MEKANISME PENJADWALAN DAN OPTIMALISASI MULTI-ALGORITMA LOAD BALANCING DAN ANYCAST PADA CONTENT DELIVERY NETWORK (CDN)

TESIS

Oleh:
Hillman Akhyar Damanik
1611601053

PROGRAM STUDI MAGISTER ILMU KOMPUTER (MKOM)
PROGRAM PASCASARJANA
UNIVERSITAS BUDI LUHUR
JAKARTA
2018

MEKANISME PENJADWALAN DAN OPTIMALISASI MULTI-ALGORITMA LOAD BALANCING DAN ANYCAST PADA CONTENT DELIVERY NETWORK (CDN)

TESIS

Diajukan untuk memenuhi salah satu persyaratan
memperoleh gelar Magister Ilmu Komputer (M. Kom)

Oleh:
Hillman Akhyar Damanik
1611601053

PROGRAM STUDI MAGISTER ILMU KOMPUTER (MKOM)
PROGRAM PASCASARJANA
UNIVERSITAS BUDI LUHUR
JAKARTA

2018

PROGRAM STUDI MAGISTER ILMU KOMPUTER
PROGRAM PASCASARJANA
UNIVERSITAS BUDI LUHUR

LEMBAR PERNYATAAN

Saya yang bertanda tangan di bawah ini,

Nama Mahasiswa : Hillman Akhyar Damanik
Nomor Induk Mahasiswa : 1611601053
Konsentrasi : Rekayasa Komputasi Terapan
Program Studi : Strata 2
Fakultas/Program : Ilmu Komputer.

Menyatakan bahwa tesis yang berjudul:

Mekanisme penjadwalan dan Optimalisasi Multi-Algoritma load Balancing dan Anycast pada Content Delivery Network (CDN)

1. Merupakan hasil karya tulis ilmiah sendiri dan bukan merupakan karya yang pernah diajukan untuk memperoleh gelar akademik oleh pihak lain.
2. Saya izinkan untuk dikelola oleh Universitas Budi Luhur sesuai dengan norma hukum dan etika yang berlaku.

Pernyataan ini saya buat dengan penuh tanggung jawab dan saya bersedia menerima konsekuensi apapun sesuai aturan apabila di kemudian pernyataan ini tidak benar.

Jakarta, 2 Agustus 2018

Hillman Akhyar Damanik

PROGRAM STUDI MAGISTER ILMU KOMPUTER
PROGRAM PASCASARJANA
UNIVERSITAS BUDI LUHUR

LEMBAR PENGESAHAN

Nama Mahasiswa	:	Hillman Akhyar Damanik
Nomor Induk Mahasiswa	:	1611601053
Konsentrasi	:	Rekayasa Komputasi Terapan
Judul Tesis	:	Mekanisme Penjadwalan dan Optimalisasi Multi-Algoritma *Load Balancing* dan *Anycast* Pada *Content Delivery Network* (CDN)

Jakarta, 2 Agustus 2018

Tim Penguji: Tanda Tangan:

Ketua,
Dr. Krisna Adiyarta M., S.Kom, M. Sc .....

Anggota,
Dr. Ir. Jan Everhard Riwurohi, M.T .....

Pembimbing,
Dr. Anton Satria Prabuwono, S.T, S.Si, M.M .....

Ketua Program Studi

Dr. M. Syafrullah, M.Kom, M.Sc

ABSTRAK

Content Delivery Networks (CDNs) telah berevolusi untuk mengatasi keterbatasan internet yang melekat dalam hal *quality of service* (QoS) saat mengakses konten web. CDN mereplikasi konten dari *server* asal ke *server cache* yang tersebar di seluruh dunia, untuk menyampaikan konten kepada pengguna akhir secara handal dan tepat waktu. Distribusi konten di internet menggabungkan pengembangan teknologi komputasi *high-end* dengan infrastruktur jaringan berkinerja tinggi dan teknik pengelolaan replika terdistribusi. Tujuan dari penelitian ini melakukan beberapa kriteria parameter optimalisasi dan peninjauan mekanisme serta *load balancing* pada *Content Delivery Network* (CDN) dengan membandingkan, menentukan dan mengevaluasi teknik penjadwalan algoritma *load balancing* yaitu Algoritma *Round Robin* (ARR), Algoritma *Weighted Round Robin* (AWRR) dan *Algoritma Least Connection* (ALC) pada sumber daya *Content Delivery Networks* (CDNs), yang diharapkan dapat menghasilkan beban *traffic* yang seimbang. Dan teknik *anycast geolocation* dalam memetakan *node server replica* pada *request node host* dalam kecepatan pengiriman paket dari segi *quality of service* (QoS). Berdasarkan hasil pengujian simulasi CDNs, dengan metode algoritma penjadwalan dan teknik *geolocation* diperoleh peningkatan kinerja dan perbedaan dari parameter nilai *Quality of Service* (QoS). Peningkatan persentase pengujian CDN menghasilkan *average throughput* meningkat menjadi, yaitu ALC: 43%, ARR: 28%, AWRR: 28% dan teknik *geolocation*: 37%. Nilai *average packet* loss dari 44.29% menurun menjadi, ALC: 41.36%, ARR: 30.53%, AWRR: 30.11% dan teknik *geolocation*: 40.10%. nilai *average delay* dari 31% menurun menjadi 76%, ARR: 71%, AWRR: 75% dan teknik *geolocation*: 77%.

Kata Kunci: *Content delivery network (CDN), load balancing, least connection, round robin, weighted round robin, geo-dns, quality of service (*Q*oS, application layer anycast, anycast geo-dns.*

ABSTRACT

Content Delivery Networks (CDNs) have evolved to overcome the limitations of the internet inherent in quality of service (QoS) when accessing web content. CDN replicates content from the origin server to the cache server that is spread all over the world, to deliver content to end users reliably and on time. Content distribution on the internet combines the development of high-end computing technology with high-performance network infrastructure and distributed replica management techniques. The purpose of this study is to perform several parameters of mechanism optimization and review mechanism as well as load balancing on the Content Delivery Network (CDN) by comparing, determining and evaluating load balancing algorithm scheduling techniques, namely Round Robin (ARR) Algorithm, Weighted Round Robin Algorithm and Algorithm Least Connection (ALC) on Content Delivery Networks (CDNs) resources, which are expected to generate balanced traffic loads. And anycast geolocation technique in mapping replica server nodes on host request nodes in the speed of package delivery in terms of quality of service (QoS). Based on the results of testing the simulation of CDNs, the scheduling algorithm and geolocation techniques obtained performance improvements and differences in the value of Quality of Service (QoS) parameters. The increase in the percentage of CDN testing resulted in an average throughput increase to, namely ALC: 43%, ARR: 28%, AWRR: 28% and geolocation technique: 37%. Average packet loss value from 44.29% decreased to, ALC: 41.36%, ARR: 30.53%, AWRR: 30.11% and geolocation technique: 40.10%. the average delay value from 31% decreased to 76%, ARR: 71%, AWRR: 75% and geolocation technique: 77%.

Keyword: *content delivery network (CDN), load balancing, least connection, round robin, weighted round robin, geo-dns, quality of service (*QoS*, application layer anycast, anycast geo-dns.*

KATA PENGANTAR

Bismillaahirrahmaanirrahiim.
Assalamu'alaikum, Wr. Wb.

Tiada sepatah katapun yang patut penulis ucapkan kecuali ucapan tahmid kehadirat Allah SWT.karena berkat rahmat dan ridhaNya penulis bisa menyelesaikan proposal penelitian tesis yang berjudul mekanisme penjadwalan dan optimalisasi multi-algoritma *load balancing* dan *anycast* DNS pada *Content Delivery Network* (CDN). Tujuan dari penulisan proposal penelitian tesis ini adalah sebagai salah satu syarat untuk menyusun tesis pada Program Studi Magister Ilmu Komputer, Universitas Budi Luhur Jakarta.

Dalam penyusunan tesis ini penulis mendapatkan banyak bimbingan dan bantuan dari berbagai pihak.rasa dan ucapan terima kasih penulis persembahkan kepada semua pihak yang telah membantu penulis dalam menyusun proposal penelitian tesis ini:

1. Ayahanda dan Ibunda yang telah memberikan kasih sayang, nasehat, dukungan dan semangat serta pengorbanan dan do'a yang tiada pernah berhenti mengalir.
2. Kakanda Haza Rini Damanik dan Adikanda Luway Nadhar Damanik, Ibnu Qushay Damanik, Udhay Aslam Damanik, Halil Udwan Damanik, Andhini Damanik, Ahmad Alhad Damanik dan Afif Baihaqy yang telah memberikan nasehat, dukungan dan semangat dan do'a yang tiada pernah berhenti mengalir.
3. Bapak Prof. Dr. sc. agr. Ir. Didik Sulistyanto selaku Rektor Universitas Budi Luhur.
4. Bapak Prof Dr. Suhartono, MBA, M.A. selaku Direktur Program Pascasarjana Universitas Budi Luhur.
5. Bapak Dr. M. Syafrullah, M.Kom, M.Sc selaku Ketua Program Studi Magister Ilmu Komputer Universitas Budi Luhur.

6. Bapak Prof. Anton Satria Prabuwono, Ph.D. selaku pembimbing tesis yang banyak memberikan bimbingan dan arahan.
7. Seluruh Dosen Program Studi Magister Ilmu Komputer Universitas Budi Luhur yang banyak memberikan ilmu selama perkuliahan.
8. Rekan-rekan mahasiswa MKOM Universitas Budi Luhur, terima kasih atas kebersamaan, kerja keras dan dukungan semangatnya.

Penulis menyadari masih banyak kekurangan dari proposal penelitian tesis ini, oleh karena itu penulis mengharapkan kritik dan saran yang bersifat membangun demi kesempurnaan penelitian tesis nantinya. Semoga proposal penelitian tesis ini masih dapat memberikan manfaat dari keterbatasannya.

Alhamdulillaahirabbil'aalamiin.
Wassalamu'alaikum Wr. Wb

Jakarta, Agustus 2018

Hillman Akhyar Damanik

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB I
PENDAHULUAN

1.1 Latar Belakang

Tujuan dari *Content Delivery Networks* (CDNs) adalah untuk menyajikan konten kepada pengguna akhir dengan kinerja tinggi. Untuk melakukan itu, CDNs mengukur *latency* khususnya QOS pada jalur *server* ke pengguna dan kemudian memilih *server* terbaik yang tersedia untuk setiap pengguna. Untuk CDNs yang besar, jalur pemantauan dari ribuan *server* ke jutaan pengguna merupakan tugas yang menantang karena ukurannya (Gursun, 2017). *Content delivery networks* (CDNs) adalah infrastruktur terdistribusi besar dari *server* replika yang ditempatkan di lokasi strategis (Pallis and Vakali 2006), (Pathan *et al*. 2008) Dengan mereplikasi konten *server* asal di *server* replika, konten dikirim ke pengguna akhir dengan nilai *quality of service* yang berkurang.

Dengan meningkatnya perluasan penggunaan Internet dan aplikasi volume tinggi, beberapa situs web populer sekarang menghasilkan trafik Internet yang sangat tinggi. Cisco memperkirakan bahwa, secara global, 71% dari semua trafik Internet akan melalui CDN pada tahun 2021, naik dari angka 52% pada tahun 2016 (Stocker *et al*. 2017). Dalam skenario ini, menawarkan *Quality of Service* (QoS) yang baik kepada pengguna dengan biaya rendah akan menjadi masalah yang menantang bagi situs web yang memotivasi tren peningkatan menuju CDN (Anjum dan Karamshuk 2017), (Cisco: *Forecast and Methodology*, *Online* 2018) Fitur utama CDN adalah penyebaran *server* replika dan *cache* konten populer lebih dekat ke pengguna untuk akses konten yang lebih cepat. Ini tidak hanya mengurangi latensi permintaan tetapi juga memungkinkan untuk menyeimbangkan beban antara *server* konten (Dehghan *et al*. 2017). Penempatan *server* replika yang tepat dapat secara signifikan mengurangi latensi transmisi konten yang diminta dan juga akan mengurangi konsumsi *bandwidth*-nya. Selain itu, mengingat bahwa *server* memiliki kapasitas penyimpanan dan pemrosesan yang terbatas, diperlukan strategi yang sesuai untuk memilih item konten yang akan di-*cache* di *server* replika karena akan mempengaruhi kinerja CDN. Dengan

demikian, efisiensi keseluruhan yang tinggi dicapai untuk CDN ketika ia mampu mengirimkan konten dengan kinerja tinggi (dikuantifikasi sebagai latensi yang berkurang atau batas yang ketat pada QoS) dan biaya rendah (Dehghan *et al*. 2017).

CDNs telah diterapkan untuk meningkatkan kualitas layanan (QoS) dan menyediakan *load balancing* di jaringan penyedia konten dengan beban lalu lintas tinggi. Dalam CDN, konten yang berasal dari penyedia konten direplikasi melalui *server* pengganti lokal. yang ditempatkan dengan benar dan permintaan pengguna dialihkan ke *server* terbaik, berdasarkan indeks kualitas jaringan. CDNs menggunakan jaringan *cache* dan *server* terdistribusi untuk mengurangi *delay, jitter, packet loss server* dan kemacetan jaringan dan faktor lain yang mempengaruhi *quality of experience* (QoE) pengguna (Haghighi *et al*. 2018).

Permintaan pengguna, penundaan jalur, beban *server* dan biaya *bandwidth* bervariasi secara dinamis berkenaan dengan waktu dan lokasi geografis. Untuk mengoptimalkan kinerja layanan *request* dengan biaya yang wajar, manajemen lalu lintas di CDN merupakan masalah yang penting namun sangat menantang (Fan *et al*. 2018). Hal ini dapat dibagi menjadi dua sub-masalah *load balancing*. Dengan meningkatnya beban pada *edge server*, diperlukan peningkatan sumber daya *server*. Varian lain adalah untuk meningkatkan jumlah *server* dan mendistribusikan beban antar masing-masing *server*. Teknologi *load balancing* adalah salah satu elemen utama untuk CDNs. dapat memastikan bahwa permintaan pengguna mengarah ke *server edge* terdekat dengan beban minimum di jaringan, sehingga konten jaringan didistribusikan secara efisien (2) *application-layer anycast* juga disebut L7 *anycast,* sangat bergantung pada *redirection* DNS dan IP *unicast*. Prosedur pemilihan *server* digambarkan pada Gambar 2.8. Seperti dijelaskan di (Calder *et al*. 2015) skema ini bekerja sebagai berikut. Pertama, pengguna meminta sumber daya, seperti *video* atau laman web HTML, melalui URL-nya. Agen pengguna menyimpulkan dari URL ini nama *hos*t, mungkin milik jaringan pengiriman konten tertentu. Dalam hal ini, peran utama kemudian dimainkan oleh *resolver* DNS lokal klien dan CDNs. *resolver* DNS menerima nama *host* yang ingin dihubungi pengguna, maka meneruskan

nama *host* ini ke CDNs yang bersangkutan. Selanjutnya, CDNs memilih simpul yang seharusnya melayani klien. Hal ini dapat dicapai dengan beberapa cara. CDNs hanya memilih *node* yang secara geografis paling dekat dengan klien. Pendekatan lain adalah memilih *node* CDNs terdekat dalam hal *latency* berkenaan dengan klien, atau yang paling dekat dalam hal jumlah *hop*. Untuk tujuan ini, *server* CDNs secara berkala meluncurkan pengukuran *ping* atau *traceroute* terhadap akses *Internet Service Provider* (ISP), dan melaporkan hasilnya ke *resolver* DNS dari CDNs (Chen *et al*. 2015).

Content Delivery Network (CDNs) (Pallis *et al*. 2006; Vakali *et al*. 2003 dan Peng, 2003) memberikan layanan yang meningkatkan kinerja jaringan dengan memaksimalkan *bandwidth*, meningkatkan aksesibilitas dan menjaga kestabilan melalui replikasi konten. CDNs menawarkan aplikasi dan layanan yang cepat dan andal dengan mendistribusikan konten ke *server cache* atau *edge* yang terletak dekat dengan pengguna (Pallis *et al*. 2003). CDN memiliki beberapa kombinasi infrastruktur penyampaian konten, *request-routing*, dan *system* distribusi. Infrastruktur pengiriman konten terdiri dari satu set *server edge* yang mengirimkan salinan konten ke pengguna akhir. Infrastruktur *request-routing* bertanggung jawab untuk mengarahkan permintaan *client* ke *server edge* yang sesuai. Ini juga berinteraksi dengan infrastruktur distribusi agar selalu *up-to-date* dengan tampilan konten yang tersimpan di CDN *cache*. Infrastruktur distribusi memindahkan konten dari *server* asal ke *server* tepi CDN dan memastikan konsistensi konten di *cache*.

Teknologi *load balancing* yang ada dalam jaringan distribusi konten (CDN) hanya menekankan distribusi beban di antara *server*, dan itu tidak membuat yang terbaik dari informasi topologi jaringan dan informasi historis dari *file* yang diakses. Teknologi *load balancing* di CDN terutama menekankan kemerataan beban pada *server*, dan mengabaikan aplikasi efisien dari informasi topologi jaringan dan riwayat akses *file*, yang mengarah ke waktu respon yang lama untuk permintaan pengguna yang menyebabkan kegagalan permintaan. Jika banyak permintaan pengguna terjadi pada saat yang sama, kinerja *server* pusat akan turun

dengan cepat, dan akhirnya memengaruhi jalannya seluruh jaringan (Bai *et al*. 2009)

Xiaofeng Zhang dan Baoqun Yin, mengusulkan sistem jaringan CDN-P2P di mana *server heterogen* beroperasi di bawah pembagian prosesor. Model yang digunakan adalah M/M/1/ N-PS, sistem jaringan CDN-P2P dengan mengadopsi strategi pengalihan *server* (Xiaofeng *et al*. 2017).

Samuel Micka dan Utkarsh Goel, mengusulkan DNS-Proxy (dp), proses sisi klien yang berbagi fungsi *load-balancing* dengan CDN dengan memilih dari *server* CDN yang diselesaikan berdasarkan kinerja jaringan dengan menggunakan 5 penyedia CDNs. menghasilkan respon *time* yg lebih efisien pada pengguna (Micka *et al*. 2015).

Qianjun Shuai, melakukan simulasi menggunakan model *Control Law Balancing* (CLB). mengusulkan algoritma yang memperhitungkan keseimbangan antara *load balancing* dan kedekatan pengarahan (*edge server*) dari sisi pengguna (Shuai *et al*. 2017).

Xiaolong Yang, mengusulkan dan melakukan simulasi dengan menggunakan algortima *user interest aware content replica optimized placement algorithm* (UIARP). Algoritma UIARP dapat mencapai kecocokan antara menempatkan replika dan permintaan konten pengguna melalui meminimalkan waktu respon rata-rata. Hasil simulasi berupa empat aspek termasuk waktu respon rata-rata, tingkat pencocokan respon permintaan, *load balancing* dan tingkat pemanfaatan replika yang berdekatan, yang memverifikasi keefektifan dari algoritma (Yang *et al*. 2014).

Dalam penelitian ini, penulis menerapkan solusi yang efektif untuk penyeimbangan beban dalam *Content Delivery Network* (CDN) dengan menggunakan metode penjadwalan algoritma penjadwalan *load balancing* yaitu *Least Connection, Round Robin* dan *Weighted Round Robin*. Penerapan perbandingan teknik *anycast geolocation* dalam memetakan *server*, pada *request host* dalam kecepatan pengiriman paket dari segi *quality of service* (QoS) pada wilayahnya. Dengan melakukan analisis pada parameter tersebut, sehingga dapat

mengeksplorasi keuntungan, kelemahan, peluang masa depan dalam pengaplikasian CDN dari nilai *quality of service* (QoS).

1.2 Masalah Penelitian

Permasalahan pada penelitian ini dijelakan secara detail pada 3 bagian pokok, yaitu identifikasi masalah, batasan masalah dan rumusan masalah. Penjelasan pada 3 bagian pokok tersebut dijabarkan sebagai berikut:

1.2.1 Identifikasi Masalah

Pada tahap identifikasi masalah pada penelitian ini, penulis melakukan beberapa tahap untuk mengidentifikasi permasalahan dalam penelitian, yaitu:

1. Lalu lintas teknologi komunikasi data yang yang semakin komplek, terutama saat ini pada teknologi *Content Delivery Network* (CDN), yang hanya menekankan distribusi beban di antara *server* replika. Sehingga sebuah *single link* pada *server-server* replika yang tersebar akan mengalami beban yang sangat tinggi ketika *realtime traffic* melakukan permintaan pada sebuah infrastruktur *server* replika.
2. Aspek penurunan nilai parameter *quality of service* (QoS), termasuk didalamnya *paket loss, delay*, *throughput,* waktu respon dan *overload* beban *traffic* pada saat beban trafik memadati suatu koneksi pada tujuan sebuah *server* replika CDNs. Sehingga layanan terdegradasi dengan penundaan akses yang tinggi dan *respon time* yang lama menyebabkan gangguan pada pengguna.
3. Saat ini CDNs hanya menyediakan layanan *best-effort* pada konektivitas server replika pada Infrastruktur *server edge*. Namun di masa depan layanan QoS (*Quality of Service*) akan perlu menarik lebih banyak pengguna. Umumnya, CDNs menggunakan mekanisme pemilihan *server* yang memilih *server* dalam jaringan untuk memenuhi permintaan pengguna. Metode yang paling umum adalah yang berikutnya untuk mengarahkan permintaan pengguna ke *server* terdekat. Menggunakan

mekanisme layanan nama domain (DNS) untuk mengukur beban *server* dan memilih *server* terdekat dengan wilayahnya.

1.2.2 Batasan Masalah

Adapun ruang lingkup yang dijadikan penulis sebagai pembatasan masalah adalah sebagai berikut:

1. Penelitian difokuskan pada model:
   - Teknik penjadwalan algoritma *load balancing: Least Connection* (LC), *Round Robin* (RR) dan *Weighted Round Robin (*WRR*).*
   - *Distributed location server* dengan Geo-DNS CDN pada *Content Delivery Networks* (CDNs).
2. Tahap analisis dan perancangan penelitian akan menghasilkan konsep *load balancing* dari metode yang akan digunakan.
3. Aspek yang dihasilkan berupa parameter nilai *Quality of Service* (QoS) termasuk didalamnya *paket loss, delay* dan *throughput* pada saat beban trafik memadati suatu koneksi pada tujuan sebuah *server*.
4. *Simulator* yang digunakan adalah *Network Simulator* 2.35 (NS2) dan perancangan topologi terdiri dari 3 *server* dan 20 *Host*. Dan *packet* multimedia yang dikirimkan hanya pada proses pengiriman (*upload*).

1.2.3 Rumusan Masalah

Perumusan masalah yang akan dilakukan kajian pada penelitian ini adalah:

1. Bagaimana menentukan algoritma *load balancing* di dalam jaringan *Content Delivery Network* (CDNs) yang diterapkan. Dengan mendistribusikan beban *traffic* pada permintaan pengguna, pada jalur perutean transmisi (*client node*) ke tujuan (*node server replica*). Dengan tujuan meminimalisasi nilai *Quality of Service* (QoS) berdasarkan parameter-parameter *throughput*, *packet loss* dan *end-to-end delay*.
2. Bagaimana mengarahkan dan mencari jalur terbaik pada *request* sebuah *host* pada *node server replica* terdekatnya, dengan teknik *geolocation domain* di dalam jaringan *Content Delivery Network* (CDNs) yang

diterapkan. Dengan meminimalisasi jarak antara pengguna dengan *server* penyedia layanan. Ditinjau dari parameter-parameter *throughput*, *packet loss* dan *end-to-end delay*. Sehingga penggunaan *bandwidth* pada indeks *throughput* dapat dimaksimalkan dengan kapasitas yang tersedia. Kemudian nilai indeks *packet loss* dan *delay* dapat diminimalisasi.

3. Bagaimana perbandingan sumber daya *Non-Content Delivery Network* (Non-CDNs) dengan *Content Delivery Network* (CDNs) dengan metode penerapan *load balancing* dan *anycast geolocation* domain.

1.3 Tujuan dan Manfaat Penelitian

1.3.1 Tujuan Penelitian

Berdasarkan Rumusan masalah yang sudah diuraikan, maka tujuan penelitian adalah:

1. Menentukan metode *load balancing* pada karakteristik dan penerapan sumber daya *Content Delivery Networks* (CDNs) dengan metode yang akan diterapkan.
2. Membuat teknik penjadwalan *load balancing* dan metode *anycast geolocation* dengan menggunakan algoritma *load balancing Least Connection* (LC), *Round Robin* (RR) dan *Weighted Round Robin* (WRR). Sehingga penurunan performansi pada parameter nilai *quality of service* (QoS) dapat diminimalisasikan dengan menggunakan metode algoritma penjadwalan. dan diharapkan dapat menghasilkan beban *traffic* yang seimbang, sehingga penurunan nilai performansi dari *throughput, packet loss,* dan *delay* dapat teratasi.
3. Mengevaluasi kinerja metode penjadwalan algoritma *Least Connection* (LC), *Round Robin* (RR) *dan Weighted Round Robin* (WRR) dan metode *anycast geolocation*, pada *Content Delivery Networks* (CDNs).

1.3.2 Manfaat Penelitian

Manfaat dari penelitian ini diharapkan dapat memberikan kontribusi sebagai berikut:

1. Mengetahui performansi metode penjadwalan *load balancing Least Connection* (LC), *Round Robin* (RR) dan *Weighted Round Robin* (WRR).
2. Mengetahui kelebihan dan kekurangan metode penjadwalan algoritma, *Least Connection* (LC), *Round Robin* (RR) dan *Weighted Round Robin* (WRR) pada karakteristik *Content Delivery Network* (CDN).
3. Mengetahui atau mendapatkan nilai kinerja performansi dan pemetaan dari sumber daya *Non-Content Delivery Network* (CDN) dengan *Content Delivery Network* (CDNs) dengan metode penerapan *anycast* Geo-DNS.

1.4 Tata Urut Penulisan

Naskah pada penelitian ini disusun dengan tata urut penulisan sebagai berikut:

BAB I PENDAHULUAN

Membahas latar belakang masalah penelitian, proses penjadwalan dan pendistribusian mekanisme penjadwalan dan optimalisasi multi-algoritma *load balancing* dan *anycast* DNS pada *Content Delivery Network* yang diteliti.identifikasi masalah,batasan masalah, rumusan masalah, tujuan dan manfaat penelitian, sistematika penulisan, dan daftar istilah yang digunakan dalam penulisan.

BAB II LANDASAN TEORI DAN KERANGKA KONSEP

Membahas tinjauan pustaka yang berkaitan dengan topik bahasan mekanisme penjadwalan dan optimalisasi multi-algoritma *load balancing* dan *anycast* DNS pada *Content Delivery Network* Tinjauan studi, tinjauan obyek penelitian, kerangka konsep dan hipotesis.

BAB III METODOLOGI DAN RANCANGAN PENELITIAN

Membahas mengenai mekanisme penjadwalan algoritma *load balancing* dan pendistribusian *anycast* Geo-DNS.Tahapan menguraikan mengenai metode penelitian, metode pemilihan sampel, metode pengumpulan data, sumber dan

instrument data, teknik analisis, rancangan dan pengujian sistem, langkah-langkah penelitian dan jadwal penelitian.

BAB IV PENUTUP

Pada tahap ini penulis membahas kesimpulan yang diperoleh berdasarkan hasil pencarian masalah penelitian, studi pustaka, tinjauan penelitian, tinjauan obyek penelitian dan metodologi penelitian.

1.5 Daftar Pengertian

Beberapa pengertian istilah yang dipergunakan di dalam proposal penelitian ini sebagai definisi kamus maupun definisi operasional, sebagai berikut:

Tabel I-1.Daftar Pengertian Istilah

CDN	*Content Delivery Network*
LC	*Least Connection*
RR	*Round Robin*
WRR	*Weighted Round Robin*
Application-layer Anycast	Pemetaan pengguna internet berdasarkan letak geografis
QoS	Seperangkat pengukuran kinerja (*Delay, packet loss, availability, bandwidth utilization (throughput)*
Delay	Waktu yang dibutuhkan oleh sebuah paket data terhitung dari saat pengiriman oleh *transmitter* sampai saat diterima oleh *receiver*
Throughput	Jumlah bit yang diterima dengan sukses perdetik melalui sebuah sistem atau media komunikasi (kemampuan sebenarnya suatu jaringan dalam melakukan pengiriman data)
Jitter	Variasi- variasi dalam panjang antrian, dalam waktu pengolahan data, dalam waktu yang dibutuhkan untuk retransmisi data (karena jalur yang digunakan juga berbeda), dan juga dalam waktu penghimpunan ulang paket-paket di akhir perjalanan.
Packet Loss	Banyaknya paket yang hilang selama proses transmisi ke tujuan
Traffic	Jumlah data yang bergerak melintasi jaringan pada suatu titik waktu tertentu
Anycast	Teknologi lapisan jaringan untuk pemilihan *server* yang transparan dan mengarahkan ulang. Dalam pendekatan ini, alamat IP yang sama diberikan ke beberapa *server* pengganti yang berada secara distributif.

DNS	*Domain Name System* (DNS) adalah sistem penamaan *key* yang digunakan di Internet
Origin servers	*Server* dari penyedia konten, berisi informasi yang akan didistribusikan atau diakses oleh klien.
Surrogates	*Server* pengganti adalah *server* replika dari *server* asal, bertindak sebagai *server proxy* atau *cache* dengan kemampuan untuk menyimpan dan mengirimkan konten.
Client	Klien adalah individu atau pengguna yang meminta dan mendownload sepotong konten yang tersimpan di suatu tempat di CDNs.
Access network	*Client* mengakses layanan yang disediakan oleh CDNs melalui berbagai *access networks*.
Distribution network	Menghubungkan *server* asal dengan *Multimedia Streaming Servers* (MSSs) untuk menghadirkan objek media dalam media streaming CDN.
Content Manager	Mengendalikan objek media yang tersimpan di setiap pengganti, memberikan informasi ini kepada *redirector* untuk mendapatkan setiap *client* dilayani oleh pengganti yang paling cocok.
DNS server	Server DNS menghadirkan dan memproses semua permintaan DNS terkait domain yang dikelola oleh sistem CDN. Ketika klien mengakses CDN, langkah pertama terdiri dari menyediakan alamat IP yang sesuai dengan pengganti CDN terbaik.
Redirector	Modul ini memberikan kecerdasan pada sistem, karena memperkirakan *server* pengganti yang paling memadai untuk setiap permintaan dan klien yang berbeda.
edge servers	*Server edge* CDN menentukan peningkatan waktu untuk menemukan replika
ISP	*Internet Service Provider*
Caching	Inti dari layanan pengiriman konten (CDN)
IPTD	IP *Packet Transfer Delay*
IPDV	IP *Packet Delay Variation*
IPLR	*Packet loss Ratio*
NS2	*Network Simulator*
NAM	*Network Animator*
OTcL	*(Tool Command Language) used for specifying scenarios and events*.
Evalvid	Evaluasi Video

BAB II
LANDASAN TEORI DAN KERANGKA KONSEP

2.1 Tinjauan Pustaka

2.1.1 Content Delivery Network (CDN)

Dengan pesatnya perkembangan internet dan semakin banyaknya konten media, semakin jelas bahwa distribusi konten yang efektif dapat meningkatkan kinerja web (Manfredi *et al.* 2010). *Content Delivery Network* (CDN) (Vakali *et al*. 2003) telah terbukti menjadi solusi yang handal untuk meningkatkan ketersediaan konten dalam konteks menghindari perubahan infrastruktur jaringan yang ada. Ide inti dari CDN adalah mendistribusikan sumber konten dari pusat data ke *server* replika yang berada dekat dengan pengguna dan secara dinamis mengalihkan permintaan klien berdasarkan mekanisme yang sesuai, yang disebut sebagai *load balancing* (Binder *et al.* 2013). Dengan demikian, dimungkinkan untuk mengurangi *latency*, meningkatkan kualitas layanan, dan menurunkan tekanan pada *network backbone* secara bersamaan. Selain itu, dengan memonitor parameter jaringan seperti *load, throughput*, dan status kesehatan setiap *server*, CDN dapat menghapus *node* yang cacat pada saat jaringan berada dalam keadaan tak terduga, yang membuat keseluruhan jaringan menjadi lebih baik dan stabil.

Content Delivery Networks (CDNs) muncul sebagai jaringan *overlay* di internet untuk memberikan dukungan yang lebih baik untuk memberikan konten komersial daripada yang tersedia dengan menggunakan layanan *transport* paket internet dasar yang terbaik. Karena volume, kompleksitas, dan heterogenitas lalu lintas Internet telah berkembang luas dari berbagai kompleksitas, demikian juga Penyedia Layanan Internet (ISP) dan CDNs yang menyediakan layanan yang digunakan untuk menyampaikan lalu lintas ini. Pentingnya CDNs dalam ekosistem Internet telah berkembang secara signifikan seiring berjalannya waktu. memperkirakan bahwa CDNs akan segera menangani lebih dari setengah dari keseluruhan masalah global di internet.

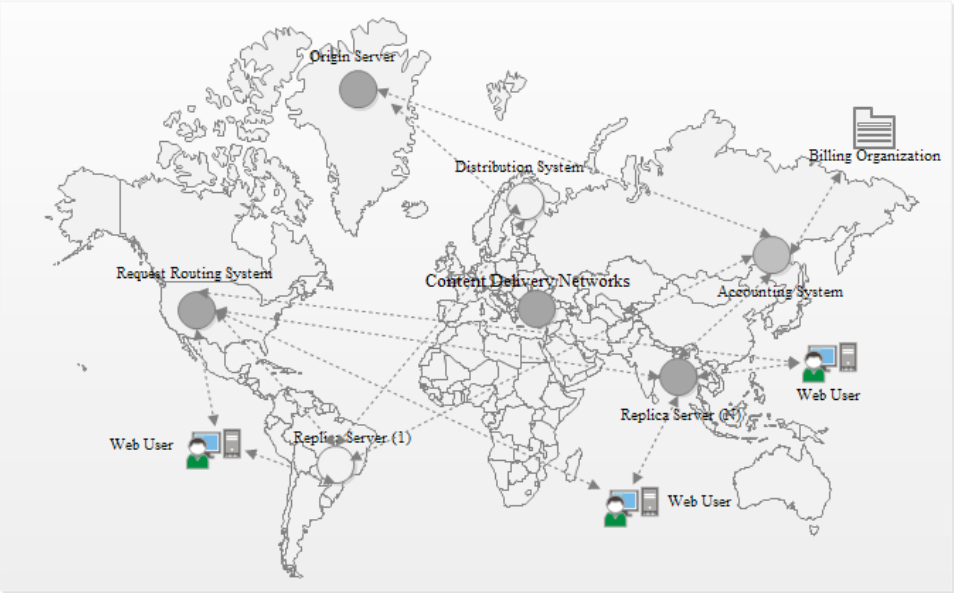
CDNs adalah kumpulan elemen jaringan yang diatur agar pengiriman konten lebih efektif ke pengguna akhir (Panchal *et al.* 2014). Kolaborasi antara komponen CDNs terdistribusi dapat terjadi di atas simpul di lingkungan homogen dan heterogen. CDNs dapat mengambil berbagai bentuk dan struktur. Mereka dapat dipusatkan, infrastruktur hirarkis di bawah kontrol administratif tertentu, atau sistem yang sepenuhnya terdesentralisasi. Ada juga berbagai bentuk *internetworking* dan kontrol berbagi antar entitas CDNs yang berbeda. Pertimbangan umum dalam merancang CDNs dapat ditemukan pada (Verma, 2002) fungsi khas CDNs yang meliputi:

- *Request* layanan pengiriman dan pengiriman ulang konten untuk mengarahkan permintaan ke *server* pengganti terdekat yang sesuai dengan menggunakan mekanisme menghindari kemacetan.
- Layanan distribusi untuk mereplikasi konten *cache* ke *server* pengganti yang akan didistribusikan menuju *server* asal.
- Layanan negosiasi konten untuk memenuhi kebutuhan spesifik setiap pengguna individual atau kelompok pengguna.
- Layanan manajemen untuk mengelola komponen jaringan dan memantau serta melaporkan penggunaan konten.

2.1.1.1 Topologi Arsitektur CDNs

Content Delivery Network (CDN) adalah jaringan yang didistribusikan secara geografis, yang berisi sejumlah *server* konten dan *router*. Sebagai aturan yang terdiri dari *main node* (asal), dan *node caching* (*edges*). CDN khas memiliki beberapa kluster *server* web yang direplikasi tersebar di seluruh dunia yang terletak di tepi jaringan tempat pengguna terhubung. CDN mendistribusikan konten ke sekumpulan *server* web untuk mengirimkan konten kepada pengguna akhir dengan cara yang dapat diandalkan. Saat pengguna mengirim permintaan ke *server* utama, permintaan ini akan diarahkan ke titik terdekat. Melalui *link*, jalur jaringan antara *server* dan pengguna, sehingga situs atau sumber lainnya jauh lebih cepat. Manfaat CDNs (Elgeldawy *et al*. 2016) (1) Meningkatkan kecepatan pengiriman konten pengguna dari seluruh dunia akan bisa mendapatkan konten

untuk rute jaringan optimal dalam waktu singkat dan dari titik terdekat. (2) Mengurangi beban pada *server* asal, semua pengguna konten statis akan men*download* dari server *caching*, dan dengan demikian beban pada *server* asal berkurang, secara signifikan informasi yang tersimpan hanya perlu tetap *up to date*.



Gambar II-1. Arsitektur Jaringan CDNs
(Sumber: Kyryk M, Pleskanka N, Pitsyk M. TCSET 2016)

Gambar II-1 menunjukkan arsitektur umum sistem CDN yang melibatkan empat komponen utama:

1. Komponen pengiriman konten yang terdiri dari *server* asal dan satu set *server* replika yang mengirimkan salinan konten ke pengguna akhir.
2. Komponen permintaan *routing* yang bertanggung jawab untuk mengarahkan permintaan klien ke *server edge* yang tepat dan untuk berinteraksi dengan komponen distribusi agar tetap mendapatkan informasi terkini tentang konten yang tersimpan di cache CDN.
3. Komponen distribusi yang memindahkan konten dari *server* asal ke *server edge* CDN dan memastikan konsistensi konten di *cache*.

4. Komponen *accounting* yang menyimpan log akses dan catatan klien penggunaan *server* CDN. Informasi ini digunakan untuk pelaporan lalu lintas dan penagihan berbasis penggunaan oleh penyedia konten itu sendiri atau oleh organisasi penagihan pihak ketiga.

Semua permintaan pengguna dikirim ke *server* terdekat atau *server* dengan beban terendah untuk mempercepat respons. Saat ini, CDNs hanya menyediakan layanan *best-effort*. Namun di masa depan layanan QoS (*Quality of Service*) akan perlu menarik lebih banyak pengguna. Umumnya, CDNs menggunakan mekanisme pemilihan *server* yang memilih *server* dalam jaringan untuk memenuhi permintaan pengguna. Metode yang paling umum adalah yang berikutnya untuk mengarahkan permintaan pengguna ke *server* terdekat menggunakan mekanisme layanan nama domain (DNS) untuk mengukur beban *server* dan memilih *server* dengan beban terendah. untuk memilih *server* yang menyediakan *delay* pengiriman terpendek berdasarkan *bandwidth* yang tersedia sepanjang rute ke *server*. maka perlu menggunakan sumber daya jaringan dan *server* seefisien mungkin (Elgeldawy *et al*. 2016). kondisi utama fungsi objektif harus dipastikan untuk mengarsipkan kualitas layanan CDN yang tepat.

2.1.2 Qos Parameter

QoS (*Quality of Service*) seperti IP *Packet Transfer Delay* (IPTD), IP *Packet Delay Variation* (IPDV) dan *packet loss Ratio* (IPLR). Internet baru-baru ini bekerja di bawah protokol TCP / IP (*Transmission Control Protocol / Internet Protocol*) dan didasarkan pada dua dasar utama, yaitu jaringan hanya menawarkan satu kelas layanan yang diberi nama *best effort service*, dan sumber daya jaringan terlalu banyak terbelakang untuk meminimalkan kerugian paket dan penundaan paket. Penyedia layanan internet bertujuan untuk menyediakan pengiriman paket kepada pengguna secepat mungkin, namun mereka jauh dari penjaminan, *quality of service* (QoS) yang diukur dengan nilai maksimum parameter yang diijinkan dari parameter seperti IPTD, IPDV dan IPLR (Mansour *et al*. 2001).

2.1.2.1 *Delay* (ms)

Berapa banyak waktu yang dihabiskan dalam jaringan tunggal, biasanya ditunjukkan dalam hitungan detik (*s*). Penundaan dalam jaringan dapat menyebabkan *packet drop* (Dobrian *et al*. 2011) tapi tidak memungkinkan untuk dapat menghindarinya. Setiap simpul sebuah sistem menambahkan penundaan. Pengolahan *delay, queuing delay*, *delay* transmisi dan *delay* propagasi adalah salah satu penundaan yang secara signifikan dapat mempengaruhi sistem.

Tabel II-1. Kategori *Delay*

Kategori Delay	Besar *Delay (ms)*	Indeks
Sangat Bagus	<150	4
Bagus	150 s/d 300	3
Sedang	300 s/d 450	2
Buruk	>450	1

(Sumber:*Telecommunications and Intranet Protocol Harmonization Over Networks* (TIPHON))

Dengan Persamaan sebagai berikut:

$$Delay = \frac{\text{jumlah } packet \text{ pengiriman data } (bit)}{jumlah\ packet\ (bit/s)}$$

2.1.2.2 *Throughput* (bits/s)

Mayoritas dari Internet saat ini adalah *throughput oriented*. *Throughput* seringkali lebih penting daripada penundaan.kecepatan *(rate) transfer* data efektif, yang diukur dalam *bps*. *Throughput* merupakan jumlah total kedatangan paket yang sukses yang diamati pada tujuan selama interval waktu tertentu dibagi oleh durasi interval waktu tersebut. Misalnya, waktu mulai *video*, yang sangat penting untuk pengalaman pengguna (Sanchi *et al*. 2013).

