran TCP N = 2. Pengujian akan berlangsung selama 70 detik. Dimulai pada detik ke-5 ketika *traffic* dibangkitkan, dan selesai pada detik ke-65.

3.3.2 Skenario 2 – *Medium Congestion*

Pada skenario 2 ini akan diberikan *traffic* jaringan dengan *Medium Congestion.* Dimana jumlah aliran TCP N = 8. Pengujian akan berlangsung selama 70 detik. Dimulai pada detik ke-5 ketika *traffic* dibangkitkan, dan selesai pada detik ke-65.

3.3.3 Skenario 3 – *High Congestion*

Pada skenario 3 ini akan diberikan *traffic* jaringan dengan *High Congestion.* Dimana jumlah aliran TCP N = 14. Pengujian akan berlangsung selama 70 detik. Dimulai pada detik ke-5 ketika *traffic* dibangkitkan, dan selesai pada detik ke-65.

3.3.4 Skenario 4 – *Very High Congestion*

Pada skenario 4 ini akan diujikan pengaruh *traffic* jaringan LAN terhadap link ke Internet. Aliran TCP N = 14 akan dibangkitkan di antara Klien dan Router, di saat yang sama Klien lain mengirim *traffic* dengan aliran TCP N = 14 ke Server (Internet). Total TCP Flows adalah 28. *Bottleneck* 0.50 Mbps/ 2.00 Mbps. Kategori *traffic Very High Congestion*.

3.4 Strategi Pengujian

Sebelum Sebelum dilakukan pengujian terhadap sistem yang telah dibuat, dilakukan pengukuran terlebih dahulu terhadap no-AQM (Drop Tail). Secara *default router* memiliki mekanisme FIFO atau *First In First Out* untuk setiap paket yang masuk dan keluar. Tujuan pengukuran ini adalah untuk membandingkan keadaan *router* yang telah menggunakan AQM CoDel dengan Drop Tail. Parameter dari sisi *user* yang akan diukur adalah *latency*, *throughput*, dan *packet loss*. Bentuk pengukuran akan dilakukan dengan menciptakan kondisi *bufferbloat*, kemudian kondisi tersebut dicapai dengan memberikan *traffic* dengan *load* tinggi (*Heavy Congestion*) terhadap *bottleneck* seperti yang telah didefinisikan sebelumnya.

3.5 Parameter Uji

Terdapat tiga parameter uji yang digunakan dalam menganalisis performa AQM:

- *Latency*

  Merupakan satuan waktu yang dibutuhkan paket saat berada di jaringan pada selang waktu paket masuk antrian pada node sender sampai paket diterima oleh receiver, seperti yang ditunjukan pada persamaan (1).

  $$\text{Latency (ms)} = \text{Waktu saat paket diterima} - \text{Waktu saat paket dikirimkan} \quad \ldots (1)$$

- *Throughput*

  Merupakan jumlah total byte yang diterima oleh receiver pada selang waktu saat paket pertama dan terakhir dikirimkan, seperti yang ditunjukan pada persamaan (2).

  $$\text{Throughput (Mbps)} = \frac{\text{Jumlah paket diterima}}{\text{Periode pengamatan}} \quad \ldots\ldots (2)$$

- *Packet Loss*

  Mengindikasikan rata-rata *packet loss* yang dirasakan pada sisi *client*. Yang diukur dalam tugas akhir ini adalah perbandingan jumlah paket yang dibuang dalam jaringan dengan jumlah paket yang dikirimkan selama selang waktu pengujian dilakukan. Dapat dilihat pada persamaan (3)

  $$\text{Packet loss (\%)} = \frac{\text{Jumlah paket yang diterima}}{\text{Jumlah paket yang dikirimkan}} \times 100\% \quad \ldots (3)$$

