ANALISIS UNJUK KERJA ALGORITMA *COOPERATIVE Q-LEARNING* SEBAGAI KENDALI KONGESTI PADA JARINGAN OPORTUNISTIK

SKRIPSI

Diajukan Untuk Memenuhi Salah Satu Syarat

Memperoleh Gelar Sarjana Komputer

Program Studi Informatika

Oleh:

Maria Christinne Aprijany Tjung

185314037

PROGRAM STUDI INFORMATIKA

JURUSAN INFORMATIKA

FAKULTAS SAINS DAN TEKNOLOGI

UNIVERSITAS SANATA DHARMA

YOGYAKARTA

2022

PERFORMANCE ANALYSIS OF COOPERATIVE Q-LEARNING ALGORITHM AS CONGESTION CONTROL IN OPPORTUNISTIC NETWORKS

THESIS

Present as Partial Fullfillment of the Requirements

For The Degree of *Sarjana Komputer*

In Informatics Department

By:

Maria Christinne Aprijany Tjung

185314037

INFORMATICS STUDY PROGRAM

DEPARTMENT OF INFORMATICS

FACULTY OF SCIENCE AND TECHNOLOGY

SANATA DHARMA UNIVERSITY

YOGYAKARTA

2022

HALAMAN PERSETUJUAN SKRIPSI

ANALISIS UNJUK KERJA ALGORITMA *COOPERATIVE Q-LEARNING* SEBAGAI KENDALI KONGESTI PADA JARINGAN OPORTUNISTIK

Oleh:

Maria Christinne Aprijany Tjung (185314037)

Telah disetujui oleh:

Dosen Pembimbing

Bambang Soelistijanto, S.T., M.Sc., Ph.D. **Tanggal, 25 Juli 2022**

HALAMAN PENGESAHAN SKRIPSI

ANALISIS UNJUK KERJA ALGORITMA *COOPERATIVE Q-LEARNING* SEBAGAI KENDALI KONGESTI PADA JARINGAN OPORTUNISTIK

Dipersiapkan dan disusun oleh:

MARIA CHRISTINNE APRIJANY TJUNG

NIM: 185314037

Telah dipertahankan di depan panitia penguji

Pada tanggal 19 Juli 2022

Dan dinyatakan memenuhi syarat

Susunan Panitia Penguji

Nama Lengkap		Tanda Tangan
Ketua	: Vittalis Ayu S.T., M.Cs.	.....
Sekretaris	: Henricus Agung Hermawan S.T., M.Kom.	.....
Anggota	: Bambang Soelistijanto S.T., M.Sc., Ph.D	.....

Yogyakarta, 25 Juli 2022

Fakultas Sains dan Teknologi

Universitas Sanata Dharma

Pjs. Dekan,

Ir. Damar Widjaja Ph.D.

MOTTO

“Biarkan saja mereka tertawa. Kalau tidak pernah berjuang sampai akhir, kita tidak akan pernah melihatnya walau ada di depan mata.”

- Marshall D Teach (One Piece)”

PERNYATAAN LEMBAR KEASLIAN KARYA

Saya menyatakan dengan sesungguhnya bahwa di dalam skripsi yang saya tulis ini tidak memuat karya atau bagian karya orang lain, kecuali yang telah disebutkan dalam kutipan daftar pustaka, sebagaimana layaknya karya ilmiah.

Yogyakarta, 19 Juli 2022

Penulis,

Maria Christinne Aprijany Tjung

LEMBAR PERSETUJUAN PUBLIKASI KARYA ILMIAH UNTUK KEPENTINGAN AKADEMIS

Yang bertanda tangan dibawah ini, saya mahasiswa Universitas Sanata Dharma:

Nama : Maria Christinne Aprijany Tjung

NIM : 185314037

Demi pengembangan ilmu pengetahuan, saya memberikan kepada Perpustakaan Universitas Sanata Dharma karya ilmiah yang berjudul:

ANALISIS UNJUK KERJA ALGORITMA *COOPERATIVE Q-LEARNING* SEBAGAI KENDALI KONGESTI PADA JARINGAN OPORTUNISTIK

Beserta perangkat yang diperlukan (bila ada). Dengan demikian saya memberikan kepada Perpustakaan Universitas Sanata Dharma hak untuk menyimpan, mengalihkan dalam bentuk media lain, mengolahnya dalam bentuk pangkalan data, mendistribusikannya secara terbatas, dan mempublikasikannya di Internet atau media lain untuk kepentingan akademis tanpa meminta ijin dari saya maupun memberikan royalty kepada saya selama tetap mencantumkan nama saya sebagai penulis.

Demikian pernyataan ini saya buat dengan sebenarnya.

Yogyakarta, 19 Juli 2022

Penulis,

Maria Christinne Aprijany Tjung

ABSTRAK

Kongesti merupakan suatu kondisi dimana beban pada jaringan melebihi kapasitas jaringan itu sendiri dan dapat menyebabkan turunnya kinerja jaringan, sehingga suatu kendali diperlukan untuk mencegah terjadinya kongesti. Pada Internet, kongesti biasanya dikendalikan oleh *Transmission Control Protocol* (TCP), yang menerapkan mekanisme *additive-increase*/*multiplicative-decrease* (AIMD) dalam mengontrol jumlah pesan yang disebarkan. Mekanisme TCP-AIMD ini diadopsi oleh algoritma *Retiring Replicants* (RR) sehingga dapat diterapkan di Jaringan Oportunistik yang memiliki waktu tunda yang lama dalam menyampaikan informasi. Pada RR, kongesti diamati berdasarkan suatu nilai kongesti (*Congestion Value*, CV). Namun, berdasarkan pengamatan peneliti, CV pada RR mengalami fluktuasi yang cukup sering sehingga dapat menyebabkan kinerja jaringan menjadi kurang optimal. Pada penelitian ini, diterapkan suatu metode untuk memperbaiki RR dengan menerapkan *Reinforcement Learning: Q-Learning*, yang membuat *node* mampu untuk belajar secara mandiri dalam mengurangi fluktuasi terhadap CV. Tetapi, pada kebanyakan algoritma *learning*, penerapannya dilakukan secara *independent* tanpa adanya kerja sama antara setiap *agent*, sedangkan masalah kongesti merupakan masalah umum yang dihadapi oleh semua *node* di jaringan. Menurut penelitian penulis, proses *learning* yang dilakukan secara *cooperative* memiliki hasil yang lebih baik dibandingkan proses *learning* yang dilakukan secara *independent*. Karena itu, di penelitian ini penulis mengimplementasikan algoritma *Q-Learning* yang bersifat *cooperative*, sehingga semua *node* dapat bekerja sama dalam memperbaiki fluktuasi pada CV guna menurunkan beban pada jaringan.