Tabel II-2. Kategori *Throughput*

Kategori *Throughput*	*Throughput %*	Indeks
Sangat Bagus	100	4
Bagus	75	3
Sedang	50	2
Buruk	>25	1

(Sumber: *Telecommunications and Intranet Protocol Harmonization Over Networks* (TIPHON))

Dengan Persamaan sebagai berikut:

$$Throughput\ \frac{\sum_i \quad \text{jumlah data dikirim } (bit)}{\sum_i \quad \text{waktu pengiriman data } (s)}$$

2.1.2.3 *Packet Loss*

Menghitung *packet loss* adalah dengan menghitung tingkat kerugian keseluruhan, yang sama dengan jumlah total lalu lintas yang hilang dibagi dengan jumlah total lalu lintas masukan selama periode waktu tertentu. Karena kita menggunakan ukuran paket konstan dalam simulasi kita, tingkat kerugian dapat dinyatakan sebagai (Shuai *et al*. 2017):

Tabel II-3. Kategori *Packet Loss*

Kategori *Packet Loss*	Besar *Packet Loss (%)*	Indeks
Sangat Bagus	0	4
Bagus	3	3
Sedang	15	2
Buruk	25	1

(Sumber: *Telecommunications and Intranet Protocol Harmonization Over Networks* (TIPHON))

$$Packet\ loss = \frac{\sum_i \quad packet\ data\ dikirm - paket\ data\ diterima}{\sum_i \quad paket\ data\ dikirim} x100\%$$

2.1.3 Algoritma *Load Balancing*

Load balancing memainkan peran penting dalam meningkatkan skalabilitas dan stabilitas di *Content Delivery Networks* (CDNs) untuk memenuhi meningkatnya permintaan *bandwidth* (Koerner and Kao, 2012.)

Load balancing adalah metodologi distribusi beban kerja berbagai sumber jaringan melalui jalur yang sesuai. Metodologi ini memungkinkan mencapai pemanfaatan sumber daya yang optimal, memaksimalkan *throughput* dan meminimalkan waktu transmisi (Rao *et al*. 2014) *client* mengirimkan permintaan ke *server*. Awalnya *server* akan mengecek antrian, jika batas antrian melebihi *server* akan mengalihkan kepada *client* lain dalam merespon permintaan. Setiap detik (*t*), *server* mengarahkan informasi statusnya ke tetangga dan pada saat bersamaan menunggu untuk informasi mereka bila permintaan baru tercapai di *server*, selanjutnya akan mengkonfirmasikan adanya tetangga dengan beban lebih rendah (Bai *et al*. 2009).

Pemodelan *load balancing* pada CDNs, dari *request client*, pemilihan *server edge* dan pada keseluruhan *server* secara dinamis, dan menggambarkan *load balancing* di CDN, pertama perlu membuat model jaringan. Asumsikan bahwa ada *server* jaringan *N* dalam jaringan, dan kapasitas *server* ini adalah *C1, C2, ..., Cn*, zona yang bertanggung jawab adalah *Z1, Z2, ..., Zn*, dan beban jaringan *W1, W2*, ... , *Wn* masing-masing. Ada *file M* yang didistribusikan di jaringan. Biaya akses *file-file* ini adalah *O1, O2*, ..., *Om* masing-masing, dan frekuensi akses yang sesuai adalah *f1, f2, ..., fm*. Pada *load balancing* ideal, beban jaringan yang didistribusikan ke masing-masing *server* harus berada dalam rasio langsung terhadap kapasitas pelayanan *server* itu sendiri. Oleh karena itu, *server G* harus memenuhi persamaan (1) (Bhagat dan Budhwant, 2016):

$$\sum_{j=1}^{\frac{WG}{N}} Wj = \sum_{j=1}^{\frac{CG}{N}} Cj$$

Definisi 1: $W_K = \sum_{x=\alpha 1}^{\alpha e} fxOx$ dimana W_K artinya total load *server K* di CDN, f_x berarti akses frekuensi *file x* yang tersimpan di *server*, O_x berarti biaya akses file x tersimpan di *server*, file yang tersimpan di *server* meliputi *α1, α2, ..., αe*.

2.1.3.1 Algoritma *Static Load Balancing*

Algoritma *static load balancing* sama-sama menghitung antara *node* dalam proses perhitungannya, terlepas dari beban sebenarnya dari *server* aktual dan kinerja penanganan yang berbeda di antara *server*. Algoritma statis memilih *server* tanpa bergantung pada informasi tentang status sistem pada saat keputusan. Algoritma statis tidak memerlukan mekanisme pengambilan data dalam sistem, yang berarti tidak ada *overhead* komunikasi yang diperkenalkan. Algoritma ini jelas merupakan solusi tercepat karena tidak mengadopsi proses seleksi (Song *et al*. 2015).

2.1.3.2 Algoritma *Dynamic Load Balancing*

Sedangkan algoritma *dynamic load balancing* mendasarkan keputusannya pada keadaan pada suatu titik waktu tertentu. Tapi algoritma *load balancing* dinamis memerlukan komunikasi konstan ke *node server*, untuk mengumpulkan informasi tentang *node server* secara *real-time* untuk *load balancing*, dan ini meningkatkan lalu lintas jaringan, menghabiskan banyak waktu (Pradhan *et al*. 2016).

Tabel II-4. Perbandingan Algoritma *Static* dan *Dynamic Load Balancing*

No.	*Static Load Balancing*	*Dynamic Load Balancing*
1	Pekerjaan di antara prosesor sebelum pelaksanaan algoritma	Mendistribusikan pekerjaan di antara prosesor selama pelaksanaan algoritma
2	Matrix-Matrix *Computation*	Pemrograman grafik paralel dan perhitungan elemen hingga adaptif
3	Mudah untuk mendesain dan mengimplementasikan	Algoritma yang membutuhkan *load balancing* dinamis agak lebih rumit
4	Perilaku algoritma *static load balancing* dapat diprediksi	Perilaku algoritma dinamis *load balancing* tidak dapat diprediksi
5	Algoritma *load balancing static* memiliki pemanfaatan sumber daya yang lebih rendah	Algoritma *load balancing* dinamis memiliki pemanfaatan sumber daya yang relatif lebih baik
6	Algoritma *load balancing static* kurang dapat diandalkan (*less reliable*)	Algoritma *load balancing* dinamis lebih *realible*
7	Algoritma *static load balancing* menyeimbangkan biaya *overhead* yang lebih rendah	Algoritma *load balancing* dinamis menimbulkan lebih banyak *overhead*

8	Algoritma *load balancing static* bebas dari *thrashing* prosesor	Algoritma *load balancing* yang dinamis menimbulkan *thrashing* prosesor yang besar
9	Tidak ada metode akurat untuk memperkirakan waktu eksekusi	Mudah untuk memperkirakan waktu eksekusi
10	Kurang efisien	Lebih Efisien
11	Penundaan komunikasi minimal	Lebih banyak penundaan komunikasi

(Sumber: Prajapati, R, &Rathod, D dan Khanna, S.*International Journal for Technological Research in Engineering* 2015)

2.1.3.3 *Round Robin (RR) Algorithm*

RRA adalah salah satu algoritma penjadwalan tertua, paling sederhana, paling bagus dan paling banyak digunakan, dirancang khusus untuk sistem *time-sharing*. Sebuah unit kecil waktu, yang disebut *slice* atau *quantum time* yang didefinisikan (Singh *et al*. 2010) Algoritma *Round Robin* (RRA) adalah algoritma statis populer yang diterapkan pada sistem *load balancing* CDN. Ini membagi trafik yang masuk antar *server* secara *round robin*.

Algoritma *Round-Robin* (RR) adalah salah satu algoritma *load balancing* paling sederhana yang digunakan.Menggunakan daftar melingkar dan *pointer* ke *server* yang dipilih terakhir untuk membuat keputusan pengiriman. *Round robin* mengirimkan permintaan pengguna ke *server* web dari *server* web pertama ke yang terakhir berdasarkan pesanan. Itu berarti jika S_i adalah simpul yang dipilih terakhir. permintaan baru ditugaskan ke S_{i+1}, di mana $i = (i+1)$ mod N dan N adalah jumlah *node server*.

Keuntungan dari algoritma ini adalah kesederhanaannya. Saat menggunakan algoritma ini, peneliti tidak perlu mencatat status semua *server* CDNs. Untuk menjamin pengiriman cepat sejumlah besar permintaan per detik. Jadi, algoritma *Round-Robin* adalah salah satu solusi tercepat untuk mencegah *request* menjadi *bottleneck* pada tujuan distribusi menuju *server* (Xu and Wang, 2015).

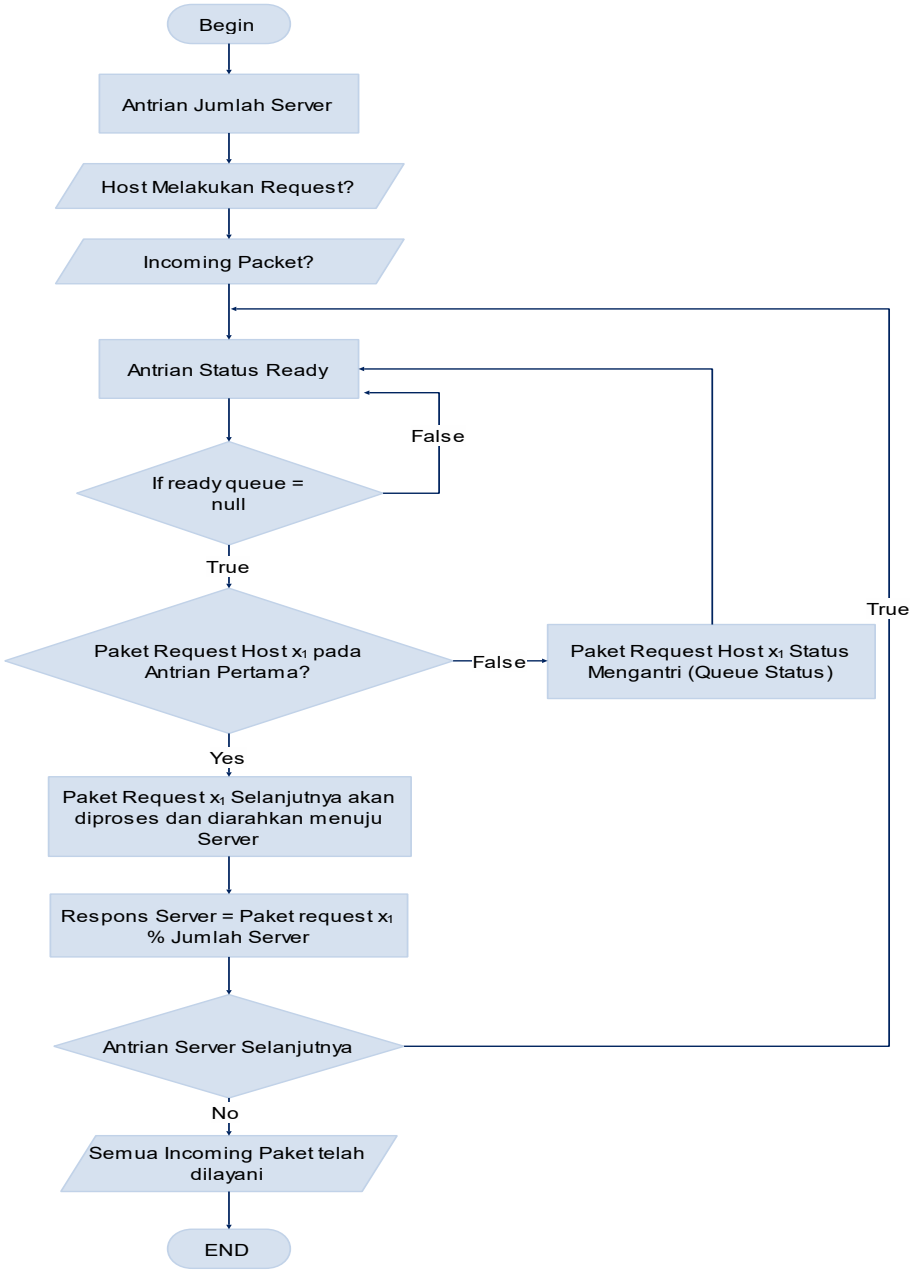
Fungsi dasar yang hanya meneruskan permintaan dengan memilih *server* tujuan melalui algoritma *round-robin*. Setelah *host-request* memilih *server*, selanjutnya mendapat jalur jaringan termudah saat status *ready queue* dari sumber

ke tujuan untuk memuat keseimbangan jaringan. Ciri utamanya adalah *server* secara konstan menerima jumlah permintaan yang sama, yang dapat berguna untuk layanan yang biaya permintaannya selalu sama (Trajano dan Fernandez, 2016].

Berikut representasi dari algoritma *load balancing round robin*. Berdasarkan *flowchart* pada gambar II.2, algoritma penerapan *load balancing* dengan menggunakan metode *round robin* adalah sebagai berikut:

1. Inisiasi antrian jumlah *server* yang akan digunakan.
2. Paket *host request* sebagai input yang akan didistribusikan dengan teknik *load balancing* berdasarkan metode penjadwalan *round robin*.
3. Paket *request* akan mendapatkan antrian status *ready* yang akan berada pada antrian.
4. Semua proses diurutkan dalam urutan yang meningkat yang berarti proses dengan waktu *burst* yang lebih kecil mendapatkan prioritas yang lebih tinggi dan proses dengan waktu *burst* yang lebih besar mendapatkan prioritas yang lebih rendah.
5. *While* (*ready queue! = NULL*).
6. *If*: *paket request* x_1 berada di antrian pertama, jika paket *request* x_1 berada di antrian pertama
7. *Then*: *host request* paket akan diproses dan diarahkan kepada *server*.
8. *Else*: Menentukan *server* yang akan melayani paket yang masuk (*incoming packet*) dengan melakukan modulo jumlah *server* terhadap *server* ke-*N*.
9. Memeriksa apakah dilanjutkan ke *server* selanjutnya. *If value=true*, maka ada paket *request* yang masih berada dalam antrian sehingga mengulangi langkah (3) (4) (5) (6) (7) dan (8). *If value=true*, maka semua paket *request* telah dilayani.
10. *End of while*
11. *End*

Berikut *flowchart* dari algoritma *load balancing round robin* dan penjabaran algoritma diatas:

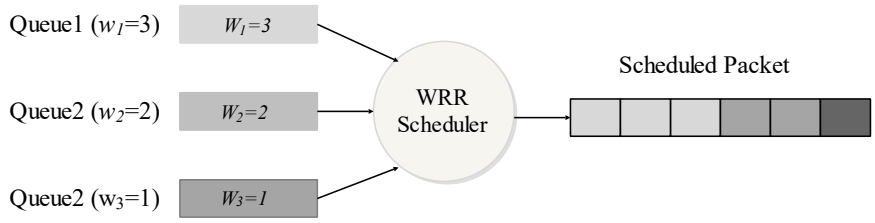


Gambar II-2. *Flowchart* Algoritma *Round Robin* (antrian dan penjadwalan *load lalancing*) (Sumber: Mohit Tomar, Master of Science in Electrical Engineering, California State University 2017)

2.1.3.4 *Weighted Round Robin (WRR) Algorithm*

Penjadwalan Algoritma WRR dirancang untuk menangani *server* dengan kapasitas pemrosesan yang lebih baik. Setiap *server* dapat diberi bobot, nilai integer yang menunjukkan kapasitas pemrosesan. *Server* dengan bobot yang lebih tinggi menerima koneksi baru lebih dulu daripada yang memiliki bobot lebih sedikit, dan *server* dengan bobot lebih tinggi mendapatkan lebih banyak koneksi daripada yang memiliki bobot lebih sedikit dan *server* dengan bobot yang sama mendapatkan koneksi yang sama (Saboori *et al*. 2012).

Metode penjadwalan algoritma WRR bekerja melayani antrian input. Bobot adalah variabel yang menunjukkan berapa banyak paket yang harus dikirim untuk setiap siklus dari setiap antrian. Jika antrian memiliki lebih sedikit paket daripada nilai berat, penjadwal WRR mengeluarkan jumlah paket yang ada dan mulai melayani antrian berikutnya.



Gambar II-3 Operation of Conventional WRR

Sumber: Patel, Z., & Dalal, U. *International Journal of Wireless & Mobile Networks* (IJWMN) 2014

Metode algoritma penjadwalan WRR didasarkan pada penetapan bobot pecahan Øi untuk setiap antrian layanan sehingga penjumlahan semua antrian layanan sama dengan satu.

$$\sum_{i=1}^{N} \emptyset i = 1 \qquad (1)$$

Karena berat adalah pecahan dan kita ingin menentukan jumlah paket bilangan bulat yang akan dilayani dari setiap antrian, berat pecahan dikalikan dengan bilangan bulat konstan M. Produk dibulatkan ke bilangan bulat lebih besar terdekat untuk memperoleh bobot bilangan bulat w_i. Nilai berat bilangan bulat dari

setiap antrian ini menentukan jumlah paket yang akan dilayani dari antrean tersebut. Jumlah total dari nilai-nilai counter ini disebut sebagai *round robin length*. Berat integer dari i^{th} antrian (Patel *et al*. 2014] adalah:

$$wi = \lceil \emptyset i * M \rceil \qquad (2)$$

Sum (Jumlah) koneksi N aktif yang ada didefinisikan sebagai panjang *round robin* W dan diberikan oleh:

$$w = \sum_{i=1}^{N} wi = M \qquad (3)$$

Asumsikan bahwa laju tautan keluar adalah *r*, dan *rate* yang ditawarkan ke i^{th} koneksi adalah:

$$ri = \frac{wi}{W} r \qquad (4)$$

Maka meningkatnya jumlah koneksi pada *rate*. Ketika jumlah koneksi *N* meningkat, persamaan persamaan (1) mengatakan bahwa bobot koneksi individu $\emptyset i$ menurun dan ini mengurangi w_i. Karena jumlah dari semua berat tetap konstan, W tetap tidak berubah dan karenanya per persamaan (4) tingkat koneksi yang menurun.

Untuk algoritma penjadwalan tertentu, parameter seperti tingkat transmisi tautan *output*, *rate* yang dialokasikan, dan jumlah koneksi dapat mempengaruhi latensi.Menentukan *worse-case latency* untuk koneksi *i* untuk penjadwal WRR konvensional. Asumsikan bahwa ada antrian sambungan N yang sedang mundur dan penjadwal saat ini melayani w_i^{th} dengan paket dari i^{th} koneksi. Karena panjang siklus adalah *W*, mungkin ada banyak paket *W-wi* yang akan dilayani dari antrian *N-1* lainnya sebelum $(wi + 1)^{th}$ paket dari antrian*i*koneksi dilayani. Oleh karena itu, *worse-case latency* dari i^{th} untuk koneksi adalah:

$$\emptyset i, wrr = (W - wi)\frac{Li}{r} \text{ W } (1 - \emptyset i)\frac{Li}{r} \qquad (5)$$

Di mana L_i adalah panjang maksimum paket yang dimiliki koneksi i^{th}. *worse-case latency* ini meningkat ketika Øi menurun dengan peningkatanjumlah koneksi. Oleh karena itu memiliki karakteristik tuning *latency* yang tidak efisien. Untuk menghitung total *latency* yang dialami oleh paket, *latency* antrian harus ditambahkan ke persamaan (5).

Kesetaraan proporsional $\eta_{PF} = 1$ karena koneksi *i (j)* dapat memimpin koneksi lain *j (i)* paling banyak dengan *wi (wj)* paket. Untuk mengukur keadilan *worse-case latency*, metrik yang disebut *Worst case Fair Index* (WFI) didefinisikan untuk mengkarakterisasi *server* antrian yang adil. Sebuah *server* dikatakan untuk menjamin WFI dari C_i untuk koneksi *i*, jika untuk waktu *t* penundaan suatu paket tiba di *t* dibatasi.

$$di < ai \ + \frac{Qi\,(t)}{ri} + Ci \qquad (6)$$

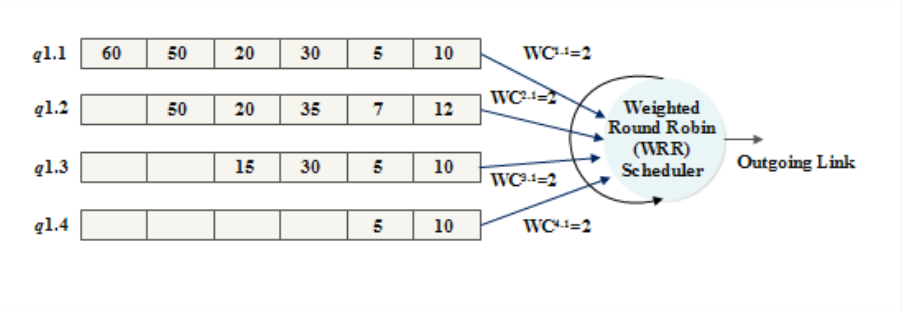
Dimana *Qi (t)* adalah ukuran antrian koneksi *i* pada waktu kedatangan paket *t* dan C_i disebut indeks *fair case* terburuk untuk koneksi *i*. Misalkan paket koneksi baru tiba pada waktu *t* ketika *server* baru saja menyeberang *i*, dan misalkan *backlog* koneksi pada waktu *t* (dilambangkan dengan *Qi (t))* adalah kelipatan dari w_i. Kemudian, paket ini bergerak setelah waktu maksimum [$Q_i r_i$ + $(W + w_i + 1)\ L_i/_r$]. Jadi WFI dari WRR scheduler diberikan oleh:

$$Ci, WRR = di - ai - \frac{Qi\,(t)}{ri}$$
$$= (W - wi + 1)\ \frac{Li}{r} \qquad (7)$$

Ketika jumlah koneksi meningkat, w_i menurun dan karenanya WFI meningkat. Jadi peningkatan jumlah koneksi pada *scheduler* menurunkan nilai WFI.

Berdasarkan skenario selanjutnya, suatu *burst* pada *input traffic* menyajikan tantangan besar untuk menciptakan penundaan dan kehilangan paket serta menurunkan *throughput* dalam jaringan IEEE 802.16, yang mengarah pada

penurunan kinerja. Seperti pada sebuah kelas mungkin memiliki lebih banyak paket yang tiba daripada nilai *counter* beratnya yang terkait, tidak dapat ditangani dalam satu putaran, seperti yang ditunjukkan pada Gambar II.4. Pada gambar ini menunjukkan bahwa penjadwal WRR menetapkan bobot statis 2 ke setiap penghitung antrian (*W*), yang berarti hanya dua paket yang dapat dikirim dari setiap antrian. Karena penghitung berkurang sebesar 1 untuk setiap paket, selanjutnya mencapai nol sebelum semua paket dalam antrian telah ditransmisikan, seperti yang ditunjukkan pada Gambar II.5, dan karena penjadwal WRR untuk antrian ini ditangguhkan hingga *reset counter* berikutnya.



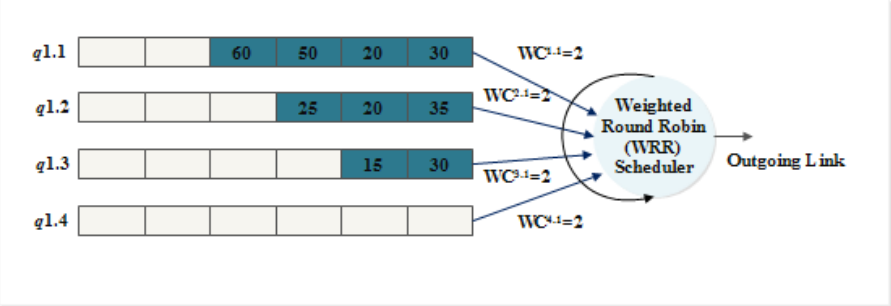
Gambar II-4.Ilustrasi Algoritma WRR (Antrian dan Penjadwalan)
(Sumber: Ibrahim Said, EURASIP Journal 2014)

Sebagai contoh, jika *q*i, 1 mewakili antrian *i* (*i* = 1, 2,...., *n*) pada putaran 1, maka *scheduler* dalam paket jadwal Gambar II.4 sebagai:

$$q1{,}1\ q2{,}1\ \rightarrow q3{,}1 \rightarrow q4{,}1 \rightarrow q1{,}1 \rightarrow q2{,}1 \rightarrow q3{,}1 \rightarrow q4{,}1$$

Penyelesaian urutan di atas merupakan putaran. *Scheduler* sehingga mentransmisikan dua paket dari setiap antrian dalam satu putaran.

Gambar II.5 menunjukkan bahwa sebelum reset berikutnya, total sembilan paket tetap dalam antrian menunggu transmisi. Oleh karena itu, paket yang menunggu di antrian akan tertunda lebih lanjut. Selain itu, dengan lalu lintas *burst input* yang berat, panjang antrian akan terus tumbuh dengan cepat, yang juga cenderung menyebabkan peningkatan *packet loss*. Pada akhirnya, akan ada pengurangan *throughput* sebagai akibat dari hilangnya paket.

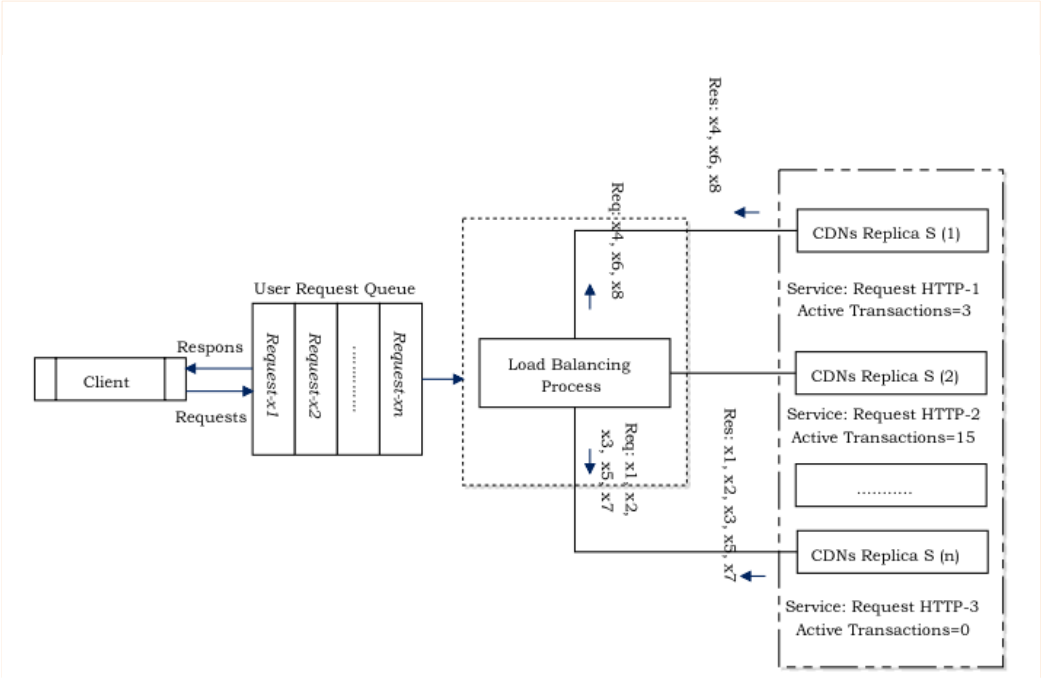


Gambar II-5.Ilustrasi Algoritma WRR sebelum reset counter. Antrian ditampilkan setelah bobot telah habis. (Sumber: Ibrahim Said, EURASIP Journal 2014)

2.1.3.5 Least Connection (LC) Algorithm

Penjadwalan Algoritma *least connection* bertugas mengarahkan koneksi jaringan ke server dengan jumlah koneksi paling sedikit.Algoritma penjadwalan ini adalah salah satu algoritma penjadwalan dinamis sehingga perlu menghitung jumlah koneksi untuk setiap *server* secara dinamis untuk memperkirakan bebannya. Penyeimbang beban mencatat nomor koneksi masing-masing *server traffic*, meningkatkan nomor koneksi *server* saat koneksi baru dikirim ke tujuan, dan mengurangi nomor koneksi *server* saat sambungan selesai atau batasan waktu (Sen, 2011). Metode penjadwalan algoritma *least connection* akan dijelaskan pada gambar II.6 berikut. Skema diagram berikut menunjukkan bagaimana *server* CDNs memilih layanan untuk *load balancing* dengan menggunakan metode koneksi terkecil. Dengan mempertimbangkan tiga layanan berikut:

1. Layanan-HTTP-1 menangani 3 transaksi aktif.
2. Layanan-HTTP-2 menangani 15 transaksi aktif.
3. Layanan-HTTP-3 tidak menangani transaksi aktif apa pun.



Gambar II-6. Ilustrasi Mekanisme Metode Penjadwalan Algoritma *Least Connection*
(Sumber: https://docs.citrix.com: online 2018)

Berikut representasi dari algoritma load balancing *least connection*. Berdasarkan skema pada gambar II.6. Dalam skema diagram ini, *server* CDNs memilih layanan untuk setiap koneksi masuk dengan memilih *server* dengan transaksi aktif paling sedikit. Koneksi diteruskan sebagai berikut:

1. Layanan-http-3 menerima permintaan pertama, karena tidak menangani transaksi aktif apa pun.layanan tanpa transaksi aktif dipilih terlebih dahulu.
2. Layanan-http-3 menerima permintaan kedua dan ketiga karena layanan memiliki jumlah transaksi aktif paling sedikit berikutnya.
3. Layanan http-1 menerima permintaan keempat karena layanan-http-1 dan layanan-3 memiliki jumlah transaksi aktif yang sama, *server* menggunakan metode *round robin* untuk memilih di antara mereka.
4. Layanan-http-3 menerima permintaan kelima.
5. Layanan-http-1 menerima permintaan keenam, dan seterusnya, hingga kedua layanan http-1 dan layanan http-3 menangani jumlah permintaan yang sama seperti layanan-http-2.

Catatan: Jika koneksi ke layanan http-2 dekat, mungkin mendapat koneksi baru sebelum masing-masing dua layanan lainnya memiliki 15 transaksi aktif.

Tabel II-5 berikut menjelaskan bagaimana koneksi didistribusikan dalam pengaturan pengaturan beban tiga layanan yang dijelaskan di atas.

Koneksi Masuk	Layanan Dipilih	Jumlah Koneksi Aktif Saat Ini	Keterangan
Request-1	*Service*-HTTP-3 (N = 0)	1	Layanan-HTTP-3 memiliki koneksi aktif paling sedikit.
Request-2	*Service*-HTTP-3 (N = 1)	2	
Request-3	*Service*-HTTP-3 (N = 2)	3	
Request-4	*Service*-HTTP-1 (N = 3)	4	Layanan-HTTP-1 dan Layanan-HTTP-3 memiliki jumlah koneksi aktif yang sama.
Request-5	*Service*-HTTP-3 (N = 3)	4	
Request-6	*Service*-HTTP-1 (N = 4)	5	
Request-7	*Service*-HTTP-3 (N = 4)	5	
Request-8	*Service*-HTTP-1 (N = 5)	6	

Tabel II-5. Pendistribusian layanan tiga layanan mekanisme metode penjadwalan algoritma *least connection* (Sumber: https://docs.citrix.com: online 2018)

Least Connection Algorithm Pseudocode

```
Serv = {Serv0, Serv1,..., Servn-1},
W(i) is Servi the weight of server Servi,
C(Servi) is the current connection number of server Si,
for (m = 0; m < n; m++)
{
      if (W(Servm) > 0)
      {
            for (i = m+1; i ¡ n; i++)
            {
      if (W(Servi) ≤ 0)
            continue;
      if (C(Servi) < C(Servm))
            m = i;
}
            return Servm;
            }
                  return NULL;
}
```

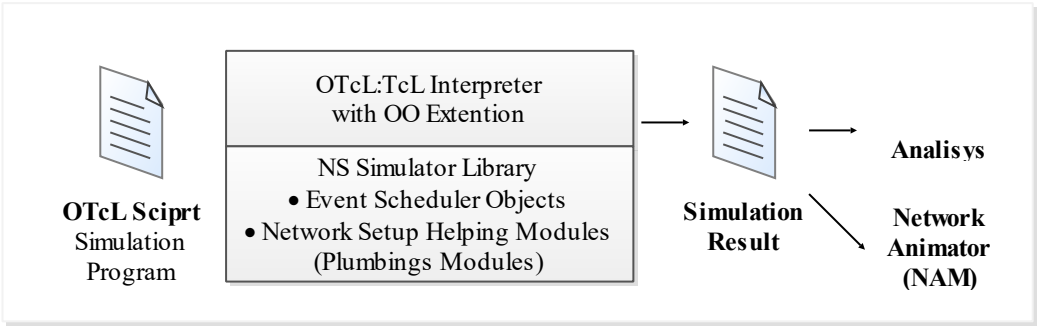
2.1.4 *Network Simulator*

Ada sejumlah simulator jaringan yang tersedia di luar sana. Ada yang komersial dan ada juga yang *open source*. Simulator umumnya terdiri dua tipe kontinu dan diskrit (Burbank *et al*. 2011) Model diskrit hanya mempertimbangkan momen diskrit dalam waktu yang sesuai dengan kejadian yang mempengaruhi jaringan simulasi. Ini disebut sebagai *discrete event simulations* (DES), dan memerlukan perangkat lunak simulasi untuk menjaga sebuah *clock* sehingga waktu simulasi saat ini dapat dipantau. Antara kejadian tersebut, tidak ada yang terjadi di jaringan dan waktu antara kejadian. Simulasi terus menerus mempertimbangkan semua titik pada saat resolusi perangkat keras *host* terbatas (semua simulasi diskrit karena berjalan pada platform digital). Metode diskrit paling sering digunakan untuk simulasi jaringan. Simulasi bisa berupa lokal (berjalan di satu komputer) atau didistribusikan di banyak komputer di jaringan komputer (tidak disimulasikan, namun komputer simulasi saling terkait). Ada juga beberapa simpul simulasi (lokal atau terdistribusi) dan beberapa simpul nyata dalam kombinasi (emulasi).

2.1.4.1 *Network Simulator* 2

Network Simulator 2 (NS-2) (Canne, 2018) adalah alat simulasi yang populer. Otnes dan Haavik menerapkan simulator untuk menguji pada sebuah protokol. Menggunakan NS-2 untuk simulasi (Nicolaou dan See, 2007), juga menguji protokol. NS2 (Ezreik dan Gheryani, 2012) menemukan bahwa *simulator* jaringan (NS2), digunakan sebagai alat untuk merancang hasil simulasi adalah *transfer* informasi yang aman antar *node*. NS2 adalah simulator diskrit dan mendukung *multicast* dan beberapa protokol *routing* untuk jaringan kabel dan nirkabel (lokal dan satelit). *Simulator* ditulis dalam C/C++ dan dihubungkan dengan TCL/OTCL. Distribusi antara kedua bahasa tersebut adalah bahwa kernel *simulator* dan modul jaringan ditulis dalam C/C++ dan disusun karena kinerjanya. *Interfacing* dengan *simulator* dilakukan dengan TCL/OTCL dimana jaringan akan diinisiasi, topologi dibangun dan berbagai kejadian dalam simulasi dikonfirmasikan. Dengan distribusi bahasa ini, modul yang dikompilasi berkinerja

baik, namun tidak perlu mengkompilasi ulang setiap kali ada perubahan pada topologi. Berikut alur blok diagram *Network Simulator* 2:



Gambar II-7 Blok Diagram NS2

NS2 mendefinisikan skenario simulasi menggunakan *node* jaringan, protokol, topologi jaringan, aplikasi spesifik dan bentuk *output* yang dibutuhkan dalam skrip OTCL.*Interpreter* OTCL menghubungkan *script* tertulis ke komponen C ++ yang dikompilasi melalui hubungan OTcl yang menciptakan satu lawan satu objek OTcl untuk masing-masing objek C ++. Setelah pelaksanaan simulasi, hasil simulasi ditangkap dengan cara yang berbeda seperti pada penelusuran *file* yang digunakan untuk menganalisis hasilnya dengan menggunakan teknik statistik dan analisis yang berbeda. NS2 dilengkapi dengan *network animator* (NAM) yang menampilkan simulasi visual ke pengguna akhir.

2.1.5 *Anycast*

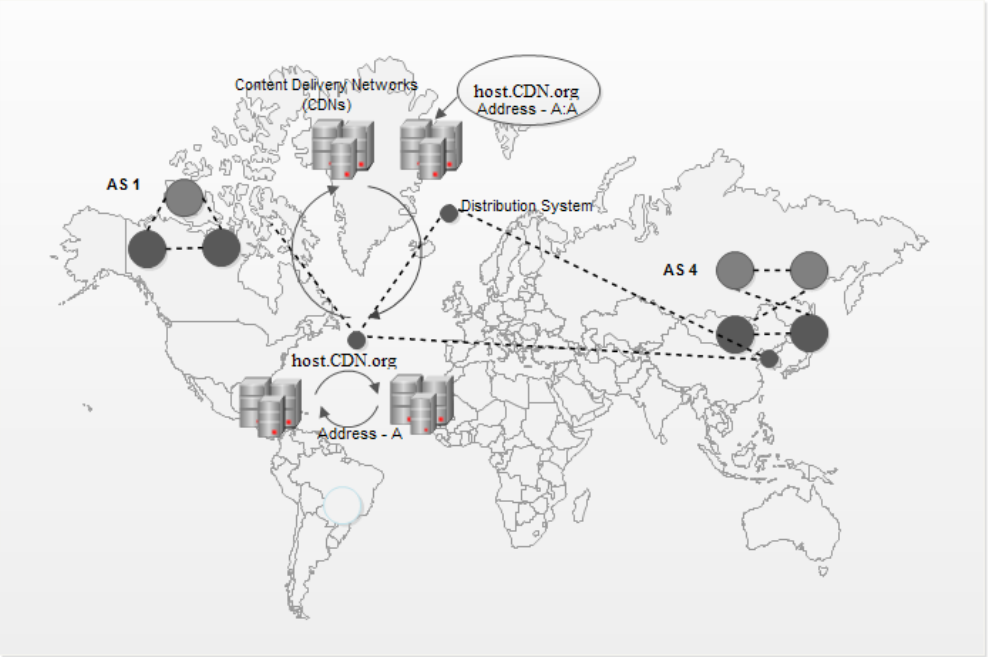
Anycast adalah paradigma yang relatif baru untuk pengelolaan CDNs, dan sudah ada beberapa CDN komersial yang ada saat ini menggunakan *anycast* (Swildens, 2009; Sarat *et al*. 2006). Dengan *anycast*, beberapa *server proxy* berbagi alamat IP yang sama. *Anycast* bergantung pada protokol *routing* untuk mengarahkan permintaan layanan ke salah satu *proxy* yang direplikasi secara *geolocation*, melalui jalur *network-path* (Bret dan Prashanth, 2018). Mekanisme berbasis *anycast* memiliki keuntungan karena mudah diterapkan dan dipelihara.

Anycast menawarkan sebuah metode untuk membuat layanan alamat IP yang tersedia untuk sistem perutean dari beberapa lokasi sekaligus.

2.1.6 *Application-layer Anycast*

Application layer anycast sangat bergantung pada *redirection* DNS dan IP *unicast*. Prosedur pemilihan *server* digambarkan pada Gambar II.8. Pertama, pengguna meminta sumber daya, seperti *video* atau laman web HTML, melalui URL-nya. Agen pengguna menyimpulkan dari URL ini nama *host*, mungkin milik CDNs tertentu. Dalam kasus ini, peran utama kemudian dimainkan oleh *resolver* DNS lokal klien dan CDNs setelah *resolver* DNS menerima nama *host* yang ingin dihubungi pengguna, ia meneruskan nama *host* ini ke CDNs yang bersangkutan.

Selanjutnya, CDNs memilih simpul yang melayani *client*. Hal ini dapat dicapai dengan beberapa cara (Leduc, 2018) Misalnya, CDNs hanya memilih *node* yang secara geografis paling dekat dengan *client*. Pendekatan lain adalah memilih node CDNs terdekat dalam hal *latency* berkenaan dengan *client*, atau yang paling dekat dalam hal jumlah *hop*. Untuk tujuan ini, *server* CDN secara berkala meluncurkan pengukuran *ping* atau *traceroute* terhadap akses *Internet Service Providers* (ISP), dan melaporkan hasilnya ke *resolver* DNS CDNs. CDNs juga bisa mengembalikan hal yang sama alamat IP, terlepas dari pengguna. Dalam kasus ini, alamat IP adalah alamat IP-*layer-anycast* yang terkait dengan beberapa (atau bahkan semua) *node* CDNs untuk menyeimbangkan beban di antara *server* yang berbeda. Masalah utama dengan metode ini adalah *client* diwakili oleh *resolver* DNS-nya, yang mungkin relatif jauh dari pengguna dan karenanya tidak harus mewakili dia dengan sangat baik (Chen *et al*. 2015) pemilihan *server* mungkin kurang optimal dalam hal kinerja. untuk pengguna Sebagai alternatif, seleksi ini dapat dilakukan di sisi *client*: teknik yang mungkin adalah bahwa sumber yang bisa diambil berisi serangkaian alamat IP dan bahwa *client* memilih yang optimal berdasarkan *latency*.



Gambar II-8. *Application Layer Anycast*

Client ingin mengetahui alamat IP yang dipetakan ke *hostname* host.CDN.org. *Resolver* DNS meneruskan permintaan ke CDN untuk mendapatkan alamat IP dari *server* "terbaik" untuk klien ini (Chen *et al*. 2015).

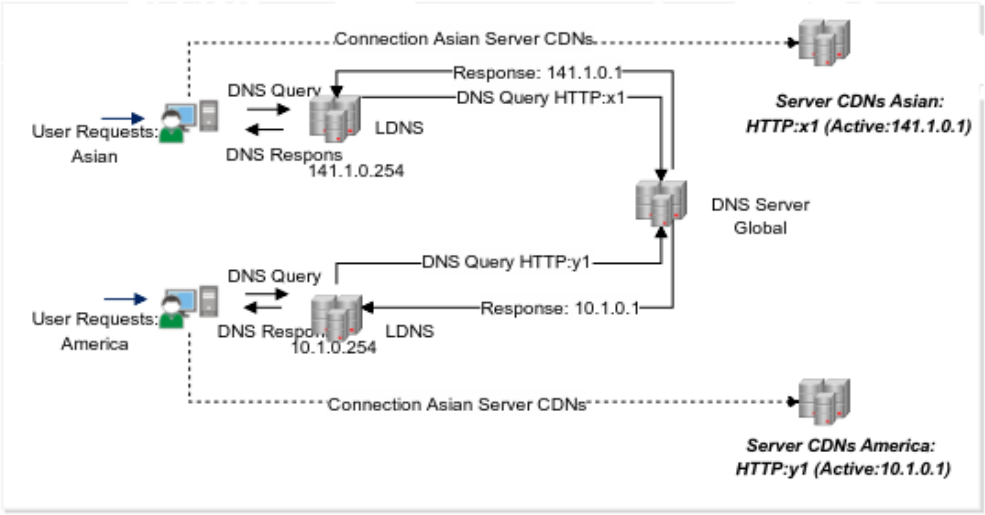
2.1.7 *Anycast in Content Delivery Network*

Sebagian besar penelitian sebelumnya (Calder, 2015; Chen, 2015; Torres, 2011; Adhikari; 2015; Bottger; 2016; Casas; 2014; Casas, 2014 dan Adhikari, 2012) bertujuan untuk mengkarakterisasi penyebaran *server* layanan populer yang bergantung pada layanan *layer* 7 atau *layer* 4 OSI *layer*. (Casas, 2014; Torres, 2011) fokus pada karakterisasi infrastruktur *server youtube* dan geolokasi *server*. Dalam mempelajari pendekatan *anycast* DNS CDNs yang akan diterapkan pada penelitian ini, memanfaatkan kemampuan *Domain Name System* yang telah diintegrasikan, dengan melakukan pengarahan terhadap *query* pengguna kelokasi *server* terdekat (Casas *et al*. 2014).