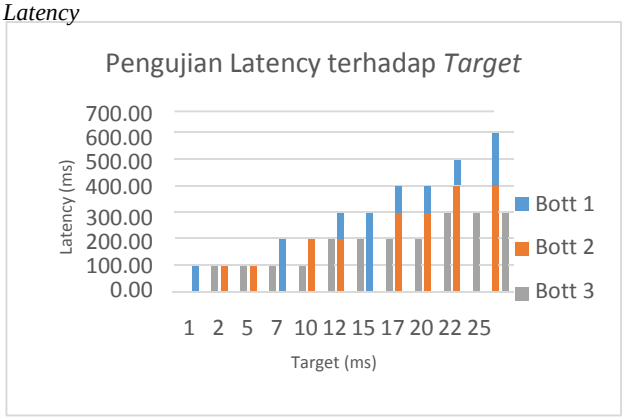
4. Pengujian dan Analisis

4.1 Analisis Hasil Pengujian Skenario

Pada jurnal ini hanya ditulis tentang skenario bagian kedua saja yaitu skenario dengan *medium congestion*, karena skenario ini lah yang digunakan sebagai pembanding antara Codel dan Drop Tail, dengan penjelasan sebagai berikut:

4.1.1 Skenario 2 – *Medium Congestion*

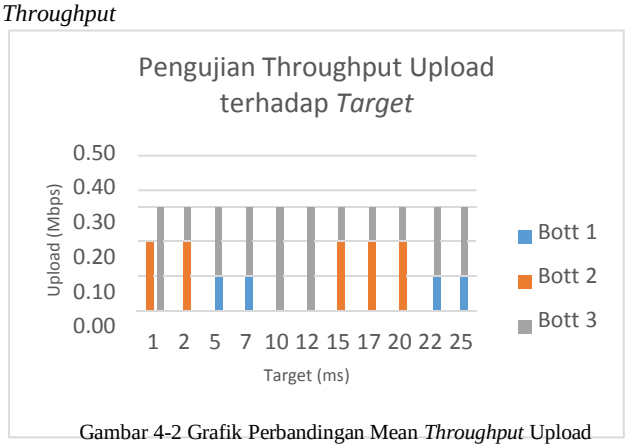
- *Latency*



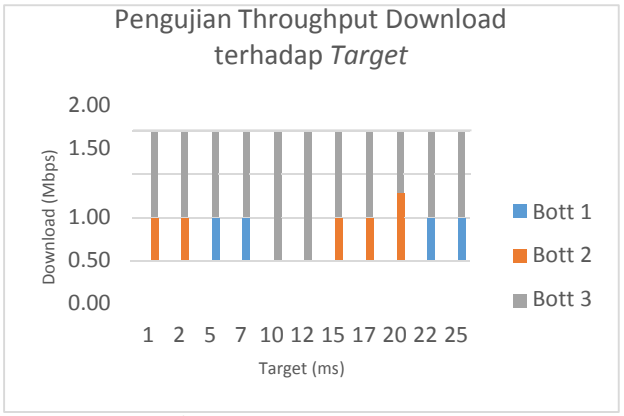
Gambar 4-1 Grafik Perbandingan Mean *Latency* Skenario 2

Gambar 4-5 menunjukkan pengujian parameter *latency* terhadap parameter *target* CoDel. Hampir sama seperti pada skenario 1 dilihat dari *bandwidth bottleneck*-nya, menunjukkan bahwa semakin besar *bandwidth bottleneck*, maka akan semakin rendah *latency*. Sementara seiring pertambahan nilai *target* hanya menyebabkan *latency* menjadi semakin tinggi. Dibandingkan dengan parameter default CoDel (*target* 5 ms), selisih *latency* nya cukup jauh. Dari skenario ini nilai *target* dengan *latency* terendah adalah 1 ms.