Kata kunci: Penanganan Kongesti, *Cooperative Agents*, *Reinforcement learning*, *Q-Learning*, Jaringan Oportunistik.

ABSTRACT

Congestion is a condition where the network load exceeds the capacity of the network. Congestion decreases the network performance. Therefore, a control is needed to prevent the congestion. In the Internet, congestion is handled by the Transmission Control Protocol (TCP), which applies an additive-increase/multiplicative-decrease (AIMD) mechanism to control the number of messages transferred. Retiring Replicants (RR) is an algorithm that adopts TCP-AIMD mechanism in order that it can be applied to Opportunistic Networks which have long delays in transmitting information. In RR, congestion is observed by a congestion value (CV). Otherwise, based on our observations, CV on RR fluctuates quite often that it can cause network performance to be less than optimal. In this study, a method is applied to improve RR by applying Reinforcement Learning: Q-Learning, which makes nodes able to learn independently in reducing fluctuations in CV. However, in most learning algorithms, the implementation is done independently without any cooperation between each agent, while the congestion problem is a common problem faced by all nodes in the network. According to our research, the learning process carried out in a cooperative manner has better results than the learning process carried out independently. Therefore, in this study we implement a Q-Learning algorithm that is cooperative to make all nodes can work together in optimizing CV in order to reduce the network load.

Keywords: Congestion Control, Cooperative Agents, Reinforcement Learning, Q-Learning, Opportunistic Networks.

KATA PENGANTAR

Puji dan syukur penulis panjatkan kehadirat Tuhan Yang Maha Esa, atas segala berkat, kasih dan kebaikan-Nya, sehingga penulis mampu untuk menyelesaikan penelitian tugas akhir yang berjudul "ANALISIS UNJUK KERJA ALGORITMA *COOPERATIVE Q-LEARNING* SEBAGAI KENDALI KONGESTI PADA JARINGAN OPORTUNISTIK" dengan baik dan lancar. Penelitian ini merupakan salah satu syarat yang harus dipenuhi untuk mendapatkan gelar Sarjana Komputer pada Program Studi S1 Informatika Fakultas Sains & Teknologi Universitas Sanata Dharma.

Penelitian ini tidak akan selesai tanpa adanya dukungan, semangat dan bantuan dari banyak pihak. Oleh karena itu, pada kesempatan ini penulis ingin mengucapkan terima kasih kepada:

1. Tuhan Yesus dan Bunda Maria, untuk segala kebaikan, anugerah, kasih, berkat dan penyertaan di dalam segala perkara yang penulis hadapi, sehingga penulis dapat sampai di titik ini.
2. Ayah tercinta Herson Tjung, yang selalu mendukung baik secara materi, rohani dan jasmani, serta selalu senantiasa mengantar-jemput ke kampus sehingga tugas akhir ini dapat terselesaikan dengan baik.
3. Adik tersayang Mario Leonardo Tjung, yang selalu memberikan dukungan dan semangat, serta menghibur dengan segala *jokes* yang membuat tertawa.
4. Bapak Bambang Soelistijanto, S.T., M.Sc., Ph.D. selaku dosen pembimbing Tugas Akhir penulis yang telah banyak membimbing penulis dan memberikan motivasi-motivasi yang sangat menguatkan penulis.
5. Ibu Vittalis Ayu, S.T., M.Cs. sebagai dosen yang selalu mendukung dan menyemangati penulis dalam menyelesaikan Tugas Akhir ini.
6. Bapak Ir. Kartono Pinaryanto S.T., M.Sc. selaku dosen pembimbing akademis penulis yang telah memberikan banyak bimbingan selama masa studi.

7. Bapak Budi Cahyono, Ibu Lani, Yayasan SCH, Bruder Flafianus, Ibu Imelda Lasakar, Ence Hendro, Pak Yohanes, Ence Yunus dan semua pihak yang telah membantu penulis dalam segala kebutuhan perkuliahan.
8. Teman seperjuangan Tugas Akhir, Acha dan Desy yang telah bersama-sama dengan penulis melewati segala suka-duka dalam penelitian Tugas Akhir ini, serta selalu menyemangati dan mendukung penulis. Dan memberikan hiburan (bermain UNO, mendengarkan lagu, serta menonton film horor dan anime) disela-sela mengerjakan Tugas Akhir ini.
9. Sahabat tersayang penulis Ade Melisa Suy yang selalu mendukung dan menyemangati penulis dalam segala hal.
10. Teman-teman Jarkom angkatan 2018 untuk segala kenangan dan kebersamaannya.
11. Teman-teman selama masa perkuliahan yang senantiasa mendukung penulis kapan pun.
12. Sahabat-sahabat SMGM yang selalu mendoakan dan mendukung penulis sehingga penulis mampu menyelesaikan Tugas Akhir ini.

Penulis menyadari bahwa dalam penulisan Tugas Akhir ini masih banyak memiliki kekurangan yang penulis sadari maupun tidak sadari. Oleh karena itu, penulis mengharapkan kritik dan saran yang membangun agar menjadi lebih baik kedepannya. Akhir kata, penulis mengharapkan Tugas Akhir ini dapat menjadi referensi dalam pembelajaran dan bermanfaat bagi pengembangan ilmu pengetahuan di bidang Informatika.

Penulis,

Maria Christinne Aprijany Tjung

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

DAFTAR RUMUS

BAB I
PENDAHULUAN

1.1.Latar Belakang

Pengiriman pesan yang efektif dan efisien merupakan tujuan utama yang ingin dicapai pada Jaringan Oportunistik[1] yang memiliki keterlambatan (*delay*) tinggi dan topologi yang selalu berubah-ubah, akibat dari *node* pada jaringan yang bergerak secara terus-menerus (*mobile*). Namun, karena ingin meningkatkan tingkat keberhasilan pengiriman, seringkali pesan disebarkan dalam jumlah replikasi yang banyak, padahal hal ini dapat memicu terjadinya kongesti jika jumlah pesan yang disebarkan di dalam jaringan melebihi kapasitas jaringan itu sendiri (*overhead*). Pada jaringan tradisional atau yang biasa kita kenal dengan Internet, kongesti ditangani oleh *Transmission Control Protocol* (TCP) yang menerapkan mekanisme *additive-increase/multiplicative-decrease* (AIMD) untuk menaikkan/mengurangi jumlah pesan yang dikirimkan. Akan tetapi, dikarenakan waktu tunda yang besar di Jaringan Oportunistik, mekanisme penanganan kongesti dari TCP menjadi kurang efektif untuk diterapkan akibat respon dari pihak pengirim yang lama dalam menangani kongesti.