2.1.8 *Anycast Geolocation* DNS

Meningkatkan efisiensi pada segi *quality of service* (QoS) pengguna akhir, Content Distribution Networks (CDNs) secara efektif mengeksploitasi *Domain Name System* (DNS) untuk mengarahkan pengguna akhir ke replika konten terdekat dengan skala waktu singkat. Sementara penggunaan DNS telah membawa keuntungan yang signifikan bagi CDN (Qin *et al.* 2014).

Server DNS menghadirkan dan memproses semua permintaan DNS terkait domain yang dikelola oleh sistem CDN. Ketika klien mengakses CDN, langkah pertama terdiri dari menyediakan alamat IP yang sesuai dengan pengganti CDN terbaik. Berbagai strategi dapat ditargetkan pada tahap ini mulai dari tugas statis sederhana hingga fungsi yang lebih kompleks. Berdasarkan status pengganti dan jaringan (Qin *et al*.2014). Pendekatan *anycast geolocation* CDNs diterapkan pada penelitian ini. Teknik ini memanfaatkan kemampuan DNS yang telah diintegrasikan dengan *Geolocation*-IP database.dengan adanya integrasi ini, DNS *server* mampu melakukan pengarahan terhadap *query* pengguna kelokasi *server* terdekat. Prinsip kerjanya adalah seperti pada gambar II-9 dibawah.



Gambar II-9. *Anycast Geo*-DNS CDNs
(Sumber: F. Chen, R. K. Sitaraman, and M. Torres 2015)

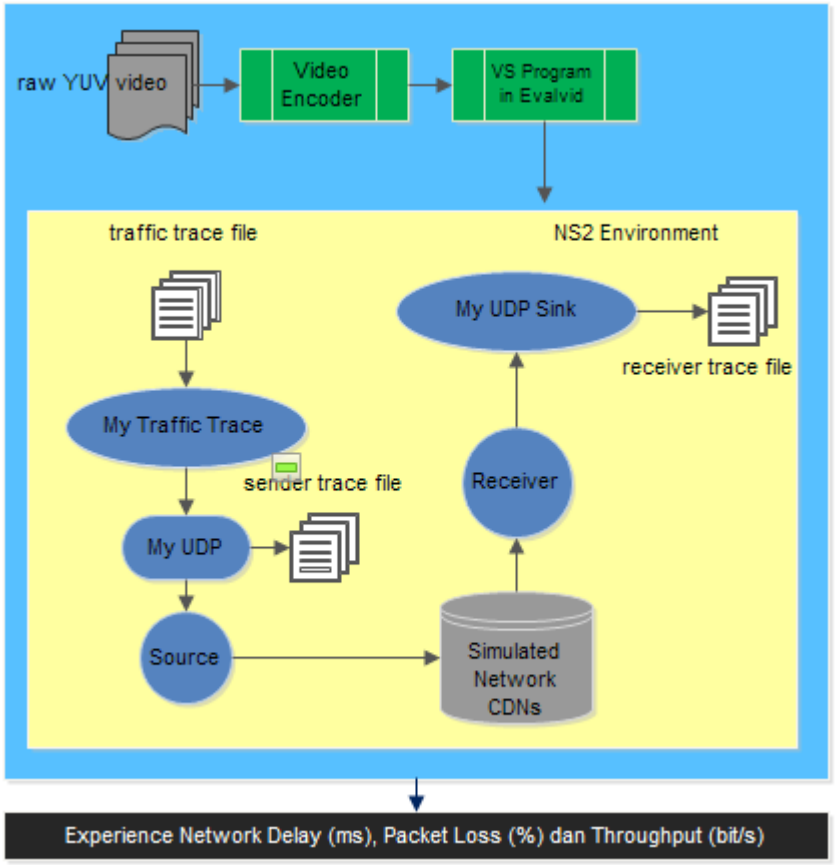
Berikut representasi dari algoritma kebijakan *geolocation* DNS untuk mencapai redireksi lalu lintas berdasarkan lokasi fisik *user* yang melakukan

request. Berdasarkan skema pada gambar II-9 proses kerja *geolocation* adalah sebagai berikut:

1. Selama proses *request* domain, pengguna mencoba untuk terhubung ke *server* CDN Asian HTTP: *x*1 (*Active*: 141.1.0.1). Selanjutnya menghasilkan *request* domain DNS yang dikirim ke *server* DNS. Biasanya, ini adalah *server* DNS yang disediakan oleh ISP lokal yang bertindak sebagai penyelesai *caching*, dan disebut sebagai LDNS.
2. Jika nama DNS tidak ada di *cache* lokal LDNS, *server* LDNS meneruskan permintaan ke *server* DNS yang berwenang untuk HTTP:*x*1 (*Active*: 141.1.0.1). *Server* DNS otoritatif merespons dengan catatan yang diminta (HTTP:*x*1 (*Active*: 141.1.0.1)) ke *server* LDNS, yang pada gilirannya akan men-*cache record* secara lokal sebelum mengirimnya *user*.
3. *Server* DNS akan melihat IP *Address user* yang melakukan *query request* kemudian akan memilih IP *Address* dari *server* yang paling dekat dengan lokasi geografis *user*.
4. *Server* DNS otoritatif biasanya melihat permintaan resolusi *domain* yang berasal dari *server* LDNS dan bukan dari *user*. Karena itu, alamat IP sumber dalam permintaan resolusi domain seperti yang terlihat oleh *server* DNS otoritatif adalah *server* LDNS dan bukan dari *user*. Namun, menggunakan alamat IP dari *server* LDNS saat mengkonfigurasi permintaan *query* berdasarkan geo-lokasi memberikan perkiraan yang adil tentang lokasi geografis *user*.

2.1.9 Integrasi Arsitektur NS2 dan Evalvid

EvalVid adalah kerangka kerja dan *tool-set* untuk mengevaluasi kualitas video yang dikirim melalui jaringan komunikasi fisik atau simulasi. Selain mengukur parameter QoS (*Quality of Service*) dari jaringan yang mendasari, seperti *delay*, *jitter*, *packet loss* serta *throughput*. Arsitektur terintegrasi dari ns-2 dan EvalVid ditunjukkan pada gambar II.10.



Gambar II-10. Integrasi Arsitektur NS2 dan Evalvid

Tiga agen simulasi terhubung, yaitu MyTrafficTrace, MyUDP, dan MyUDPSink, diimplementasikan antara NS2 dan EvalVid. Antarmuka ini dirancang untuk membaca *file* pelacakan video atau untuk menghasilkan data yang diperlukan untuk mengevaluasi kualitas video yang dikirim. Gambar II.10 mengilustrasikan kerangka penilaian QoS untuk lalu lintas video yang dimungkinkan oleh perangkat yang menggabungkan EvalVid dan NS2 (Ukani, 2015).

2.2 Tinjauan Studi

(Jena and Ahmad, 2013) Melakukan penelitian mengenai algoritma *load balancing*: (i) *Round Robin* (ii) *Active monitoring* dan (3) *Throttled* yang efisien dan disempurnakan dan secara eksperimental untuk memverifikasi bagaimana

meminimalkan waktu respon dan waktu proses melalui simulasi dengan *cloud analyst*. Peneliti menemukan algoritma *round robin load balancing* sebagai hasil yang efisien. Peneliti juga menemukan bahwa parameter: waktu respons, waktu pemrosesan data hampir sama dalam hal pemantauan aktif dan penyeimbangan beban.

(**Dasguptaa *et al*. 2013)** Penelitian ini mengusulkan strategi *load balancing* baru menggunakan *Genetic Algorithm* (GA). Algoritma ini berkembang untuk menyeimbangkan beban infrastruktur *cloud* saat mencoba meminimalkan rentang dari set tugas yang diberikan. Strategi *load balancing* yang diusulkan telah disimulasikan menggunakan *simulator cloud analyst*. Hasil simulasi untuk aplikasi sampel yang khas menunjukkan bahwa algoritma yang diusulkan mengungguli pendekatan yang ada seperti *First Come First Serve (*FCFS*), Round Robin (*RR*) dan Stochastic Hill Climbing (*SHC*)*.

(**Narayanan *et al*. 2017)** Penelitian ini menerapkan tentang sumber daya yang diatur dan diakses secara terpusat. Saat permintaan dibuat ke penyedia layanan *cloud*, permintaan harus dilakukan oleh salah satu dari beberapa *server* yang tersedia dan memuat pada *server* harus seimbang. Penelitian ini menyediakan algoritma *load balancing* yang efisien untuk lingkungan *cloud*, dan menggabungkan manfaat dari algoritma *load balancing divisible* dan algoritma *weight round robin*. Hasil simulasi menunjukkan bahwa metode ini efisien dan nilai waktu pemrosesan dan waktu respon menghasilkan nilai yang rendah dibandingkan dengan metode lain. Selain itu metode ini juga menghilangkan kelemahan metode *round robin* tradisional.

(**Goel *et al*. 2015)** Penelitian menunjukkan bahwa pemilihan *server* CDN berbasis DNS dapat diperbaiki untuk sepenuhnya mempercepat proses CDN. Peneliti mengusulkan DNS-Proxy (*dp*), proses sisi klien yang berbagi fungsi *load-balancing* dengan CDN dengan memilih dari antara *server* CDN yang diselesaikan berdasarkan kinerja jaringan *lastmile*. Studi pengukuran peneliti

tentang infrastruktur CDN yang disebarkan oleh lima penyedia CDN besar menunjukkan bahwa *dp* mengurangi waktu buka halaman web rata-rata 29%. Jika dp telah *resolve* domain, pengurangan waktu buka halaman web adalah sebanyak 40%. Akhirnya, *dp* mengurangi waktu buka objek Web statis individual sebanyak 43%. Peneliti berpendapat bahwa *dp* memungkinkan penggunaan yang lebih efektif dari infrastruktur pengiriman konten yang ada dan merupakan strategi pelengkap untuk peningkatan ketersediaan konten geografis.

(**Benchaita *et al*. 2016)** Penelitian ini menyajikan skema yang fleksibel dan algoritma pengoptimalan, berdasarkan teori Lyapunov, untuk perutean permintaan dalam CDN.Pendekatan adalah memberikan kualitas layanan yang stabil kepada *client*, sekaligus meningkatkan penundaan pengiriman konten.teori ini juga mengurangi biaya transportasi data untuk ISP dalam hal peak manajemen *traffic*. Dalam penelitian ini, peneliti bertujuan untuk meningkatkan efektivitas solusi pengiriman konten. Untuk tujuan ini, kontribusi utama peneliti adalah desain metode perutean permintaan yang dioptimalkan, tetapi generik, yaitu: SORT. Algoritma di balik SORT didasarkan pada teori optimasi Lyapunov, dan sangat cocok untuk mengoptimalkan kinerja sambil mempertahankan stabilitas jaringan.Juga, untuk menyediakan suatu proses *routing* yang efektif, tetapi fleksibel, suatu modul tabel *routing* yang dimodifikasi dan terintegrasi.

(Wang *et al*. 2018) Penelitian ini telah menyajikan survei komprehensif CDN berbasis DNS. Sementara pengalihan *server* berbasis DNS menunjukkan keunggulannya dibandingkan dengan pendekatan lain, DNS jarak jauh menimbulkan masalah penting dalam hal kinerja CDN. ECS, *name extension*, dan *direct resolution* adalah tiga proposal yang menjanjikan untuk mengatasi masalah DNS jarak jauh. Skema *pseudononymizing client* yang diusulkan adalah metode yang paling efektif untuk mengatasi kedua masalah privasi. Sebuah studi komparatif dari semua solusi CDN berbasis DNS menunjukkan bahwa setiap model memiliki kelebihan dan kekurangan. Secara khusus, ECS ditemukan kekurangan transparansi meskipun kinerja CDN yang baik, dan *direct resolution*

ditunjukkan untuk menilai privasi pengalihan dengan mengorbankan peningkatan kompleksitas klien. Sebagai substrat dasar dari internet saat ini, DNS secara konstan ditingkatkan untuk beradaptasi dengan teknologi dan layanan yang muncul yang muncul. Itu selalu berisiko untuk meluncurkan solusi DNS baru pada skala Internet tanpa sepenuhnya menguji dan memvalidasinya. Oleh karena itu, implementasi yang ekstensif, percobaan, dan pemodelan kinerja dan evaluasi solusi CDN berbasis DNS yang muncul akan menjadi prioritas utama untuk komunitas DNS dan CDNs pada masa yang akan datang.

Tabel II-6. Ringkasan Tinjauan Studi

Penulis	Judul Penelitian	Metode	Hasil Penelitian
[Jena and Ahmad, 2013]	*Response Time Minimization of Different Load Balancing Algorithms in Cloud Computing Environment* (2013)	*1. Round Robin* *2. Active monitoring* *3. Throttled*	Pada penelitian ini membahas algoritma *load balancing* yang efisien dan disempurnakan secara eksperimental dan memverifikasi bagaimana meminimalkan waktu respon dan waktu proses. Salah satu hasil *respon time* algoritma: 1. *Round Robin*= 187.41 (ms) 2. *Active monitoring*= 187.52 (ms) 3. *Throttled*= 187.47 (ms) Salah satu hasil proses time algoritma: 1. *Round Robin*= 11.25 (ms) 2. *Active monitoring*= 11.34 (ms) 3. *Throttled*= 11.37 (ms)
(Dasguptaa *et al*. 2013)	*A Genetic Algorithm (GA) based Load Balancing Strategy for Cloud Computing* (2013)	*1. Genetic Algorithm (GA)* *2. Round Robin Algorithm (RRA)* *3. Stochastic Hill Climbing (SHC)* *4. First Come First Serve (FCFS)*	Hasil penelitian pada *load balancing* strategi menggunakan dalam *response time* (RT) *in* (ms): 1. *Genetic Algorithm* (GA): 330.54 2. *Round Robin Algorithm* (RRA): 341.87 3. *Stochastic Hill* Climbing (SHC): 336.96 4. FCFS: 349.26 Analisis hasil menunjukkan bahwa strategi yang diusulkan untuk *load balancing* tidak hanya mengungguli beberapa teknik yang ada tetapi juga menjamin persyaratan QoS.

(Narayanan *et al*. 2017)	*Efficient Load Balancing Algorithm For Cloud Computing Using Divisible Load Scheduling And Weighted Round Robin Methods* (2017)	*1. Round Robin Algorithm (RR)* *2. Weight Round Robin Algoritm (WRR)* *3. Divisible Weight Round Robin (DWRR)*	Penelitian ini menyediakan algoritma *load balancing* yang efisien untuk lingkungan *cloud*, menggabungkan manfaat dari *divisible load balancing algorithm and weighted round robin algorithm* Parameter *respon time*: 1. *Round Robin: Average* (ms)=288.17, Min (ms) =2.97, Max (ms) = 38,930.83. 2. WRR: *Average* (ms) =653.67, Min (ms)= 38.46, Max (ms)=36,203.93 3. *Divisible* WRR: Average (ms) = 653.67, Min (ms) = 38.46 , Max(ms) =36,203.93 Parameter *Processing time*: 1. *Round Robin*: *Average* (ms) = 608.53, Min (ms) = 46.20, Max (ms) = 29,887.38. 2. WRR: *Average* (ms) = 258.85, Min (ms) =1.26, Max (ms) = 42,632.49. 3. *Divisible* WRR: *Average* (ms) = 249.7 , Min (ms)= 1.38, Max (ms) = 33,086.29
(Goel *et al*. 2015)	*Faster Web through Client-assisted CDN Server Selection* (2015)	*DNS-Proxy (dp)* *CDN Server Selection*	Peneliti mengusulkan DNS-*Proxy* (*dp*), Studi pengukuran tentang infrastruktur CDN yang disebarkan oleh lima penyedia CDN dan menunjukkan bahwa *dp* mengurangi waktu buka halaman web rata-rata 29%. Jika dp me *resolve* domain, pengurangan waktu buka halaman web adalah sebanyak 40%. Peneliti berpendapat bahwa *dp* memungkinkan penggunaan infrastruktur pengiriman konten yang lebih efektif dan mewakili strategi pelengkap untuk peningkatan ketersediaan konten geografis secara terus-menerus.

(Benchaita *et al*. 2016)	*Stability and Optimization of DNS-based Request Redirection in CDNs* (2016)	*SORT (Stable and Optimized Request Routing)*	1. Penelitian menerapkan algoritma pengoptimalan, berdasarkan teori Lyapunov: perutean permintaan dalam CDNs memberikan kualitas layanan yang stabil kepada *client*, sekaligus meningkatkan penundaan pengiriman konten. Teori ini juga mengurangi biaya transportasi data untuk ISP dan melebihi teknik yang ada dalam hal *peak* manajemen *traffic*. Algoritma SORT menunjukkan hasil: penundaan pengiriman terkecil= Peningkatan rasio hit 20%. 2. *Request* perutean pada SORT efisiensi *cache* = 100% (Pemrosesan pada lalu lintas server)
(Wang *et al*. 2018)	*Evolution and challenges of DNS-Based CDNs* (2018)	*DNS-Based CDNs()pseudononymizing scheme)*	Penelitian ini telah menyajikan survei komprehensif CDN berbasis DNS. Sementara pengalihan *server* berbasis DNS menunjukkan keunggulannya dibandingkan dengan pendekatan lain, DNS jarak jauh menimbulkan masalah penting dalam hal kinerja CDN. ECS, *name extension*, dan *direct resolution* adalah tiga proposal yang menjanjikan untuk mengatasi masalah DNS jarak jauh.

Berdasarkan tinjauan studi pada Tabel II-6 diatas ada beberapa perbedaan pada pemilihan konsep penerapan algoritma yang digunakan. Berdasarkan pemilihan topik penelitian dari tinjauan studi tersebut, penulis menentukan dan menerapkan beberapa metode algoritma penjadwalan *load balancing* pada teknologi *cloud Content Delivery Networks* (CDNs) yakni: (i) *Least Connection* (LC) (ii) *Round Robin* (RR) (iii) *Weighted Round Robin* (WRR) dan menerapkan metode *anycast Geolocation*-DNS untuk optimalisasi perutean *traffic* jalur transmisi CDNs. Nilai *output* yang dihasilkan adalah parameter QOS pada perutean jalur transmisi CDNs. Dengan menerapkan pembebanan *traffic* secara seimbang dari sumber *client node* ke tujuan *server node*. Model penjadwalan algoritma yang akan diterapkan dapat mengatasi *load traffic* pada teknologi perutean jalur transmisi CDNs menuju server *edge* (*replica server*). Dengan memperhatikan nilai performansi dari *throughput*, *packet loss*, dan *delay.*

Analisis penelitian mempertimbangkan beberapa parameter *quality of service* yang dijabarkan sebagai berikut:

1. Pemanfaatan sumber daya perutean paket data: digunakan untuk memastikan pemanfaatan yang tepat dari semua sumber daya algoritma penjadwalan, yang terdiri dari keseluruhan penerapan. Faktor ini harus dioptimalkan agar memiliki algoritma *load balancing* yang efisien. Ditinjau dari nilai parameter performansi dari *throughput*, *packet loss*, dan *delay.*
2. Skalabilitas: kemampuan suatu algoritma untuk melakukan *load balancing* yang seragam dalam suatu sistem perutean jalur transmisi. Dengan jumlah *node* pada penerapan CDN dan Non-CDN, dengan hasil yang diharapakan yaitu agoritma dengan skalabilitas yang efisien.
3. Performansi: digunakan untuk memeriksa dan mengevaluasi, seberapa efisien algoritma yang diterapkan. Misalnya mengurangi *delay*, *packet loss* dan *throughput* dari parameter QoS yang dapat diterima.

Pada Tabel II-6 sebelumnya teknik algoritma penjadwalan beban pada jalur transmisi yang berbeda yang diusulkan. Tabel II-7 memberikan analisis

komparatif *output* teknik *load balancing* dan *geolocation* yang berbeda sehubungan dengan parameter kinerja yang berbeda.

Tabel II-7. Perbandingan teknik *load balancing* dan *geolocation* domain pada penelitian model penjadwalan algoritma *balancing least connection* (ALC), *round robin* (ARR) dan *weighted round robin* (AWRR) yang akan diterapkan.

Technic Metric	***Throughput***	***Overhead***	***Fault Tolerance***	***Migration Time***	***Response Time***	***Resource Utilization***	***Scalability***	***Performance***
Round Robin	*Yes*	*Yes*	*No*	*No*	*Yes*	*Yes*	*Yes*	*Yes*
Weight Round Robin	*Yes*	*Yes*	*No*	*Yes*	*Yes*	*Yes*	*Yes*	*Yes*
Least Connection	*Yes*	*Yes*	*Yes*	*No*	*Yes*	*Yes*	*Yes*	*Yes*
Geolocation Domain	*Yes*	*Yes*	*No*	*No*	*Yes*	*Yes*	*Yes*	*Yes*

Tabel II-7. *Teknik Load Balancing Pada Penelitian*
(Sumber: Hitesh Bheda, H. B. (2015)

2.3 Tinjauan Obyek Penelitan

Obyek utama penelitian ini merupakan perkembangan internet yang sangat cepat, pada penggunaan konten yang sangat komplek. *Content Delivery Networks* (CDNs) merupakan bagian penting dari ekosistem internet dan membangun lapisan untuk pengiriman konten yang lebih cepat. berkembangnya topologi internet dan pertumbuhan lalu lintas, ada kebutuhan untuk mempelajari penyebaran *cache* CDN untuk mengoptimalkan biaya serta memenuhi persyaratan kinerja.

Pada penelitian ini penulis mempelajari masalah pengoptimalan penyebaran CDN multi-AS di inti Internet. Selanjutnya berfokus pada penerapan algoritma *load balancing* untuk mengetahui pendistribusian beban trafik pada dua atau lebih jalur koneksi secara seimbang. Penerapan akan dilakukan dengan perbandingan CDN dan Non-CDN menggunakan algoritma *Least Connection* (LC), *Round Robin* (RR) dan *Weighted Round Robin* (WRR). Selanjutnya pemetaan pada beberapa *server* dan *client* untuk mengetahui *request* suatu *client* dalam mencari

jalur terdekat sebuah *server* ketika *client* tersebut melakukan *request*, dengan teknik *geolocation* domain. Adapun nilai yang akan dianalisis adalah *quality of service* (QoS) termasuk didalamnya *paket loss, delay* dan *throughput*.

2.4 Kerangka Konsep

Berdasarkan identifikasi masalah, tujuan penelitian, kajian teori, studi dari penelitian sebelumnya, dan tinjauan obyek penelitian, maka dapat dibangun kerangka konsep penelitian tentang Mekanisme Penjadwalan dan Optimalisasi Multi-Algoritma *Load Balancing* dan *Anycast* DNS pada *Content Delivery Network* (CDN).

Kerangka konsep menguraikan proses-proses sebagai berikut:

1. Perkembangan Teknologi CDNs saat ini
    - Lalu lintas teknologi komunikasi data yang yang semakin komplek, terutama saat ini pada teknologi *Content Delivery Network* (CDN), yang hanya menekankan distribusi beban di antara *server* replika. Sehingga sebuah *single link* pada *server-server* replika yang tersebar akan mengalami beban yang sangat tinggi ketika *realtime traffic* melakukan permintaan pada sebuah infrastruktur *server* replika.
    - Aspek penurunan nilai parameter *quality of service* (QoS), termasuk didalamnya *paket loss, delay, throughput*, waktu respon dan *overload* beban *traffic* pada saat beban trafik memadati suatu koneksi pada tujuan sebuah *server* replika CDNs. Sehingga layanan terdegradasi dengan penundaan akses yang tinggi dan *respon time* yang lama menyebabkan gangguan pada pengguna.
    - Saat ini CDNs hanya menyediakan layanan *best-effort* pada konektivitas *server* replika pada Infrastruktur *server edge*. Namun di masa depan layanan QoS (*Quality of Service*) akan perlu menarik lebih banyak pengguna. Umumnya, CDNs menggunakan mekanisme pemilihan *server* yang memilih *server* dalam jaringan untuk memenuhi permintaan pengguna. Metode yang paling umum

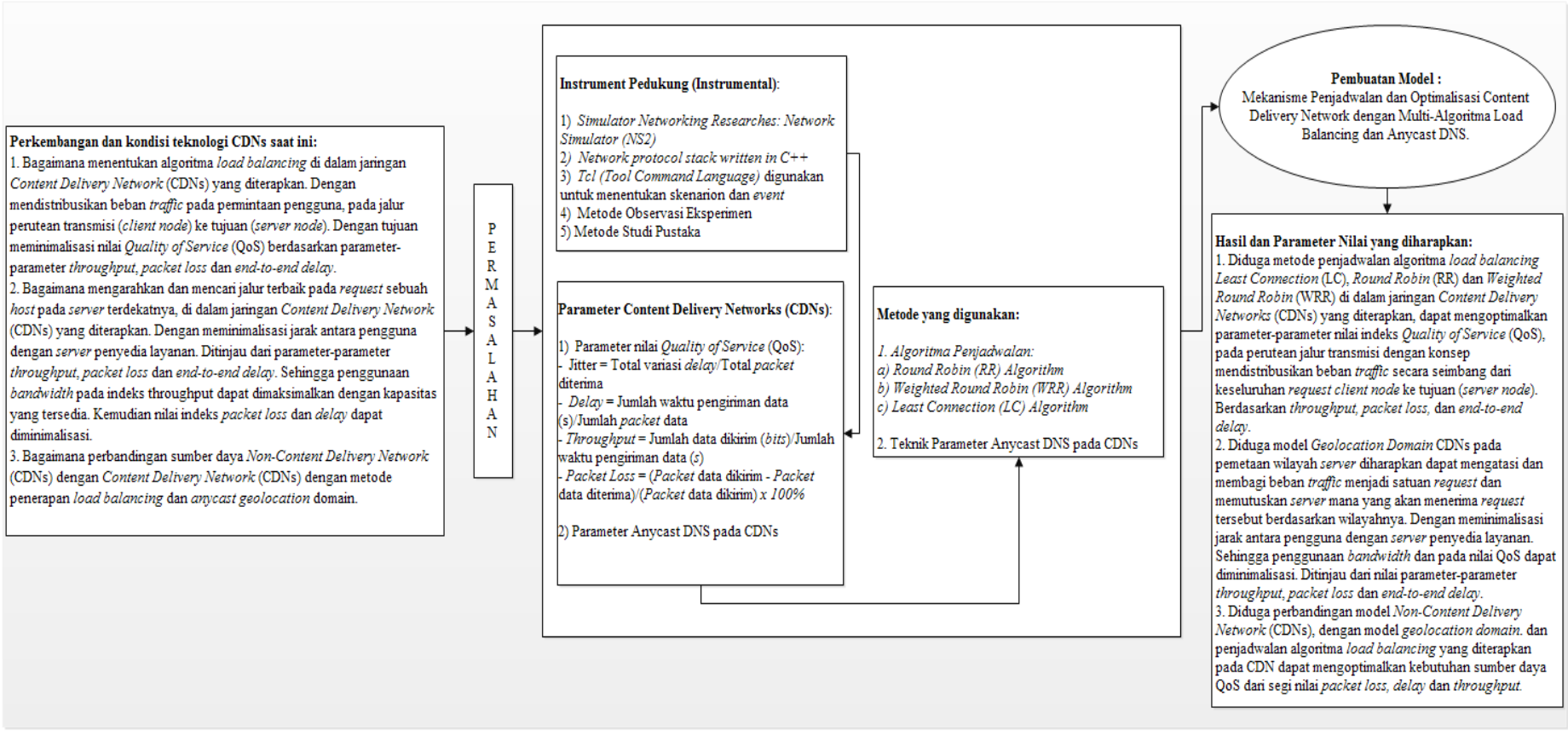
adalah yang berikutnya untuk mengarahkan permintaan pengguna ke *server* terdekat. Menggunakan mekanisme layanan nama domain (DNS) untuk mengukur beban *server* dan memilih *server* terdekat dengan wilayahnya.

2. Permasalahan
3. Instrument Pendukung (Instrumental)
   - *Simulator Networking Researches: Network Simulator (NS2)*
   - *Network protocol stack written in C++*
   - *Tcl (Tool Command Language)* digunakan untuk menentukan skenarion dan *event*
   - Metode Observasi Eksperimen
   - Metode Studi Pustaka
4. Parameter *quality of service* Content Delivery Networks (CDNs):
   - Parameter nilai *quality of service* (QoS): *Delay* = Jumlah waktu pengiriman data (s) / jumlah *packet* data.
   - *Throughput* = Jumlah data dikirim (*bits*)/Jumlah waktu pengiriman data (*s*).
   - *Packet Loss* = (*Packet* data dikirim - *Packet* data diterima)/(*Packet* data dikirim) *x 100%.*
   - Parameter *Application-layer Anycast*: *Anycast Geolocation* DNS
5. Metode yang digunakan:
   a) Teknik Algoritma Penjadwalan
      - *Round Robin (RR) Algorithm*
      - *Weighted Round Robin (WRR) Algorithm*
      - *Least Connection (LC) Algorithm*
   b) Teknik *Anycast Geolocation Domain Name System* (DNS)
6. Pembuatan Model

   Mekanisme Parameter *Anycast* dan Optimalisasi *Load Balancing Content Delivery Networks* (CDNs) dengan Multi-Algoritma.

7. Pembuatan Model Teknik Algoritma Penjadwalan dan *Anycast Geolocation* DNS.
8. Pengujian dan Perbandingan
   - Algoritma Penjadwalan: *Round Robin (RR) Algorithm, Weighted Round Robin (WRR) Algorithm* dan *Least Connection (LC) Algorithm*.
   - Teknik *Anycast Geolocation* DNS pada CDNs.
9. Hasil dan Parameter yang diharapkan.
   - Diduga metode penjadwalan algoritma *load balancing Least Connection* (LC), *Round Robin* (RR) dan *Weighted Round Robin* (WRR) di dalam jaringan *Content Delivery Networks* (CDNs) yang diterapkan, dapat mengoptimalkan parameter-parameter nilai indeks *Quality of Service* (QoS), pada perutean jalur transmisi dengan konsep mendistribusikan beban *traffic* secara seimbang dari keseluruhan *request client node* ke tujuan (*replica server*). Berdasarkan *throughput, packet loss, dan end-to-end delay*.
   - Diduga model *Geolocation Domain* CDNs pada pemetaan wilayah *server* diharapkan dapat mengatasi dan membagi beban *traffic* menjadi satuan *request* dan memutuskan *server* mana yang akan menerima *request* tersebut berdasarkan wilayahnya. Dengan meminimalisasi jarak antara pengguna dengan *server* penyedia layanan. Sehingga penggunaan *bandwidth* dan pada nilai QoS dapat diminimalisasi. Ditinjau dari nilai parameter-parameter *throughput, packet loss* dan *end-to-end delay*.
   - Diduga perbandingan model *Non-Content Delivery Network* (Non-CDNs), dengan model *geolocation domain* dan penjadwalan algoritma *load balancing* yang diterapkan pada CDNs, dapat meningkatkan kualitas layanan. Mengurangi *latency* serta menurunkan tekanan *load pada network node* menuju *replica server* secara bersamaan. Selain itu, dengan memonitor parameter jaringan seperti *throughput* dan status kesehatan setiap *server* CDNs.

Kerangka Konsep: Mekanisme Penjadwalan dan Optimalisasi *Content Delivery Network* dengan Multi-Algoritma *Load Balancing* dan *Anycast* DNS.



Gambar II-11. Kerangka Konsep Penelitian

2.5 Hipotesis

Berdasarkan kerangka konsep yang telah dikemukakan maka pernyataan penelitian ini dapat dirumuskan sebagai berikut:

1. Diduga metode penjadwalan algoritma *load balancing Least Connection* (LC), *Round Robin* (RR) dan *Weighted Round Robin* (WRR) di dalam jaringan *Content Delivery Networks* (CDNs) yang diterapkan, dapat mengoptimalkan parameter-parameter nilai indeks *Quality of Service* (QoS), pada perutean jalur transmisi dengan konsep mendistribusikan beban *traffic* secara seimbang dari keseluruhan *request client node* ke tujuan (*replica server*). Berdasarkan *throughput, packet loss,* dan *end-to-end delay*.
2. Diduga model *Geolocation Domain* CDNs pada pemetaan wilayah *server* diharapkan dapat mengatasi dan membagi beban *traffic* menjadi satuan *request* dan memutuskan *server* mana yang akan menerima *request* tersebut berdasarkan wilayahnya. Dengan meminimalisasi jarak antara pengguna dengan *server* penyedia layanan. Sehingga penggunaan *bandwidth* dan pada nilai QoS dapat diminimalisasi. Ditinjau dari nilai parameter-parameter *throughput*, *packet loss* dan *end-to-end delay*.
3. Diduga perbandingan model *Non-Content Delivery Network* (CDNs), dengan model *geolocation domain* dan penjadwalan algoritma *load balancing* yang diterapkan pada CDNs, dapat meningkatkan kualitas layanan. Mengurangi *latency* serta menurunkan tekanan *load* pada *network backbone* secara bersamaan. Selain itu, dengan memonitor parameter jaringan seperti *throughput* dan status kesehatan setiap *server* CDNs.

BAB III
METODOLOGI DAN RANCANGAN PENELITIAN

3.1 Jenis Penelitian

Penelitian model mekanisme penjadwalan dan optimalisasi multi-algoritma *load balancing* dan *anycast* DNS pada *Content Delivery Networks* (CDN) merupakan jenis penelitian eksperimen (*Research Experiment*). Hakekat penelitian eksperimen (*experimental research*) adalah penelitian yang dilakukan untuk mengetahui pengaruh pemberian suatu *treatment* atau perlakuan terhadap subjek penelitian.

Metode dan konsep yang dilakukan dalam penelitian dengan mempelajari teori-teori, teknik dan konsep mengenai teknik *load balancing* dan algoritma penjadwalan antrian yaitu *Least Connection (LC), Round Robin (RR)* dan *Weight Round Robin* (WRR). Selanjutnya mengklasifikasikan metode *anycast DNS* dan *mapper* pada distribusi konsep yang akan dimodelkan.

Sistem model yang dilakukan dengan menggunakan *network simulator* NS2. NS2 adalah *simulator* kejadian diskrit dan mendukung *multicast* dan beberapa protokol routing untuk jaringan kabel dan nirkabel, dan transmisi TCP / IP dan untuk penjadwalan acara diskrit. *Simulator* ditulis dalam C/C++ dan dihubungkan dengan TCL/OTCL. Distribusi antara dua bahasa bahwa kernel *simulator* dan modul jaringan ditulis dalam C/C++ dan dikompilasi. Antarmuka dengan *simulator* dilakukan dengan TCL/OTCL di mana jaringan akan dimulai, topologi yang dibangun dan berbagai peristiwa dalam simulasi dikonfigurasi. Dengan distribusi bahasa ini, modul yang dikompilasi bekerja dengan baik, tetapi tidak perlu untuk mengkompilasi ulang setiap kali ada perubahan dalam topologi. *Simulator* NS2 akan diterapkan pada simulasi *output* untuk menghasilkan parameter kinerja pada jaringan *Content Delivery Networks* (CDNs).

3.2 Metode Pengumpulan Data

Pada penelitian ini metode yang digunakan dalam pengumpulan data adalah:

1. Metode Observasi

   Pada metode observasi yang digunakan adalah observasi eksperimen yang dilakukan dengan mengklasifikasikan dan mendata serta meneliti perkembangan pada bidang tertentu yang akan disimulasikan.Klasifikasi dilakukan pada konsep dan teknik algoritma penjadwalan: *Least Connection* (LC), *Round Robin* (RR) dan *Weight Round Robin* (WRR). Serta klasifikasi metode *anycast* Geo-DNS berdasarkan teknik *anycast* DNS dalam hal pemetan sebuah *request user*. Proses pendataan da penelitian yang dilakukan pada teknologi CDNs saat ini, yang telah mendapat banyak perhatian penelitian. Teknologi CDNs menggabungkan pengembangan teknologi komputasi *high-end* dengan infrastruktur jaringan berkinerja tinggi dan teknik pengelolaan replika terdistribusi.

2. Metode Studi Pustaka

   Studi pustaka yang dilakukan dengan melakukan pengumpulan data yang diarahkan kepada pencarian data dengan mempelajari, meneliti dan membaca referensi-referensi terkait dengan penelitian melalui jurnal tesis dan internet dalam melakukan penelitian.

   Pencarian data dan referensi jurnal-jurnal dan referensi terkait yang relevan dengan topik penelitian yang akan diterapkan, yakni teknologi CDNs saat ini dan teknik algoritma penjadwalan dan teknik *anycast* Geo-DNS berdasarkan *anycast* Geo-DNS. Sebagai penyeimbangan beban *traffic* pada sumber daya *Content Delivery Networks* (CDN) berupa nilai *quality of service* (QoS) termasuk didalamnya *paket loss, delay* dan *throughput*.

3. Metode Dokumentasi

   Pengambilan data melalui dokumen tertulis maupun elektronik dari lembaga/institusi berdasarkan (*Scimago Journal & Country Rank*) terkait penelitian yang akan diterapkan. Mencakup *Content Delivery Networks* (CDN) yang bertujuan untuk mengatasi keterbatasan yang melekat pada Internet. Konsep utama pada dasar teknologi ini adalah pengiriman pada titik-titik *edge* atau *replica server* jaringan di dekat area permintaan, serta

meningkatkan kinerja yang dirasakan pengguna serta membatasi dalam hal biaya.berfokus pada bidang penelitian utama di bidang CDNs, menunjukkan kinerja, dan menganalisis strategi yang ada untuk penempatan dan manajemen replika, pengukuran *server*, pemilihan *replica* terbaik dan permintaan pengarahan ulang.

3.3 Instrumentasi

Instrumentasi yang digunakan dalam mencakup penelitian ini adalah:

1. Instrumentasi dalam tahap pengumpulan data dengan metode observasi. Pada tahap ini peneliti melihat dan menganalisa dalam hal pengembangan teknologi pada bidang *Content Delivery Network* (CDN). khususnya pada sistem sebuah interkoneksi global internet pada internet *traffic* dan *network traffic* pada CDNs. teknologi dan layanan yang digunakan untuk mentransformasikan dan mekanisme pendistribusiannya. Parameter nilai pada kinerja jaringan *end-to-end latency* dan layanan pada *quality of service* (QoS). Optimalisasi perbandingan Teknik *Anycast* Geo-DNS dalam memetakan server pada *request host* dalam kecepatan pengiriman paket dari segi *quality of service* (QoS). Dengan melakukan analisis pada parameter tersebut, sehingga dapat mengeksplorasi keuntungan, kelemahan, peluang masa depan dalam pengaplikasian CDN dari nilai *quality of service* (QoS).
2. Instrumentasi dalam tahap studi pustaka.

   Peneliti melakukan studi *literature* dengan mengetahui dan dan mempelajari konsep-konsep model dalam system mekanisme dan teknik mendistribusikan layanan dan teknik distribusi beban *traffic* menggunakan teknik algoritma penjadwalan antrian dan teknik *anycast* CDN pada suatu kualitas jalur *traffic* internet pada *Content Delivery Networks* (CDNs).
3. Instrumentasi Dokumentasi

   Pada tahap ini peneliti melakukan pencarian data mengenai hal ataupun variabel yang berupa jurnal penelitian baik pada *e-journal* digital dan buku.Tujuan tahap dokumentasi ini adalah untuk melengkapi hal-hal pada

data-data yang masih diperlukan pada penerapan penelitian. Pencarian dokumentasi yang relevan dengan masalah penelitian. Yakni teknik algoritma penjadwalan yang digunakan yang akan membandingkan pad teknologi *Content Delivery Networks* (CDNs).

4. Instrumentasi Kebutuhan Perangkat Keras (*Hardware*) dan Perangkat Lunak (*Software*).
   a. Identifikasi Kebutuhan Perangkat Keras (*Hardware*)
      Simulasi penelitian menggunakan *hardware* dengan spesifikasi:
      - *Processor* : Intel (R) Core (TM) i3-2120T
      - RAM : 4 Gb
      - Hardisk : 520 Gb
      - Sistem Operasi : Linux Ubuntu 16.04 LTS
   b. Identifikasi Kebutuhan Perangkat Lunak (*Software*)
      - Linux Ubuntu.
      - *Simulator Networking Researches*: *Network Simulator* (NS2)
      - *Network protocol stack written in* C++
      - Tcl (*Tool Command Language*)

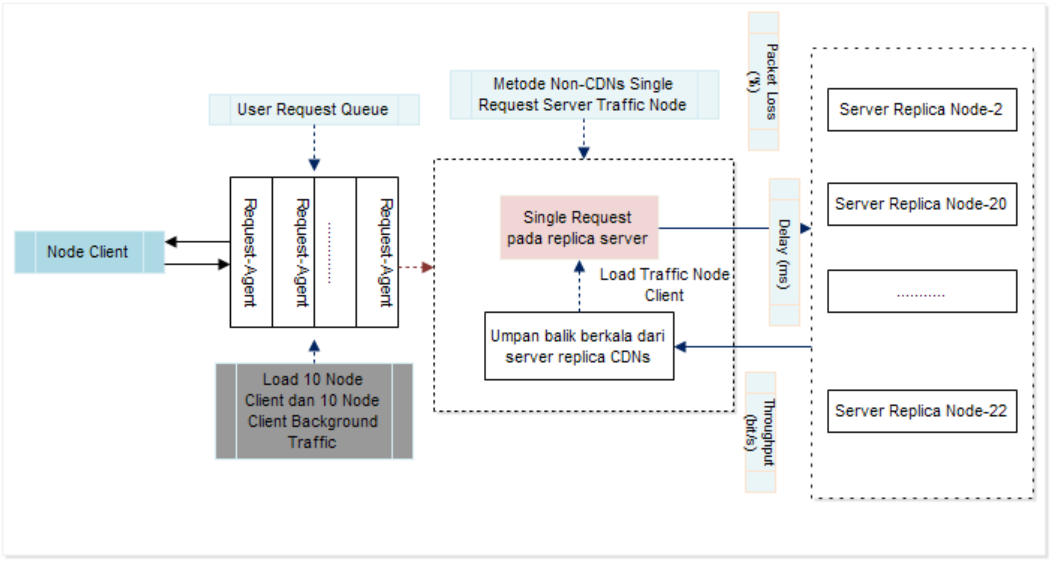
3.4 Teknik Analisis, Perancangan dan Pengujian

3.4.1 Teknik Analisis

Pada tahap analisis penulis melakukan konsep pada perencanaan sistem kinerja pada sebuah *Content Delivery Network* (CDN) pada *node replica server*. Saat ini *Content Provider* (CP) khususnya CDNs *provider* melakukan pengembangan dengan melakukan distribusi *server-server replica* ataupun *edge* pada masing-masing wilayah ataupun negara untuk menawarkan nilai tambah pada pengiriman konten internet.