- *Throughput*



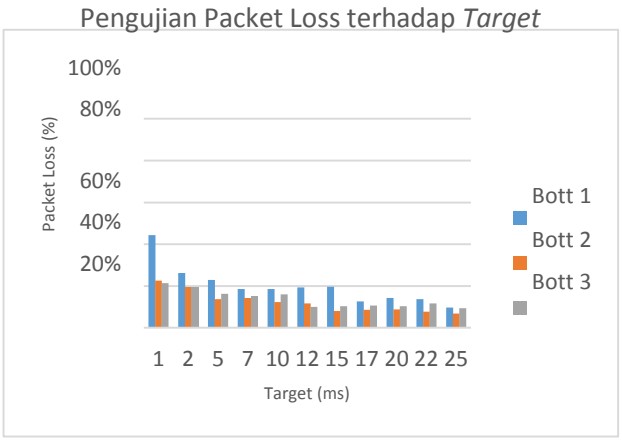
Gambar 4-2 Grafik Perbandingan Mean *Throughput* Upload Skenario 2



Gambar 4-3 Grafik Perbandingan Mean Throughput Download Skenario 2

Gambar 4-6 dan 4-7 menunjukkan pengujian parameter *throughput* upload dan download terhadap parameter *target* CoDel. Dilihat dari grafik, berapapun nilai *target* yang diujikan, *throughput* menunjukkan nilai yang relatif stabil. Utilisasi *throughput* upload lebih buruk dari hasil pada skenario 1 yaitu hanya mendekati 70%, sementara utilisasi download sedikit lebih baik dari hasil pada skenario 1 mendekati 80%.

- *Packet Loss*



Gambar 4-4 Grafik Perbandingan Mean *Packet Loss* Skenario 2

Pada Gambar 4-8 menunjukkan pengujian parameter *target* terhadap *packet loss*. Dari grafik dapat dilihat bahwa pada nilai *target* 1 – 15 ms, *packet loss* menunjukkan akan paling tinggi bila dibandingkan dengan yang lainya.

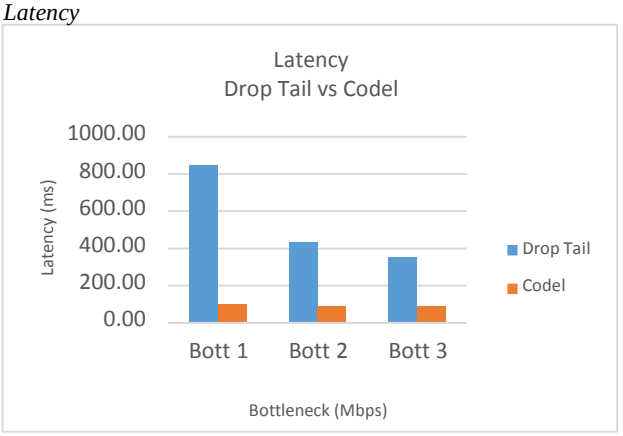
4.1.2 Analisis Hasil Pengujian Setiap Skenario

Dari hasil pengujian skenario 1 sampai 4, didapat nilai *target* 1 ms dengan *latency* terendah. Utilisasi *throughput* relatif stabil untuk semua skenario, berapapun nilai *target*, *bottleneck bandwidth*, dan *traffic* yang diberikan. Sementara *packet loss* pada setiap skenarionya cukup tinggi, sebagai kompensasi untuk mendapatkan *latency* yang rendah.

4.2 Analisis Perbandingan Drop Tail dan Codel

Setelah melakukan pengujian setiap skenario, didapat nilai *target* dari parameter inputan CoDel dengan hasil yang menunjukkan *latency* paling rendah yaitu 1 ms. Berikut akan ditampilkan grafik perbandingan antara *router* yang menggunakan AQM (Drop Tail) dengan *router* yang menggunakan AQM (CoDel). Data CoDel yang dibandingkan adalah data dengan nilai *target* yang memiliki *latency* paling rendah. Trafik yang digunakan *High Congestion.*