Algoritma *Retiring Replicants* (RR)[2] telah memperbaiki metode AIMD yang dmiliki oleh TCP untuk dapat menangani kongesti di Jaringan Oportunistika. Jumlah pesan yang dikirimkan, dinaikkan/diturunkan berdasarkan nilai kongesti (*congestion value*, CV) yang diamati. CV pada algoritma RR merupakan rasio dari jumlah pesan yang dibuang terhadap jumlah pesan yang diterima (replikasi). Namun, berdasarkan pengamatan penulis, CV yang dihasilkan pada algoritma RR sangat sering berfluktuasi dan dapat menyebabkan kondisi jaringan menjadi kurang stabil. Akibatnya, kinerja yang dihasilkan untuk mengurangi *overhead* menjadi kurang optimal.

Pada penelitian ini, diusulkan sebuah perbaikan pada algoritma RR dengan menerapkan metode *machine learning* yang membuat sebuah *node* (*agent*) mampu untuk mempelajari kondisi (*state*) lingkungannya

(*environment*) dan kemudian menerapkan sebuah tindakan (*action*) yang dapat meningkatkan kinerja jaringan. Algoritma *Reinforcement Learning: Q-Learning* merupakan salah satu algoritma yang cukup populer digunakan. Bahkan, telah diterapkan dalam mengontrol kongesti pada Jaringan Oportunistik[3]. Akan tetapi, pada kebanyakan penelitian untuk mengendalikan kongesti di Jaringan Oportunistik, hanya menggunakan informasi lokal untuk mengamati terjadinya kongesti, salah satunya adalah pengamatan kepadatan *buffer*. Jika kepadatan *buffer* tinggi, maka itu menandakan terjadinya kongesti yang juga disebut sebagai kemacetan *buffer* (*buffer congestion*). Padahal, Thompson[2] mengatakan bahwa kondisi *buffer* setiap saat selalu hampir penuh meskipun kongesti tidak terjadi.

Pada penelitian yang dilakukan penulis, selain memperbaiki fluktuasi CV pada RR, penggunaan mekanisme *Q-Learning* dalam mengontrol kongesti diterapkan tidak berdasarkan *buffer*, melainkan menggunakan CV untuk mengamati tanda terjadinya kongesti. Akan tetapi, pada kebanyakan algoritma *learning*, proses pembelajaran dilakukan secara *independent* oleh setiap *agents* tanpa adanya kerja sama. Sedangkan masalah kongesti merupakan masalah umum yang dihadapi oleh semua *node* di jaringan, sehingga membutuhkan kerjasama dari setiap *node* agar kondisi dalam jaringan bisa semakin stabil. Maka pada penelitian ini, penulis mengusulkan *Cooperative Q-Learning Congestion Control* (CQLCC) yang mengimplementasikan algoritma *Q-Learning* yang bersifat *cooperative*[4] dalam menangani kongesti, sehingga semua *node* dapat bekerja sama dalam menurunkan beban pada jaringan dengan tujuan mengurangi terjadinya kongesti.

1.2.Rumusan Masalah

Berdasarkan latar belakang diatas, rumusan masalah yang didapat adalah seberapa efektif dan efisien unjuk kerja dari algoritma *Q-Learning* yang *cooperative* dalam mengurangi *overhead* pada jaringan untuk mengatasi kongesti.

1.3.Batasan Masalah

Batasan masalah pada penelitian ini adalah sebagai berikut:

1. Menggunakan *The One Simulator* dalam pengujian[5]
2. Menggunakan algoritma *Cooperative Q-Learning*
3. Setiap *node* hanya dapat memilih 1 tindakan *(action)* pada sebuah keadaan *(state).*
4. *State-action space*-nya adalah 4×8.
5. *State space*-nya bersifat diskrit.
6. Menggunakan parameter unjuk kerja:
   - *Delivery Ratio*
   - *End-to-end Latency*
   - *Goodput*
   - *Overhead*

1.4.Tujuan Penelitian

Tujuan pada penelitian ini adalah untuk melihat unjuk kerja dari algoritma *Cooperative Q-Learning* dalam mengatasi kongesti di Jaringan Oportunistik.

1.5.Manfaat Penelitian

Manfaat dilakukannya penelitian ini adalah untuk membantu mengatasi kongesti di Jaringan Oportunistik dengan menerapkan pembelajaran mandiri pada setiap *node*, yaitu dengan algoritma *Cooperative Q-Learning.*

1.6.Metodologi Penelitian

Adapun metodologi penelitian dan langkah-langkah yang digunakan dalam pelaksanaan penelitian ini adalah sebagai berikut:

1. **Studi Literatur**

   Mengumpulkan berbagai macam referensi dan mempelajari teori-teori yang mendukung penulisan, seperti:

   a. Teori Kongesti di Jaringan Oportunistik

b. Teori Algoritma *Retiring Replicants*
c. Teori *Reinforcement Learning: Q-Learning*
d. Teori *Multi-Agent Reinforcement Learning: Independent Vs. Cooperative Agents*

2. Perancangan

Tahapan ini merupakan rancangan skenario yang digunakan dalam penelitian yang terdiri dari:

a. Pergerakan *node* yang digunakan Haggle03-Infocom 5[6] dan Random Waypoint

3. Pembangunan Simulasi dan Pengumpulan Data

Simulasi yang digunakan pada penelitian ini menggunakan *The One Simulator*[5].

4. Analisis Data Simulasi

Dalam tahap ini penulis menganalisis hasil pengukuran yang diperoleh pada proses simulasi. Analisis dihasilkan dengan melakukan pengamatan dari beberapa kali pengukuran dengan menggunakan parameter simulasi yang berbeda.

5. Penarikan Kesimpulan

Penarikan kesimpulan ini didasarkan pada beberapa hasil dari parameter unjuk kerja yang diperoleh pada proses analisis data.