Analisis pada penelitian mekanisme penjadwalan dan optimalisasi multi-algoritma *load balancing* dan *anycast* Geo-DNS pada CDNs dilakukan dengan beberapa teknik algoritma yang digunakan yaitu *Round Robin*, *Weight Round Robin* (WRR) dan *Least Connection* (LC). Teknik algoritma yang digunakan akan membandingkan teknologi *Content Delivery Networks* (CDNs) dengan Non-

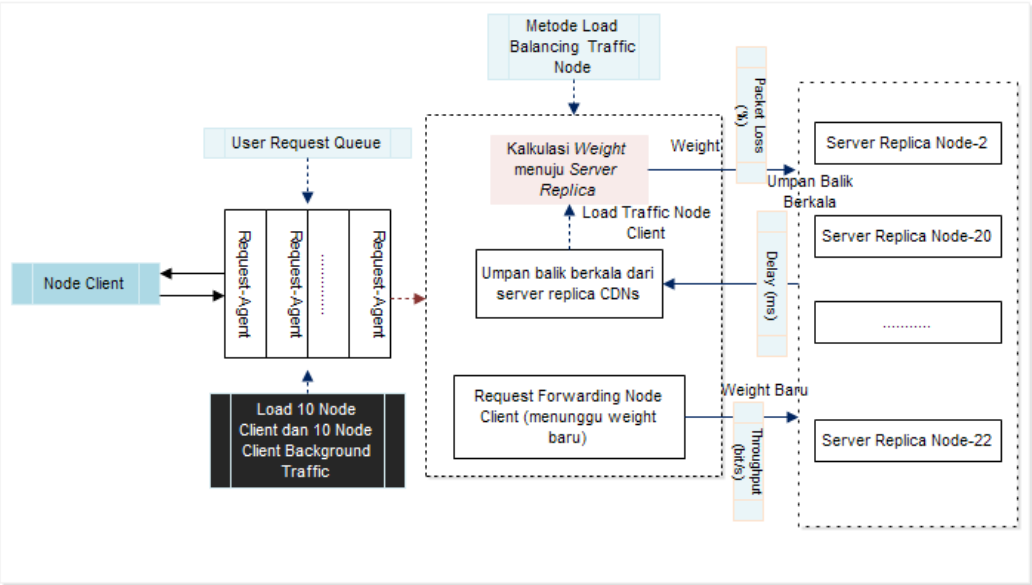
CDNs (*request single server*). Sebagai acuan tingkat kualitas layanan yang dijamin oleh infrastruktur *request* pada CDNs pada penelitian ini, maka digunakan standar internasional telekomunikasi sebagai acuan besaran parameter. Adapun parameter-parameter QoS yang diukur adalah *delay*, *packet loss* dan *throughput*. Proses analisis pada metode Non-CDNs pada *request single server* yang akan diterapkan dapat dilihat pada gambar III-1 sebagai berikut:



Gambar III-1 Proses analisis metode Non-CDN *Request Single Server*

Pada tabel III-1 dipetakan proses konfigurasi *node replica server* dan *client node* yang dialokasikan pada masing-masing *request node client*, prosesnya adalah menghubungkan masing-masing *client node* ke *server replica node* berdasarkan *single request*.

Proses analisis pada metode *load balancing* yang akan diterapkan dapat dilihat pada gambar III-2. Penerapan penjadwalan algoritma *load balancing* yang diterapkan dapat memanfaatkan sumber daya perutean paket *request* pengguna, yang digunakan untuk memastikan pemanfaatan yang tepat dari semua sumber daya algoritma penjadwalan pada perutean beban *traffic*, yang terdiri dari keseluruhan penerapan. Faktor ini harus dioptimalkan agar memiliki algoritma *load balancing* yang efisien.



Gambar III-2 Proses analisis metode *load balancing* CDNs

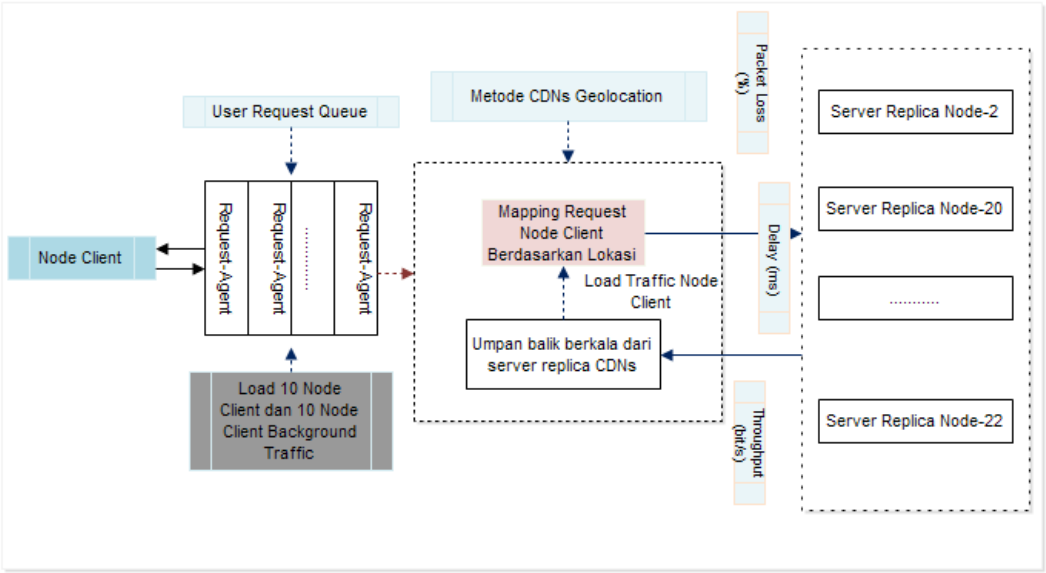
Kemudian nilai *quality of service* (QoS), akan dihitung dengan perbandingan tersebut mengacu pada standar *Quality of Service* (QoS) *Telecommunications and Internet Protocol Harmonization Over Networks* (TIPHON).

Mekanisme pembagian beban pada *node replica server* dengan menggunakan teknik *load balancing* akan didistribusikan pada beberapa jalur koneksi secara seimbang. Dengan tujuan teknik ini dapat memaksimalkan nilai nilai *quality of service* (QoS) dan memperkecil waktu tanggap dan menghindari *overload* pada salah satu jalur koneksi.

Selanjutnya pada parameter QoS dan optimalisasi *request* pengguna akan dipetakan dengan teknik *anycast Geolocation*, dimana teknik ini akan melayani pengguna berdasarkan *node replica server* terdekat dalam responnya.

Pada pemetaan wilayah *node replica server* dan *node client* akan dipetakan pada *request node client* dengan membagi beban *traffic* menjadi satuan *request* dan memutuskan *server* mana yang akan menerima *request* tersebut berdasarkan wilayahnya. Tujuannya adalah meminimalisasi jarak antara pengguna dengan *server* penyedia layanan. Sehingga penggunaan *bandwidth* dan pada nilai QoS

dapat diminimalisasi. Ditinjau dari nilai parameter-parameter *throughput, packet loss dan end-to-end delay*.



Gambar III-3 Proses analisis metode *Geolocation* CDNs

3.4.2 Teknik Perancangan dan Pengujian Data

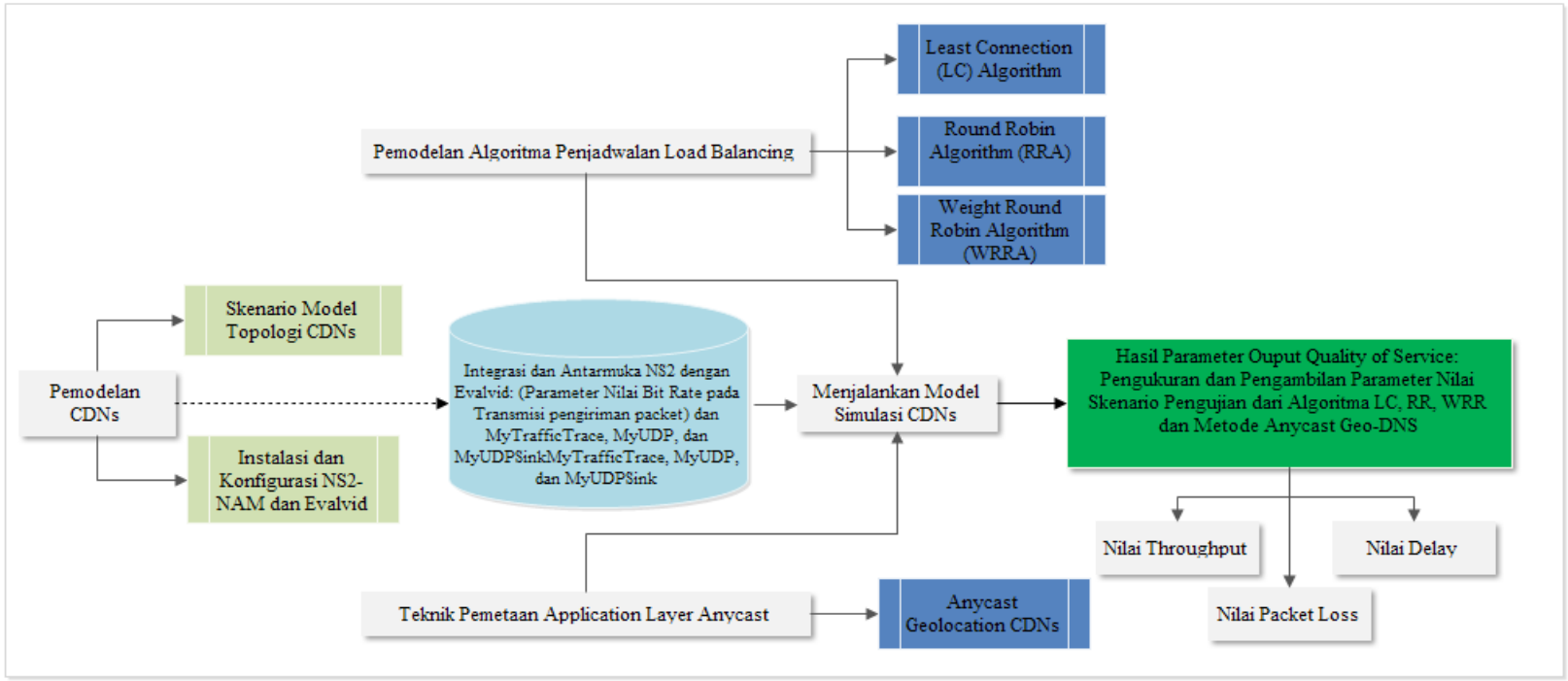
Pada proses perancangan yang dilakukan adalah sebagai berikut:

1. Pemodelan CDNs adalah Proses pemodelan pada tahap perancangan desain dan konsep pemilihan topologi jaringan yang akan diterapkan
2. Instalasi dan konfigurasi Network simulator (NS2) dan NAM pada teknik *load balancing* dan teknik *Geolocation Domain* dengan konsep pemodelan *Content Delivery Networks* (CDNs).
3. Skenario topologi model CDNs adalah perancangan akan didesain terlebih dahulu skenario topologi CDNs dari beberapa *node replica server* dan *node client*. Inisialisasi jumlah *node*, *agent*, *background traffic*, set parameter nilai *link*, ukuran *packet* pada Evalvid.
4. Integrasi NS2 dan Evalvid (Parameter Nilai *Bit Rate* pada Transmisi perutean pengiriman *packet*). Proses ini untuk mengevaluasi kualitas *packet* video yang dikirimkan melalui jaringan komunikasi simulasi. Selain mengukur parameter QoS (Quality of Service) dari jaringan yang

mendasari, seperti penundaan (*delay*), tingkat kehilangan dan jitter, dan throughput, dimana nilainya adalah [total *sent* data – total *lost* data] / *time*. Selanjutnya MyTrafficTrace, MyUDP, dan MyUDPSink, diimplementasikan antara NS2 dan EvalVid. Antarmuka ini dirancang untuk membaca file pelacakan video atau untuk menghasilkan data yang diperlukan untuk mengevaluasi kualitas video yang dikirimkan pada perutean transmisi antara *node client* dan *node replica server*.

5. Penerapan *load balancing* perutean akan diterapkan dengan metode Algoritma *Round Robin* (ARR), Algoritma *Weight Round Robin* (AWRR) dan Algoritma *Least Connection* (ALC). Perhitungan akan dilakukan pada masing-masing nilai *quality of service* (QoS), merekam parameter *ouput quality of service*. Pengukuran dan pengambilan parameter nilai skenario pengujian dari algoritma LC, RR, WRR.
6. Konsep optimalisai *anycast Geolocation* akan disimulasikan untuk mendapatkan parameter *quality of service* pada sebuah *host* ketika melakukan *request* pada *node server replica*. Pemetaan akan dilakukan pada *request host* tersebut, dengan memetakan *node server replica* terdekat untuk merespon. Disimulasikan pada *Network Simulator* 2 (NS2) dengan menggunakan dua jenis bahasa pemrograman, yaitu C++ akan digunakan sebagai inti proses simulasi dan Bahasa TCL untuk konfigurasi desain jaringan CDNs.
7. Merekam parameter *ouput quality of service* dengan pengukuran dan pengambilan parameter nilai skenario pengujian dari algoritma ALC, ARR, AWRR dan teknik metode *anycast* Geo-DNS
8. Hasil parameter dan optimalisasi metode algoritma penjadwalan *load balancing* dan *anycast* Geo-DNS CDNs.

Model: Teknik Perancangan Mekanisme Penjadwalan dan Optimalisasi Multi-Algoritma *Load Balancing* dan *Anycast* DNS pada *Content Delivery Network*.



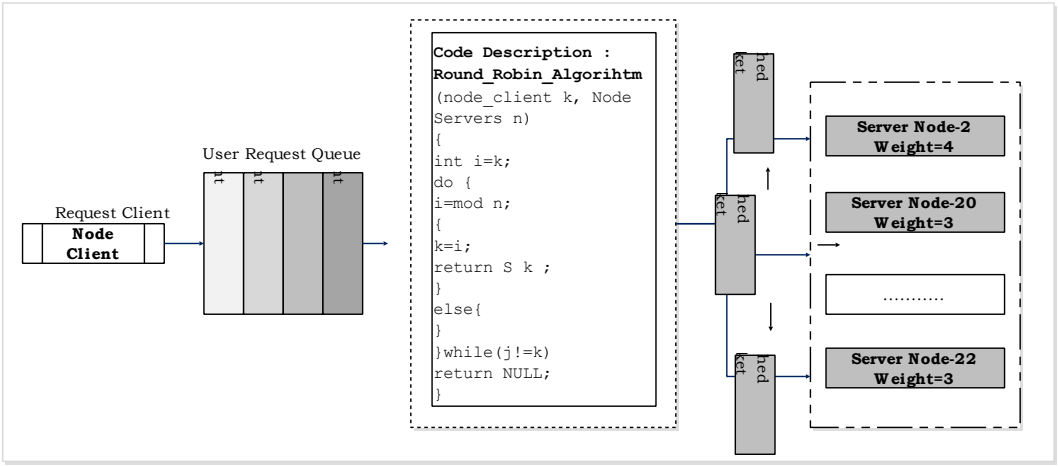
Gambar III-4. Teknik Perancangan Mekanisme Penjadwalan dan Optimalisasi Multi-Algoritma *Load Balancing* dan *Anycast* DNS pada *Content Delivery Network*

3.4.3 Teknik Pengujian Sistem

Teknik pengujian sistem akan dilakukan perbandingan *quality of service* (QoS) pada nilai *load balancing* dan *anycast* Geo-DNS, dengan tujuan mendistribusikan pekerjaan beban *node replica server* secara seimbang untuk memaksimalkan *throughput*, *latency* dan *packet loss*. Penulis akan mengevaluasi dan membandingkan dari tiga motede *load balancing* dengan melakukan perbandingan dari metode tersebut. Selanjutnya tahap kedua *anycast* Geo-DNS sebagai evaluasi dari proses respon *node replica server* pada *host* yang melakukan *request*.

Teknik pengujian sistem dari ke-empat metode akan diuraikan secara ringkas pada konsep dibawah ini.

1. Metode Penjadwalan Algoritma *Round Robin* (ARR)

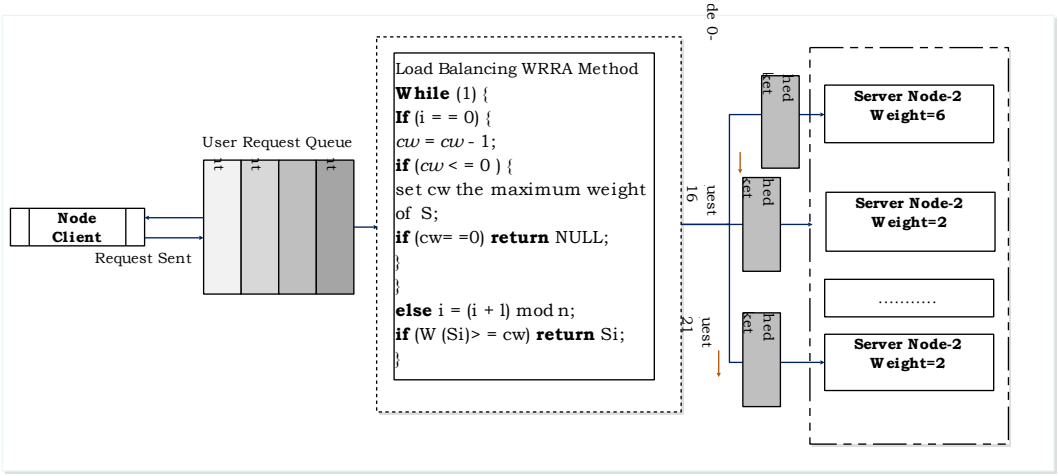


Gambar III-5. Metode Penjadwalan Algoritma *Round Robin* (ARR)

Ketika *client node* 0 melakukan *request* maka *load balancer* akan menghubungkan *client node* tersebut ke *node replica server* 2. *Request* dari *client node* kedua *load balancer* akan menghubungkannya *node replica server* 20. *Request* dari *client node* ketiga *load balancer* akan menghubungkannya ke *node replica server* 22. *Round robin* mengirimkan permintaan pengguna ke *node replica server* pertama dan ke yang terakhir berdasarkan *request*. Selanjutnya S_i adalah simpul yang dipilih terakhir.

2. Metode Penjadwalan Algoritma *Weight Round Robin* (AWRR)

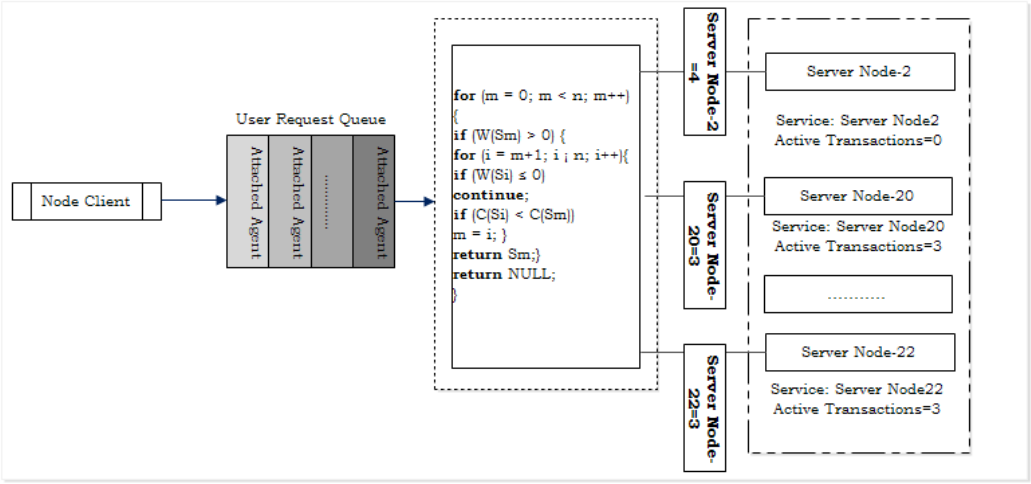
Penjadwalan WRR dapat memperlakukan simpul dari kapasitas pemrosesan yang berbeda. Setiap *node* dapat diberi bobot (Wi) dan nilai *integer* yang menunjukkan kapasitas pemrosesan, dengan bobot *default* 1.



Gambar III-6. Metode Penjadwalan Algoritma *Weight Round Robin* (AWRR)

Penjadwalan ini berfungsi bahwa *node server* S = (S2, S20, S22) Indeks *i* adalah *node replica server* yang dipilih terakhir dalam S, dan *cw* adalah bobot saat ini. Variabel *i* dan *cw* pertama diinisialisasi ke nol. jika semua W (Si) = 0, tidak ada *node replica server* yang tersedia, semua koneksi untuk *node replica server* akan di *drop*.

3. Metode Penjadwalan Algoritma *Least Connection* (LC)



Gambar III-7. Metode Penjadwalan Algoritma *Least Connection* (LC)

Metode *Least Connection* memperhitungkan transaksi aktif dipilih terlebih dahulu (yang tidak memiliki beban). Nilai parameter simulasi adalah:

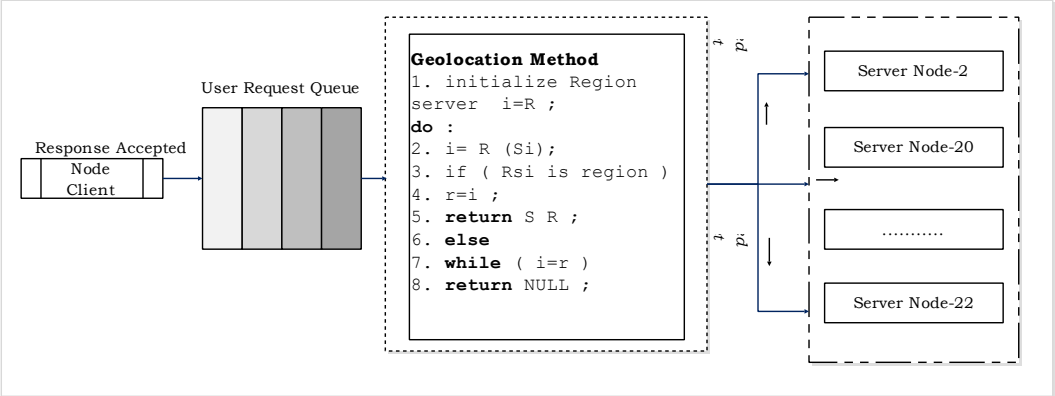
1. *Node replica server* (CDNs)-2 menangani 0 transaksi aktif.
2. *Node replica server* (CDNs)-20 menangani 3 transaksi aktif.
3. *Node replica server* (CDNs)-22 menangani 3 transaksi aktif.

1. *Node replica server* 2 menerima permintaan *client node* 0-1–2-3. dengan *node traffic client* ke *server* adalah= UDP 0-UDP 1-UDP9-UDP16.

2. *Node replica server* 20 menerima permintaan *client node* 4-5–6. dengan *node traffic client* ke *server* adalah= UDP 7-UDP 14-UDP19.

3. *Node replica server* 22 menerima permintaan *client node* 7-8–9. dengan *node traffic client* ke *server* adalah= UDP 8-UDP 15-UDP21.

4. Metode Teknik *Anycast Geolocation Domain Name System*



Gambar III-8. Metode Teknik *Anycast Geolocation Domain Name System*

Metode Teknik *Geolocation Domain Name System* (GDNS) menggunakan *anycast* domain untuk mendistribusikan replika geografis dari server nama DNS. Berdasarkan skema pada gambar III-5, dalam skema yang diterapkan pada simulasi, *node client* memilih layanan untuk setiap koneksi masuk dengan memilih *server* pada masing-masing *geolocation*. koneksi diteruskan dengan *Geolocation* A terdapat *node server* 2 yang melayani permintaan dari *node client* 0,1,9 dan 16. *Geolocation* B terdapat *node replica server* 22 yang melayani

permintaan dari *client* 7, 8, dan 21. *Geolocation* C terdapat *node replica server* 20 yang melayani permintaan dari *client* 14, 15 dan 19.

Pengujian Mekanisme Penjadwalan dan Optimalisasi Multi-Algoritma *Load Balancing* dan *Anycast* pada *Content Delivery Network* (CDN), mengacu pada standar *Quality of Service* (QoS) *Telecommunications and Internet Protocol Harmonization Over Networks* (TIPHON).

Dalam lingkup jaringan *Content Delivery Network* (CDN), pengukuran QoS akan dilakukan seberapa baik jaringan pada penerapan CDNs pada metode *load balancing* dan *geolocation*. Hasil nilai parameter QoS digunakan akan mengukur sekumpulan atribut kinerja yang telah dispesifikasikan dengan suatu layanan multimedia yaitu video. QoS menawarkan kemampuan untuk mendefinisikan atribut-atribut layanan jaringan CDNs yang disediakan, baik secara kualitatif maupun kuantitatif.

Parameter-parameter nilai general aspek dari *Quality of Service* (QoS) akan diuraikan pada tabel berikut.

1. Indeks *Packet Loss Level*

   Packet loss merupakan nilai indeks parameter yang menggambarkan suatu kondisi lingkungan jaringan yang menunjukkan jumlah total paket yang hilang, dapat terjadi karena *collision* dan *congestion* pada komunikasi data. Tabel III-1 menunjukkan peringkat dan kriteria *packet loss*.

   - Paket Loss biasanya dihitung berdasarkan paket id.
   - Menghitung kerugian, jaringan menyediakan id paket.
   - ID paket ini sering diambil dari IP melalui jaringan.

   Persamaan *packet loss*:

   packet loss $PL_T = 100 \frac{\text{nT Received}}{\text{nT Sent}}$ Dimana:

   nT_{Sent}: *number of type* T *packets sent*

   $nT_{received}$: *number of type* T *packets received*

2. Indeks *Throughput*

Kecepatan transmisi indeks (*rate*) transfer data efektif, yang diukur dalam bps. Throughput merupakan jumlah total kedatangan paket yang mencapai tujuan, yang diamati pada tujuan selama interval waktu tertentu dibagi oleh durasi interval waktu tersebut. Persamaan perhitungan *throughput*:

Tabel III-2. Indeks *Throughput*

Kategori Degradasi	*Throughput* %	Indeks
Sangat Bagus	100	4
Bagus	75	3
Sedang	50	2
Buruk	>25	1

Dengan Persamaan sebagai berikut:

$$Throughput\ \frac{\sum \text{jumlah data dikirim } (bit)}{\text{waktu pengiriman data } (s)}$$

3. Indeks *Delay*

Delay adalah nilai indeks waktu yang dibutuhkan paket dari sumber untuk mencapai tujuan, karena adanya antrian (*queue*), atau mengambil rute pada transmisi yang lain untuk menghindari kemacetan. Peringkat dan kriteria indeks *delay* diperlihatkan pada Tabel III-3 sebagai berikut:

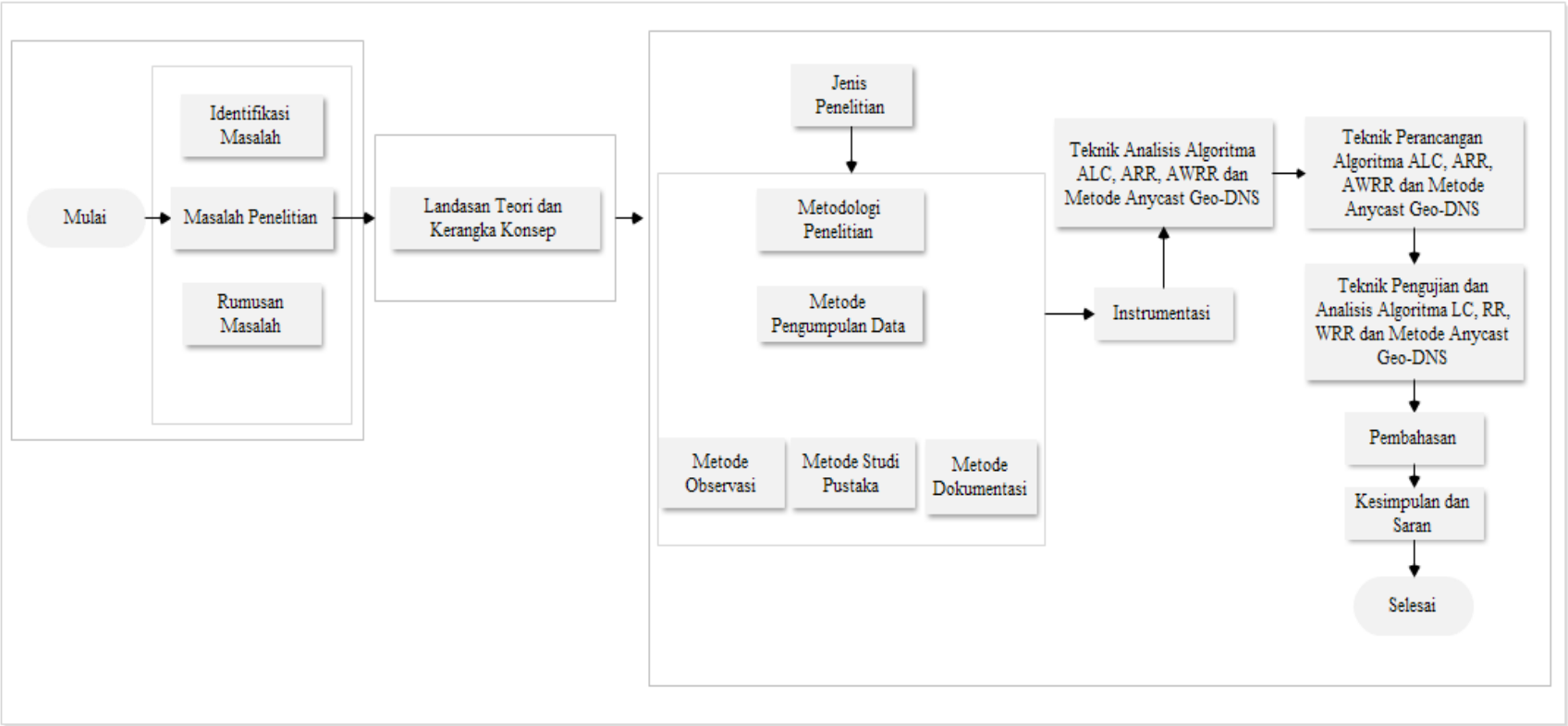
Tabel III-3. Indeks *Packet Delay*

Kategori Degradasi	*Packet Delay* %	Indeks
Sangat Bagus	<150	4
Bagus	150 s/d 300	3
Sedang	300 s/d 450	2
Buruk	>450	1

Dengan Persamaan sebagai berikut:

$$Delay = \frac{\text{jumlah } packet \text{ pengiriman data } (bit)}{jumlah\ packet\ (bit/s)}$$

Langkah-langkah penelitian: Model: Teknik Perancangan Mekanisme Penjadwalan dan Optimalisasi Multi-Algoritma *Load Balancing* dan *Anycast* DNS pada *Content Delivery Network.*



Gambar III-9. Langkah-langkah Penelitian

3.5 Langkah-langkah Penelitian

Dalam penelitian ini keseluruhan proses yang dilalui harus melalui beberapa tahapan. Tahapan yang dilakukan meliputi: masalah penelitian, tinjauan studi, metodologi, perancangan sistem, pembuatan model, dan pengujian sistem. Langkah- langkah pada tahapan pelaksanaan penelitian sebagai berikut:

1. **Identifikasi Masalah**

   Langkah awal dari penelitian ini adalah dengan mengidentifikasi masalah pada teknologi CDNs. Kriteria parameter optimalisasi dan peninjauan mekanisme serta penerapan *load balancing pada Content Delivery Networks* (CDNs), Selanjutnya hasil dari langkah ini adalah rumusan masalah.

2. **Rumusan Masalah**

   - Bagaimana menentukan algoritma *load balancing* di dalam jaringan *Content Delivery Network* (CDNs) yang diterapkan. Dengan mendistribusikan beban *traffic* pada permintaan pengguna, pada jalur perutean transmisi (*client node*) ke tujuan (*node replica server*). Dengan tujuan meminimalisasi nilai *Quality of Service* (QoS) berdasarkan parameter-parameter *throughput, packet loss* dan *end-to-end delay*.
   - Bagaimana mengarahkan dan mencari jalur terbaik pada *request* sebuah *host* pada *node replica server* terdekatnya, dengan teknik *geolocation* domain di dalam jaringan *Content Delivery Network* (CDNs) yang diterapkan. Dengan meminimalisasi jarak antara pengguna dengan *server* penyedia layanan. Ditinjau dari parameter-parameter *throughput, packet loss* dan *end-to-end delay*. Sehingga penggunaan *bandwidth* pada indeks *throughput* dapat dimaksimalkan dengan kapasitas yang tersedia. Kemudian nilai indeks *packet loss* dan *delay* dapat diminimalisasi.
   - Bagaimana perbandingan sumber daya *Non-Content Delivery Network* (Non-CDNs) dengan *Content Delivery Network* (CDNs) dengan metode penerapan *load balancing* dan *anycast geolocation domain*.

3. **Tinjauan Studi**

   Tinjauan studi ini dilakukan untuk mendapatkan teori yang berkaitan dengan topik bahasan teknologi CDNs, Algoritma penjadwalan dan pendistribusian

anycast Geo-DNS pada *node replica server* terdistribusi. Hasil dari langkah ini adalah literatur yang terkait dengan rumusan masalah.

- Mengumpulkan data dan variabel yang dibutuhkan pada teknologi CDNs.
- Mempelajari konsep teknik *load balancing*.
- Mempelajari teknik penjadwalan antrian *Least Connection* (LC), *Round Robin* (RR) dan *Weighted Round Robin* (WRR)).
- Mempelajari metode pemodelan pada *anycast* Geo-DNS berdasarkan teknik *anycast* DNS.

4. Jenis Penelitian

Pada tahap jenis penelitian akan dilakukan pemilihan jenis penelitian yang digunakan: penelitian eksperimen (*Research Experiment*), Metode yang dilakukan dalam penelitian dengan mempelajari teori-teori dan konsep mengenai teknologi CDNs saat ini dan teknik algoritma penjadwalan dan teknik *anycast* Geo-DNS berdasarkan *anycast* Geo-DNS. Sebagai penyeimbangan beban *traffic* pada sumber daya content delivery network (CDN) berupa nilai *quality of service* (QoS) termasuk didalamnya *paket loss, delay* dan *throughput.* Klasifikasi dilakukan pada konsep dan teknik algoritma penjadwalan: *Least Connection* (LC), *Round Robin* (RR) dan *Weight Round Robin* (WRR).

5. Metode Pengumpulan Data

Pada tahap pengumpulan data dilakukan pemilihan metode pengumpulan data yang akan digunakan dalam penelitian ini:

- Metode studi pustaka.
- Metode observasi.
- Metode dokumentasi.

6. Instrumentasi

Pada tahap instrumentasi dilakukan pemilihan instrumen yang akan digunakan dalam pengumpulan data:

- Instrumentasi dalam tahap pengumpulan data dengan metode observasi.

- Instrumentasi dalam tahap studi pustaka.
- Instrumentasi Dokumentasi.

7. Teknik Perancangan Sistem

Pada tahap ini dilakukan perancangan terhadap pemodelan teknik dan parameter konsep yang akan diterapkan.

8. Pembuatan Model

Pada tahap ini akan dilakukan pemodelan: Mekanisme Parameter dan Optimalisasi *Load Balancing* CDNs dengan *anycast* DNS dan Multi-Algoritma.

9. Teknik Pengujian Sistem

Pada tahap ini pemodelan akan dilakukan pengujian dari hasil *output* parameter pengujian Non-CDNs, *load balancing* CDNs dan *Geolocation* CDNs.

3.6 Jadwal Penelitian

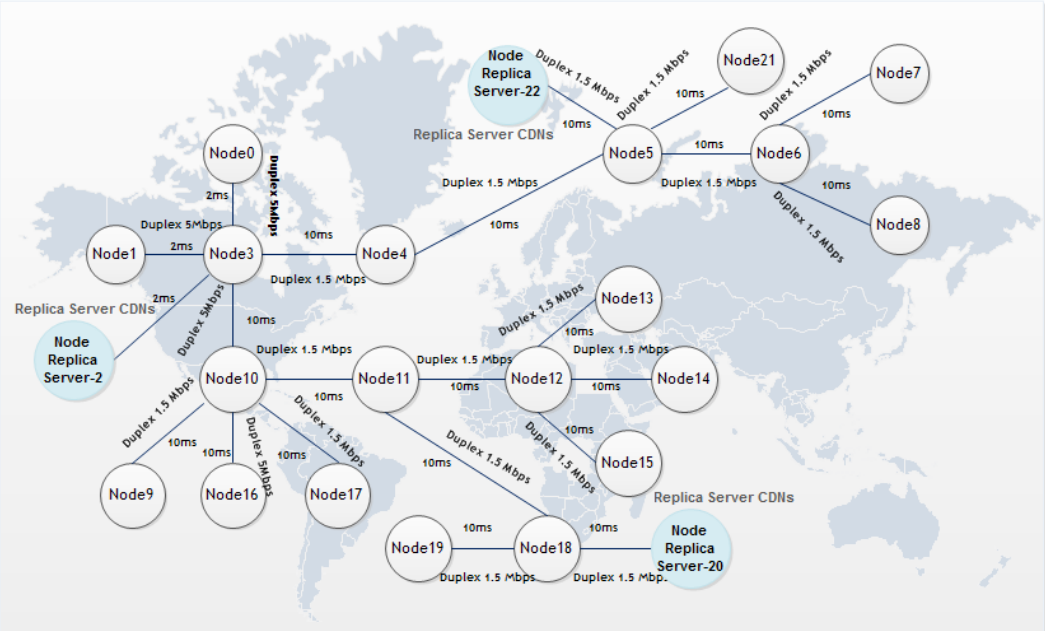
No.	Kegiatan	JANUARI				FEBRUARI				MARET				APRIL				MAY				JUNE				JULY			
		1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
1	Pencarian dan pemilihan objek penelitian																												
2	Studi melalui media internet, jurnal internasional dari objek penelitian																												
3	Perumusan masalah penelitian																												
4	Penentuan topik dan pembimbing tesis																												
5	Pengumpulan bahan literatur referensi																												
6	Penyusunan kerangka landasan pemikiran (tinjauan pustaka/studi sampai dengan kerangka konsep dan hipotesis)																												
7	Penyusunan metodologi penelitian (jenis penelitian metode sampling, metode pengumpulan data, instrumentasi, teknik analisis, perancangan, pengujian data)																												
8	Pembuatan Laporan Penelitian																												

Tabel III-10. Jadwal Penelitian

BAB IV
PEMBAHASAN

4.1. Perancangan *Network* Topologi *Single Server* dan *Load Balancing Content Delivery Network* (CDN)

Desain sistem simulasi dan spesifikasi perancangan, serta modul *traffic* pada penulisan penelitian dimodelkan seperti gambar IV-1. Desain konfigurasi jaringan yang dimodelkan menyerupai jaringan *Content Delivery Network* (CDN) seperti ditunjukkan pada Gambar IV-1, dimana jaringan CDN yang dimodelkan dengan jaringan CDN terdiri dari 3 *node replica server* dan 20 *client node*. Dimana dari 20 *client node* yang melakukan permintaan layanan ada 10 *client node*. Sementara 10 *client node* dijadikan *background traffic*.



Gambar IV.1. Desain Perancangan Model Topologi Mekanisme Penjadwalan Dan Optimalisasi Multi-Algoritma *Load Balancing dan Anycast* DNS Pada *Content Delivery Network* (CDN)

Pada model *layout* simulasi pada gambar IV-1. memiliki spesifikasi perancangan sebagai berikut:

Tabel IV-1. Spesifikasi Perancangan Topologi *Content Delivery Network (CDN)*

No.	***Node Server***	***Node***	***Node Traffic***	***Link Mode***	***Bandwidth***	***Propagation Delay***
1	***Node Server* 2**	*Client Node*	*Node*-2	*Duplex*	1.5 dan 5 Mbps	2 dan 10 ms
2		*Client Node*	*Node*-3	*Duplex*	1.5 dan 5 Mbps	3 dan 10 ms
3		*Client Node*	*Node*-4	*Duplex*	1.5 dan 5 Mbps	4 dan 10 ms
4		*Client Node*	*Node*-5	*Duplex*	1.5 dan 5 Mbps	5 dan 10 ms
5	***Node Server* 20**	*Client Node*	*Node*-6	*Duplex*	1.5 dan 5 Mbps	6 dan 10 ms
6		*Client Node*	*Node*-10	*Duplex*	1.5 dan 5 Mbps	7 dan 10 ms
7		*Client Node*	*Node*-11	*Duplex*	1.5 dan 5 Mbps	8 dan 10 ms
8		*Client Node*	*Node*-12	*Duplex*	1.5 dan 5 Mbps	9 dan 10 ms
9	***Node Server* 22**	*Client Node*	*Node*-13	*Duplex*	1.5 dan 5 Mbps	10 dan 10 ms
10		*Client Node*	*Node*-17	*Duplex*	1.5 dan 5 Mbps	11 dan 10 ms
11		*Client Node*	*Node*-18	*Duplex*	1.5 dan 5 Mbps	12 dan 10 ms
12		*Client Node*	*Node*-20	*Duplex*	1.5 dan 5 Mbps	13 dan 10 ms

Tabel IV-2. Spesifikasi Modul *traffic* Pengiriman Paket Evaluasi Video (*evalvid*) Pada Transmisi *Content Delivery Network* (CDN)

Parameter	**Keterangan *File***
Nama *traffic video*	bridge_cif.yuf
Kecepatan *frame*	30fps
Tipe *Frame*	IPP
Codec	MPEG4
bit rate (bps)	289197
Sender	MP4trace RTP/UDP
Trace File	IP *packet dump sender*
GPAC	*libgpac_static.lib*

4.2. Instalasi pada modul-modul tool simulasi Network Simulator (NS) 2

Simulasi pada penelitian ini dengan menggunakan sistem operasi Linux Ubuntu 16.04 LTS dan NS2.35 beserta modul evaluasi *traffic video* (*evalvid*). Proses-proses instalasi implementasi adalah sebagai berikut:

4.2.1. Instalasi *Network Simulator* (NS) 2

Pada tahap instalasi dilakukan dengan *command line* pada *update* dan instal *packages-package* pada modul NS-2 dan *traffic video evalvid*.

```
:sudo apt-get update
:sudo apt-get install gcc-5.4.0
:sudo apt-get install build-essential autoconf automake libxmu-dev
:tar xvzf ns-allinone-2.35.tar.gz -C /home/ajn16lts
:./install
```

Selanjutnya penambahan baris-baris ini perintah dibawah pada *home* direktori ke yang sama pada sistem komputer dan pastikan untuk mengubah nomor versi pada *package* yang akan diterapkan.