- *Latency*

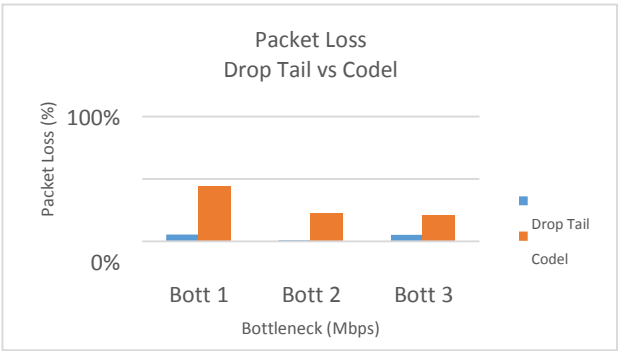


Gambar 4-5 Perbandingan *Latency* Drop Tail vs Codel

Gambar 4-13 menunjukkan perbandingan mean *latency* Drop Tail dan CoDel dalam waktu satu menit dengan berbagai perubahan *bottleneck bandwidth*. Kondisi *bottleneck* yang diberikan adalah *bottleneck* 1 (up/down): 0.25 Mbps/ 0.50 Mbps, *bottleneck* 2 (up/down): 0.40 Mbps/ 1.00 Mbps, dan *bottleneck* 3 (up/down): 0.50 Mbps/ 2.00 Mbps. Hasil sepertinya yang diharapkan. CoDel mampu menjaga *latency* tetap rendah bila dibandingkan dengan Drop Tail. Paket tidak dapat sampai di tujuan lebih cepat dari waktu yang diperlukannya untuk mengirim paket pada *bottleneck.* Paket harus mengalami minimum *latency* tertentu pada *bottleneck.*

CoDel tidak mengizinkan sebuah paket berada di dalam *queue* lebih dari 1 ms (sesuai hasil pengujian *target*), maka CoDel men-*drop* paket dan mencegah terjadinya kongesti di *buffer*. Berbeda halnya pada Drop Tail, semua paket dibiarkan masuk satu per satu ke dalam buffer. Ketika *burst packet* dengan jumlah besar datang, *buffer* tidak mampu menampungnya, *queue* menjadi panjang, sehingga menimbulkan *delay/latency* yang tinggi. Drop Tail hanya melakukan *drop* paket ketika *bufferoverflow.*

- *Packet Loss*

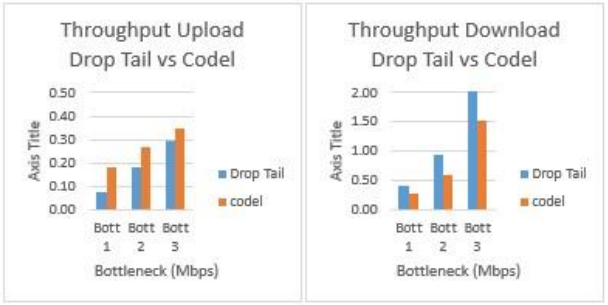


Gambar 4-6 Perbandingan *Packet Loss* Drop Tail vs Codel

Gambar 4-14 menunjukkan perbandingan mean *packet loss* Drop Tail dan CoDel dalam waktu satu menit dengan berbagai perubahan *bottleneck bandwidth.* Pada bagian ini, rasio mean *throughput* dan *packet loss* hanya sebagai indikasi untuk melihat

kemampuan CoDel dalam menangani utilisasi *bandwidth*. Seperti yang dijelaskan pada bab 2, fokus utama CoDel adalah mengontrol *delay/latency.* CoDel menunjukkan mean *packet loss* lebih tinggi dari pada Drop Tail. CoDel men-*drop* paket sebanyak yang diperlukan agar dapat menjaga *latency* di bawah nilai *target* (1 ms). Drop Tail menunjukkan mean *packet loss* paling rendah, hal ini karena Drop Tail hanya melakukan *drop* paket ketika terjadi *buffer overflow* saja. Hasil mean *packet loss* ini berdampak pada *throughput* yang didapat.