1.7.Sistematika Penulisan

Sistematika penulisan dibagi menjadi beberapa bab dengan susunan sebagai berikut:

BAB I PENDAHULUAN

Bab ini membahasa tentang latar belakang, rumusan masalah, Batasan masalah, tujuan penelitian, manfaat penelitian, metodologi penelitian dan sistematika penulisan.

BAB II LANDASAN TEORI

Bab ini berisi tentang dasar teori yang digunakan sebagai dasar dalam melakukan penelitian ini.

BAB III DESAIN ALGORITMA

Bab ini berisi tentang desain algoritma yang dibangun untuk menangani kongesti di Jaringan Oportunistik dengan menerapkan algoritma *Cooperative Q-Learning*.

BAB IV PENGUJIAN DAN ANALISIS

Bab ini berisi tentang skenario simulasi, parameter simulasi dan matriks unjuk kerja dan analisis hasil simulasi.

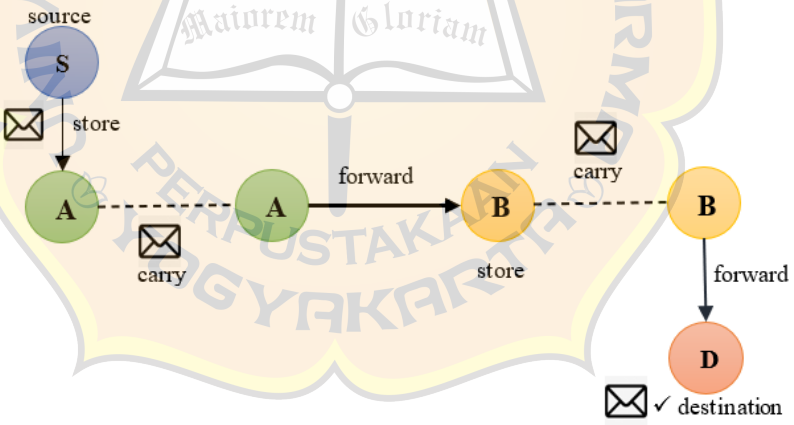
BAB V KESIMPULAN

Bab ini berisi tentang kesimpulan dari hasil pengujian dan analisis simulasi, serta saran untuk peneliti selanjutnya.

BAB II
LANDASAN TEORI

2.1.Kongesti di Jaringan Oportunistik

Kongesti yang terjadi di Jaringan Oportunistik disebabkan oleh jumlah replikasi yang berlebihan di dalam jaringan. Hal ini mengakibatkan *buffer* yang dimiliki oleh *node* menjadi kebanjiran dan penurunan tingkat keberhasilan pengiriman pesan. Pada Internet, kongesti biasanya ditangani oleh TCP yang menggunakan mekanisme AIMD dalam mengendalikan jumlah pesan yang disebarkan didalam jaringan. Akan tetapi, berbeda dengan Internet yang memiliki infrastruktur dan *end-to-end path*, di Jaringan Oportunistik pesan dikirimkan dengan mekanisme *store-carry-forward* (SCF) dan tidak memiliki *end-to-end path*. Akibatnya, pengiriman pesan akan memiliki waktu tunda yang sangat lama.

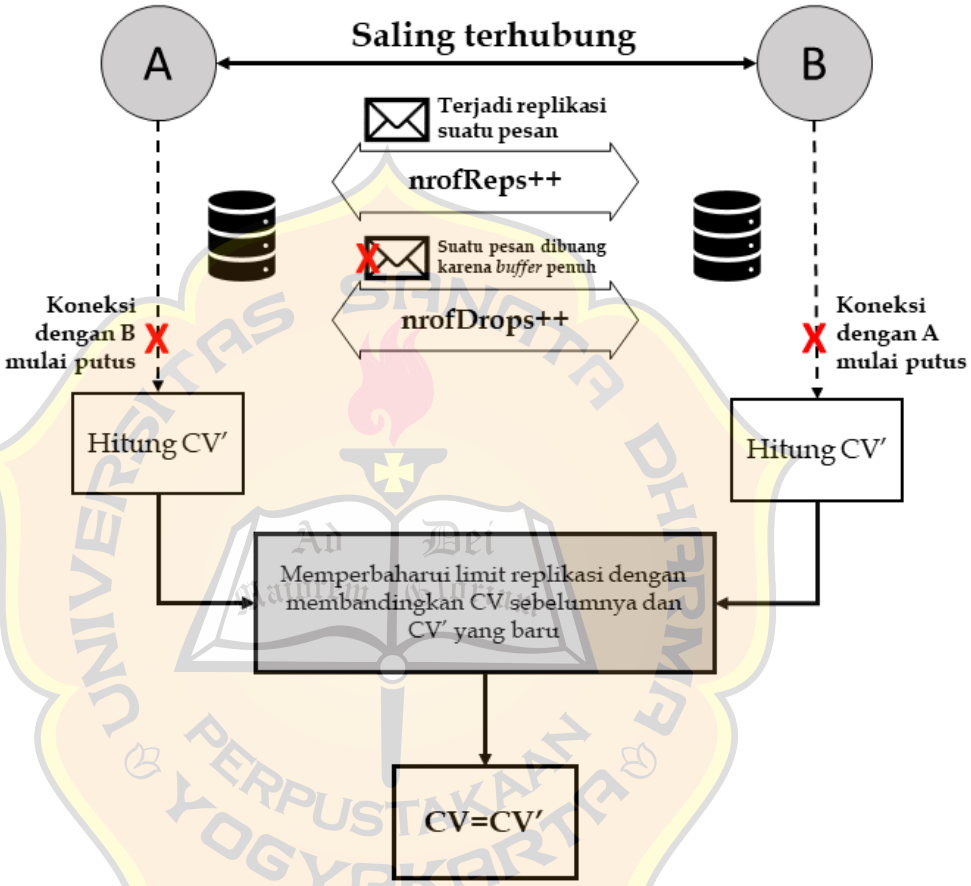


Gambar 2. 1. Mekanisme *Store-Carry-Forward* (SCF)

Pengiriman pesan dengan waktu tunda yang lama pada Jaringan Oportunistik, mengakibatkan TCP-AIMD tidak dapat diterapkan dengan baik pada jaringan ini. Karena, saat terjadi kongesti, maka informasi kongesti tersebut akan lambat diterima oleh *node*, dan akibatnya respon tersebut menjadi kurang efektif. Oleh karena itu, diperlukan suatu perbaikan pada mekanisme

penanganan kongesti agar dapat diterapkan dengan baik di Jaringan Oportunistik.