Setelah proses instalasi dilakukan, selanjutnya adalah melakukan *edit environment* pada *file* .bashrc yang terletak pada *home/ajn16lts/ns2all-in-2.35*.

```
# LD_LIBRARY_PATH
OTCL_LIB=/home/ajn16lts/ns-allinone-2.35/otcl-1.14
NS2_LIB=/home/ajn16lts/ns-allinone-2.35/lib
X11_LIB=/usr/X11R6/lib
USR_LOCAL_LIB=/usr/local/lib
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$OTCL_LIB:
$NS2_LIB:$X11_LIB:$USR_LOCAL_LIB
# TCL_LIBRARY
TCL_LIB=/home/ajn16lts/ns-allinone-2.35/tcl8.5.10/library
USR_LIB=/usr/lib
export TCL_LIBRARY=$TCL_LIB:$USR_LIB
# PATH
XGRAPH=/home/ajn16lts/ns-allinone-2.35/bin:/home/ajn16lts/
ns-allinone-2.35/tcl8.5.10/unix:/home/ajn16lts/ns-allinone-2.35
/tk8.5.10/unix
#the above two lines beginning from xgraph and ending
with unix should come on the same line
NS=/home/ajn16lts/ns-allinone-2.35/ns-2.35/
NAM=/home/ajn16lts/ns-allinone-2.35/nam-1.15/
PATH=$PATH:$XGRAPH:$NS:$NAM
```

4.2.2. Instalasi Modul Evaluasi Video (EvalVid)

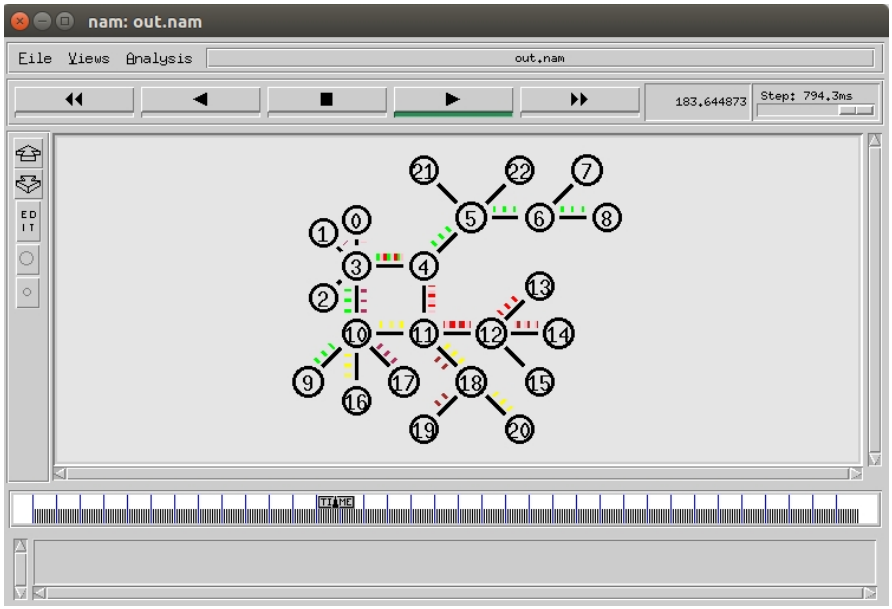
Merupakan sebuah aplikasi yang dapat digunakan untuk evaluasi unjuk kerja transmisi video dengan struktur modular, sehingga dapat disesuaikan dengan lingkungan medium transmisi yang digunakan. Modifikasi modul-modul EvalVid akan diuraikan dengan langkah-langkah sebagai berikut:

Bersamaan dengan ns-2, evaluasi transmisi video akan menggunakan EvalVid, kerangka kerja dan *toolkit* untuk penilaian terpadu pada kualitas transmisi video. Penawaran *EvalVid* antarmuka yang mudah ke alat simulasi jaringan dalam arti semua interaksi dengan mekanisme transport jaringan yang mendasari dilakukan melalui dua *trace file*. Jadi relatif mudah untuk mengintegrasikan *EvalVid* dalam berbagai simulasi desain topologi yang akan diterapkan. Berdasarkan lingkungan simulasi ini, eksperimen dengan berbagai jenis topologi jaringan serta lalu lintas data yang berbeda skenario pada masing-masing metode, dapat dilakukan dengan tujuan utama untuk melakukan investigasi menyeluruh terhadap evaluasi QoS video dari ujung ke ujung transmisi melalui model jaringan.

4.3. Pengujian dan Analisis Simulasi Perancangan Penelitian

Pengujian data hasil dari penelitian mekanisme penjadwalan dan optimalisasi multi-algoritma *load balancing* dan *anycast* DNS pada *Content Delivery Network* (CDN), akan merepresentasikan parameter nilai QoS yang akan diperoleh dari hasil simulasi yang sudah dilakukan. Data-data tersebut berupa *trace file* utama simulasi pada pengiriman dan penerimaan paket-paket MyUDP.

Hasil *trace file* tersebut akan diolah untuk mendapatkan parameter *packet loss, delay dan throughput*. hasil dari *trace file* inilah yang akan dijadikan parameter perbandingan yang telah ditentukan dan juga berdasarkan hasil pengamatan. Kemudian divisualisasikan dalam bentuk grafik untuk mengetahui perbedaan pada setiap kondisi pada masing-masing parameter. Berikut dibawah pada gambar IV.2 hasil simulasi *trace file* pada desain Algoritma *Load Balancing Geolocation* Domain pada *Content Delivery Network* (CDN). Hasil simulasi pada NS-2 akan menghasilkan *file* yang berekstensi txt. Data *file* tersebut dikonversi ke ms.*excel* untuk menghitung nilai *Quality of Service* dari masing-masing metode yang diterapkan.



Gambar IV-2. SimulasiModel Topologi Mekanisme Penjadwalan Dan Optimalisasi Multi-Algoritma *Load Balancing* dan *Geolocation* pada CDN dengan NAM NS2.

4.3.1. Penerapan Metode Penjadwalan Load Balancing

Penerapan metode penjadwalan *load balancing* pada jaringan CDN ditempatkan diantara *client* dan *node replica server* seperti yang ditunjukkan pada Gambar IV.2. Ketika ada *client node* yang ingin terhubung ke *node replica server* maka metode penjadwalan *load balancing* akan memilih *server* mana yang akan melayani *client node* tersebut dengan salah satu dari tiga algoritma yaitu *Least Connection* (LC), *Round Robin* (RR), dan *Weighted Round Robin* (WRR) dan metode *geolocation* domain.

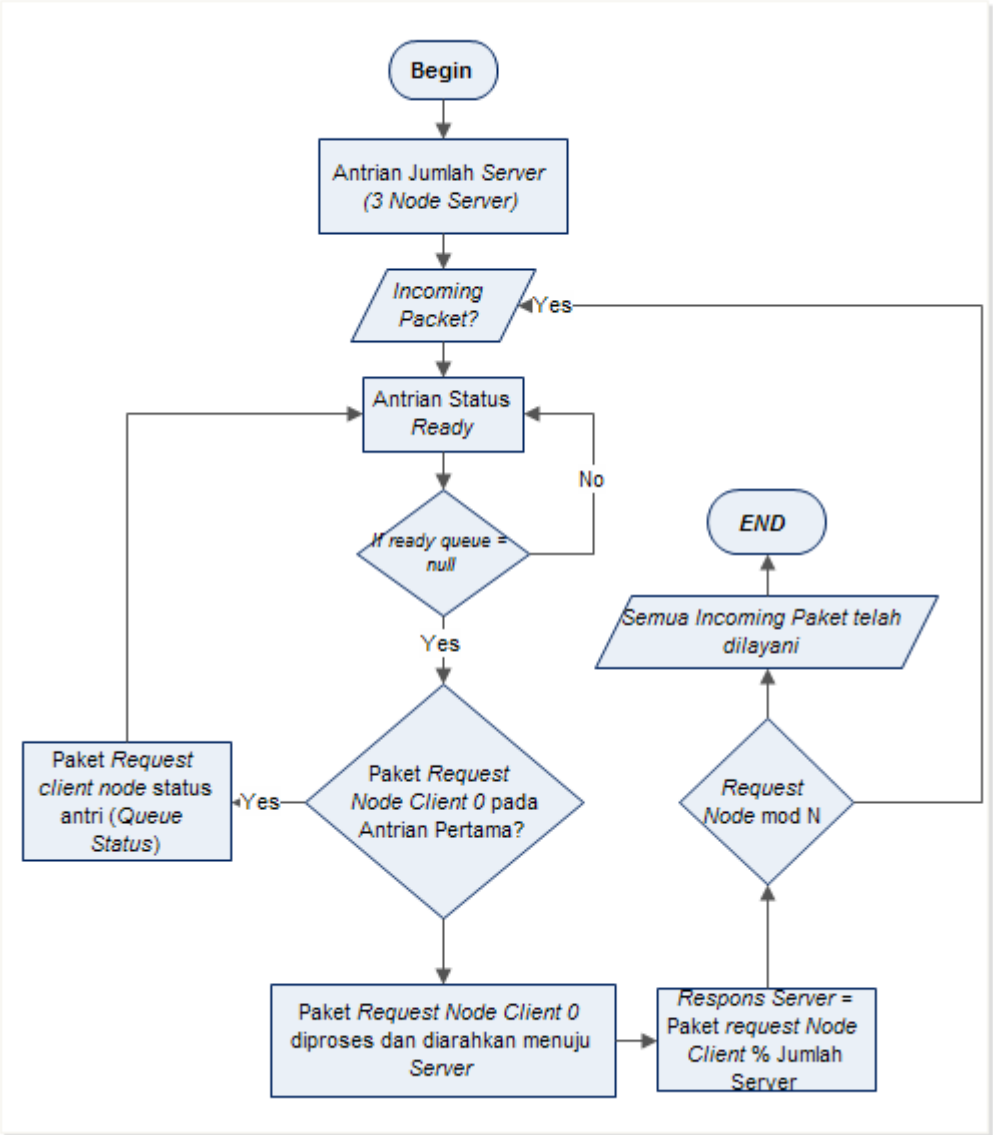
4.3.1.1. Metode Penjadwalan Algoritma *Round Robin* (ARR)

Metode *load balancing* dengan algoritma *round robin* menghubungkan *client node* dengan *server node replica* berdasarkan waktu kedatangan dan urutan *node replica server* yang ada. Ketika *client node* 0 melakukan *request* maka *load balancer* akan menghubungkan *client node* tersebut ke *node replica server* 1. *Request* dari *client node*

kedua *load balancer* akan menghubungkannya ke *server node replica* 2. *Request* dari *client node* ketiga *load balancer* akan menghubungkannya ke *node replica*

server replica 3. *Round robin* mengirimkan permintaan pengguna ke *node replica server* pertama dan ke yang terakhir berdasarkan *request*. Itu berarti jika S_i adalah simpul yang dipilih terakhir.

Berikut representasi *flowchart* dari algoritma *load balancing round robin* dan penjabaran penjelasan diatas:



Gambar IV-3. *Flowchart* Algoritma *Round Robin* (antrian dan penjadwalan *load balancing*)

Tabel IV-3. Tabel pemetaan posisi *server node replica* berdasarkan metode algoritma *Round Robin* (ARR)

Client Node	***Packet Request Node Client***	***Server Receive***	***Sequence Server***	***Current Weight Server***
Client Node0	Client Node0	Node Server Replica2	Node Server Replica0	Node Replica Server 2=4
Client Node1	Client Node9	Node Server Replica20	Node Server Replica1	
Client Node2	Client Node16	Node Server Replica22	Node Server Replica2	
Client Node3	Client Node19	Node Server Replica2	Node Server Replica0	Node Replica Server 20=3
Client Node4	Client Node7	Node Server Replica20	Node Server Replica1	
Client Node5	Client Node8	Node Server Replica22	Node Server Replica2	
Client Node6	Client Node16	Node Server Replica2	Node Server Replica0	Node Replica Server 22=3
Client Node7	Client Node21	Node Server Replica20	Node Server Replica1	
Client Node8	Client Node9	Node Server Replica22	Node Server Replica2	
Client Node9	Client Node14	Node Server Replica2	Node Server Replica0	

Pada tabel IV-3 dipetakan proses konfigurasi *node server replica* dan *client node* yang dialokasikan pada masing-masing *request node client*, prosesnya adalah menghubungkan masing-masing *client node* ke *node server replica* ke berdasarkan konsep *first in first out*.

Ketika *client node* 0 melakukan *request* maka *load balancer* akan menghubungkan *client node* tersebut ke *node server replica* 0. Urutan pada *server* yaitu dimulai dari *node server replica* 0, *node server replica* 20 dan *node server replica* 22. *Request* dari *client node* pertama *load balancer* akan menghubungkannya ke *node server replica* 20. *Request* dari *client node* kedua *load balancer* akan menghubungkannya ke *server node* 22. *Round robin* mengirimkan permintaan pengguna ke *node server replica* pertama dan ke yang terakhir berdasarkan *request*, sampai keseluruhan *request* dari *client node* terpenuhi.

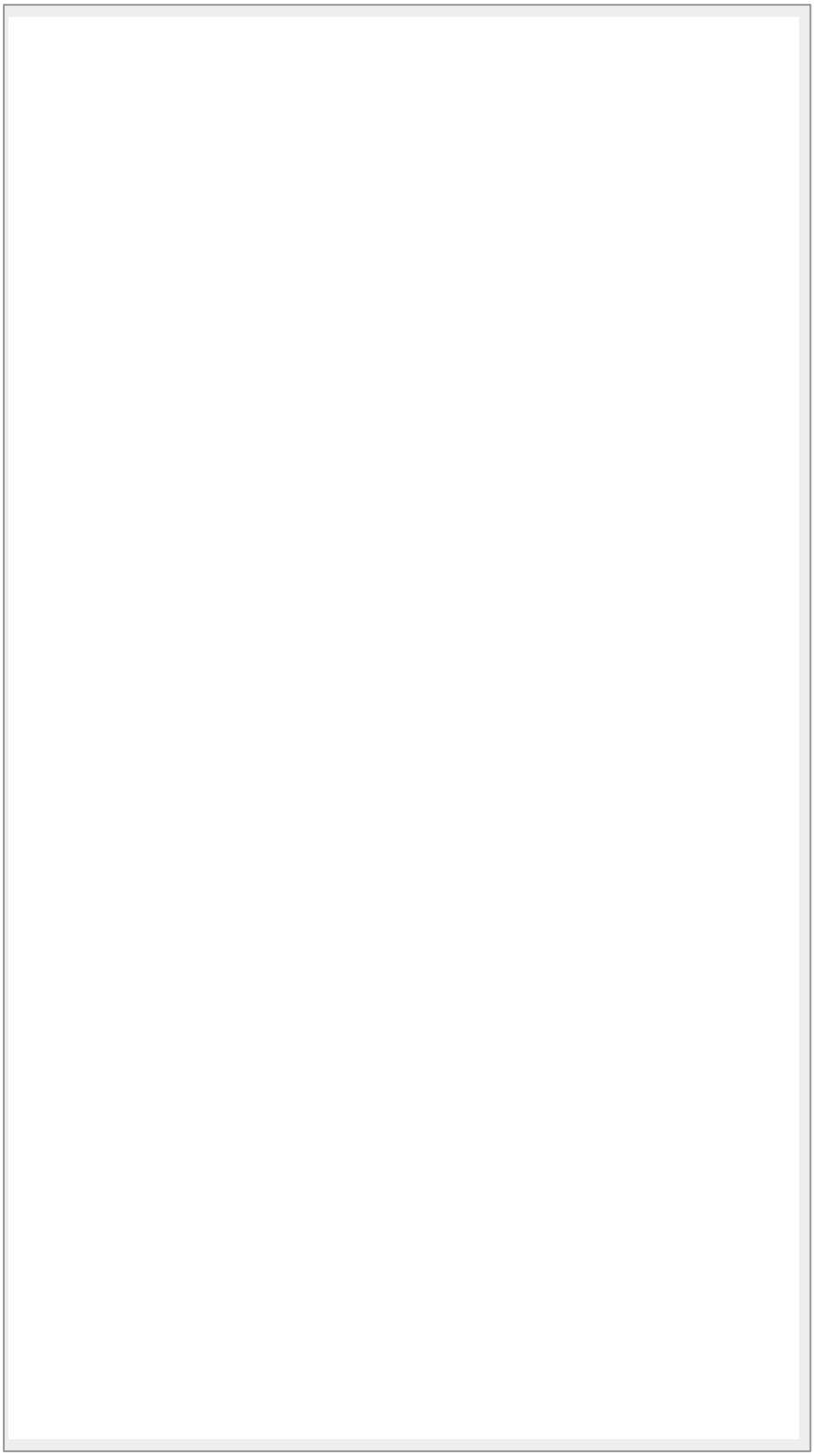
Pseudocode algoritma ARR adalah didefinisikan seperti gambar IV.4 sebagai berikut:

```
Pseudocode ARR didefinisikan:
Output: select the replica server to be assigned task.
Round_Robin_Algorithm(node_client k, node replica
server n)
1. Settings and initialize variables i=k ;
do :
2. i=mod n ;
3. k=i ;
4. return S k ;
5. else nothing to handle ;
6. while ( I ! =k )
7. return NULL ;
Code Description :
Round_Robin_Algorihtm(node_client k, node replica
server n)
{
int i=k;
do {
i=mod n;
{
k=i;
return S k ;
}
else{
}
}while(j!=k)
return NULL;
}
```

Gambar IV-4. *Pseudocode* Algoritma Round Robin
(antrian dan penjadwalan *load balancing*)

Berikut dibawah ini akan diuraikan metode *load balancing* round robin dan dengan perhitungan pada desain topologi yang diterapkan:

Berdasarkan perancangan simulasi pada *request client node* menuju masing-masing *node server replica*, berikut pada gambar VI.5 dijelaskan bagaimana pemetaan untuk klasifikasi *request* tersebut pada algoritma *round robin*.



Gambar IV-5. Perhitungan Algoritma *Round Robin* (antrian dan penjadwalan *load balancing*)

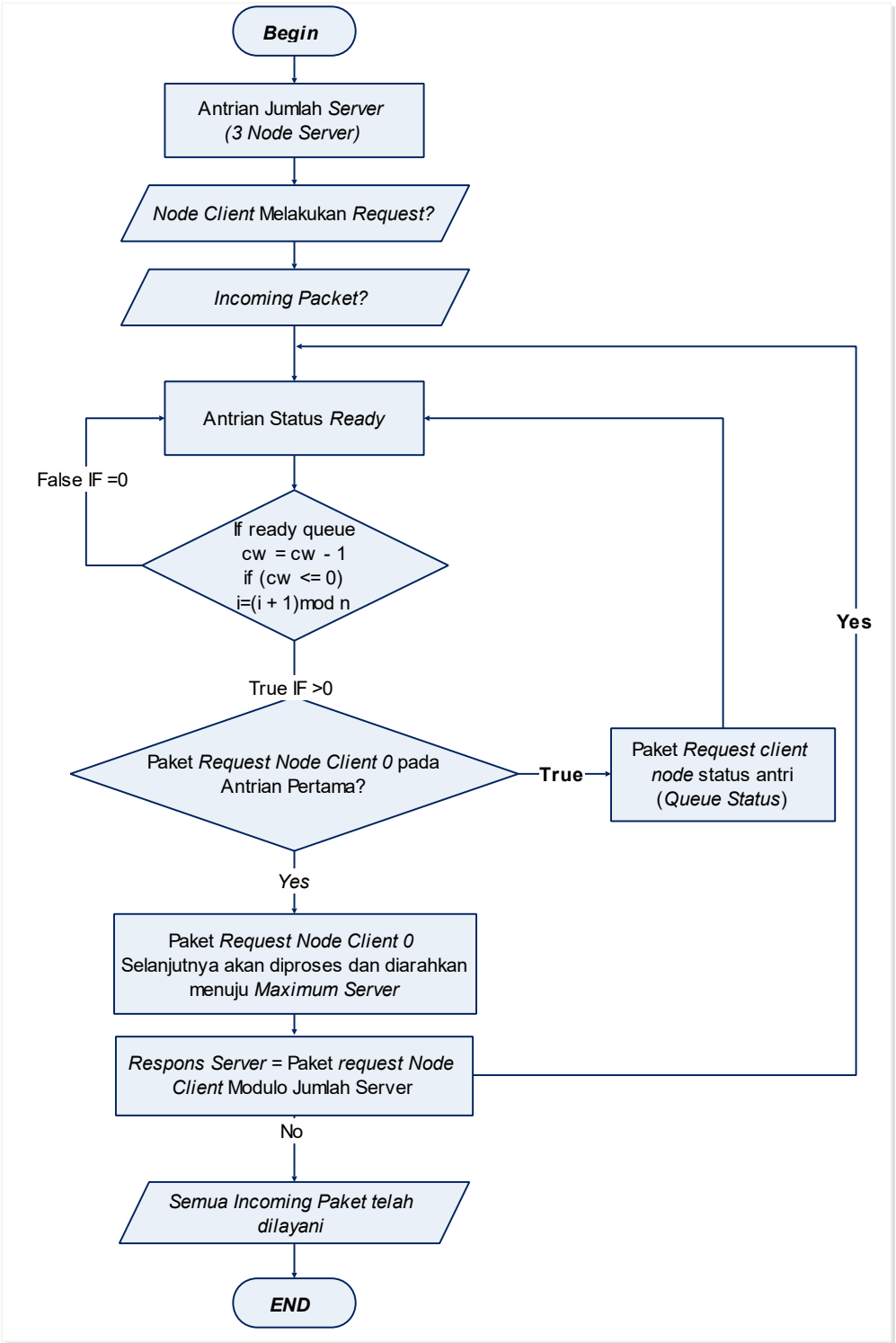
Tabel IV-4. Implementasi metode simulasi penjadwalan algoritma *Round Robin* (ARR), dijelaskan pada *script* TcL:

#Melampirkan traffic to *Node Server*-2: *Node Server*-20: *Node Server*-22
#*create source traffic client node to server node*
#*Create a UDP agent and attach it to node* n0
set udp_0 [new Agent/myUDP] $udp_0 set_filename udpSend0
$udp_0 set packetSize_ 1052 $ns attach-agent $n2 $udp_0
set udp_3 [new Agent/myUDP] $udp_3 set_filename udpSend1
$udp_3 set packetSize_ 1052 $ns attach-agent $n2 $udp_3
set udp_6 [new Agent/myUDP] $udp_6 set_filename udpSend7
$udp_6 set packetSize_ 1052 $ns attach-agent $n2 $udp_6
set udp_9 [new Agent/myUDP] $udp_9 set_filename udpSend8
$udp_9 set packetSize_ 1052 $ns attach-agent $n2 $udp_9
set udp_1 [new Agent/myUDP] $udp_1 set_filename udpSend9
$udp_1 set packetSize_ 1052 $ns attach-agent $n20 $udp_1
set udp_4 [new Agent/myUDP] $udp_4 set_filename udpSend14
$udp_4 set packetSize_ 1052 $ns attach-agent $n20 $udp_4
set udp_7 [new Agent/myUDP] $udp_7 set_filename udpSend15
$udp_7 set packetSize_ 1052 $ns attach-agent $n20 $udp_7
set udp_2 [new Agent/myUDP] $udp_2 set_filename udpSend16
$udp_2 set packetSize_ 1052 $ns attach-agent $n22 $udp_2
set udp_5 [new Agent/myUDP] $udp_5 set_filename udpSend19
$udp_5 set packetSize_ 1052 $ns attach-agent $n22 $udp_5
set udp_8 [new Agent/myUDP] $udp_8 set_filename udpSend21
$udp_8 set packetSize_ 1052 $ns attach-agent $n22 $udp_8

4.3.1.2 Metode Penjadwalan Algoritma *Weight Round Robin* (WRR)

Algoritma *weighted round-robin* adalah pengembangan lebih lanjut dari algoritma *round-robin*. Algoritma ini dapat mempertimbangkan beban *node server replica* berdasarkan kapasitasnya permintaan dari *client node*. *Pseudocode* algoritma WRR adalah didefinisikan seperti gambar 2 sebagai berikut:

Berikut representasi *flowchart* dari algoritma *load balancing round robin* dan penjabaran penjelasan diatas:



Gambar IV-6. *Flowchart* Algoritma *Weight Round Robin* (antrian dan penjadwalan *load balancing*)

Tabel IV-5. Tabel pemetaan posisi *server* berdasarkan metode algoritma *Weight Round Robin* (AWRR)

Client Node	***Packet Request***	***Server Receive***	***Sequence Server***	***Current Weight Server***
Client Node0	*Client Node0*	*Node Server Replica2*	*Node Server Replica0*	*Node Server Replica 2=6*
Client Node1	*Client Node1*	*Node Server Replica2*	*Node Server Replica0*	
Client Node2	*Client Node15*	*Node Server Replica20*	*Node Server Replica 1*	
Client Node3	*Client Node19*	*Node Server Replica22*	*Node Server Replica 2*	*Node Server Replica 20=2*
Client Node4	*Client Node7*	*Node Server Replica2*	*Node Server Replica 0*	
Client Node5	*Client Node8*	*Node Server Replica2*	*Node Server Replica 0*	
Client Node6	*Client Node16*	*Node Server Replica20*	*Node Server Replica 1*	*Node Server Replica 22=2*
Client Node7	*Client Node21*	*Node Server Replica22*	*Node Server Replica 2*	
Client Node8	*Client Node9*	*Node Server Replica2*	*Node Server Replica 0*	
Client Node9	*Client Node14*	*Node Server Replica2*	*Node Server Replica 0*	

Pada tabel IV-5 dipetakan proses konfigurasi *node server replica* dan *client node* yang dialokasikan pada masing-masing *request node client*, prosesnya adalah menghubungkan masing-masing *client node* ke *node server replica* ke berdasarkan konsep pembebanan pada banyaknya *request client* ke *node server replica*.

Ketika *client node* 0 melakukan *request* maka *load balancer* akan menghubungkan *client node* tersebut ke *node server replica* 0. Urutan pada *server* yaitu dimulai dari *node server replica* 0, *node server replica* 20 dan *node server replica* 22. *Request* dari *client node* pertama *load balancer* akan menghubungkannya ke *node server replica* 0. *Request* dari *client node* kedua *load balancer* akan menghubungkannya ke *node server replica* 20. *Round robin* mengirimkan permintaan pengguna ke *node server replica* yang memiliki jumlah terbanyak pada *request client node*, sampai keseluruhan *request* dari *client node* terpenuhi.

Pseudocode algoritma AWRR adalah didefinisikan seperti gambar IV.7 sebagai berikut:

Gambar IV-7. *Pseudocode* Algoritma *Weighted Round Robin* (antrian dan penjadwalan *load balancing*)

Berikut dibawah ini akan diuraikan metode *load balancing weight round robin* dan dengan perhitungan pada desain topologi yang diterapkan. Berdasarkan perancangan simulasi pada *request client node* menuju masing-masing *node server replica*, dan uraian pada *flowchart* dan *pseudocode* pada gambar VI.7. Selanjutnya pada gambar IV.8 akan dijelaskan bagaimana pemetaan dan perhitungan untuk klasifikasi *request client node* menuju *node server replica* tersebut pada algoritma *weight round robin*.

1. *Client node* (0) dilayani oleh *server* ke:

Server = Paket request 0 jumlah *node server* i=(i + 1)mod n; if(i == 0){ cw=cw-1;*Server*=0 mod 3

if (cw <= 0)

Server Node=6-0 Mod3=0

***Server* = 0 (*Server Node*-2)**

2. *Client node* (1) dilayani oleh *server* ke:

Server = Paket *request* 1 jumlah *node server* i=(i + 1)mod n; if(i == 0){ cw=cw-1;
Server = i=(i+1) mod N; *Server*=6 mod 3

if (cw <= 0)

Server Node=6-0 Mod3=0

***Server* = 0 (*Server Node*-2)**

3. *Client node* (2) dilayani oleh *server* ke:

Server = Paket *request* 15 jumlah *node server* i=(i + 1)mod n; if(i == 0){ cw=cw-1;*Server*=2 mod 3

cw=cw-1;

if (i == 0)

Server Node=2-1;Mod3=1

Server = 1 (Server *Node*-20)

4. *Client node* (3) dilayani oleh *server* ke:

Server = Paket *request* 19 jumlah *node server* *i=(i + 1)mod n; if(i == 0){ cw=cw-1;Server*=2 mod 3

i=(i + 1)mod n;if(i==0)

Server Node=2 Mod3=2

Server = 0 (*Server Node*-22)

5. *Client node* (4) dilayani oleh *server* ke:

Server = Paket *request* 7 jumlah *node server* i=(i + 1)mod n; if(i == 0){ cw=cw-1;Server=6 mod 3

if (cw <= 0)

Server Node=6-0 Mod3=0

Server = 0 (*Server Node*-2)

6. *Client node* (5) dilayani oleh *server* ke:

Server = Paket *request* 8 jumlah *node server* i=(i + 1)mod n; if(i == 0){ cw=cw-1;*Server*=6 mod 3

if (cw <= 0)

Server Node=6-0 Mod3=0

Server = 0 (*Server Node*-2)

7. *Client node* (6) dilayani oleh *server* ke:

Server = Paket *request* 16 jumlah *node server* i=(i + 1)mod n; if(i == 0){ cw=cw-1;*Server*=0 mod 3

cw=cw-1;

if (i == 0)

Server Node=2-1 Mod3=1

Server = 1 (*Server Node*-20)

8. *Client* node (7) dilayani oleh *server* ke:

Server = Paket *request* 21 jumlah *node server* i=(i + 1)mod n; if(i == 0){ cw=cw-1;*Server*=0 mod 3

i=(i + 1)mod n; if(i == 0)

Server Node=2 Mod3=2

Server = 2 (*Server Node*-22)

9. Client node (8) dilayani oleh *server* ke:

Server = Paket *request* 9 jumlah *node server* i=(i + 1)mod n; if(i == 0){ cw=cw-1;*Server*=0 mod 3

if (cw <= 0)

Server Node=6-0 Mod3=0

Server = 0 (*Server Node*-2)

10. *Client node* (9) dilayani oleh *server* ke:

Server = Paket *request* 14 jumlah *node server* i=(i + 1)mod n; if(i == 0){ cw=cw-1;*Server*=0 mod 3

if (cw <= 0)

Server Node=6-0 Mod3=0

Server = 0 (*Server Node*-2)

Gambar IV-8. Perhitungan Algoritma *Weight Round Robin*

Tabel IV-6. Implementasi metode simulasi penjadwalan algoritma *Weight Round Robin* (AWRR), dijelaskan pada *script* TcL:

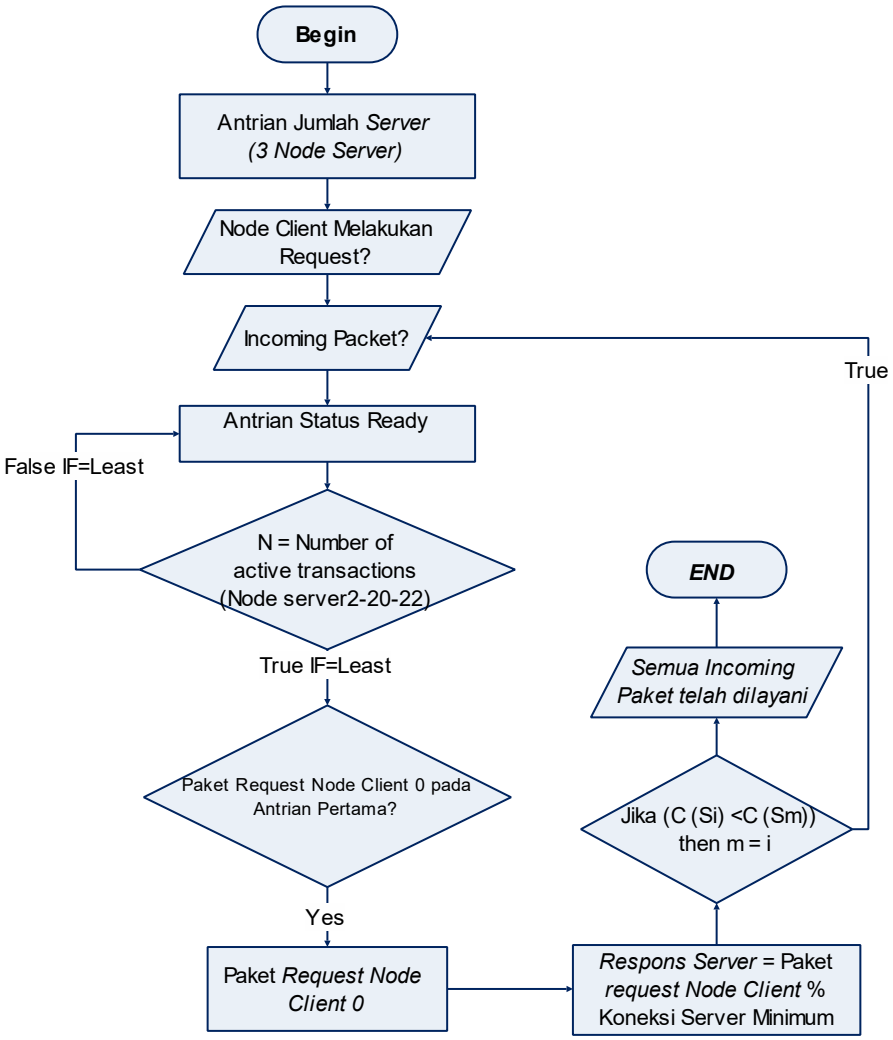
#Melampirkan traffic to *Node Server*-2: *Node Server*-20: *Node Server*-22
#*create source traffic client node to server node*
#*Create a UDP agent and attach it to node* n0
set udp_0 [new Agent/myUDP] $udp_0 set_filename udpSend0
$udp_0 set packetSize_ 1052 $ns attach-agent $n2 $udp_0
set udp_1 [new Agent/myUDP] $udp_1 set_filename udpSend1
$udp_1 set packetSize_ 1052 $ns attach-agent $n2 $udp_1
set udp_4 [new Agent/myUDP] $udp_4 set_filename udpSend7
$udp_4 set packetSize_ 1052 $ns attach-agent $n2 $udp_4
set udp_5 [new Agent/myUDP] $udp_5 set_filename udpSend8
$udp_5 set packetSize_ 1052 $ns attach-agent $n2 $udp_5
set udp_8 [new Agent/myUDP] $udp_8 set_filename udpSend9
$udp_8 set packetSize_ 1052 $ns attach-agent $n2 $udp_8
set udp_9 [new Agent/myUDP] $udp_9 set_filename udpSend14
$udp_9 set packetSize_ 1052 $ns attach-agent $n2 $udp_9
set udp_2 [new Agent/myUDP] $udp_2 set_filename udpSend15
$udp_2 set packetSize_ 1052 $ns attach-agent $n20 $udp_2
set udp_6 [new Agent/myUDP] $udp_6 set_filename udpSend16
$udp_6 set packetSize_ 1052 $ns attach-agent $n20 $udp_6
set udp_3 [new Agent/myUDP] $udp_3 set_filename udpSend19
$udp_3 set packetSize_ 1052 $ns attach-agent $n22 $udp_3
set udp_7 [new Agent/myUDP] $udp_7 set_filename udpSend21
$udp_7 set packetSize_ 1052 $ns attach-agent $n22 $udp_7

Tabel IV-6 merupakan representasi dari algoritma *load balancing weighted round robin*. Berdasarkan skema dan pemetaan pada Tabel IV-6 yang diterapkan pada simulasi, *node server replica* memilih layanan untuk setiap koneksi masuk dengan memilih *server* dengan transaksi aktif paling sering muncul, selanjutnya *node server replica* yang memiliki transaksi aktif paling kecil akan menerima *request client* selanjutnya.

4.3.1.3. Metode Penjadwalan Algoritma *Least Connection* (ALC)

Algoritma *Least-Connection scheduling* menilai status beban *real-time* dari *node server* melalui nomor koneksi yang direkam pada *load balancer*, mengirimkan yang *packet* ke simpul server yang memiliki jumlah koneksi menuju *node server replica*. *Pseudocode* algoritma ALC adalah didefinisikan seperti gambar IV-10 sebagai berikut:

Berikut representasi *flowchart* dari algoritma *load balancing least connection* dan penjabaran penjelasan diatas:



Gambar IV-9. *Flowchart* Algoritma *Least Connection* (antrian dan penjadwalan *load balancing*)

Tabel IV-7. Tabel pemetaan posisi *node server replica* berdasarkan metode algoritma *Least Connection* (ALC):

Client Node	***Packet Request***	***Server Receive***	***Sequence Server***	***Current Weight Server***
Client Node0	*Client Node0*	*Server Node replica2*	*Server Node replica 0*	*Node Server Replica 2=4*
Client Node1	*Client Node1*	*Server Node replica 2*	*Server Node replica 0*	
Client Node2	*Client Node9*	*Server Node replica 20*	*Server Node replica 1*	
Client Node3	*Client Node16*	*Server Node replica 22*	*Server Node replica 2*	*Node Server Replica 20=3*
Client Node4	*Client Node7*	*Server Node replica 2*	*Server Node replica 0*	
Client Node5	*Client Node14*	*Server Node replica 20*	*Server Node replica 1*	
Client Node6	*Client Node19*	*Server Node replica 22*	*Server Node replica 2*	*Node Server Replica 22=3*
Client Node7	*Client Node8*	*Server Node replica 2*	*Server Node replica 0*	
Client Node8	*Client Node15*	*Server Node replica 20*	*Server Node replica 1*	
Client Node9	*Client Node21*	*Server Node replica 22*	*Server Node replica 2*	

Pada tabel IV-7 dipetakan proses konfigurasi *node server replica* dan *client node* yang dialokasikan pada masing-masing *request node client*, prosesnya adalah menghubungkan masing-masing *client node* ke *node server replica* ke berdasarkan konsep pembebanan pada banyaknya *request client* ke *node server replica*. Ketika transaksi aktif memiliki request *node client* yang sama *load balancer* akan menghitungnya secara *round robin*.