- *Throughput*



Gambar 4-7 Perbandingan *Throughput* Drop Tail vs Codel

Gambar 4-15 menunjukan perbandingan mean *throughput* upload dan download Drop Tail dan Codel dalam waktu satu menit dengan berbagai perubahan *bottleneck bandwidth.* Kapasitas bandwidth yang dapat disediakan pada jalur upload lebih kecil dari pada jalur download. Hal ini menunjukkan bahwa pada bottlenck tinggi, Codel akan banyak melakukan *drop* paket. Sehingga *throuhput* upload Codel lebih tinggi dari Drop Tail. Sementara pada *throughput* download Codel kalah unggul dari Drop Tail. Mean *packet loss* Codel menunjukkan angka yang tinggi sekali sehingga *throughput* yang didapat pun tidak tinggi, hal ini sebagai kompensasi untuk mendapatkan *latency* yang rendah

5. Kesimpulan dan Saran

5.1 Kesimpulan

Berdasarkan Berdasarkan hasil dari beberapa pengujian yang diterapkan pada skenario dapat diambil kesimpulan sebagai berikut:

1. Hasil pengujian pada skenario 1 sampai dengan 4 dengan berbagai kondisi *bottleneck* dan intensitas *traffic*, didapat nilai *target* dengan *latency* terendah sebesar 1 ms. Hal ini membuktikan bahwa pada *bandwidth* rendah (kurang dari 2 Mbps), nilai *target* tersebut lebih baik dari nilai *target* yang direkomendasikan oleh CoDel.

   Bila ditinjau dari *throughput*, hasil pengujian menunjukkan nilai yang relatif stabil, walaupun utilisasinya tidak maksimal. Sementara *packet loss* menunjukkan nilai yang bertolak belakang dengan pertambahan nilai *target*. Semakin rendah nilai *target* maka akan semakin besar *packet loss*-nya.
2. Pada berbagai kondisi *bottleneck* dengan *traffic* 14 koneksi TCP Codel juga lebih unggul dari pada Drop Tail dalam menjaga *latency* tetap rendah. Hal ini disebabkan karena CoDel selalu menjaga agar *queue* tidak penuh dan menjaga *delay* pada *queue* di bawah nilai *target.*
   Throughput pada CoDel sedikit lebih rendah bila dibandingkan dengan Drop Tail. Sementara *packet loss* pada CoDel menunjukkan presentase yang cukup tinggi. Hal tersebut disebabkan karena mekanisme Codel dalam melakukan *drop* paket.
3. CoDel baik dengan nilai *target* 1 ms (sesuai hasil pengujian) atau 5 ms (sesuai rekomendasi CoDel) masih belum cukup untuk dapat menggantikan Drop Tail, karena walaupun CoDel dapat menjaga *latency* tetap rendah, CoDel belum mampu memaksimalkan *throughput* dan meminimalkan *packet loss* pada *bandwidth* asimetris rendah dan berbagai intensitas *traffic*..

5.2 Saran

Saran-saran di bawah ini merupakan proses tindak lanjut dari penulis yang belum terealisasikan, antara lain sebagai berikut:

1. Menguji parameter CoDel dengan nilai target 1 ms pada berbagai jenis aplikasi real-time, karen latency yang rendah saja nampaknya belum cukup bila throughput belum maksimal dan packet loss masih tinggi.
2. Membandingkan CoDel dengan FQ_CoDel. FQ_CoDel adalah Active Queue Management gabungan antara CoDel dengan SFQ (Stochastic Fair Queueing).

Daftar Pustaka

[1] Braden, R., dan teman teman. 1998. "Recommendations on Queue Management and Congestion Avoidance in the Internet, RFC2309 (Informational), Internet Engineering Task Force." *Internet Engineering Task Force, RFC2309 (Informational).* April. http://www.ietf.org/rfc/rfc2309.txt.

[2] Gettys, J., Kathleen N. 2011. "Bufferbloat: Dark Buffers in the Internet." *AQM Queue.* November. http://queue.acm.org/detail.cfm?id=2071893.