2.2.Algoritma *Retiring Replicants*



Gambar 2. 2. Mekanisme *Retiring Replicants*

Algoritma *Retiring Replicants*[2] adalah mekanisme untuk mengendalikan kongesti pada Jaringan Oportunistik yang mengadopsi metode TCP-AIMD. Jumlah pesan yang dikirimkan, akan dinaikkan/diturunkan berdasarkan CV yang merupakan rasio dari jumlah pesan yang dibuang terhadap jumlah pesan yang direplikasi. Naik/turunnya CV dari sebuah *node*,

dihitung menggunakan rumus *Exponentially Weighted Moving Average* (EWMA) dengan persamaan sebagai berikut:

$$CV' = \alpha \cdot (d/r) + (1 - \alpha) \cdot CV$$

Rumus 2. 1. *Congestion Value*

Dimana:

$d = nrofDrops + peer.nrofDrops$

$r = nrofReps + peer.nrofReps + \sum_{m \in stored\ messages\ m}(hops(m) - 1))$

$CV' =$ *Congestion Value* yang baru

$\alpha =$ *learning rate*

$CV =$ *Congestion Value* sebelumnya

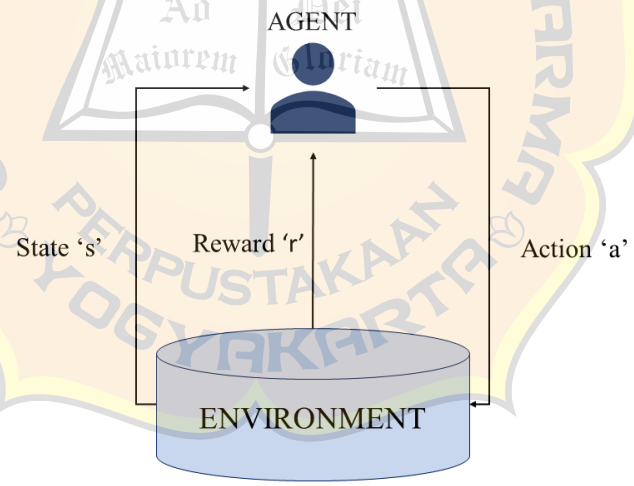
Seperti yang diperlihatkan pada Gambar 2.2, saat *node* A memiliki koneksi dengan *node* lain (*node* B), dan kemudian A menerima suatu pesan, maka A akan menambahkan nilai dari jumlah replikasi yang diterima olehnya. Demikian juga yang terjadi pada *node* B ketika menerima sebuah pesan. Namun, ketika ada pesan yang harus dibuang karena *buffer* penuh, maka *node* akan menambahkan nilai jumlah pesan yang dibuang olehnya. Setelah koneksi antara kedua *node* akan putus, maka kedua *node* akan saling menukarkan informasi tentang jumlah replikasi dan jumlah pesan yang dibuang, dan kemudian menghitung CV yang baru dan mengamatinya dengan CV yang lama untuk memperbaharui nilai limit. Jika CV yang baru lebih kecil dari nilai CV sebelumnya, maka jumlah pesan yang dikirimkan oleh *node* dalam suatu koneksi akan dinaikkan sesuai dengan nilai *additive increase* (AI) yang ditentukan. Sebaliknya, jika CV yang baru lebih besar dari nilai CV sebelumnya, maka jumlah pesan yang dikirimkan oleh *node* dalam suatu koneksi akan diturunkan sesuai dengan nilai *multiplicative decrease* (MD) yang ditentukan.

Akan tetapi, menurut pengamatan penulis, RR memiliki suatu kelemahan. Nilai CV yang di *update* secara terus menerus mengakibatkan CV tersebut berfluktuasi secara tajam dan cepat[2]. Sehingga, hal ini akan

mengakibatkan kinerja jaringan yang kurang optimal dalam mengatasi kongesti. Karena itu, diperlukan suatu perbaikan untuk menekan fluktuasi yang terjadi pada CV dengan menerapkan mekanisme pembelajaran mesin yang dapat menguatkan pengetahuannya, sehingga setiap *node* mampu untuk mengambil keputusan yang tepat terkait dengan naik/turunnya CV.

2.3.*Reinforcement Learning: Q-Learning*

Reinforcement Learning merupakan sebuah teknik dari *machine learning* yang pembelajarannya dilakukan dengan cara menemukan *action* yang dapat memaksimalkan *reward* yang didapatkan. Pada pembelajaran ini, *action* yang harus diambil tidak langsung diketahui dari awal, melainkan dengan mencoba *actions* yang ada dan menemukan *action* mana yang memberikan *reward* paling baik[7].



Gambar 2. 3. Model dari *Reinforcement Learning*

Model dari *Reinforcement Learning* dapat dilihat pada Gambar 2.3. *Agent* akan melakukan *actions* yang akan memberikan dampak pada *state* dan juga lingkungan (*environment)* miliknya, kemudian akan menerima *reward* yang mengindikasikan kualitas dari *action* yang dilakukan dan peralihan *state.* Jika *action* yang dilakukan memberikan dampak yang baik, maka akan

diberikan *reward* positif. Namun, jika *action* yang dilakukan memberikan dampak yang tidak baik, maka akan diberikan *reward* negatif.

Salah satu model *Reinforcement Learning* yang populer digunakan adalah algoritma *Q-Learning*. Pada *Q-Learning*, suatu tindakan diterapkan pada suatu keadaan lingkungan berdasarkan nilai Q (*Q-Values*) yang dihitung berdasarkan persamaan berikut:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$

Rumus 2. 2. *Q-Learning* [7]

Q-Values setiap pasangan *state-action* disimpan di dalam *Q-Table* dan akan digunakan serta diperbaharui pada setiap kali pengambilan keputusan.

Pada penerapannya dalam menangani kongesti, lingkungan yang diamati untuk menentukan *state* saat ini adalah CV. Tindakan yang akan diterapkan nanti diharapkan mampu mengurangi fluktuasi pada CV dan mampu membuat kinerja jaringan dalam mengurangi beban dan kongesti menjadi lebih optimal. Akan tetapi, pada kebanyakan algoritma *learning*, penerapannya dilakukan secara *independent* tanpa adanya kerja sama antara setiap *agent*. Padahal, kongesti merupakan masalah bersama yang dihadapi oleh setiap *node* yang ada di dalam jaringan. Karena itu, penerapan *learning* yang dilakukan secara kooperatif diperlukan dalam menangani kongesti guna meningkatkan hasil yang lebih optimal lagi.