Ketika *client node* 0 melakukan *request* maka *load balancer* akan menghubungkan *client node* tersebut ke *node server replica* 0. Urutan pada *server* yaitu dimulai dari *node server replica* 0, *node server replica* 20 dan *node server replica* 22. *Request* dari *client node* pertama *load balancer* akan menghubungkannya ke *node server replica* 0. *Request* dari *client node* kedua *load balancer* akan menghubungkannya ke *server node* 20. *Least connection* mengirimkan permintaan pengguna ke *server node* yang memiliki jumlah terkecil selanjutnya pada *request client node*, sampai keseluruhan *request* dari *client node* terpenuhi.

```
Least_Connection_Algorithm (node_client m, Node_Server n)

1.Menentukan server node 2 transaksi aktif least_connection;

2.Temukan Link to_server dengan koneksi least transaksi aktif;

3.Cari server transaksi aktif selanjutnya node server replica 20
continue node server replica 22, kembalikan Sm;

4. Kembalikan NULL jika tidak menemukan server.
Deskripsi Kode:

Least_Connection_Algorithm (node m, Server n)
      Pseudocode ALC didefinisikan:
      Jumlah Server Node set S = {S Node2, S Node20,S Node22},

      W(Si) Bobot of server Si,
      C(Si) Koneksi Indeks Server  Si,

      for (m = 0; m < n; m++) {
          if (W(Sm) > 0) {
              for (i = i < n; i++) {
                  if (W(Si) <= 0)
                      continue;
                  if (C(Si) < C(Sm))
                      m = i;
              }
              return Sm;
          }
      }
return NULL;
```

Gambar IV-10. *Pseudocode* Algoritma *Least Connection* (antrian dan penjadwalan *load balancing*)

Berdasarkan skema pada Tabel IV-10, dalam skema yang diterapkan pada simulasi, *node server replica* memilih layanan untuk setiap koneksi masuk dengan memilih *server* dengan transaksi aktif dan koneksi diteruskan sebagai berikut:

1. *Node Server Replica* 2 menerima permintaan *client node* 0-1–2-3. dengan *node traffic client* ke *server* adalah= UDP 0-UDP 1-UDP9-UDP16.
2. *Node Server Replica* 20 menerima permintaan *client node* 4-5–6. dengan *node traffic client* ke *server* adalah= UDP 7-UDP 14-UDP19.
3. *Node Server Replica* 22 menerima permintaan *client node* 7-8–9. dengan *node traffic client* ke *server* adalah= UDP 8-UDP 15-UDP21.

```
1. Client node (0-1) dilayani oleh server ke:
Server = Paket request 0; jumlah node server
Server = Server Node-2 menerima permintaan
client node-0 dan client node-1 (transaksi aktif 0)
2. Client node (2) dilayani oleh server ke:
Server = Paket request 1; jumlah node server
Server Node-20 menerima permintaan
node client-2 (transaksi aktif 3)

3. Client node (3) dilayani oleh server ke:
Server = Paket request 9; jumlah node server
Server Node-22 menerima permintaan
node client-3 (transaksi aktif 3)

4. Client node (4) dilayani oleh server ke:
Server = Paket request 7; jumlah node server
Server Node-2 menerima permintaan
node client-4 (transaksi aktif 0)

5. Client node (5) dilayani oleh server ke:
Server = Paket request 14; jumlah node server
Server Node-20 menerima permintaan
node client-5 (transaksi aktif 3)

6. Client node (6) dilayani oleh server ke:
Server = Paket request 19; jumlah node server
Server Node-22 menerima permintaan
node client-6 (transaksi aktif 3)
7. Client node (7) dilayani oleh server ke:
Server = Paket request 8; jumlah node server
Server Node-2 menerima permintaan
node client-7 (transaksi aktif 0)
8. Client node (8) dilayani oleh server ke:
Server = Paket request 15; jumlah node server
Server Node-20 menerima permintaan
node client-8 (transaksi aktif 3)
9. Client node (8) dilayani oleh server ke:
Server = Paket request 21; jumlah node server
Server Node-22 menerima permintaan
node client-22 (transaksi aktif 3)

1.  Node Server Replica-2 menangani 0 transaksi aktif.
2.  Node Server Replica-20 menangani 3 transaksi aktif.
3.  Node Server Replica-22 menangani 3 transaksi aktif.
```

Gambar IV-11. Perhitungan Algoritma *Least Connection* (antrian dan penjadwalan *load balancing*)

Tabel IV-8. Implementasi metode simulasi penjadwalan algoritma *Least Connection* (ALC), dijelaskan pada *script* TcL:

#Melampirkan traffic to *Node Server-2*: *Node Server-20*: *Node Server-22*
#*create source traffic client node to server node*
#*Create a UDP agent and attach it to node* n0
set udp_0 [new Agent/myUDP] $udp_0 set_filename udpSend0
$udp_0 set packetSize_ 1052 $ns attach-agent $n2 $udp_0
set udp_1 [new Agent/myUDP] $udp_1 set_filename udpSend1
$udp_1 set packetSize_ 1052 $ns attach-agent $n2 $udp_1
set udp_4 [new Agent/myUDP] $udp_4 set_filename udpSend7
$udp_4 set packetSize_ 1052 $ns attach-agent $n2 $udp_4
set udp_7 [new Agent/myUDP] $udp_7 set_filename udpSend8
$udp_7 set packetSize_ 1052 $ns attach-agent $n2 $udp_7
set udp_2 [new Agent/myUDP] $udp_2 set_filename udpSend9
$udp_2 set packetSize_ 1052 $ns attach-agent $n20 $udp_2
set udp_5 [new Agent/myUDP] $udp_5 set_filename udpSend14
$udp_5 set packetSize_ 1052 $ns attach-agent $n20 $udp_5
set udp_8 [new Agent/myUDP] $udp_8 set_filename udpSend15
$udp_8 set packetSize_ 1052 $ns attach-agent $n20 $udp_8
set udp_3 [new Agent/myUDP] $udp_3 set_filename udpSend16
$udp_3 set packetSize_ 1052 $ns attach-agent $n22 $udp_3
set udp_6 [new Agent/myUDP] $udp_6 set_filename udpSend19
$udp_6 set packetSize_ 1052 $ns attach-agent $n22 $udp_6
set udp_9 [new Agent/myUDP] $udp_9 set_filename udpSend21
$udp_9 set packetSize_ 1052 $ns attach-agent $n22 $udp_9

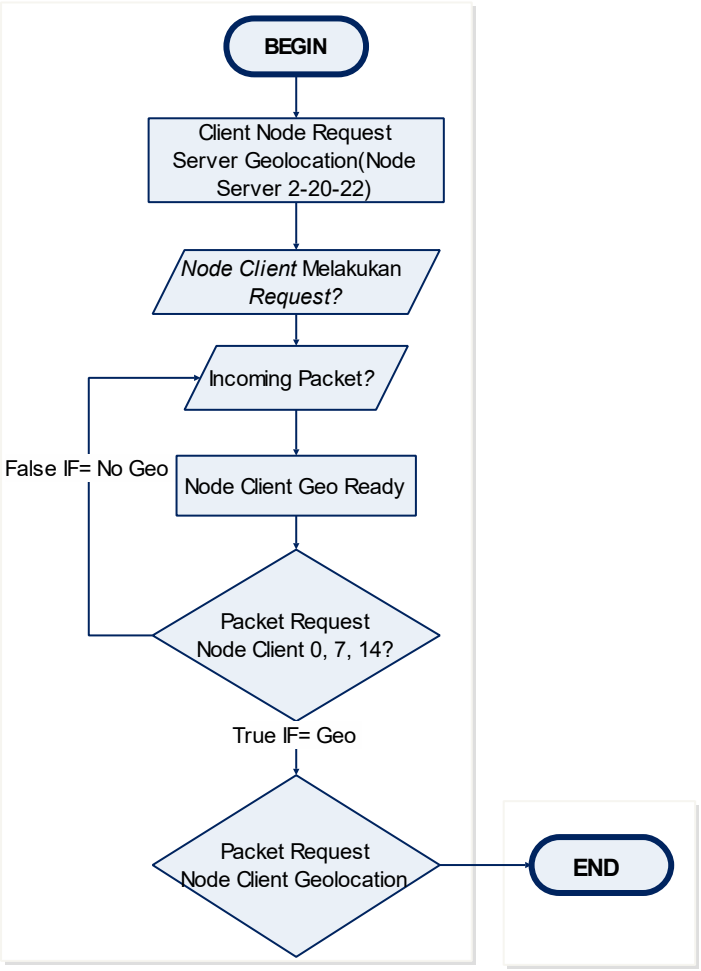
Tabel IV-8 merupakan representasi dari script TcL algoritma *load balancing least connection*. Berdasarkan skema dan pemetaan pada Tabel IV-8 yang diterapkan pada simulasi, *node server replica* memilih layanan untuk setiap koneksi masuk dengan memilih *node server replica* dengan transaksi aktif dengan nilai modulo paling kecil menuju *node server replica*.

4.3.1.4 *Anycast Geolocation Domain Name System* (DNS)

Pada penerapan *Geolocation Domain Name System* (DNS) ada beberapa spesifikasi yang akan dilakukan pada *node server replica* dan *client server*. Pada

table 4.5 akan diperlihatkan pemetaan terhadap setiap *node client* menurut lokasi yang terdekat dengan *node server replica* nya. *Geolocation* A terdapat *node server replica* 2 yang melayani permintaan dari *client* 0,1,9 dan 16. *Geolocation* B terdapat *node server replica* 22 yang melayani permintaan dari *client* 7, 8, dan 21. *Geolocation* C terdapat *node server replica* 20 yang melayani permintaan dari *client* 14, 15 dan 19.

Berikut representasi *flowchart* dari algoritma *geolocation* domain dan penjabaran penjelasan diatas:



Gambar IV-12. *Flowchart* Algoritma *Geolocation Domain Name System*

Tabel IV-9. Tabel pemetaan posisi *server* berdasarkan metode algoritma *Geo-Location* Domain:

Client Node	***Packet Request***	***Server Receive***	***Sequence Server***	***Current Weight Server***
*Client Node*0	*Client Node*0	*Node Server Replica*2	*Node Server Replica* 0	*Node Server Replica* 2=4 =*Region* A
*Client Node*1	*Client Node*1	*Node Server Replica* 2	*Node Server Replica* 0	
*Client Node*7	*Client Node*7	*Node Server Replica* 2	*Node Server Replica* 0	
*Client Node*8	*Client Node*8	*Node Server Replica* 2	*Node Server Replica* 0	
*Client Node*9	*Client Node*9	*Node Server Replica* 20	*Node Server Replica* 1	*Node Server Replica* 20=3 =*Region* B
*Client Node*14	*Client Node*14	*Node Server Replica* 20	*Node Server Replica* 1	
*Client Node*15	*Client Node*15	*Node Server Replica* 20	*Node Server Replica* 1	
*Client Node*16	*Client Node*16	*Node Server Replica* 22	*Node Server Replica* 2	*Node Server Replica* 22=3 *Region* C
*Client Node*19	*Client Node*19	*Node Server Replica* 22	*Node Server Replica* 2	
Client Node21	*Client Node*21	*Node Server Replica* 22	*Node Server Replica* 2	

Pada tabel IV-9 dipetakan proses konfigurasi *node server replica* dan *client node* yang dialokasikan pada masing-masing *request node client*, prosesnya adalah menghubungkan masing-masing *client node* ke *node server replica* ke berdasarkan konsep *geolocation* seperti yang sudah diklasifikasikan seperti tabel IV.9.

Ketika *client node* 0 melakukan *request* maka *load balancer* akan menghubungkan *client node* tersebut ke *node server replica* 2, sebagai *node server geolocation* A. Urutan pada *server* yaitu dimulai dari *node server replica* 20 dan *node server replica* 22. *Request* dari *client node* pertama *load balancer* akan menghubungkannya ke *node server replica* 0. *Request* dari *client node* ketujuh dan kedelapan *geolocation node server* akan menghubungkannya ke *node server replica* 2. Karena masih pada satu tujuan *node server replica geolocation* A. Selanjutnya, *Request* dari *client node* kesembilan *geolocation node server* akan

menghubungkannya ke *node server replica* 20 dan *node server replica* 22, sampai keseluruhan *request* dari *client node* terpenuhi.

```
Pseudocode Geolocation didefinisikan:
1. Client node menentukan node server replica dan lanjutkan ke
langkah 2;
2. Melintasi geolocation node server (2-20-22)
3. Temukan node server replica dengan koneksi pada masing-masing
wilayah;
4. Kembalikan NULL jika tidak menemukan node server replica yang
sesuai.
Deskripsi Kode:

Pseudocode Geolocation_Algorithm
Jumlah Replica Server Node set S = {S Node2, S Node20,S Node22},
R(Si) wilayah of replica server Si,
C(Si) Koneksi Indeks Server  Si,
1. initialize Region replica server  i=R ;
do :
2. i= R (Si);
3. if ( Rsi is region )
4. r=i ;
5. return S R ;
6. else
7. while ( i=r )
8. return NULL ;


Keterangan :
S = jumlah replica server (S Node2, S Node20, S Node22 )
i = indeks dari region replica server yang terpilih
```

Gambar IV-13. *Pseudocode* Algoritma *Geolocation Domain*

Berdasarkan skema pada gambar IV-13, dalam skema yang diterapkan pada simulasi, *node client* memilih layanan untuk setiap koneksi masuk dengan memilih *server* pada masing-masing *geolocation*. koneksi diteruskan dengan *Geolocation* A terdapat *node server replica* 2 yang melayani permintaan dari *node client* 0,1,9 dan 16. *Geolocation* B terdapat *node server replica* 22 yang melayani permintaan dari *client* 7, 8, dan 21. *Geolocation* C terdapat *node server replica* 20 yang melayani permintaan dari *client* 14, 15 dan 19.

Gambar IV-14. Perhitungan Algoritma *Geolocation* Domain

Gambar IV-15, IV-16 dan IV-17 adalah implementasi metode simulasi algoritma *Geolocation* domain yang dijelaskan pada *script* TcL berikut:

Gambar IV-15. Pemetaan *node client* menurut *geolocation*

Gambar IV-16. Pemetaan *node client* sebagai penerima *request* menuju *node server*

Gambar IV-17. Pemetaan *node client* dan *node server*

Konfigurasi *script* TCL adalah: $n2 merupakan *server node*-2 dan diberi nama UDP_0, UDP_1, UDP_7 dan UDP_8. Selanjutnya pada konfigurasi *client node* sebagai penerima $n0 sebagai *client node* 0, $n1 sebagai *client node* 1, $n9 sebagai *client node* 9 dan $n16 sebagai *client node* 16 yang diberi nama $null_0, $null_1, $null_9 dan $null_16.

4.4. Hasil Pengujian Dan Analisis Data

Pengujian dan analisis data akan menganalisis *output* yang dihasilkan dari 20 kali percobaan pada simulasi pada NAM di NS-2. Analisis data *output* tersebut diklasifikasikan untuk mengetahui hasil penggunaan beberapa algoritma pada teknik *load balancer* dan juga metode *geolocation* yang diterapkan. Parameter kinerja yang diperoleh antara lain *delay, packet loss* dan *throughtput.*

4.4.1. Pengujian dan Analisis Topologi Tanpa Konfigurasi *Content Delivery Network* (CDN)

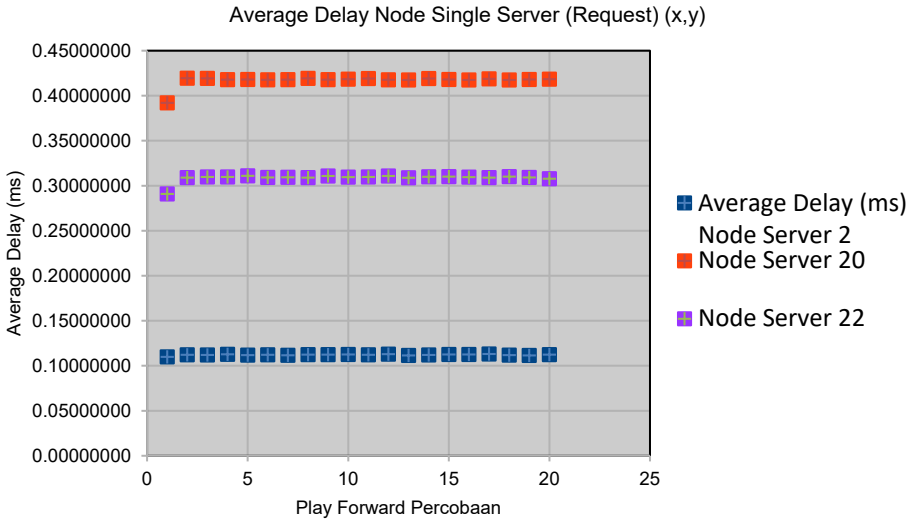
Pengujian topologi jaringan yang dimaksud adalah *single node server*, dimana hanya 1 *node server* masing-masing yang akan melayani ketika satu *node client* dan keseluruhan yang melakukan *request*.

4.4.1.1. *Transmission Average Delay* (ms)

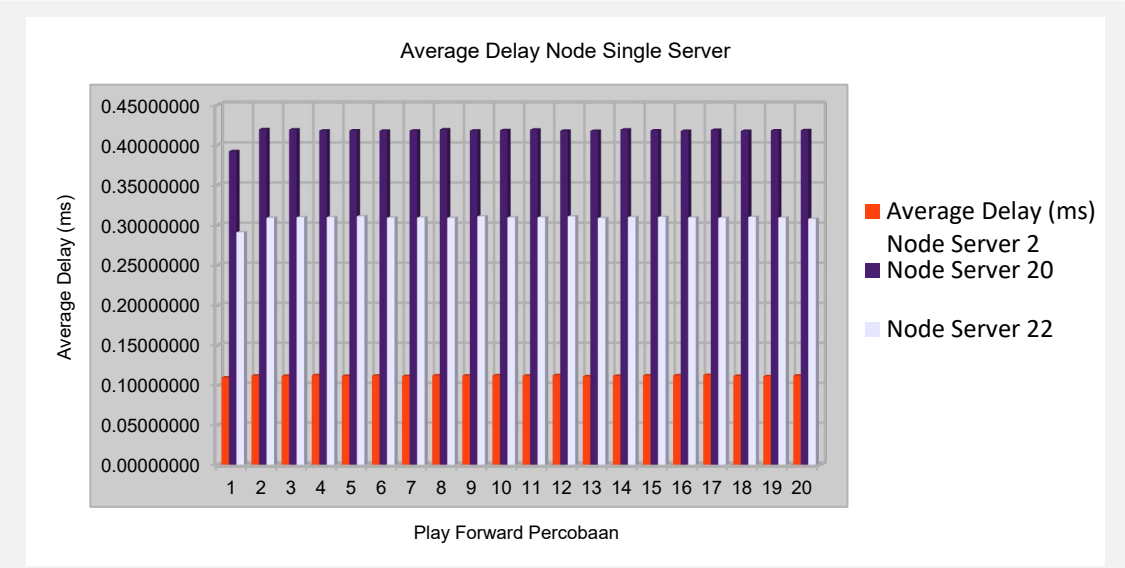
Pada proses data hasil dari *average delay* dilakukan *play forward* percobaan sebanyak 20 kali seperti pada tabel V-10. Pada proses *play forward* sebanyak 20 kali percobaan untuk *node single server* paling besar di *node server* 20 yaitu sebesar 0.41700328 detik dan yang paling kecil terdapat pada *node server* 2 yaitu sebesar 0.11217142 detik. Selanjutnya pada *node server* 22 memiliki *average delay* sebesar 0.30871480 detik. Pemetaan simulasi perubahan setiap *play forward* yang dilakukan dapat dilihat pada Gambar IV-9. Dari Gambar IV-9 dapat dilihat bahwa karakteristik perubahan setiap *play forward* yang dilakukan pada masing-masing *node client* dan *node server* terhadap parameter nilai *average delay* tidak terdapat perubahan yang signifikan.

Tabel IV-10. Data hasil *play forward average delay* pada topologi jaringan *node single server*

Play Forward	*Average Delay* (ms)		
	Node Server 2	*Node Server 20*	*Node Server 22*
1	0.10993220	0.39218870	0.29099920
2	0.11229980	0.41957750	0.30911770
3	0.11207700	0.41927060	0.30974140
4	0.11284050	0.41788990	0.30982870
5	0.11197000	0.41805040	0.31099270
6	0.11227630	0.41771140	0.30924740
7	0.11173620	0.41793260	0.30940530
8	0.11245490	0.41939040	0.30907900
9	0.11236820	0.41787150	0.31087480
10	0.11263310	0.41839280	0.30958610
11	0.11225350	0.41907990	0.30978410
12	0.11287670	0.41770090	0.31083050
13	0.11147280	0.41752440	0.30887370
14	0.11198290	0.41915870	0.30997240
15	0.11257390	0.41814040	0.31012920
16	0.11266690	0.41751590	0.30955390
17	0.11311480	0.41874240	0.30904130
18	0.11195670	0.41746770	0.31012070
19	0.11152520	0.41807610	0.30933040
20	0.11241680	0.41838340	0.30778670
Average Delay (ms)	0.11217142	0.41700328	0.3087148



Gambar IV-18. Karakteristik *average delay node request single server*



Gambar IV-19. *Average Delay Node Request Single Server*

Gambar IV-18 dan IV-19 menunjukkan perbandingan *delay* dari urutan *play forward* simulasi yang dilakukan. Grafik pemetaan perubahan setiap *play forward* yang dilakukan dapat dilihat pada gambar IV-18 dan IV-19. Dari gambar IV-18 dan IV-19 dapat dilihat bahwa karakteristik perubahan setiap *play forward* yang dilakukan pada masing-masing *node client* dan *node server* terhadap parameter nilai *average delay* tidak terdapat perubahan yang signifikan. Pada masing-masing *request client* menuju *node server* terdapat ketidakseimbangan *pada average delay* yang dihasilkan.

4.4.1.2. *Average Packet Loss* (ms)

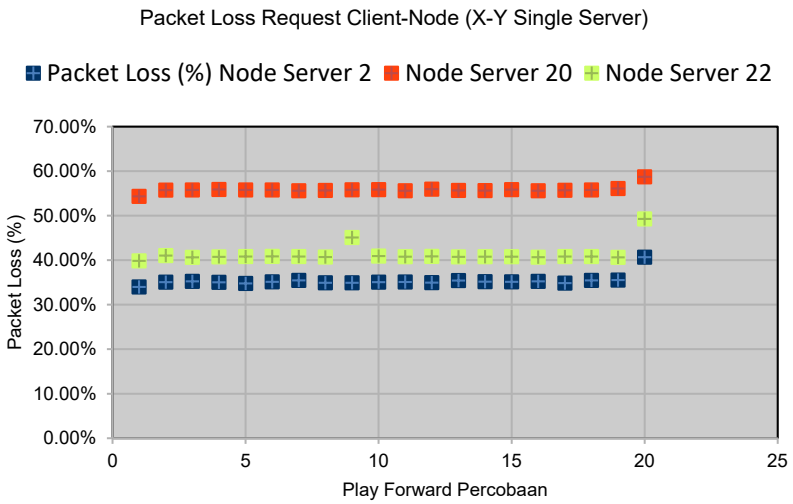
Pada proses data hasil dari *packet loss* dilakukan *play forward* percobaan sebanyak 20 kali seperti pada tabel IV-11. Pada proses *play forward* sebanyak 20 kali percobaan, untuk *node single server* paling besar di *node server* 2 yaitu sebesar 55.84%. Paling kecil terdapat pada *node server* 20 yaitu sebesar 35.36%. Selanjutnya pada *node server* 22 memiliki *packet loss* sebesar 41.38%. Grafik pemetaan perubahan setiap *play forward* yang dilakukan dapat dilihat pada Gambar IV-21. Dari Gambar IV-21 dapat dilihat bahwa karakteristik perubahan setiap *play forward* yang dilakukan pada masing-masing *node client* dan *node server* terhadap parameter nilai *packet loss* tidak terdapat perubahan yang

signifikan. Pada masing-masing *request node client* masih mengalami peningkatan pada persentasi *average delay* yang dihasilkan. Waktu tunda yang terjadi disebabkan oleh proses pengiriman paket data ke *node server* yang menjadi tujuannya.

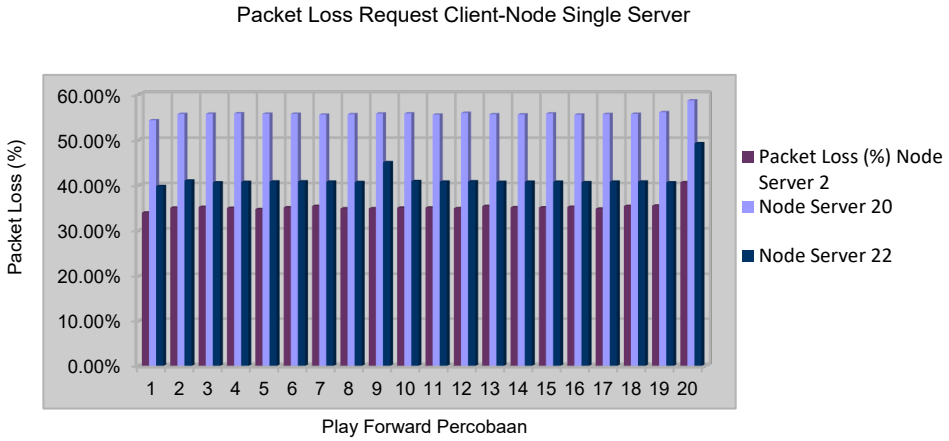
Tabel IV-11. Data hasil *play forward average delay* pada topologi jaringan *node single server*

Play Forward	***Packet Loss (%)***		
	Node Server 2	***Node Server 20***	***Node Server 22***
1	34.00%	54.34%	39.83%
2	35.08%	55.73%	41.05%
3	35.24%	55.78%	40.65%
4	35.03%	55.87%	40.75%
5	34.76%	55.79%	40.83%
6	35.15%	55.77%	40.84%
7	35.45%	55.58%	40.80%
8	34.91%	55.68%	40.70%
9	34.93%	55.82%	45.08%
10	35.07%	55.85%	40.92%
11	35.10%	55.58%	40.79%
12	34.95%	55.97%	40.86%
13	35.44%	55.66%	40.75%
14	35.16%	55.63%	40.78%
15	35.13%	55.84%	40.78%
16	35.26%	55.59%	40.68%
17	34.86%	55.72%	40.80%
18	35.45%	55.77%	40.81%
19	35.53%	56.11%	40.65%
20	40.68%	58.72%	49.27%
Average Packet Loss (%)	35.36%	55.84%	41.38%

Tabel IV.11 *packet loss node request single server* (Non-CDN)



Gambar IV-20. Karakteristik *average packet loss node request single server* (Non-CDN)



Gambar IV-21. Perbandingan *Packet Loss (%) node request* pada *Single Server* (Non-CDN)

Pada gambar IV-20 dan IV-21 menunjukkan hasil karakteristik dan perbandingan *packet loss* pada topologi jaringan *single server* (Non-CDN). Dimana *request node client* menuju *node server* dengan melakukan percobaan *play forward* sebanyak 20 kali. Dari parameter nilai yang dihasilkan *average delay* tersebut masih dalam kategori nilai garansi. Penyebab *packet loss* yaitu antrian yang melebihi kapasitas *buffer* pada setiap *node*.

4.4.1.3. *Throughput* (*bits/s*)

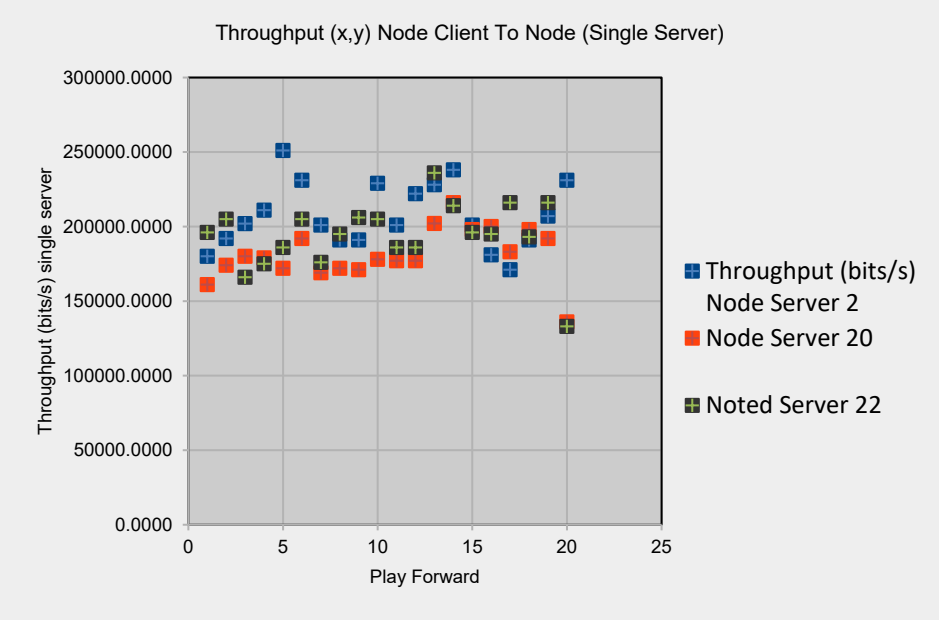
Pada proses data hasil dari *throughput* dilakukan *play forward* percobaan sebanyak 20 kali seperti pada tabel 4.9. Pada proses *play forward* sebanyak 20 kali percobaan untuk *node single server* paling besar di *node server* 2 yaitu sebesar 189274.875 bit/s dan yang paling kecil terdapat pada *node server* 20 yaitu sebesar 130256.078 bit/s. Selanjutnya pada *node server* 22 memiliki *throughput* sebesar 172152.6 bit/s.

Tabel IV-12. Data hasil *play forward throughput* pada topologi jaringan *node single server* (Non-CDN)

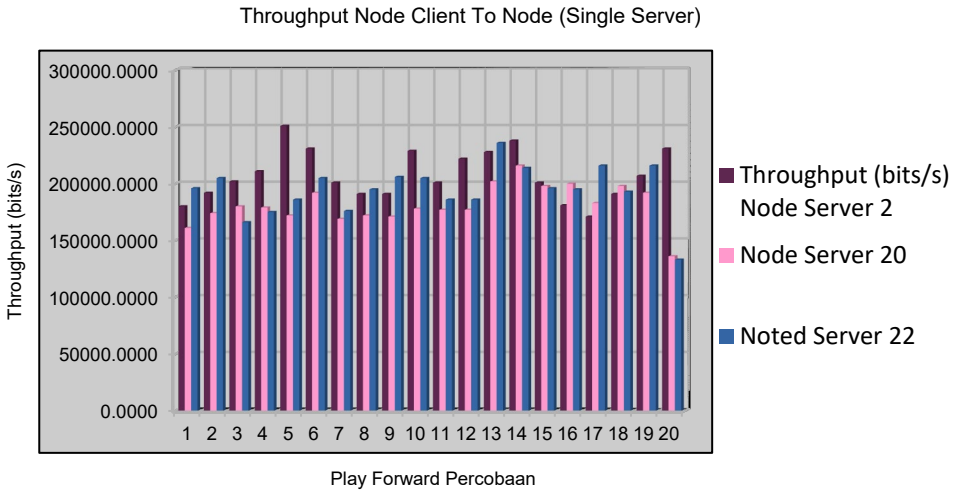
Play Forward	***Throughput* (bits/s)**		
	Node Server 2	***Node Server 20***	***Noted Server 22***
1	180049.11	161049.1117	196049.1176
2	192049.11	174049.1117	205047.1145
3	202049.11	180049.1117	166049.1176
4	211049.11	179049.1117	175047.1145
5	251049.11	172049.1117	186049.1176
6	231049.11	192049.1117	205047.1145
7	201049.11	169041.1102	176049.1176
8	191049.11	172057.1132	195047.1145
9	191043.11	171041.1102	206049.1226
10	229049.11	178057.1132	205047.1158
11	201055.11	177049.1117	186047.1187
12	222059.11	177049.1117	186051.1165
13	228041.11	202057.1132	236049.1247
14	238039.11	216041.1102	214047.1158
15	201049.11	198049.1117	196047.1227
16	181049.11	200049.1117	195047.1158
17	171049.11	183057.1132	216047.1184
18	191043.11	198041.1102	193047.1158
19	207049.11	192057.1132	216047.1179
20	231055.11	136041.1102	133047.1158
Average	189274.875	130256.078	172152.6000

Grafik pemetaan perubahan setiap *play forward* yang dilakukan dapat dilihat pada Gambar IV-22. Dari Gambar IV-22 dapat dilihat bahwa karakteristik

perubahan setiap *play forward* yang dilakukan pada masing-masing *node client* dan *node server* terhadap parameter nilai *throughput* tidak terdapat keseimbangan nilai yang didapatkan saat terjadi pengiriman paket dari *client node* menuju *node server*.



Gambar IV-22 Karakteristik *throughput node request single server* (Non-CDN)



Gambar IV-23. Perbandingan *Throughput (bits/s) node request* pada *Single Server* (Non-CDN)

Pada gambar IV-22 adalah *throughput* dari *node server* melalui *single request*. *Horizontal axis* adalah tingkat koneksi percobaan dengan *play forward*

sebanyak 20 kali dengan *single server request*. *Vertikal axis* adalah *throughput* pada masing-masing *request client* menuju *node server* berdasarkan *single request*. Gambar IV-22 bahwa ketika tingkat nilai *request* pada masing-masing *node client* menuju *server* memiliki nilai *throughput* yang tidak seimbang ketika melakukan *request*.

4.4.2. Pengujian dan Analisis Topologi Jaringan dengan *Content Delivery Network* (CDN)

Pada pembahasan diatas sebelumnya sudah dilakukan pengujian dan analisis topologi jaringan, yang didesain tanpa menggunakan penerapan *load balancing* pada CDN. Serta penerapan pada metode *geolocation* domain, selanjutnya pembahasan CDN akan dilakukan pengujian dan analisis dengan membandingkan *single server request* dengan CDN dengan parameter yang sama yang akan diuji yakni pada nilai *delay, packet loss dan throughput*.

4.4.2.1. *Average Transmission Delay* (ms)

Parameter nilai pada masing-masing *delay* dapat dilihat pada tabel IV-13. pada proses *play forward* sebanyak 20 kali percobaan untuk *node load balancing* terlihat mengalami penurunan yang signifikan pada masing-masing *request client*. Dimana rata-rata nilai *delay* yang ditampilkan adalah pada ketiga *node server replica*. Pada tabel IV-13 terlihat nilai *average delay* pada *single server* mengalami lonjakan ketika mengalami *request* dari *client*. Berbeda dengan nilai *delay* yang ada pada ketiga server dengan menggunakan teknik *load balancing* CDN dan juga teknik *geolocation*.

Perbandingan pada setiap perubahan parameter nilai metode yang dilakukan dapat dilihat pada table IV-13. Dari table IV-13 dapat dilihat hasil simulasi metode dengan perulangan percobaan *play forward* menunjukkan bahwa, nilai *average delay* dari masing-masing *client node* ketika metode yang diterapkan menghasilkan nilai *delay* yang berkurang. Hal ini ditunjukkan pada Tabel IV-13. *Delay* pada metode Non-CDN, mengalami *average delay* yang lebih tinggi dari metode CDN. *Average delay* perbandingan, non CDN mengalami *delay* berkisar

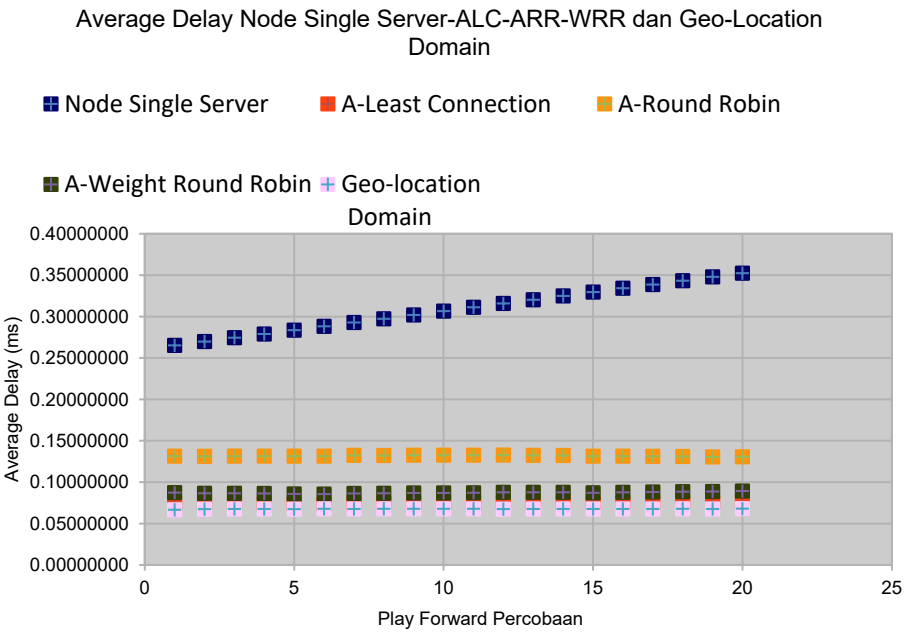
0.308879515 (31%) ARR: 0.131706725 (13%), AWRR: 0.087243966 (9%), ALC: 0.07509255 (8%) dan *Geolocation* lebih kecil yaitu 0.0680760 (7%).

Tabel IV-13. Data hasil *average delay* pada topologi jaringan *node single server* (Non-CDN) dan metode ARR, AWRR, ALC dan *Geolocation*.

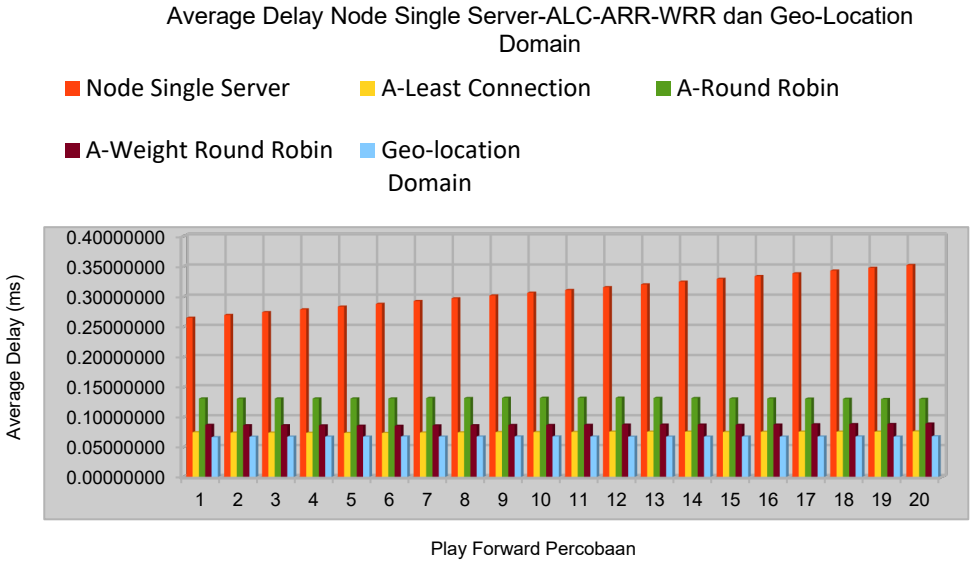
Play Forward	***Node Single Server***	***A-Least Connection***	***A-Round Robin***	***A-Weight Round Robin***	***Geo-location Domain***
1	0.26520160	0.07495000	0.13147117	0.08728748	0.0666300
2	0.26983380	0.07420100	0.13118712	0.08653831	0.0674160
3	0.27442550	0.07420400	0.13142296	0.08654160	0.0675570
4	0.27902430	0.07397300	0.13142416	0.08631024	0.0676090
5	0.28363250	0.07373800	0.13142225	0.08584321	0.0675260
6	0.28822040	0.07374500	0.13142928	0.08561769	0.0678290
7	0.29280480	0.07443700	0.13235394	0.08630979	0.0675890
8	0.29739410	0.07467300	0.13212445	0.08654541	0.0677210
9	0.30200860	0.07514000	0.13259155	0.08701251	0.0678060
10	0.30659500	0.07514100	0.13259207	0.08701303	0.0678290
11	0.31117990	0.07537100	0.13259025	0.08724377	0.0677500
12	0.31579800	0.07582900	0.13281489	0.08770097	0.0674100
13	0.32035610	0.07583200	0.13235265	0.08770385	0.0675340
14	0.32497230	0.07583300	0.13212126	0.08770501	0.0675840
15	0.32958040	0.07537000	0.13142581	0.08724212	0.0676800
16	0.33416130	0.07560100	0.13142417	0.08770560	0.0676090
17	0.33872820	0.07583500	0.13119360	0.08817270	0.0676950
18	0.34330970	0.07583700	0.13096280	0.08840701	0.0677700
19	0.34789980	0.07583200	0.13049276	0.08863465	0.0675580
20	0.35246400	0.07630900	0.13073736	0.08934437	0.0680760
Average	**0.308879515**	**0.07509255**	**0.131706725**	**0.087243966**	**0.0680760**

Pada gambar IV-23 adalah *average delay* dari metode penerapan ARR, AWRR, ALC dan Geolocation, melalui *client request*. *Horizontal axis* adalah tingkat koneksi percobaan dengan *play forward* sebanyak 20 kali dengan *single server request* dan penerapan *load balancing dengan CDN* ARR, AWRR, ALC dan *Geolocation*. Vertikal *axis* adalah *delay* pada masing-masing *request client* menuju *node server* berdasarkan *single request* dan *load balancing* CDN ARR, AWRR, ALC dan *Geolocation*.

Delay maksimum yang direkomendasikan oleh ITU untuk aplikasi multimedia dan suara adalah 150 ms. pada penerapan teknik *load balancing* pada CDN termasuk memenuhi kategori *excellent* pada media transmisinya karena nilainya adalah 0.0 dan 0.1 ms.



Gambar IV-23. Karakteristik *average delay* algoritma penjadwalan ARR-AWRR-ALC dan *Geolocation* (CDN)



Gambar IV-24. Perbandingan *delay* Algoritma penjadwalan ARR-AWRR-ALC dan *Geolocation* (CDN)

4.4.2.2 *Packet Loss* (%)

Parameter nilai pada masing-masing *packet loss* dapat dilihat pada tabel IV-14. pada proses *play forward* sebanyak 20 kali percobaan untuk *node load balancing* terlihat mengalami penurunan nilai *loss* pada masing-masing *server* ketika *request client*. Pada tabel IV-14 terlihat nilai *average delay* pada *single server* mengalami lonjakan ketika mengalami *request* dari *client*. Berbeda dengan nilai *packet loss* dengan menggunakan teknik *load balancing* CDN dan juga teknik *geolocation*.

Perbandingan pada setiap perubahan parameter nilai metode yang dilakukan dapat dilihat pada table IV-14. Dari table IV-14 dapat dilihat hasil simulasi metode dengan perulangan percobaan *play forward* menunjukkan bahwa, nilai *average packet loss* dari masing-masing *client node* ketika metode yang diterapkan menghasilkan nilai *packet loss* yang berkurang. *Packet loss* pada metode Non-CDN, mengalami *average packet loss* yang lebih tinggi dari metode CDN. *Average packet loss* non CDN mengalami *packet loss* berkisar 44.19%, ARR: 13.67%, AWRR: 14.09%, ALC: 2.84% dan *Geolocation* lebih kecil yaitu 0.041%.

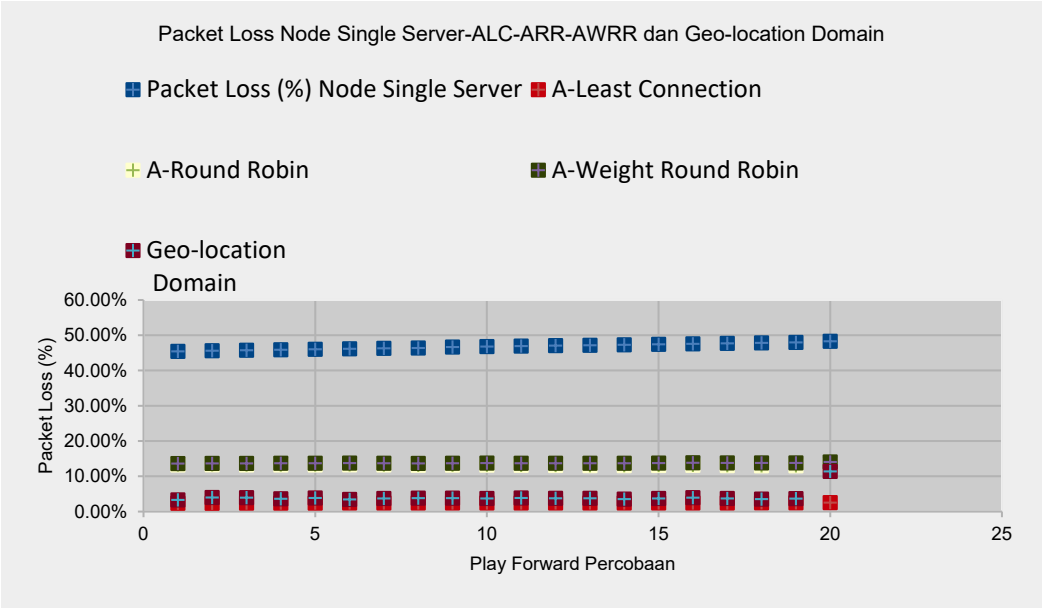
Tabel IV-14. Data hasil *average packet loss* pada topologi jaringan *node single server* (Non-CDN) dan metode ARR, AWRR, ALC dan *Geolocation*.