[3] Gettys, J., Kathleen, N., dan teman-teman. 2014. *http://www.bufferbloat.net.* Agustus 12. http://www.bufferbloat.net/.

[4] Greg W., Dan R. 2013. "Active Queue Management Algorithms DOCSIS 3.0." *CableLabs.* April. http://www.cablelabs.com/wp-content/uploads/2013/11/Active_Queue_Management_Algorithms_DOCSIS_3_0.pdf.

[5] Hoiland-Jorgensen, Toke. 2012. "Battling Bufferbloat: An experimental comparison of four approaches to queue management in Linux Master module project Computer Science." *RUDAR (Roskilde University Digital Archive.* Desember. http://rudar.ruc.dk/handle/1800/9322.

[6] Høiland-Jørgensen, Toke. 2014. "Netperf Wrapper - Python wrapper to run multiple simultaneous netperf instances and aggregate the results." Accessed November 2014. github.com/tohojo/netperf-wrapper.

[7] Jacobson, V., Kathleen, N. 2012. "Controlling Queue Delay - A modern AQM is just one piece of the solution to bufferbloat." *Asscociation for Computing Machinery (ACM Queue).* Mei 6. http://queue.acm.org/detail.cfm?id=2209336.

[8] Naeem, K., David, R., Michael, W. 2014. "The new AQM kids on the block: An experimental evaluation of CoDel and PIE." *IEEE Xplore* 85-90.

[9] Nichols, K., Jacobson, V. 2014. "Controlled Delay Active Queue Management." *Internet Engineering Task Force.* Oktober 24. http://www.ietf.org/id/draft-ietf-aqm-codel-00.txt.

[10] Preethi Rao V., Mohit P. Tahiliani, Udaya Kumar K. Shenoy. 2014. "Analysis of sfqCoDel for Active Queue Management." *IEEE Xplore* 262-267.

[11] Radika, V., Jason, B., Grenville, A. 2014. "Buffer size estimation of TP LINK TL-PA211KIT HomePlug AV adapters." *Centre for Advanced Internet Architectures (CAIA) in the Faculty of Science, Engineering and Technology at Swinburne University of Technology.* September. http://caia.swin.edu.au/reports/130417A/CAIA-TR-130417A.pdf.

[12] Raghuvanshi, D.M., B. Annappa, and Mohit P. T. 2013. "On the Effectiveness of CoDel for Active Queue Management." *IEEE Computer Society, In Proceedings of Third International Conference on Advanced Computing & Communication Technologies, ACCT* 107114.

[13] Ryu, Seungwan. 2002. "Active Queue Management (AQM) based Internet Congestion ControlU." *University at Buffalo.* 1 Oktober. http://www.cse.buffalo.edu/~qiao/cse620/fall04/AQM-Fall04.pdf.

[14] Sally, F., Van, J. 1993. "Random Early Detection Gateways for Congestion Avoidance." *Lawrence Berkeley Laboratory.* Agustus. http://www.icir.org/floyd/papers/early.twocolumn.pdf.

[15] Sharma, Tanvi. 2014. "Controlling Queue Delay (CoDel) to counter the Bufferbloat Problem in Internet." *INPRESSCO International Journal of Current Engineering and Technology.* Juni 5. http://inpressco.com/wp-content/uploads/2014/07/Paper1992210-2215.pdf.

[16] Speedy. 2014. *Sepintas Teknologi ADSL.* November 1. http://opensource.telkomspeedy.com/wiki/index.php/Sepintas_Teknologi_ADSL.

[17] Taht, Dave. 2012. "RFC: Realtime Response Under Load (rrul) test specification." *GMANE.* September 6. http://article.gmane.org/gmane.network.routing.bufferbloat/940/.

[18] Tannenbaum, A.S. 2011. "Computer Network 5th Edition." 393. New Jersey: Prentice Hall, Inc.