2.4.*Cooperative Q-Learning*

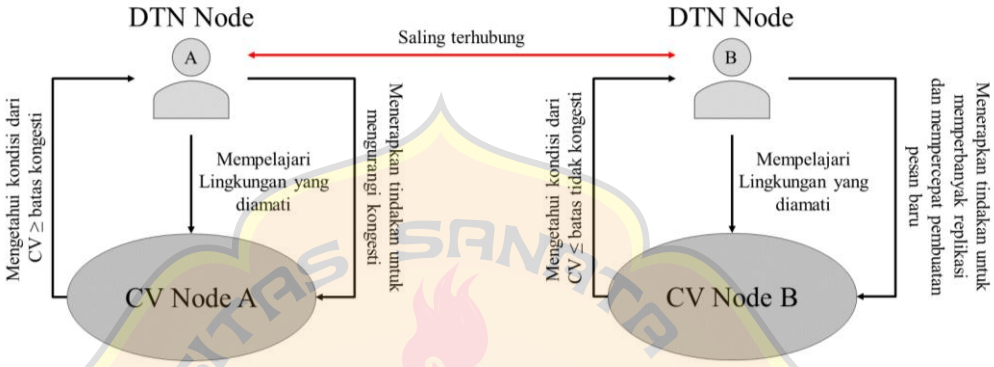
Cooperative Agents Reinforcement Learning merupakan sebuah mekanisme pembelajaran penguatan yang setiap *agent*-nya saling bekerja sama. Pada kebanyakan mekanisme *learning*, setiap *agents* belajar secara mandiri tanpa adanya kerja sama antara satu dengan yang lain. Padahal, manusia yang cerdas tidak hanya belajar dari *trial and error*, tetapi juga melalui kerjasama dengan sesamanya, seperti berbagi informasi, pengalaman dan ilmu yang dipelajari, serta bekerja sama dalam mengerjakan suatu tugas[4].

Gambar 2. 4. Contoh ilustrasi pekerjaan tanpa adanya kerja sama

Bayangkan suatu kondisi, dimana terdapat sebuah ruang kelas yang terdiri dari beberapa murid. Kelas yang bersih dan menyenangkan adalah suatu kondisi yang ingin dicapai. Namun, seperti yang terlihat pada Gambar 2.4, jika ada seorang murid yang berusaha untuk membersihkan kelas dan mengurangi sampah, tetapi masih ada murid yang membuang sampah di dalam kelas dan tidak menjaga kebersihannya, maka kondisi kelas yang bersih dan menyenangkan tidak dapat dicapai dengan optimal. Sebaliknya, jika semua murid berusaha menjaga kebersihan kelas bersama-sama, maka kondisi kelas yang diinginkan dapat dicapai dengan lebih optimal lagi.

Hal tersebut juga berlaku dalam penanganan kongesti. Kongesti merupakan suatu kondisi dimana jumlah replikasi pesan yang disebarkan di dalam jaringan, melebihi kapasitas jaringan itu sendiri. Karena itu, kongesti merupakan sebuah permasalahan bersama yang dihadapi oleh semua *node* di dalam jaringan. Sehingga, untuk mengatasi kongesti yang terjadi pada jaringan secara optimal, tidak bisa dilakukan secara mandiri oleh setiap *node*. Melainkan diperlukan suatu kerja sama sehingga kondisi yang diinginkan dapat terwujud dengan baik.

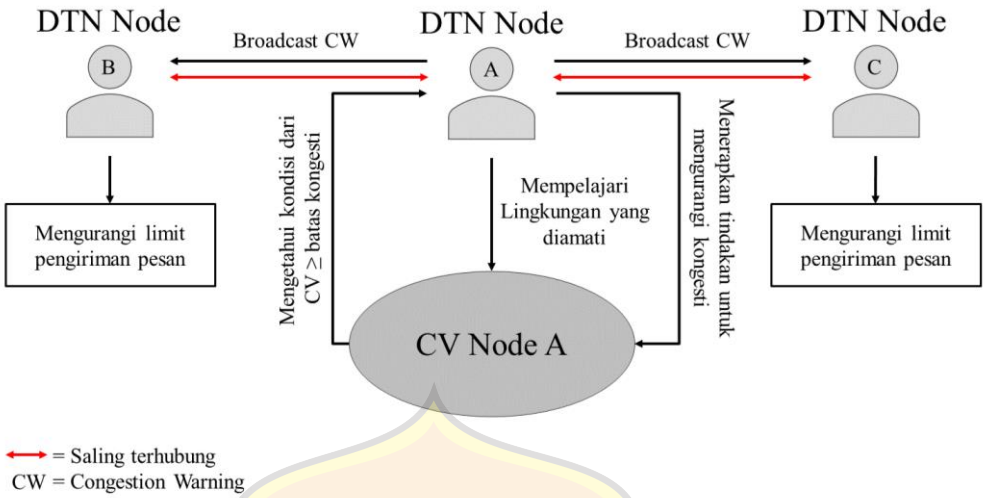
Pada penerapan algoritma *Q-Learning* untuk memperbaiki nilai CV pada RR[2] guna meningkatkan kinerja jaringan dalam mengurangi *overhead* untuk mencegah kongesti, informasi yang ada pada CV, telah saling menukarkan informasi jumlah pesan yang dibuang dan jumlah pesan yang direplikasi dengan suatu *peer* yang akan putus koneksinya.



Gambar 2. 5. Skema IQLCC yang tidak memiliki kerja sama

Namun, tanpa adanya suatu tindakan bersama untuk mencegah kongesti dalam jaringan, setiap *node* hanya bertindak menurut pengamatannya sendiri atau belajar secara *independent* (*Independent Q-Learning Congestion Control*, IQLCC). Sehingga, kurang adanya sinergi yang baik antara setiap *node* yang akan mengakibatkan hasil yang kurang optimal.

Seperti yang dapat kita lihat pada Gambar 2.5, *node* A telah mempelajari lingkungan yang diamatinya dan mengetahui bahwa nilai CV miliknya melebihi batas kongesti (*Congestion Threshold*). Sehingga, *node* A menyimpulkan bahwa lingkungan saat ini sedang berada di *state "congested"* dan menerapkan tindakan untuk mengurangi kongesti di jaringan, misalkan dengan cara mengurangi limit pengiriman pesan atau mengurangi pembuatan pesan baru. Akan tetapi jika pada *node* yang lain, yaitu *node* B yang mengamati bahwa kongesti tidak terjadi dan menerapkan tindakan untuk menambah limit pengiriman pesan atau mempercepat pembuatan pesan baru, maka tindakan yang diterapkan oleh *node* A untuk mengurangi kongesti menjadi kurang efektif.