Play Forward	***Packet Loss (%)***				
	Node Single Server	***A-Least Connection***	***A-Round Robin***	***A-Weight Round Robin***	***Geo-location Domain***
1	45.46%	2.37%	13.24%	13.64%	3.31%
2	45.63%	2.39%	13.26%	13.69%	3.99%
3	45.76%	2.41%	13.24%	13.69%	3.97%
4	45.89%	2.42%	13.24%	13.71%	3.63%
5	46.02%	2.43%	13.24%	13.71%	3.88%
6	46.16%	2.44%	13.25%	13.75%	3.47%
7	46.29%	2.45%	13.21%	13.71%	3.72%
8	46.42%	2.44%	13.15%	13.69%	3.83%
9	46.65%	2.44%	13.17%	13.74%	3.82%
10	46.79%	2.44%	13.17%	13.77%	3.73%
11	46.93%	2.45%	13.13%	13.73%	3.89%
12	47.06%	2.45%	13.10%	13.72%	3.79%
13	47.19%	2.45%	13.06%	13.70%	3.78%
14	47.32%	2.44%	13.06%	13.73%	3.56%

15	47.45%	2.46%	13.08%	13.78%	3.74%
16	47.58%	2.48%	13.12%	13.86%	3.97%
17	47.71%	2.48%	13.04%	13.83%	3.72%
18	47.85%	2.49%	12.99%	13.82%	3.54%
19	47.97%	2.49%	12.97%	13.83%	3.67%
20	48.29%	2.55%	13.16%	14.08%	11.42%
Average Packet Loss (%)	**44.19%**	**2.84%**	**13.67%**	**14.09%**	**0.041**

Pada gambar IV-25 adalah *average packet loss* dari metode penerapan ARR, AWRR, ALC dan *Geolocation*, melalui *client request*. *Horizontal axis* adalah tingkat koneksi percobaan dengan *play forward* sebanyak 20 kali dengan *single server request* dan penerapan *load balancing dengan CDN* ARR, AWRR, ALC dan *Geolocation*. Vertikal *axis* adalah *average delay* pada masing-masing *request client* menuju *node server* berdasarkan *single request* dan *load balancing* CDN ARR, AWRR, ALC dan *Geolocation*.

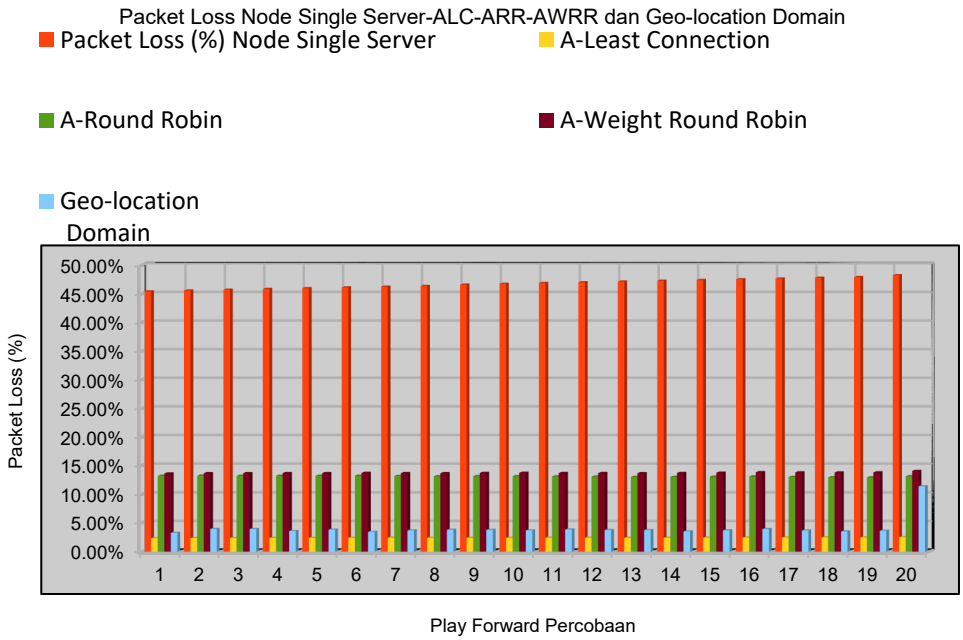
Packet loss maksimum yang direkomendasikan oleh TIPHON untuk aplikasi multimedia dan suara adalah 0-15%. pada penerapan teknik *load balancing* pada CDN termasuk memenuhi kategori sangat bagus pada media transmisinya karena nilainya adalah 0%.



Gambar IV-25. Karakteristik *packet loss* Algoritma penjadwalan ARR-AWRR-ALC dan *Geolocation* (CDN)

Grafik pemetaan perubahan setiap *play forward* yang dilakukan dapat dilihat pada gambar IV.26. Dari Gambar IV.26 dapat dilihat bahwa karakteristik perubahan setiap *play forward* yang dilakukan pada masing-masing *node client* dan *node server* terhadap parameter nilai *average packet loss* terdapat perubahan yang signifikan pada *node server* yang menggunakan teknik *load balancing* pada server CDN dan teknik *geolocation* yang diterapkan.

Pada gambar Gambar IV.26 LC metode *geolocation* yang diterapkan mengalami packet loss yang paling rendah 0.04% selanjutnya ALC: 2.84%, ARR: 13.67%, AWRR: 14.09%.



Gambar IV-26. Perbandingan *delay* Algoritma penjadwalan ARR-AWRR-ALC dan *Geolocation* (CDN)

4.4.2.3. *Throughput (*bits/s)

Parameter nilai pada masing-masing *throughput* dapat dilihat pada tabel IV-15. Pada proses simulasi dengan *play forward* sebanyak 20 kali percobaan untuk *node server load balancing* terlihat nilai *throughput,* akan semakin tinggi pada jalur transmisi dan dengan demikian besarnya *packet* yang berhasil dikirim, juga akan semakin bagus. terutama pada teknik *geolocation*, ALC dan AWRR. Pada tabel IV-6 terlihat nilai rata-rata *throughput* pada *single server* mengalami

penurunan yaitu 157318.6295 bits/s ketika mengalami *request* dari *client*. Berbeda dengan nilai *throughput* yang ada pada keempat metode dengan menggunakan teknik *load balancing* CDN dan *geolocation* domain.

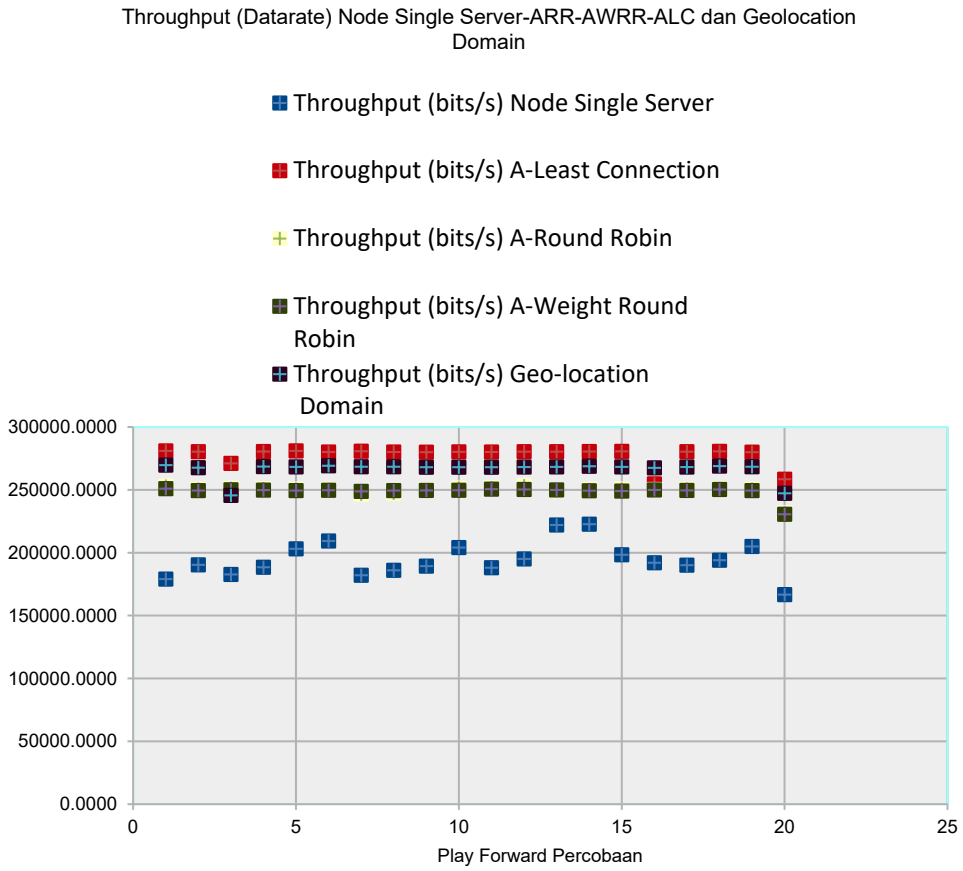
Average throughput pada metode Non-CDN, mengalami *average throughput* yang lebih rendah dari metode yang diterapak dengan CDN. *Average throughput* non CDN mengalami penurunan berkisar 191210.9321 bits/s, ARR: 249568.3884 bits/s, AWRR: 248803.4251 bits/s, ALC: 277585.9487 bits/s dan *Geolocation* 266181.7355 bits/s.

Tabel IV-15. Data hasil *average throughput* pada topologi jaringan *node single server* (Non-CDN) dan metode ARR, AWRR, ALC dan *Geolocation*.

Play Forward	***Throughput (bits/s)***				
	Node Single Server	**A-Least Connection**	**A-Round Robin**	**A-Weight Round Robin**	***Geo-location Domain***
1	179049.1133	281036.1670	253021.0330	250925.4000	269700.6252
2	190381.7789	280361.3670	250547.0000	249443.6000	267600.6394
3	182715.7799	271078.0000	251187.9670	250129.1000	245622.5947
4	188381.7789	280532.1330	250932.7330	249870.9000	268500.6381
5	203049.1133	280970.4000	250541.6670	249427.4000	268300.6305
6	209381.7789	280150.7670	250467.6330	249657.0330	269200.6246
7	182046.4461	280792.2670	246542.5330	248756.8000	268400.6460
8	186051.1127	280110.8330	247307.5000	249474.9330	268400.6007
9	189377.7804	280046.5670	249940.8670	249601.0670	268000.6290
10	204051.1121	280215.3000	251406.2000	249725.9000	268000.6815
11	188050.4476	280173.8330	251726.0000	250564.5000	268000.6861
12	195053.1137	280384.8670	253235.0000	250367.8670	268200.6554
13	222049.1150	280321.9000	249307.9670	249985.1670	268300.6492
14	222709.1114	280545.5670	250235.6000	249315.8000	268800.6431
15	198381.7816	280665.6330	250680.2000	249042.7000	268300.7152
16	192048.4460	254785.7390	250275.0670	249894.3000	267500.6294
17	190051.1140	280420.4330	250594.9670	249595.5670	268200.6262
18	194043.7782	280672.0000	251065.1670	250291.9670	268900.6483
19	205051.1128	279955.4000	250593.5670	249471.4670	268400.6565
20	166714.4462	258499.8000	231759.1000	230527.0330	247300.4901
Average Throughput (bits/s)	**194431.9131**	**277585.9487**	**249568.3884**	**248803.4251**	**266181.7355**

Grafik pemetaan perbandingan perubahan setiap *play forward* yang dilakukan dapat dilihat pada gambar IV-7. Dari Gambar IV-7dapat dilihat bahwa

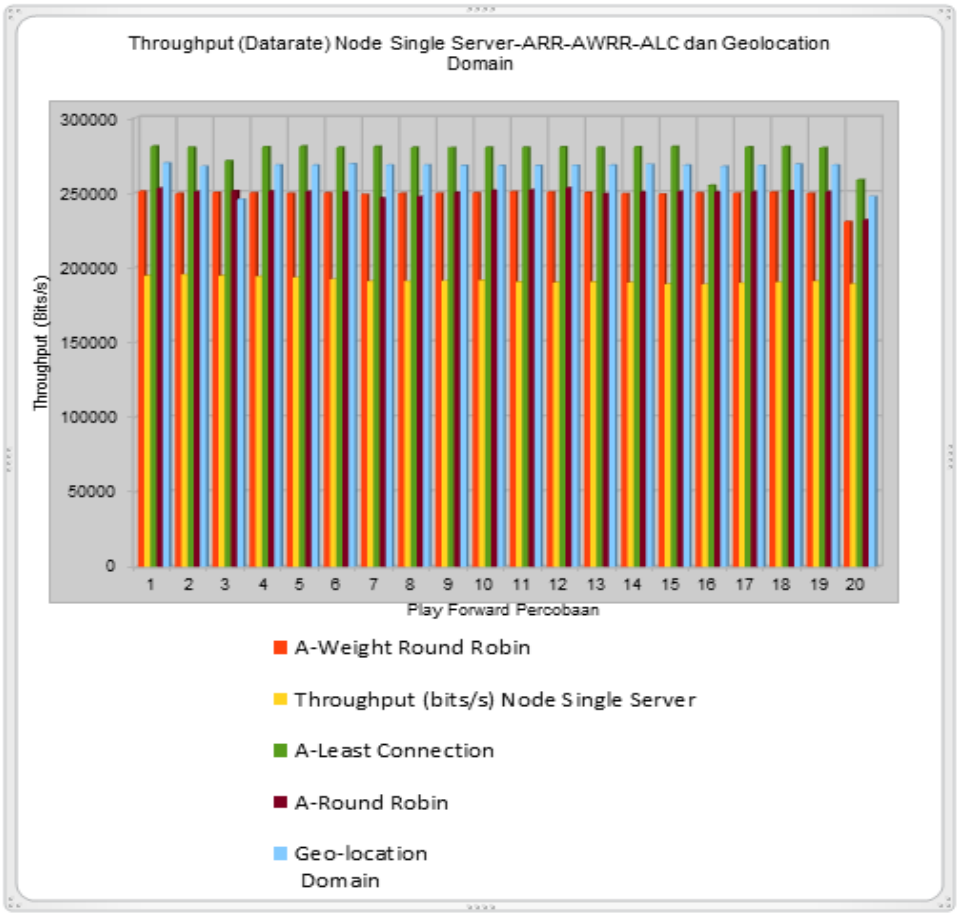
perbandingan perubahan setiap *play forward* yang dilakukan pada masing-masing *node client request* terhadap parameter nilai *average throughputs* terdapat peningkatan pada jalur menuju *server* yang menggunakan teknik *load balancing* pada *server* CDN dan metode *geolocation* yang diterapkan.



Gambar IV-27. Karakteristik *average throughput* algoritma penjadwalan ARR-AWRR-ALC dan *Geolocation* (CDN)

Pada gambar IV-27 dan IV-28 adalah *throughput* dari *node server* melalui *client request*. *Horizontal axis* adalah tingkat koneksi percobaan dengan *play forward* sebanyak 20 kali dengan *single server request* dan penerapan *load balancing* dan *geolocation* dengan CDN. Vertikal *axis* adalah *throughput* pada masing-masing *request client* menuju *node server replica* berdasarkan *single request* dan *load balancing* CDN. *Throughput* toleran maksimum yang direkomendasikan oleh ITU dan TIPHON untuk aplikasi video dan suara adalah

100%. berdasarkan hasil perhitungan lebar pipa dan besar bandwidth yang akan dilewatkan, yaitu 682960.0000 bits/s.



Gambar IV-28. Perbandingan *average throughput* algoritma penjadwalan ARR-AWRR-ALC dan *Geolocation* (CDN)

Parameter nilai *throughput node server* CDN lebih tinggi daripada jaringan *Single Server* (tanpa CDN). Yaitu nilai *throughput* metode CDN:1042139.50 dan *Single Server* (tanpa CDN):191210.93. Ini berarti dengan jaringan *node server replica* maka kemampuan jaringan akan lebih meningkat dalam mentransmisikan *bit-bit video*. Dari simulasi yang sudah dilakukan pada sistem jaringan *single server* dan CDN nilai *delay*, *packet loss*, dan *throughput* dari algoritma *least connection* dan teknik *geolocation* lebih baik daripada algoritma *round robin* dan *weighted round robin.*

BAB V
KESIMPULAN DAN SARAN

5.1. Kesimpulan

Dari hasil pembahasan perancangan, analisis dan simulasi yang telah dilakukan, ada beberapa kesimpulan yang dapat dijabarkan pada hasil-hasil yang diperoleh berdasarkan hasil simulasi pada perancangan topologi *load balancing* dan *geolocation domain* maka didapatkan nilai *Quality of Service* (QoS) sebagai berikut:

Berdasarkan hasil simulasi pada perancangan topologi *load balancing* dan *geolocation domain* maka didapatkan nilai *Quality of Service* (QoS) sebagai berikut:

1. Penerapan penjadwalan algoritma *load balancing* Algoritma *Least Connection* (ALC), Algoritma *Round Robin* (RR) dan Algoritma *Weighted Round Robin* (WRR) di dalam jaringan *Content Delivery Networks* (CDNs), yang memiliki nilai rata-rata dari parameter indeks QoS yang lebih baik adalah algoritma Algoritma *Least Connection* (ALC), teknik *Geolocation Domain*, Algoritma *Weighted Round Robin* (AWRR) dan selanjutnya Algoritma Round Robin (ARR). Sehingga sumber daya pada metode *load balancing* dan *Geolocation* yang diterapkan, menunjukkan bahwa strategi yang diusulkan untuk *load balancing* tidak hanya mengungguli beberapa teknik yang ada tetapi juga menjamin persyaratan QoS.
2. Penerapan penjadwalan algoritma *load balancing Least Connection* (LC), *Round Robin* (RR) dan *Weighted Round Robin* (WRR) serta teknik *Geolocation domain* di dalam jaringan *Content Delivery Network* (CDN), menghasilan parameter nilai yang dapat mengukur pembebanan *traffic* secara seimbang, efektif dan efisien pada sumber daya Content Delivery Network (CDN).Parameter nilai indek QoS pada masing-masing
    - Dengan algoritma ALC didapatkan *average delay* 0.07509255 ms atau turun sekitar 76%. Selanjutnya dengan penjadwalan algoritma ARR didapatkan *average delay* 0.131706725 ms atau turun sekitar

- 71%, algoritma AWRR didapatkan *average delay* 0.087243966 ms atau turun 75% dan dengan metode *geolocation* didapatkan *average delay* 0.0680760 ms atau turun sekitar 77%.
- Dengan penjadwalan algoritma ALC didapatkan packet loss 2.84% atau turun 41.36%. Dengan algoritma ARR didapatkan average packet loss 13.67% atau turun yaitu 30.53%. Selanjutnya dengan algoritma AWRR diperoleh packet loss 14.09% atau turun 30.11% dan dengan teknik geolocation diperoleh packet loss 4.00% atau turun 40.10%.
- Dengan algoritma ALC diperoleh *throughput* 277585.9487. Dengan algoritma RR diperoleh *throughput* 249568.3884 bps dengan algoritma WRR diperoleh *throughput* 248803.4251 bps. dan teknik *geolocation* diperoleh *throughput* 266181.7355 bps.

3. Nilai *output* yang dihasilkan pada parameter QOS, dengan metode algoritma *load balancing* dan metode *geolocation domain* yang diterapkan, pada perutean jalur transmisi CDNs. Pembebanan *traffic* menunjukkan bahwa hasil yang diperoleh seimbang dari sumber *client node* ke tujuan *server node* (*replica server*). Model penjadwalan algoritma yang akan diterapkan dapat mengatasi *load traffic* pada teknologi perutean jalur transmisi CDNs menuju *server edge* (*replica server*). Dengan memperhatikan nilai performansi dari *throughput*, *packet loss*, dan *average delay*.

   Pada indeks *average delay* teknik geolocation domain yang memiliki nilai delay yang paling kecil yaitu: 0.0680760, ALC: 0.07509255, AWRR: 0.087243966 dan ARR: 0.131706725. Indeks average packet loss teknik *geolocation* domain yang memiliki nilai *packet loss* yang paling kecil yaitu: 0.041%, ALC: 2.84%, ARR: 13.67% dan AWRR: 14.09%. Indeks *average throughput* diperoleh lebih baik dalam memenuhi kapasitas pengiriman packet adalah ALC: 277585.9487 bps, *Geolocation*: 266181.7355bps, ARR: 249568.3884 bps dan AWRR: 248803.4251 bps.

5.2. Saran

Berdasarkan hasil pembahasan perancangan, analisis dan simulasi yang telah dilakukan, dalam mekanisme penjadwalan dan optimalisasi multi-algoritma *load balancing* dan *anycast* DNS pada *Content Delivery Networks* (CDN), maka dapat diuraikan saran sebagai berikut:

1. Dalam penelitian ini, peneliti mensurvei berbagai algoritma *load balancing* pada teknik penjadwalan. Peneliti membahas masalah utama yang harus dipertimbangkan saat merancang algoritma *load balancing*. Peneliti membahas algoritma yang sudah diusulkan oleh berbagai peneliti dalam literatur, kelebihan dan kekurangan mereka. Di masa depan peneliti menyarankan pada merancang algoritma yang akan mempertahankan *trade-off* yang lebih baik di antara semua parameter kinerja. Baik pada lingkungan CDNs, VM *Server* dan *Cloud Computing*.
2. Penelitian selanjutnya peneliti menyarankan dalam merancang model, dapat menggunakan teknik algoritma penjadwalan yang lainnya. Begitu juga untuk jumlah dari *node server* dan *node client* yang akan diterapkan.
3. Penelitian selanjutnya peneliti menyarankan dalam merancang nilai parameter hasil dengan parameter lainnya seperti *respons time* dan *processing time* pada masing-masing *replica server*.

DAFTAR PUSTAKA

A. Vakali, and G. Pallis, “Content Delivery Networks: Status and Trends,” IEEE Internet Computing, IEEE Computer Society, pp. 68-74, November-December 2003.

A. Binder and I. Kotuliak, "Content Delivery Network Interconnect: Practical experience," 2013 IEEE 11th International Conference on Emerging eLearning Technologies and Applications (ICETA), Stara Lesna, 2013, pp. 29-33.

Bai, Y., Jia, B., Zhang, J., & Pu, Q. (2009). An efficient load balancing technology in CDN. 6th International Conference on Fuzzy Systems and Knowledge Discovery, FSKD 2009, 7, 510–514.

Bhagat, C. B., & Budhwant, D. K. (2016). Improvised Provision for Load Balancing in Content Delivery Networks, 5(10).

Cisco Visual Networking Index: Forecast and Methodology, 2016–2021, Cisco, San Jose, CA, USA, Jun. 2017.

D. C. Verma, Content Distribution Networks: An Engineering Approach, John Wiley & Sons, Inc., New York, 2002.

ETSI. (2000). Telecommunications and Intranet Protocol Harmonization Over Networks (TIPHON), End to End Quality of Service in TIPHON System.

Elgeldawy, F. A., Gerges, M., & Abdel, M. F. (2016). Performance of QOS Parameters for IPTV Through NGN.

Ehsan Saboori, Shahriar Mohammadi, Shafigh Parsazad. (2012) "A new scheduling algorithm for server farms load balancing", Industrial and Information Systems (IIS), 2010 2nd International Conference on Industrial and Information Systems .

E. S.-J. Swildens, Z. Liu, and R. D. Day, (Aug. 11 2009) “Global traffic management system using IP anycast routing and dynamic load-balancing,” uS Patent 7,574,499.

Ezreik, A., & Gheryani, A. (2012). Design and Simulation of Wireless Network using NS-2. 2nd International Conference on Computer Science and Information Technology, 3(1), 157–161.

F. Chen, R. K. Sitaraman, and M. Torres, “End-User Mapping: Next Generation Request Routing for Content Delivery,” in Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication, August 2015.

Fan, Q., Yin, H., Jiao, L., Lv, Y., Huang, H., & Zhang, X. (2018). Towards Optimal Request Mapping and Response Routing for Content Delivery Networks. IEEE Transactions on Services Computing, 1374(c), 1–1.

F. Chen, R. K. Sitaraman, and M. Torres, August 2015. “End-User Mapping: Next Generation Request Routing for Content Delivery,” in Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication.

F. Dobrian, V. Sekar, A. Awan, I. Stoica, D. Joseph,A. Ganjam, J. Zhan, and H. Zhang. (2011)“Understanding the impact of video quality on user engagement,” in SIGCOMM.

F. Chen, R. K. Sitaraman, and M. Torres, (August 2015) “End-User Mapping: Next Generation Request Routing for Content Delivery,” in Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication.

Goel, U., Wittie, M. P., & Steiner, M. (2015). Faster web through client-assisted CDN server selection. Proceedings - International Conference on Computer Communications and Networks, ICCCN.

G. Pallis, and A. Vakali, “Insight and Perspectives for Content Delivery Networks,” Communications of the ACM, Vol. 49, No. 1, ACM Press, NY, USA, pp. 101-106, January 2006.

G. Peng, “CDN: Content Distribution Network,” Technical Report TR-125, Experimental Computer Systems Lab, Department of Computer Science, State University of New York, Stony Brook, NY, 2003.

G. Pallis, and A. Vakali, “Insight and Perspectives for Content Delivery Networks,” Communications of the ACM, Vol. 49, No. 1, ACM Press, NY, USA, pp. 101-106, January 2006.

Gursun, G. (2017). Routing-aware partitioning of the internet address space for server ranking in CDNs. Computer Communications, 106, 86–99.

G. Pallis and A. Vakali, "Insight and perspectives for content delivery networks, "Communications of the ACM, vol. 49, no. 1, pp. 101-106, 2006.

G. Leduc, “Course "Computer Network Architectures and Multimedia",” 2016-2017. [Online: accessed 18 Februari 2018]. Available:

http://www.montefiore.ulg.ac.be/~leduc/cours/structure-multimedia.html.

Haghighi, A. A., Shah Heydari, S., & Shahbazpanahi, S. (2018). Dynamic QoS-Aware Resource Assignment in Cloud-Based Content-Delivery Networks. IEEE Access, 6, 2298–2309.

Hitesh Bheda, H. B. (2015). An Overview of Load balancing Techniques in Cloud Computing Environments An Overview of Load balancing Techniques in Cloud Computing Environments, 9874–9881.

Jack L Burbank, William Kasch, and Jon Ward. (2011). An Introduction to Network Modeling and Simulation for the Practicing Engineer, volume 5. John Wiley & Sons.

Kanthai Srinivasa Rao, Manjula Srinivas, Molli Srinivasa Rao. (Oct - Dec 2014). Load Balancing Performance in Content Delivery Networks (CDN's), IJCST Vol. 5, Issue 4.

Klaue, Jirka, Berthold Rathke, and Adam Wolisz;'Evalvid–A framework for video transmission and quality evaluation. Computer Performance Evaluation. Modelling Techniques and Tools". Springer Berlin Heidelberg, 2003. 255-272.

Kyryk M, Pleskanka N, Pitsyk M. (2016). QoS Mechanism in Content Delivery Network. International Conference on Modern Problems of Radio Engineering, Telecommunications and Computer Science (TCSET).

Koerner M, Kao O. (2012). Multiple service load balancing with Open Flow, IEEE HPSR.

Kanthai Srinivasa Rao, Manjula Srinivas, Molli Srinivasa Rao, (Oct - Dec 2014) Load Balancing Performance in Content Delivery Networks (CDN's), IJCST Vol. 5, Issue 4.

M. Pathan, R. Buyya and A. Vakali, "Content delivery networks: State of the art, insights, and imperatives," in Content Delivery Networks, Springer Berlin Heidelberg, 2008, pp. 3-32.

Mansour, Y. and Patt-shamir, B. (2001) 'Jitter Control in QoS Networks', 9(4), pp. 492–502.

M. Calder, A. Flavel, E. Katz-Bassett, R. Mahajan, and J. Padhye, "Analyzing the Performance of an Anycast CDN," in Proceedings of the 2015 Internet Measurement Conference, October 2015.

Michael Goetz and Ivor Nissen. (2012). Guwmanet—multicast routing in underwater acoustic networks. In Communications and Information Systems Conference (MCC), 2012 Military, pages 1–8. IEEE.

M. Calder, A. Flavel, E. Katz-Bassett, R. Mahajan, and J. Padhye, "Analyzing the Performance of an Anycast CDN," in Proceedings of the 2015 Internet Measurement Conference, October 2015.

M. Dehghan et al., ''On the complexity of optimal request routing and content caching in heterogeneous cache networks,'' IEEE/ACM Trans. Netw., vol. 25, no. 3, pp. 1635–1648, Jun. 2017.

N. Anjum, D. Karamshuk, M. Shikh-Bahaei, and N. Sastry, ''Surveyon peer-assisted content delivery networks,'' Comput. Netw., vol. 116,pp. 79–95, Apr. 2017.

Nicolas Nicolaou, Andrew See, Peng Xie, Jun-Hong Cui, and Dario Maggiorini.(2007). Improving the robustness of location-based routing for underwater sensor networks. In OCEANS 2007-Europe, pages 1–6. IEEE.

Panchal, P. et al. (2014) 'Content delivery networks in the cloud with distributed edge servers', Proceedings - 2013 IEEE International Conference on High Performance Computing and Communications, HPCC 2013 and 2013 IEEE International Conference on Embedded and Ubiquitous Computing, EUC 2013, pp. 526–532.

Patel, Z., & Dalal, U. (2014). Design and Implementation of Low Latency Weighted Round Robin (Ll- Wrr) Scheduling for High Speed (IJWMN), 6(4), 59–71.

Pettersson, J. (2013). Towards Automated Network Con guration Simulations In A Switched Ethernet Network Kateryna Mariushkina.

Pandaba Pradhan, Prafulla Ku. Behera, B N B Ray, (CMS 2016). Modified Round Robin Algorithm for Resource Allocation in Cloud Computing, International Conference CMS 2016,Procedia Computer Science 85 (2016) 878 – 890.

P. Casas, A. D'Alconzo, P. Fiadino, A. Bär, A. Finamore, and T. Zseby, "When Youtube Does not Work Analysis of QoE-Relevant Degradation in Google CDN Trac," IEEE Transactions on Network and Service Management, vol. 11, no. 4, pp. 441–457, December 2014.

P. Bret, K. Prashanth, J. Samir, and A. K. Zaid, "TCP over IP Anycast – Pipe dream or Reality?" Maret 2018. [Online]. Available:https://engineering.linkedin.com/network-performance/tcp-over-ip-anycast-pipe-dream-or-reality.

P. Casas, P. Fiadino, A. Bär, A. D'Alconzo, A. Finamore, and M. Mellia, "Youtube All Around: Characterizing YouTube from Mobile and Fixed-line Network Vantage Points," in 2014 European Conference on Networks and Communications (EuCNC).

Prajapati, R, & Rathod, D dan Khanna. (2015). Journal, International & Technological, F. (2015). Comparison of Static and Dynamic Load Balancing in, 2(7), 1337–1340.

Province, A. (n.d.). Performance Analysis of CDN-P2P Networks B ased on Processer-Sharing Queues, 2–5.2017

Qin, Z. et al. (2014) 'A CDN-based domain name system', Computer Communications. Elsevier B.V., 45, pp. 11–20.

R. Torres, A. Finamore, J. R. Kim, M. Mellia, M. M. Munafo, and S. Rao, (June 2011) "Dissecting Video Server Selection Strategies in the YouTube CDN," in Proceedings of the 2011 31st International Conference on Distributed Computing Systems.

S. Manfredi, F. Oliviero and S. P. Romano, "Distributed management for load balancing in Content Delivery Networks," 2010 IEEE Globecom Workshops, Miami, FL, 2010, pp. 579-583.

Sanchi, Pandey and V. Tyagi. 2013. Performance Analysis of Wired and Wireless Network Using NS-2 Simulator. Internationl Journal of Computer Appliction. 2013; 72 (21): 38-44.

Science, C., & Sen, A. (2011). Improving Performance of Clusters using Load Balancing Algorithms.

Shuai, Q., Wang, K., Miao, F., & Jin, L. (2017). A Cost-Based Distributed Algorithm for Load Balancing in Content Delivery Network. 2017 9th International Conference on Intelligent Human-Machine Systems and Cybernetics (IHMSC).

Song, M., Li, H., & Wu, H. (2015). 2015 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery A Decentralized Load Balancing Architecture for Cache System, 114–119.

Singh, A., Goyal, P., & Batra, S. (2010). An optimized round robin scheduling algorithm for CPU scheduling. IJCSE) International Journal on Computer Science and Engineering, 2(07), 2383-2385.

Steve McCanne et al. Ns2 documentation. [Online: accessed 18 Februari 2018].

S. Sarat, V. Pappas, and A. Terzis, 2006. "On the use of anycast in DNS,"in Computer Communications and Networks, ICCCN 2006. Proceedings. 15th International Conference on. IEEE, 2006, pp. 71–78.

T. Bottger, F. Cuadrado, G. Tyson, I. Castro, and S. Uhlig, "Open connect everywhere: A glimpse at the internet ecosystem through the lens of the net-ix cdn," 2016.

Trajano, A. F. R., & Fernandez, M. P. (2016). uLoBal : Enabling In-Network Load Balancing for Arbitrary Internet Services on SDN, (c), 62–67.

Ukani, V. S., & Detection, O. (2019). Video in Transmission, (August 2015).

V. K. Adhikari, Y. Guo, F. Hao, V. Hilt, Z. L. Zhang, M. Varvello, and M. Steiner, "Measurement Study of Net-ix, Hulu, and a Tale of Three CDNs," IEEE/ACM Transactions on Networking, vol. 23, no. 6, pp. 1984–1997, December 2015.

V. K. Adhikari, Y. Guo, F. Hao, M. Varvello, V. Hilt, M. Steiner, and Z. L. Zhang, (March 2012) "Unreeling Net-ix: Understanding and improving multi-CDN movie delivery," in 2012 Proceedings IEEE INFOCOM.

V. Stocker, G. Smaragdakis, W. Lehr, and S. Bauer, ''The growing complexity of content delivery networks: Challenges and implications for the Internet ecosystem,Telecommun. Policy, vol. 41, pp. 1003–1016,Mar. 2017.

Wang, Z., Huang, J., & Rose, S. (2018). Evolution and challenges of DNS-Based CDNs. Digital Communications and Networks, (February), 1–9.

Xu, Z., & Wang, X. (2015). A predictive modified round robin scheduling algorithm for web server clusters. Chinese Control Conference, CCC, 2015–September, 5804–5808.

Yang, X.-L., Wang, X.-X., Zhang, M., Long, K.-P., & Huang, Q. (2014). User interest-aware content replica optimized placement algorithm. Tongxin Xuebao/Journal on Communications, 35(12), 21–27.

Yeh, H., Ph, D., Chassiakos, A., & Ph, D. (2017). Comparative Analysis of Load Balancing Algorithms In Cloud Computing Presented to the Department of Electrical Engineering California State University.

LAMPIRAN

1. Parameter Nilai Hasil Quality of Service (QoS) Non-CDN and CDN

i. Total Average Delay Node Server 2 (Single Node Server)

Total *Average Delay Node Server* 2 (*Single Node Server*)

Play Forward	***Client Node0***	***Client Node1***	***Client Node7***	***Client Node8***	***Client Node9***	***Client Node14***	***Client Node15***	***Client Node16***	***Client Node19***	***Client Node21***	***Average Delay***
1	0.011137	0.011855	0.152850	0.1574180	0.0577220	0.1619860	0.1712070	0.0550890	0.1653720	0.1546860	0.1099322
2	0.011187	0.011962	0.162829	0.1645510	0.0584980	0.1638170	0.1724640	0.0553210	0.1664700	0.1558990	0.1122998
3	0.011187	0.011962	0.161682	0.1632590	0.0585300	0.1644810	0.1722010	0.0553340	0.1660040	0.1561300	0.1120770
4	0.011187	0.011962	0.164214	0.1632010	0.0585600	0.1650070	0.1736290	0.0553490	0.1674990	0.1577970	0.1128405
5	0.011187	0.011962	0.162416	0.1623390	0.0585840	0.1645860	0.1711490	0.0553660	0.1666410	0.1554700	0.1119700
6	0.011186	0.011962	0.163881	0.1630170	0.0586030	0.1642260	0.1717430	0.0553880	0.1668710	0.1558860	0.1122763
7	0.011186	0.011962	0.162173	0.1618550	0.0586010	0.1637770	0.1711420	0.0553900	0.1662740	0.1550020	0.1117362
8	0.011186	0.011962	0.163186	0.1626400	0.0585980	0.1655850	0.1722710	0.0553840	0.1668550	0.1568820	0.1124549
9	0.011186	0.011962	0.161592	0.1652730	0.0585960	0.1642610	0.1729220	0.0553840	0.1666840	0.1558220	0.1123682
10	0.011186	0.011962	0.163928	0.1627240	0.0585910	0.1661240	0.1724960	0.0553820	0.1664090	0.1575290	0.1126331
11	0.011186	0.011962	0.162446	0.1630960	0.0585820	0.1647400	0.1727910	0.0553750	0.1671810	0.1551760	0.1122535
12	0.011187	0.011962	0.163641	0.1642560	0.0585910	0.1654780	0.1723230	0.0553760	0.1680900	0.1578630	0.1128767
13	0.011187	0.011962	0.159911	0.1612060	0.0585580	0.1641280	0.1709950	0.0553540	0.1653410	0.1560860	0.1114728
14	0.011187	0.011962	0.161835	0.1623590	0.0585400	0.1643430	0.1717060	0.0553430	0.1665920	0.1559620	0.1119829
15	0.011187	0.011962	0.162935	0.1642600	0.0585450	0.1644060	0.1727760	0.0553500	0.1664650	0.1578530	0.1125739
16	0.011187	0.011962	0.163297	0.1641030	0.0585350	0.1649250	0.1729260	0.0553380	0.1676370	0.1567590	0.1126669
17	0.011187	0.011962	0.163904	0.1652050	0.0585320	0.1667220	0.1730700	0.0553330	0.1679450	0.1572880	0.1131148
18	0.011187	0.011962	0.162186	0.1615210	0.0585350	0.1645640	0.1714530	0.0553310	0.1675590	0.1552690	0.1119567
19	0.011187	0.011962	0.161566	0.1615100	0.0585130	0.1628270	0.1710090	0.0553190	0.1661940	0.1551650	0.1115252
20	0.011187	0.011987	0.162811	0.1630040	0.0591350	0.1635660	0.1705640	0.0559010	0.1681050	0.1579080	0.1124168
***Total Average Delay Node Server* 2**											**0.11217142**

ii. Total Average Packet Loss Node Server 2 (Single Node Server)

Total *Average Packet Loss Node Server* 2 (*Single Node Server*)

Play Forward	Client Node 0	Client Node1	Client Node 7	Client Node8	Client Node9	Client Node 14	Client Node15	Client Node16	Client Node19	Client Node21	*Average Packet Loss*
1	0%	0%	54.82%	55.03%	0%	51.88%	67.41%	0%	47.41%	63.45%	34.00%
2	0%	0%	61.22%	58.88%	0%	55.03%	66.50%	0%	48.22%	60.91%	35.08%
3	0%	0%	63.35%	58.38%	0%	50.05%	67.21%	0%	50.56%	62.84%	35.24%
4	0%	0%	58.78%	59.19%	0%	53.30%	67.21%	0%	51.37%	60.41%	35.03%
5	0%	0%	54.82%	58.98%	0%	52.79%	68.63%	0%	47.31%	65.08%	34.76%
6	0%	0%	61.12%	59.09%	0%	53.30%	66.09%	0%	48.83%	63.05%	35.15%
7	0%	0%	62.44%	59.59%	0%	53.60%	66.90%	0%	48.53%	63.45%	35.45%
8	0%	0%	63.96%	58.88%	0%	50.86%	64.67%	0%	46.50%	64.26%	34.91%
9	0%	0%	64.06%	56.95%	0%	51.17%	66.40%	0%	47.51%	63.25%	34.93%
10	0%	0%	62.44%	58.68%	0%	52.59%	64.97%	0%	50.76%	61.22%	35.07%
11	0%	0%	62.34%	57.56%	0%	54.11%	66.19%	0%	49.54%	61.22%	35.10%
12	0%	0%	60.00%	58.78%	0%	52.69%	70.25%	0%	47.92%	59.90%	34.95%
13	0%	0%	61.32%	61.02%	0%	53.10%	67.51%	0%	47.92%	63.55%	35.44%
14	0%	0%	61.42%	60.91%	0%	52.89%	65.79%	0%	46.90%	63.65%	35.16%
15	0%	0%	62.94%	60.41%	0%	51.57%	66.50%	0%	48.53%	61.32%	35.13%
16	0%	0%	59.09%	58.98%	0%	53.91%	68.53%	0%	48.12%	63.96%	35.26%
17	0%	0%	58.07%	59.39%	0%	50.46%	67.21%	0%	49.24%	64.26%	34.86%
18	0%	0%	61.02%	56.75%	0%	54.01%	69.75%	0%	49.75%	63.25%	35.45%
19	0%	0%	59.19%	60.51%	0%	53.91%	70.36%	0%	48.32%	63.05%	35.53%
20	0%	2.03%	64.16%	61.62%	8.22%	57.26%	71.57%	14.11%	57.16%	70.66%	40.68%
***Total Average Packet Loss Node Server* 2**											**35.36%**

iii. Total Nilai Average Throughput Node Client (Node Server 2)