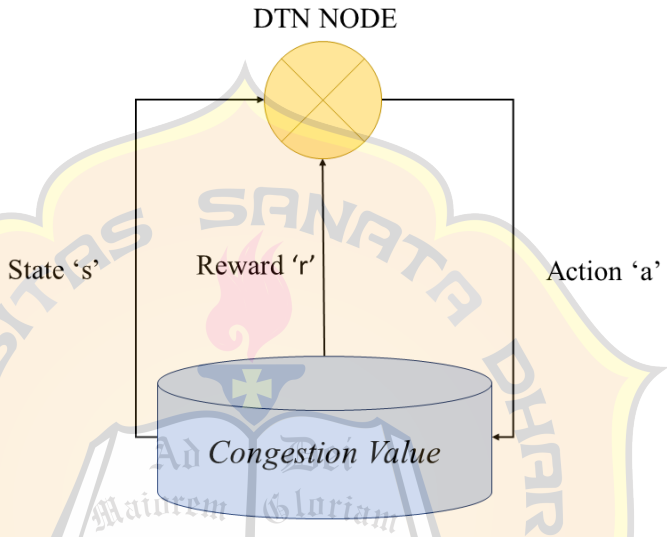
Gambar 2. 6. Skema CQLCC yang menerapkan kerja sama antara setiap *node*.

Pada penelitian ini, penulis menerapkan suatu bentuk kerja sama antara setiap *node* yang ada pada jaringan untuk mengatasi kongesti (*Cooperative Q-Learning Congestion Control*, CQLCC). Sehingga saat suatu *node* mengetahui bahwa kongesti terjadi, maka *node* tersebut akan memberitahukan kepada *node* lain di sekitarnya untuk mengambil tindakan pencegahan bersama. Seperti pada Gambar 2.6, saat *node* A mengetahui bahwa kongesti telah terjadi, maka *node* A memberitahukan peringatan kongesti kepada *node* yang berada di sekitarnya, yaitu *node* B dan C. Kemudian, *node* B dan C pun akan bekerja sama untuk mengurangi kongesti dengan mengurangi limit replikasi. Dengan diterapkannya kerja sama antara setiap *node*, maka diharapkan penerapan *Q-Learning* dalam mengurangi *overhead* guna mencegah kongesti pada jaringan dapat menjadi lebih optimal.

BAB III
DESAIN ALGORITMA

3.1.Implementasi Penanganan Kongesti Dengan *Q-Learning*

Dalam penerapannya untuk mengontrol *congestion* pada Jaringan Oportunistik, pemodelan dari *Q-Learning* terlihat pada Gambar 3.1.



Gambar 3. 1. Model *Reinforcement Learning: Q-Learning* dalam menangani kongesti di Jaringan Oportunistik.

State yang ada pada *Q-Learning* akan merepresentasikan keadaan dari CV pada sebuah *node*. Setiap *node* akan menyimpan pandangannya sendiri tentang *state* miliknya di dalam sebuah *Q-table* dan di dalam *Q-table* tersebut terdiri semua *state* yang ada dan semua *action* yang dapat diambil pada *state* tertentu; beserta nilai dari Q(s,a) untuk setiap pasangan *state* dan *action* yang didefinisikan berdasarkan persamaan berikut:

$$Q_n^*(s,a) = (1-\eta_n)Q_{n-1}^*(s,a) + \eta_n(r + \gamma V_{n-1}^*(s'))$$

Dimana

$$\eta_n = \frac{1}{1+visits_n(s,a)}$$

Dan

$$V_{n-1}^*(s') = max_{a'}[Q_{n-1}^*(s',a')]$$

Rumus 3. 1. *Q-Learning* dalam mengendalikan kongesti[3]

Dengan keterangan :

$Q_n^*(s,a)$ = Nilai *Q-Value* yang baru untuk pasangan *state-action*

$V_{n-1}^*(s')$ = nilai *reward* maksimum yang diharapkan yang diberikan oleh *state* baru dan semua *action* yang mungkin pada *state* baru

η_n = *learning rate*

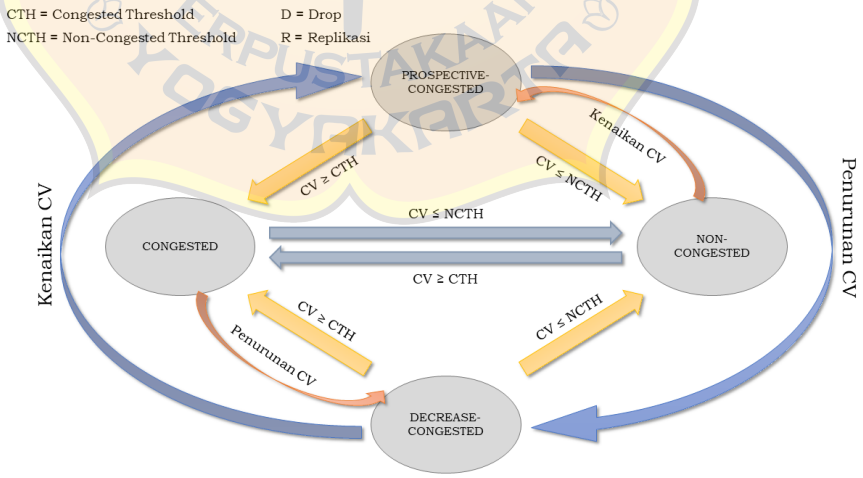
r = *reward* untuk *action* yang dilakukan

γ = *discount rate*

$visits_n(s,a)$ = Total jumlah berapa banyak pasangan *state-action* telah dikunjungi beserta iterasi ke-n.

Pada RR, CV di-*update* setiap putusnya suatu koneksi. Namun, jika banyak *node* yang koneksinya putus dalam waktu berdekatan, maka hal ini akan mengakibatkan CV tersebut sangat cepat berfluktuasi. Selain itu, karena *Q-Learning* membutuhkan waktu untuk mengamati kinerja dari suatu *action*, maka pada penelitian ini, CV tidak di-*update* setiap putusnya koneksi, melainkan jumlah pesan yang dibuang dan direplikasi dari *peer* yang lain akan dicatat setiap koneksi akan terputus, dan kemudian CV akan di-*update* dalam suatu interval waktu.