Total Nilai *Average Throughput Node Client* (*Node Server* 2)											
Play Forward	***Client Node0***	***Client Node1***	***Client Node7***	***Client Node8***	***Client Node9***	***Client Node14***	***Client Node15***	***Client Node16***	***Client Node19***	***Client Node21***	***Average Node Client***
1	110037.1210	110037.1130	320137.1200	130037.1000	290037.1000	100037.1000	110077.1200	310037.1000	110017.1000	210037.1310	180049.1105
2	110037.1130	320137.1200	210037.1210	210037.1000	110077.1200	120037.1000	110037.1000	210037.1310	310037.1000	210017.1000	192049.1105
3	210037.1000	310077.1200	310037.1000	210017.1000	110037.1310	110037.1210	120037.1130	120137.1200	330037.1000	190037.1000	202049.1105
4	130037.1000	390037.1000	210037.1310	310037.1000	210017.1000	210037.1130	120137.1200	210037.1210	210037.1000	110077.1200	211049.1105
5	210037.1000	310077.1200	110037.1000	210017.1000	110037.1310	210037.1210	210037.1130	320137.1200	430037.1000	390037.1000	251049.1105
6	330037.1000	190037.1000	210037.1310	310037.1000	210017.1000	210037.1130	120137.1200	210037.1210	210037.1000	310077.1200	231049.1105
7	210037.1210	210037.1130	220137.1200	330037.1000	190037.1000	210037.1000	110077.1200	110037.1000	210017.1000	210037.1310	201049.1105
8	110037.1130	120137.1200	210037.1210	210037.1000	110077.1200	130037.1000	190037.1000	210037.1310	310037.1000	310017.1000	191049.1105
9	130037.1000	190037.1000	210037.1310	110037.1000	210017.1000	110017.1000	210037.1130	120137.1200	210037.1210	410037.1000	191043.1085
10	210037.1210	210037.1130	220137.1200	130037.1000	190037.1000	190037.1000	210037.1000	310077.1200	410037.1000	210017.1000	229049.1074
11	110037.1130	120137.1200	210037.1210	210037.1000	210077.1200	310077.1200	130037.1000	190037.1000	210037.1310	310037.1000	201055.1125
12	210037.1130	120137.1200	210037.1210	210037.1000	310077.1200	210017.1000	210037.1130	120137.1200	210037.1210	410037.1000	222059.1128
13	210037.1000	310077.1200	110037.1000	210017.1000	210037.1310	190037.1000	210037.1000	310077.1200	310037.1000	210017.1000	228041.1071
14	210037.1000	190037.1000	210037.1310	110037.1000	210017.1000	310077.1200	130037.1000	390037.1000	210037.1310	410037.1000	238039.1082
15	130037.1000	290037.1000	210037.1310	110037.1000	210017.1000	210037.1130	120137.1200	210037.1210	210037.1000	310077.1200	201049.1105
16	210037.1210	210037.1130	120137.1200	130037.1000	190037.1000	210037.1000	310077.1200	110037.1000	110017.1000	210037.1310	181049.1105
17	110037.1130	120137.1200	210037.1210	210037.1000	110077.1200	130037.1000	190037.1000	210037.1310	310037.1000	110017.1000	171049.1105
18	110017.1000	210037.1130	120137.1200	210037.1210	210037.1000	330037.1000	190037.1000	210037.1310	110037.1000	210017.1000	191043.1085
19	190037.1000	210037.1000	310077.1200	30037.1000	210017.1000	210037.1210	210037.1130	120137.1200	190037.1000	390037.1000	207049.1074
20	310077.1200	130037.1000	190037.1000	210037.1310	310037.1000	310037.1130	120137.1200	210037.1210	210037.1000	310077.1200	231055.1125

Total Nilai *Average Throughput Node Client (Node Server 2)*	4150974.2004

2. Parameter Nilai Hasil Quality of Service (QoS) Non-CDN and CDN

i. Total Nilai Average Throughput Node Client (Node Server 2)-Penjadwalan Least Connection

Total Nilai *Average Throughput Node Client* (*Node Server* 2)-Penjadwalan *Least Connection*

Play Forward	*Client Node0*	*Client Node1*	*Client Node7*	*Client Node8*	*Client Node9*	*Client Node14*	*Client Node15*	*Client Node16*	*Client Node19*	*Client Node21*	*Average Node Client*
1	410037.1210	510037.1130	320137.1200	230037.1000	290037.1000	410037.1000	310077.1200	310037.1000	110017.1000	210037.1310	311049.1105
2	210037.1130	320137.1200	210037.1210	210037.1000	310077.1200	530037.1000	190037.1000	210037.1310	610037.1000	210017.1000	301049.1105
3	210037.1000	310077.1200	310037.1000	210017.1000	210037.1310	210037.1210	210037.1130	120137.1200	330037.1000	490037.1000	261049.1105
4	430037.1000	390037.1000	210037.1310	410037.1000	210017.1000	210037.1130	420137.1200	210037.1210	210037.1000	310077.1200	301049.1105
5	210037.1000	310077.1200	310037.1000	210017.1000	310037.1310	210037.1210	210037.1130	420137.1200	430037.1000	390037.1000	301049.1105
6	330037.1000	390037.1000	210037.1310	310037.1000	210017.1000	210037.1130	120137.1200	210037.1210	210037.1000	310077.1200	251049.1105
7	210037.1210	210037.1130	420137.1200	330037.1000	490037.1000	210037.1000	310077.1200	110037.1000	210017.1000	210037.1310	271049.1105
8	310037.1130	520137.1200	210037.1210	210037.1000	310077.1200	130037.1000	190037.1000	210037.1310	310037.1000	210017.1000	261049.1105
9	330037.1000	190037.1000	210037.1310	310037.1000	210017.1000	310017.1000	210037.1130	120137.1200	210037.1210	410037.1000	251043.1085
10	210037.1210	210037.1130	220137.1200	130037.1000	190037.1000	490037.1000	210037.1000	310077.1200	410037.1000	210017.1000	259049.1074
11	310037.1130	520137.1200	210037.1210	210037.1000	310077.1200	310077.1200	130037.1000	190037.1000	210037.1310	310037.1000	271055.1125
12	210037.1130	420137.1200	210037.1210	210037.1000	310077.1200	210017.1000	210037.1130	120137.1200	210037.1210	410037.1000	252059.1128
13	210037.1000	310077.1200	310037.1000	210017.1000	210037.1310	190037.1000	210037.1000	310077.1200	510037.1000	210017.1000	268041.1071
14	430037.1000	190037.1000	210037.1310	510037.1000	210017.1000	310077.1200	130037.1000	190037.1000	210037.1310	410037.1000	280039.1082
15	330037.1000	490037.1000	210037.1310	410037.1000	210017.1000	210037.1130	120137.1200	210037.1210	210037.1000	310077.1200	271049.1105

16	210037.1210	210037.1130	120137.1200	430037.1000	190037.1000	210037.1000	310077.1200	110037.1000	510017.1000	210037.1310	251049.1105
17	310037.1130	320137.1200	210037.1210	210037.1000	310077.1200	130037.1000	190037.1000	210037.1310	310037.1000	510017.1000	271049.1105
18	210017.1000	210037.1130	520137.1200	210037.1210	210037.1000	330037.1000	190037.1000	210037.1310	410037.1000	210017.1000	271043.1085
19	490037.1000	210037.1000	310077.1200	30037.1000	210017.1000	210037.1210	210037.1130	120137.1200	390037.1000	390037.1000	257049.1074
20	310077.1200	430037.1000	190037.1000	210037.1310	310037.1000	310037.1130	120137.1200	210037.1210	210037.1000	310077.1200	261055.1125

ii. Total Nilai *Average Throughput Node Client* Penjadwalan *Weighted Round Robin*

Total Nilai *Average Throughput Node Client* (*Node Server* 2)-Penjadwalan *Weight Round Robin*

Play Forward	*Client Node0*	*Client Node1*	*Client Node7*	*Client Node8*	*Client Node9*	*Client Node14*	*Client Node15*	*Client Node16*	*Client Node19*	*Client Node21*	*Average Node Client*
1	210037.1210	210037.1130	320137.1200	230037.1000	290037.1000	310037.1000	310077.1200	310037.1000	110017.1000	210037.1310	251049.1105
2	210037.1130	320137.1200	210037.1210	210037.1000	310077.1200	330037.1000	190037.1000	210037.1310	310037.1000	210017.1000	251049.1105
3	210037.1000	310077.1200	310037.1000	210017.1000	210037.1310	210037.1210	210037.1130	120137.1200	330037.1000	190037.1000	231049.1105
4	230037.1000	390037.1000	210037.1310	310037.1000	210017.1000	210037.1130	120137.1200	210037.1210	210037.1000	110077.1200	221049.1105
5	210037.1000	310077.1200	110037.1000	210017.1000	110037.1310	210037.1210	210037.1130	320137.1200	430037.1000	390037.1000	251049.1105
6	330037.1000	190037.1000	210037.1310	310037.1270	210017.1000	210037.1130	120137.1200	210037.1210	210037.1000	310077.1200	231049.1132
7	210037.1210	210037.1130	220137.1200	330037.1000	190037.1000	210037.1000	310077.1200	110037.1000	210017.1000	210037.1310	221049.1105
8	310037.1130	120137.1200	210037.1210	210037.1000	310077.1200	330037.1000	190037.1000	210037.1310	310037.1000	310017.1000	251049.1105
9	330037.1000	190037.1000	210037.1310	110037.1000	210017.1000	310017.1000	210037.1130	120137.1200	210037.1210	410037.1000	231043.1085
10	210037.1210	210037.1130	220137.1200	130037.1000	190037.1000	290037.1710	210037.1000	310077.1200	410037.1000	210017.1000	239049.1145
11	310037.1130	120137.1200	210037.1210	210037.1910	210077.1200	310077.1200	230037.1000	190037.1000	210037.1310	310037.1000	231055.1216
12	210037.1130	120137.1200	210037.1210	210037.1000	310077.1200	210017.1000	210037.1130	120137.1200	210037.1210	410037.1000	222059.1128
13	210037.1000	310077.1200	110037.1000	210017.1000	210037.1310	190037.1000	210037.1000	310077.1200	310037.1000	210017.1000	228041.1071
14	210037.1000	190037.1000	210037.1310	110037.1230	210017.1000	310077.1200	130037.1000	390037.1000	210037.1310	410037.1000	238039.1105
15	230037.1000	290037.1000	210037.1310	110037.1100	210017.1000	210037.1130	120137.1200	210037.1210	210037.1000	310077.1200	211049.1115
16	210037.1210	210037.1130	120137.1200	430037.1200	190037.1000	210037.1000	310077.1200	110037.1000	110017.1000	210037.1310	211049.1125

17	310037.1130	120137.1200	210037.1210	210037.1000	310077.1200	130037.1000	190037.1000	210037.1310	310037.1000	110017.1000	211049.1105
18	210017.1000	210037.1130	120137.1200	210037.1210	210037.1000	330037.1000	190037.1000	210037.1310	110037.1000	210017.1000	201043.1085
19	190037.1000	210037.1000	310077.1200	30037.1000	210017.1000	210037.1210	210037.1130	120137.1200	390037.1000	390037.1000	227049.1074
20	310077.1200	130037.1000	190037.1000	210037.1310	310037.1000	310037.1130	120137.1200	210037.1210	210037.1000	310077.1200	231055.1125

iii. Total Nilai *Average Throughput Node Client* Penjadwalan *Round Robin*

Total Nilai *Average Throughput Node Client* (*Node Server* 2)-Penjadwalan *Round Robin*											
Play Forward	***Client Node0***	***Client Node1***	***Client Node7***	***Client Node8***	***Client Node9***	***Client Node14***	***Client Node15***	***Client Node16***	***Client Node19***	***Client Node21***	***Average Node Client***
1	210037.1210	110037.1130	320137.1200	230037.1000	290037.1000	310037.1000	310077.1200	310037.1000	110017.1000	210037.1310	241049.1105
2	210037.1130	320137.1200	210037.1210	210037.1000	310077.1200	330037.1000	190037.1000	210037.1310	310037.1000	210017.1000	251049.1105
3	210037.1000	310077.1200	310037.1000	210017.1000	210037.1310	210037.1210	210037.1130	120137.1200	330037.1000	190037.1000	231049.1105
4	430037.1000	390037.1000	210037.1310	310037.1000	210017.1000	210037.1130	120137.1200	210037.1210	210037.1000	110077.1200	241049.1105
5	210037.1000	310077.1200	110037.1000	210017.1000	110037.1310	210037.1210	210037.1130	320137.1200	430037.1000	390037.1000	251049.1105
6	330037.1000	190037.1000	210037.1310	310037.1000	210017.1000	210037.1130	120137.1200	210037.1210	210037.1000	310077.1200	231049.1105
7	210037.1210	210037.1130	220137.1200	330037.1000	190037.1000	210037.1000	310077.1200	110037.1000	210017.1000	210037.1310	221049.1105
8	310037.1130	120137.1200	210037.1210	210037.1000	310077.1200	330037.1000	190037.1000	210037.1310	310037.1000	310017.1000	251049.1105
9	330037.1000	190037.1000	210037.1310	110037.1000	210017.1000	310017.1000	210037.1130	120137.1200	210037.1210	410037.1000	231043.1085
10	210037.1210	210037.1130	220137.1200	130037.1000	190037.1000	190037.1000	210037.1000	310077.1200	410037.1000	210017.1000	229049.1074
11	310037.1130	120137.1200	210037.1210	210037.1000	210077.1200	310077.1200	130037.1000	190037.1000	210037.1310	310037.1000	221055.1125
12	210037.1130	120137.1200	210037.1210	210037.1000	310077.1200	210017.1000	210037.1130	120137.1200	210037.1210	410037.1000	222059.1128
13	210037.1000	310077.1200	110037.1000	210017.1000	210037.1310	190037.1000	210037.1000	310077.1200	310037.1000	210017.1000	228041.1071
14	210037.1000	190037.1000	210037.1310	110037.1000	210017.1000	310077.1200	130037.1000	390037.1000	210037.1310	410037.1000	238039.1082
15	130037.1000	290037.1000	210037.1310	110037.1000	210017.1000	210037.1130	120137.1200	210037.1210	210037.1000	310077.1200	201049.1105
16	210037.1210	210037.1130	120137.1200	430037.1000	190037.1000	210037.1000	310077.1200	110037.1000	110017.1000	210037.1310	211049.1105

17	310037.1130	120137.1200	210037.1210	210037.1000	310077.1200	130037.1000	190037.1000	210037.1310	310037.1000	110017.1000	211049.1105
18	210017.1000	210037.1130	120137.1200	210037.1210	210037.1000	330037.1000	190037.1000	210037.1310	110037.1000	210017.1000	201043.1085
19	190037.1000	210037.1000	310077.1200	30037.1000	210017.1000	210037.1210	210037.1130	120137.1200	390037.1000	390037.1000	227049.1074
20	310077.1200	130037.1000	190037.1000	210037.1310	310037.1000	310037.1130	120137.1200	210037.1210	210037.1000	310077.1200	231055.1125

iv. Total Nilai *Average Packet Loss Node Client* Penjadwalan *Least Connection*

Total *Average Packet Loss Node Server* 2 Algoritma Penjadwalan *Least Connection*

Play Forward	***Client Node* 0**	***Client Node* 1**	***Client Node* 7**	***Client Node*8**	***Client Node* 9**	***Client Node*14**	***Client Node*15**	***Client Node* 16**	***Client Node*19**	***Client Node* 21**	***Average***
1	0%	0%	0%	0%	0%	1.74%	7.36%	0%	0%	0%	0.91%
2	0%	0%	0%	0%	0%	1.39%	10.51%	0%	0%	0%	1.19%
3	0%	0%	0%	0%	0%	1.29%	10.41%	0%	0%	0%	1.17%
4	0%	0%	0%	0%	0%	1.84%	10.41%	0%	0%	0%	1.23%
5	0%	0%	0%	0%	0%	1.49%	9.29%	0%	0%	0%	1.08%
6	0%	0%	0%	0%	0%	5.63%	9.09%	0%	0%	0%	1.47%
7	0%	0%	0%	0%	0%	8.17%	8.98%	0%	0%	0%	1.72%
8	0%	0%	0%	0%	0%	8.07%	10.20%	0%	0%	0%	1.83%
9	0%	0%	0%	0%	0%	8.88%	9.29%	0%	0%	0%	1.82%
10	0%	0%	0%	0%	0%	7.26%	10.00%	0%	0%	0%	1.73%
11	0%	0%	0%	0%	0%	8.58%	10.30%	0%	0%	0%	1.89%
12	0%	0%	0%	0%	0%	7.38%	9.49%	0%	0%	0%	1.69%
13	0%	0%	0%	0%	0%	8.98%	8.78%	0%	0%	0%	1.78%
14	0%	0%	0%	0%	0%	8.58%	1.06%	0%	0%	0%	0.96%
15	0%	0%	0%	0%	0%	8.27%	4.09%	0%	0%	0%	1.24%

16	0%	0%	0%	0%	0%	6.90%	10.80%	0%	0%	0%	1.77%
17	0%	0%	0%	0%	0%	6.95%	2.20%	0%	0%	0%	0.92%
18	0%	0%	0%	0%	0%	7.87%	7.56%	0%	0%	0%	1.54%
19	0%	0%	0%	0%	0%	8.27%	8.38%	0%	0%	0%	1.67%
20	0%	2.03%	2.49%	4.11%	3.86%	5.65%	2.89%	5.38%	6.66%	2.14%	3.52%
***Total Average Packet Loss Node Server* 2**											**1.55%**

v. Total Nilai *Average Packet Loss Node Client* Penjadwalan *Weight Round Robin*

Total *Average Packet Loss Node Server* 2 Algoritma Penjadwalan *Algoritma Round Robin*

Play Forward	***Client Node* 0**	***Client Node* 1**	***Client Node* 7**	***Client Node*8**	***Client Node* 9**	***Client Node*14**	***Client Node*15**	***Client Node* 16**	***Client Node*19**	***Client Node* 21**	***Average***
1	11.27%	10.27%	11.27%	10.27%	12.27%	23.74%	27.36%	0%	0%	0%	10.65%
2	12.77%	13.77%	12.77%	19.77%	12.77%	28.39%	29.51%	0%	0%	0%	12.98%
3	12.97%	11.97%	12.97%	11.97%	13.97%	24.29%	29.41%	0%	0%	0%	11.76%
4	19.27%	9.27%	9.27%	19.27%	11.27%	28.84%	39.41%	0%	0%	0%	13.66%
5	10.27%	10.27%	10.27%	10.27%	10.27%	27.49%	29.29%	0%	0%	0%	10.81%
6	19.27%	9.27%	9.27%	19.27%	11.27%	25.63%	39.09%	0%	0%	0%	13.31%
7	11.87%	11.87%	11.87%	11.87%	11.27%	28.17%	38.98%	0%	0%	0%	12.59%
8	11.29%	11.29%	11.29%	11.29%	10.27%	28.07%	39.20%	0%	0%	0%	12.27%
9	9.97%	9.97%	9.97%	19.97%	12.97%	28.88%	39.29%	0%	0%	0%	13.10%
10	19.27%	17.27%	9.27%	19.27%	13.27%	27.26%	39.00%	0%	0%	0%	14.46%
11	12.87%	19.87%	12.87%	19.87%	12.87%	28.58%	39.30%	0%	0%	0%	14.62%
12	19.27%	19.29%	9.27%	19.27%	10.27%	27.38%	39.49%	0%	0%	0%	14.42%
13	11.27%	9.27%	11.27%	19.27%	13.27%	28.98%	38.78%	0%	0%	0%	13.21%
14	9.27%	9.27%	9.27%	17.27%	11.27%	28.58%	31.06%	0%	0%	0%	11.60%
15	13.27%	9.27%	13.27%	12.27%	9.27%	28.27%	29.09%	0%	0%	0%	11.47%
16	10.27%	10.27%	10.27%	10.27%	10.27%	56.90%	39.80%	0%	0%	0%	14.81%

17	14.27%	9.27%	14.27%	19.27%	9.27%	56.95%	32.20%	0%	0%	0%	15.55%
18	19.27%	9.27%	9.27%	19.27%	9.27%	37.87%	37.56%	0%	0%	0%	14.18%
19	13.27%	13.27%	13.27%	13.27%	13.27%	28.27%	48.38%	0%	0%	0%	14.30%
20	15.38%	15.38%	15.38%	15.38%	15.38%	35.65%	42.89%	16.66%	16.66%	12.14%	20.09%
***Total Average Packet Loss Node Server* 2**											**13.49%**

vi. Total Nilai *Average Packet Loss Node Client* Penjadwalan *Round Robin*

Total *Average Packet Loss Node Server* 2 Algoritma Penjadwalan *Algoritma Round Robin*											
Play Forward	***Client Node* 0**	***Client Node* 1**	***Client Node* 7**	***Client Node*8**	***Client Node* 9**	***Client Node*14**	***Client Node*15**	***Client Node* 16**	***Client Node*19**	***Client Node* 21**	***Average***
1	11.27%	10.27%	11.27%	10.27%	9.27%	23.74%	27.36%	0%	0%	0%	10.35%
2	12.77%	13.77%	12.77%	19.77%	9.77%	28.39%	29.51%	0%	0%	0%	12.68%
3	12.97%	11.97%	12.97%	11.97%	9.97%	24.29%	29.41%	0%	0%	0%	11.36%
4	9.27%	9.27%	9.27%	19.27%	9.27%	28.84%	39.41%	0%	0%	0%	12.46%
5	10.27%	10.27%	10.27%	10.27%	10.27%	27.49%	29.29%	0%	0%	0%	10.81%
6	9.27%	9.27%	9.27%	19.27%	9.27%	25.63%	39.09%	0%	0%	0%	12.11%
7	11.87%	11.87%	11.87%	11.87%	11.27%	28.17%	38.98%	0%	0%	0%	12.59%
8	11.29%	11.29%	11.29%	11.29%	9.27%	28.07%	39.20%	0%	0%	0%	12.17%
9	9.97%	9.97%	9.97%	19.97%	9.97%	28.88%	39.29%	0%	0%	0%	12.80%
10	9.27%	17.27%	9.27%	19.27%	9.27%	27.26%	39.00%	0%	0%	0%	13.06%
11	12.87%	19.87%	12.87%	19.87%	9.87%	28.58%	39.30%	0%	0%	0%	14.32%
12	9.27%	19.29%	9.27%	19.27%	9.27%	27.38%	39.49%	0%	0%	0%	13.32%
13	11.27%	9.27%	11.27%	19.27%	9.27%	28.98%	38.78%	0%	0%	0%	12.81%
14	9.27%	9.27%	9.27%	17.27%	9.27%	28.58%	31.06%	0%	0%	0%	11.40%
15	13.27%	9.27%	13.27%	12.27%	9.27%	28.27%	29.09%	0%	0%	0%	11.47%

16	10.27%	10.27%	10.27%	10.27%	10.27%	56.90%	39.80%	0%	0%	0%	14.81%
17	14.27%	9.27%	14.27%	19.27%	9.27%	56.95%	32.20%	0%	0%	0%	15.55%
18	9.27%	9.27%	9.27%	19.27%	9.27%	37.87%	37.56%	0%	0%	0%	13.18%
19	13.27%	13.27%	13.27%	13.27%	13.27%	28.27%	48.38%	0%	0%	0%	14.30%
20	15.38%	15.38%	15.38%	15.38%	15.38%	35.65%	42.89%	16.66%	16.66%	12.14%	20.09%
***Total Average Packet Loss Node Server* 2**											**13.08%**

vii. Total Nilai *Average Delay Node Client* Penjadwalan *Least Connection*

Total *Average Delay Node Server* 2 Algoritma Penjadwalan *Algoritma Least Connection*											
Play Forward	***Client Node*0**	***Client Node* 1**	***Client Node* 7**	***Client Node* 8**	***Client Node*9**	***Client Node* 14**	***Client Node* 15**	***Client Node* 16**	***Client Node* 19**	***Client Node* 21**	***Average Delay (ms) Client Node***
1	0.21057	0.110739	0.096411	0.069705	0.049440	0.149166	0.149445	0.06615	0.046698	0.047976	0.099630
2	0.11057	0.010739	0.066495	0.069726	0.050046	0.152995	0.152019	0.066893	0.046698	0.047976	0.077416
3	0.11057	0.010739	0.066514	0.069746	0.050062	0.153663	0.152707	0.066898	0.046698	0.047975	0.077557
4	0.11057	0.010739	0.066465	0.069732	0.050016	0.154003	0.152934	0.066954	0.046698	0.047975	0.077609
5	0.01057	0.010739	0.066506	0.069771	0.050093	0.153345	0.152585	0.066982	0.046698	0.047975	0.067526
6	0.01057	0.010739	0.066458	0.069667	0.050124	0.155379	0.153645	0.067033	0.046698	0.047976	0.067829
7	0.01057	0.010739	0.066432	0.069616	0.050162	0.153938	0.152705	0.067055	0.046698	0.047976	0.067589
8	0.01057	0.010739	0.066454	0.069621	0.050046	0.15512	0.152925	0.067061	0.046698	0.047976	0.067721
9	0.01057	0.010739	0.066471	0.069626	0.050067	0.154728	0.154136	0.067053	0.046698	0.047976	0.067806
10	0.01057	0.010739	0.066436	0.069636	0.050119	0.155467	0.153604	0.067043	0.046698	0.047976	0.067829
11	0.01057	0.010739	0.066324	0.069593	0.050054	0.154797	0.153716	0.067036	0.046698	0.047976	0.067750
12	0.01057	0.210739	0.066363	0.069613	0.050088	0.152265	0.152752	0.067035	0.046698	0.047975	0.087410
13	0.01057	0.010739	0.066402	0.069639	0.050156	0.152748	0.153377	0.067034	0.046698	0.047975	0.067534
14	0.01057	0.210739	0.066314	0.069581	0.050214	0.153521	0.153209	0.067017	0.046698	0.047975	0.087584
15	0.01057	0.110739	0.066358	0.069608	0.050171	0.153846	0.153842	0.066989	0.046698	0.047975	0.077680

16	0.01057	0.010739	0.066458	0.069651	0.050169	0.153414	0.153438	0.066978	0.046698	0.047975	0.067609
17	0.01057	0.110739	0.06646	0.069617	0.050093	0.154262	0.15357	0.066961	0.046698	0.047975	0.077695
18	0.01057	0.010739	0.066561	0.069703	0.050108	0.15417	0.154222	0.066953	0.046698	0.047975	0.067770
19	0.01057	0.010739	0.066601	0.069774	0.050126	0.152855	0.153302	0.066942	0.046698	0.047975	0.067558
20	0.01057	0.010765	0.066846	0.070189	0.050544	0.154558	0.15471	0.067296	0.046820	0.048462	0.068076
***Total Average Delay Node Server* 2**											**0.073759**

viii. Total Nilai *Average Delay Node Client* Penjadwalan *Weighted Round Robin*

Total *Average Delay Node Server* 2 Algoritma Penjadwalan *Algoritma Least Connection*											
Play Forward	***Client Node*0**	***Client Node* 1**	***Client Node* 7**	***Client Node* 8**	***Client Node*9**	***Client Node* 14**	***Client Node* 15**	***Client Node* 16**	***Client Node* 19**	***Client Node* 21**	***Average Delay (ms) Client Node***
1	0.21057	0.110739	0.096411	0.069705	0.049440	0.249166	0.249445	0.066150	0.046698	0.047976	0.119630
2	0.11057	0.010739	0.066495	0.069726	0.050046	0.252995	0.252019	0.066893	0.046698	0.047976	0.097416
3	0.11057	0.010739	0.066514	0.069746	0.050062	0.253663	0.252707	0.066898	0.046698	0.047975	0.097557
4	0.11057	0.010739	0.066465	0.069732	0.050016	0.254003	0.252934	0.066954	0.046698	0.047975	0.097609
5	0.01057	0.010739	0.066506	0.069771	0.050093	0.253345	0.252585	0.066982	0.046698	0.047975	0.087526
6	0.01057	0.010739	0.066458	0.069667	0.050124	0.255379	0.153645	0.067033	0.046698	0.047976	0.077829
7	0.01057	0.010739	0.066432	0.069616	0.050162	0.253938	0.152705	0.067055	0.046698	0.047976	0.077589
8	0.01057	0.010739	0.066454	0.069621	0.050046	0.255120	0.152925	0.067061	0.046698	0.047976	0.077721
9	0.01057	0.010739	0.066471	0.069626	0.050067	0.254728	0.154136	0.067053	0.046698	0.047976	0.077806
10	0.01057	0.010739	0.066436	0.069636	0.050119	0.255467	0.153604	0.067043	0.046698	0.047976	0.077829
11	0.01057	0.010739	0.066324	0.069593	0.050054	0.254797	0.153716	0.067036	0.046698	0.047976	0.077750
12	0.01057	0.210739	0.066363	0.069613	0.050088	0.252265	0.152752	0.067035	0.046698	0.047975	0.097410
13	0.01057	0.010739	0.066402	0.069639	0.050156	0.252748	0.153377	0.067034	0.046698	0.047975	0.077534

14	0.01057	0.210739	0.066314	0.069581	0.050214	0.253521	0.153209	0.067017	0.046698	0.047975	0.097584
15	0.01057	0.110739	0.066358	0.069608	0.050171	0.253846	0.153842	0.066989	0.046698	0.047975	0.087680
16	0.01057	0.010739	0.066458	0.069651	0.050169	0.253414	0.153438	0.066978	0.046698	0.047975	0.077609
17	0.01057	0.110739	0.06646	0.069617	0.050093	0.254262	0.15357	0.066961	0.046698	0.047975	0.087695
18	0.01057	0.010739	0.066561	0.069703	0.050108	0.254170	0.154222	0.066953	0.046698	0.047975	0.077770
19	0.01057	0.010739	0.066601	0.069774	0.050126	0.252855	0.153302	0.066942	0.046698	0.047975	0.077558
20	0.01057	0.010765	0.066846	0.070189	0.050544	0.254558	0.154710	0.067296	0.04682	0.048462	0.078076
***Total Average Delay Node Server* 2**											**0.086259**

ix. Total Nilai *Average Delay Node Client* Penjadwalan *Round Robin*

Total *Average Delay Node Server* 2 Algoritma Penjadwalan *Algoritma Round Robin*

Play Forward	***Client Node*0**	***Client Node* 1**	***Client Node* 7**	***Client Node* 8**	***Client Node*9**	***Client Node* 14**	***Client Node* 15**	***Client Node* 16**	***Client Node* 19**	***Client Node* 21**	***Average Delay***
1	0.11057	0.110739	0.196411	0.069705	0.049440	0.349166	0.249445	0.06615	0.046698	0.047976	0.129630
2	0.11057	0.010739	0.166495	0.069726	0.050046	0.352995	0.252019	0.066893	0.046698	0.047976	0.117416
3	0.11057	0.010739	0.166514	0.069746	0.050062	0.353663	0.252707	0.066898	0.046698	0.047975	0.117557
4	0.11057	0.010739	0.166465	0.069732	0.050016	0.354003	0.252934	0.066954	0.046698	0.047975	0.117609
5	0.11057	0.010739	0.166506	0.069771	0.050093	0.353345	0.252585	0.066982	0.046698	0.047975	0.117526
6	0.01057	0.010739	0.166458	0.069667	0.050124	0.355379	0.253645	0.067033	0.046698	0.047976	0.107829
7	0.21057	0.010739	0.166432	0.069616	0.050162	0.353938	0.252705	0.067055	0.046698	0.047976	0.127589
8	0.11057	0.010739	0.166454	0.069621	0.050046	0.355120	0.252925	0.067061	0.046698	0.047976	0.117721
9	0.11057	0.010739	0.166471	0.069626	0.050067	0.354728	0.254136	0.067053	0.046698	0.047976	0.117806
10	0.21057	0.010739	0.166436	0.069636	0.050119	0.355467	0.253604	0.067043	0.046698	0.047976	0.127829
11	0.11057	0.010739	0.166324	0.069593	0.150054	0.354797	0.253716	0.067036	0.046698	0.047976	0.127750

12	0.21057	0.210739	0.166363	0.069613	0.150088	0.352265	0.252752	0.067035	0.046698	0.047975	0.157410

Total Nilai *Average Throughput Node Client Geolocation Domain Name System*

13	0.11057	0.010739	0.166402	0.069639	0.150156	0.352748	0.253377	0.067034	0.046698	0.047975	0.127534
14	0.11057	0.210739	0.166314	0.069581	0.150214	0.353521	0.253209	0.067017	0.046698	0.047975	0.147584
15	0.11057	0.110739	0.166358	0.069608	0.150171	0.353846	0.253842	0.066989	0.046698	0.047975	0.137680
16	0.11057	0.010739	0.166458	0.069651	0.150169	0.353414	0.253438	0.066978	0.046698	0.047975	0.127609
17	0.11057	0.110739	0.166460	0.069617	0.150093	0.354262	0.253570	0.066961	0.046698	0.047975	0.137695
18	0.21057	0.010739	0.166561	0.069703	0.150108	0.354170	0.254222	0.066953	0.046698	0.047975	0.137770
19	0.21057	0.010739	0.166601	0.069774	0.150126	0.352855	0.253302	0.066942	0.046698	0.047975	0.137558
20	0.31057	0.010765	0.166846	0.070189	0.150544	0.354558	0.254710	0.067296	0.04682	0.048462	0.148076
***Total Average Delay Node Server* 2**											**0.129259**

x. Total Nilai Average Throughput Node Client Geolocation Domain Name System

Play Forward	*Client Node0*	*Client Node1*	*Client Node7*	*Client Node8*	*Client Node9*	*Client Node14*	*Client Node15*	*Client Node16*	*Client Node19*	*Client Node21*	*Average Node Client*
1	279000.6750	279000.6750	279000.6750	279000.6750	279000.6750	236000.5260	229000.3260	279000.6750	279000.6750	279000.6750	269700.6252
2	279000.6750	279000.6750	279000.6750	279000.6750	279000.6750	226000.0230	218000.9710	279000.6750	279000.6750	279000.6750	267600.6394
3	279000.6750	279000.6750	279000.6750	279000.6750	279000.6750	224000.5460	220.0007	279000.6750	279000.6750	279000.6750	245622.5947
4	279000.6750	279000.6750	279000.6750	279000.6750	279000.6750	233000.8040	220000.1770	279000.6750	279000.6750	279000.6750	268500.6381
5	279000.6750	279000.6750	279000.6750	279000.6750	279000.6750	227000.3030	224000.6020	279000.6750	279000.6750	279000.6750	268300.6305
6	279000.6750	279000.6750	279000.6750	279000.6750	279000.6750	236000.6860	224000.1600	279000.6750	279000.6750	279000.6750	269200.6246
7	279000.6750	279000.6750	279000.6750	279000.6750	279000.6750	228000.6490	224000.4110	279000.6750	279000.6750	279000.6750	268400.6460
8	279000.6750	279000.6750	279000.6750	279000.6750	279000.6750	229000.5080	223000.0990	279000.6750	279000.6750	279000.6750	268400.6007
9	279000.6750	279000.6750	279000.6750	279000.6750	279000.6750	225000.4730	223000.4170	279000.6750	279000.6750	279000.6750	268000.6290
10	279000.6750	279000.6750	279000.6750	279000.6750	279000.6750	228000.9140	220000.5010	279000.6750	279000.6750	279000.6750	268000.6815
11	279000.6750	279000.6750	279000.6750	279000.6750	279000.6750	227000.7320	221000.7290	279000.6750	279000.6750	279000.6750	268000.6861
12	279000.6750	279000.6750	279000.6750	279000.6750	279000.6750	226000.5770	224000.5770	279000.6750	279000.6750	279000.6750	268200.6554
13	279000.6750	279000.6750	279000.6750	279000.6750	279000.6750	226000.7950	225000.2970	279000.6750	279000.6750	279000.6750	268300.6492
14	279000.6750	279000.6750	279000.6750	279000.6750	279000.6750	226000.2730	230000.7580	279000.6750	279000.6750	279000.6750	268800.6431
15	279000.6750	279000.6750	279000.6750	279000.6750	279000.6750	227000.7860	224000.9660	279000.6750	279000.6750	279000.6750	268300.7152
16	279000.6750	279000.6750	279000.6750	279000.6750	279000.6750	222000.4670	221000.4270	279000.6750	279000.6750	279000.6750	267500.6294
17	279000.6750	279000.6750	279000.6750	279000.6750	279000.6750	230000.8620	220000.0000	279000.6750	279000.6750	279000.6750	268200.6262
18	279000.6750	279000.6750	279000.6750	279000.6750	279000.6750	229000.2150	228000.8680	279000.6750	279000.6750	279000.6750	268900.6483
19	279000.6750	279000.6750	279000.6750	279000.6750	279000.6750	227000.8350	225000.3300	279000.6750	279000.6750	279000.6750	268400.6565
20	279000.6750	274000.5520	245000.2350	240000.3660	269000.7530	213000.8050	205000.3230	265000.2860	249000.4220	234000.4840	247300.4901
Total Nilai *Average Throughput Node Client Geolocation Domain Name System*											**266181.7355**

xi. Total Nilai Average Throughput Node Client Geolocation Domain Name System

Total *Average Packet Loss Node Client Geolocation Domain Name System*

Play Forward	***Client Node* 0**	***Client Node* 1**	***Client Node* 7**	***Client Node*8**	***Client Node* 9**	***Client Node*14**	***Client Node*15**	***Client Node* 16**	***Client Node*19**	***Client Node* 21**	***Average***
1	0%	0%	0%	0%	0%	15.74%	17.36%	0%	0%	0%	3.31%
2	0%	0%	0%	0%	0%	19.39%	20.51%	0%	0%	0%	3.99%
3	0%	0%	0%	0%	0%	19.29%	20.41%	0%	0%	0%	3.97%
4	0%	0%	0%	0%	0%	15.84%	20.41%	0%	0%	0%	3.63%
5	0%	0%	0%	0%	0%	19.49%	19.29%	0%	0%	0%	3.88%
6	0%	0%	0%	0%	0%	15.63%	19.09%	0%	0%	0%	3.47%
7	0%	0%	0%	0%	0%	18.17%	18.98%	0%	0%	0%	3.72%
8	0%	0%	0%	0%	0%	18.07%	20.20%	0%	0%	0%	3.83%
9	0%	0%	0%	0%	0%	18.88%	19.29%	0%	0%	0%	3.82%
10	0%	0%	0%	0%	0%	17.26%	20.00%	0%	0%	0%	3.73%
11	0%	0%	0%	0%	0%	18.58%	20.30%	0%	0%	0%	3.89%
12	0%	0%	0%	0%	0%	18.38%	19.49%	0%	0%	0%	3.79%
13	0%	0%	0%	0%	0%	18.98%	18.78%	0%	0%	0%	3.78%
14	0%	0%	0%	0%	0%	18.58%	17.06%	0%	0%	0%	3.56%
15	0%	0%	0%	0%	0%	18.27%	19.09%	0%	0%	0%	3.74%
16	0%	0%	0%	0%	0%	19.90%	19.80%	0%	0%	0%	3.97%
17	0%	0%	0%	0%	0%	16.95%	20.20%	0%	0%	0%	3.72%
18	0%	0%	0%	0%	0%	17.87%	17.56%	0%	0%	0%	3.54%
19	0%	0%	0%	0%	0%	18.27%	18.38%	0%	0%	0%	3.67%
20	0%	2.03%	12.49%	14.11%	3.86%	23.65%	25.89%	5.38%	10.66%	16.14%	11.42%
Total Average Packet Loss Node Client Geolocation Domain Name System											**4.12%**

xii. Total Nilai Average Delay Node Client Geolocation Domain Name System

Total ***Average Delay Node client Geolocation Domain Name System***											
Play Forward	***Client Node*****0**	***Client Node* 1**	***Client Node* 7**	***Client Node* 8**	***Client Node*****9**	***Client Node* 14**	***Client Node* 15**	***Client Node* 16**	***Client Node* 19**	***Client Node* 21**	***Average Delay (ms) Client Node***
1	0.01057	0.010739	0.066411	0.069705	0.049440	0.149166	0.149445	0.066150	0.046698	0.047976	0.066630
2	0.01057	0.010739	0.066495	0.069726	0.050046	0.152995	0.152019	0.066893	0.046698	0.047976	0.067416
3	0.01057	0.010739	0.066514	0.069746	0.050062	0.153663	0.152707	0.066898	0.046698	0.047975	0.067557
4	0.01057	0.010739	0.066465	0.069732	0.050016	0.154003	0.152934	0.066954	0.046698	0.047975	0.067609
5	0.01057	0.010739	0.066506	0.069771	0.050093	0.153345	0.152585	0.066982	0.046698	0.047975	0.067526
6	0.01057	0.010739	0.066458	0.069667	0.050124	0.155379	0.153645	0.067033	0.046698	0.047976	0.067829
7	0.01057	0.010739	0.066432	0.069616	0.050162	0.153938	0.152705	0.067055	0.046698	0.047976	0.067589
8	0.01057	0.010739	0.066454	0.069621	0.050046	0.15512	0.152925	0.067061	0.046698	0.047976	0.067721
9	0.01057	0.010739	0.066471	0.069626	0.050067	0.154728	0.154136	0.067053	0.046698	0.047976	0.067806
10	0.01057	0.010739	0.066436	0.069636	0.050119	0.155467	0.153604	0.067043	0.046698	0.047976	0.067829
11	0.01057	0.010739	0.066324	0.069593	0.050054	0.154797	0.153716	0.067036	0.046698	0.047976	0.067750
12	0.01057	0.010739	0.066363	0.069613	0.050088	0.152265	0.152752	0.067035	0.046698	0.047975	0.067410
13	0.01057	0.010739	0.066402	0.069639	0.050156	0.152748	0.153377	0.067034	0.046698	0.047975	0.067534
14	0.01057	0.010739	0.066314	0.069581	0.050214	0.153521	0.153209	0.067017	0.046698	0.047975	0.067584
15	0.01057	0.010739	0.066358	0.069608	0.050171	0.153846	0.153842	0.066989	0.046698	0.047975	0.067680
16	0.01057	0.010739	0.066458	0.069651	0.050169	0.153414	0.153438	0.066978	0.046698	0.047975	0.067609
17	0.01057	0.010739	0.06646	0.069617	0.050093	0.154262	0.153570	0.066961	0.046698	0.047975	0.067695
18	0.01057	0.010739	0.066561	0.069703	0.050108	0.15417	0.154222	0.066953	0.046698	0.047975	0.067770
19	0.01057	0.010739	0.066601	0.069774	0.050126	0.152855	0.153302	0.066942	0.046698	0.047975	0.067558
20	0.01057	0.010765	0.066846	0.070189	0.050544	0.154558	0.154710	0.067296	0.046820	0.048462	0.068076
Total Average Delay Node client Geolocation Domain Name System											**0.067609**

RIWAYAT HIDUP SINGKAT PENULIS

|+6282387308716| Hillman Akhyar Damanik

Email: 1611601053@student.budiluhur.ac.id

Sekilas dari penulis dengan nama Hillman Akhyar Damanik, lahir di SUMUT. Anak ke-2 dari 9 bersaudara dengan Pendidikan: SDN 121142 selanjutnya penulis lalu melanjutkan ke sekolah Menengah Pertama SMPN Teknik Arsitektur Bangunan 12, selanjutnya melanjutkan pendidikan jenjang SMA Erlangga Sumatera Utara. Menyelesaikan Pendidikan S1 di Universitas Muhammadiyah dengan mengambil Jurusan Teknik Informatika, dengan membuat Skripsi untuk memenuhi syarat Sarjana (S1) dengan judul: Evaluasi Kinerja *Routing Protocol Open Short Path First*. Selain aktif pada bidang akademisi penulis juga sampai saat ini memiliki pengalaman bekerja pada perusahaan Telekomunikasi Satellite yang ada di Indonesia. Penulis juga aktif pada bidang-bidang kemanusian dengan mengikuti kegiatan-kegiatan Palang Merah Indonesia.