3.1.1. *State Machine*



Gambar 3. 2. Model transisi setiap *state*

Seperti yang terlihat pada Gambar 3.1, setiap *node* yang ada pada jaringan akan mengontrol kongesti dapat berada pada *states* berikut:

Congested, *Prospective-congested*, *Decrease-congested* dan *Non-congested*. Setiap *node* memperkirakan naik-turunnya suatu CV dengan menghitung nilai rasio CV tersebut menggunakan persamaan EWMA. Transisi *state* suatu *node* dapat berbeda-beda berdasarkan naik-turunnya CV yang dialami *node* tersebut. Dibawah ini merupakan transisi *state* yang dapat terjadi pada suatu *node* dalam mengontrol kongesti:

- ***Congested → Non-Congested***
- ***Congested → Decrease-Congested***
- ***Decrease-Congested → Congested***
- ***Decrease-Congested → Prospective-Congested***
- ***Decrease-Congested → Non-Congested***
- ***Non-Congested → Prospective-Congested***
- ***Non-Congested → Congested***
- ***Prospective-Congested → Non-Congested***
- ***Prospective-Congested → Congested***
- ***Prospective-Congested → Decrease-Congested***

Transisi untuk setiap *state* yang ada pada kendali kongesti ini didefinisikan sebagai berikut:

- ***Congested* (C)** adalah *state* saat CV mencapai nilai lebih dari atau sama dengan suatu nilai batasan (*threshold*).
- ***Prospective-Congested* (PC)** adalah *state* saat terjadinya kenaikan CV berdasarkan perhitungan persamaan EWMA, namun belum mencapai kondisi *Congested*.
- ***Decrease-Congested* (DC)** adalah *state* saat terjadinya penurunan CV berdasarkan perhitungan persamaan EWMA, namun belum mencapai kondisi *Non-Congested*.
- ***Non-Congested* (NC)** adalah *state* saat CV-nya berada dibawah atau sama dengan nilai *threshold*.

3.1.2. *Action*

Untuk setiap *state*, terdapat *actions* yang dapat diterapkan untuk meningkatkan kinerja jaringan. Pada Tabel 3.1, ditunjukan *action* yang saat ini diterapkan dalam suatu *state* tertentu.

Tabel 3. 1. *Actions* yang dapat diambil oleh setiap *state*

Actions	***State***			
	C	**PC**	**NC**	**DC**
A0: Drop based on highest rate	✓	✓		
A1: Drop based on highest replications	✓	✓		
A2: Drop based on oldest TTL	✓	✓	✓	✓
A3: Drop based on oldest received time	✓	✓	✓	✓
A4: Increase message generation rate	✓	✓		
A5: Decrease message generation rate			✓	✓
A6: Decreasing number of replications	✓	✓		
A7: Increasing number of replications			✓	✓

- ***Drop based on highest rate***: *node* membuang pesan-pesan yang ada berdasarkan *rate* dari pesan-pesan tersebut, yang dihitung dengan persamaan berikut[8]:

$$Rate = \frac{K_m}{TTL_{init} - TTL}$$

Rumus 3. 2. *Messages Rate*

Dengan keterangan:

K_m : total *hops* sebuah pesan

TTL_{init} : TTL awal pesan

TTL : Sisa TTL pesan saat ini

- ***Drop based on highest replications***: *node* membuang pesan-pesan yang dimilikinya berdasarkan jumlah suatu pesan tereplikasi. Ketika *node i* yang membawa pesan *m* bertemu

dengan *node j* yang tidak membawa pesan *m*, jumlah replikasi pesan *m* dihitung berdasarkan dua kasus berikut[8]:

1) Kasus pertama adalah *node j* dipilih sebagai *node* yang akan membawa pesan *m*. Jika *node j* tidak memiliki informasi apapun tentang pesan *m*, jumlah replikasi dari pesan *m* diatur menjadi $R_m^i + 1$ pada kedua *node*. Sebaliknya, jika *node j* memiliki informasi tentang pesan *m*, maka kedua *node* akan bertukar *summary vector* dan mengatur nilai dari pesan *m* menjadi $\max(R_m^i, R_m^j) + 1$.
2) Kasus kedua adalah jika *node j* tidak dipilih *relay node* untuk pesan *m*. Jika *node j* tidak memiliki informasi apapun tentang pesan *m*, jumlah pesan *m* diatur menjadi R_m^i pada kedua *node*. Sebaliknya, jika *node j* memiliki informasi tentang pesan *m*, maka maka kedua *node* akan bertukar *summary vector* dan mengatur nilai dari pesan *m* menjadi $\max(R_m^i, R_m^j)$.

- ***Drop based on oldest TTL***: *node* membuang pesan-pesan yang dimilikinya berdasarkan *Time to Live* (TTL) yang paling kecil, yaitu pesan yang umurnya paling tua.
- ***Drop based on oldest received time***: *node* membuang pesan-pesan yang dimilikinya berdasarkan waktu pesan yang paling lama diterima olehnya.
- ***Increase message generation rate***: interval waktu pembuatan pesan ditambah dengan cara mengalikannya dengan sebuah bilangan konstan K. Pada penelitian ini, peneliti menggunakan K=2, sehingga banyak pesan yang dibuat berkurang hingga setengah.

- ***Decrease message generation rate***: interval waktu pembuatan pesan dikurangi dengan cara membaginya dengan sebuah bilangan konstan K, yang pada penelitian ini digunakan K=2.
- ***Decreasing number of replications***: mengurangi batasan (*limit*) dari jumlah pesan yang dapat dikirimlah oleh *node* dalam suatu koneksi, dengan cara mengkalikan *limit* tersebut dengan suatu nilai MD, sesuai dengan cara algoritma RR dalam mengurang *limit*-nya. Pada penelitian digunakan nilai MD=0.2.
- ***Increasing number of replications***: menambah batasan (*limit*) dari jumlah pesan yang dapat dikirimlah oleh *node* dalam suatu koneksi, dengan cara menjumlahkan *limit* tersebut dengan suatu nilai AI, sesuai dengan cara algoritma RR dalam menambah *limit*-nya. Pada penelitian digunakan nilai AI=1.

3.1.3. *Reward*

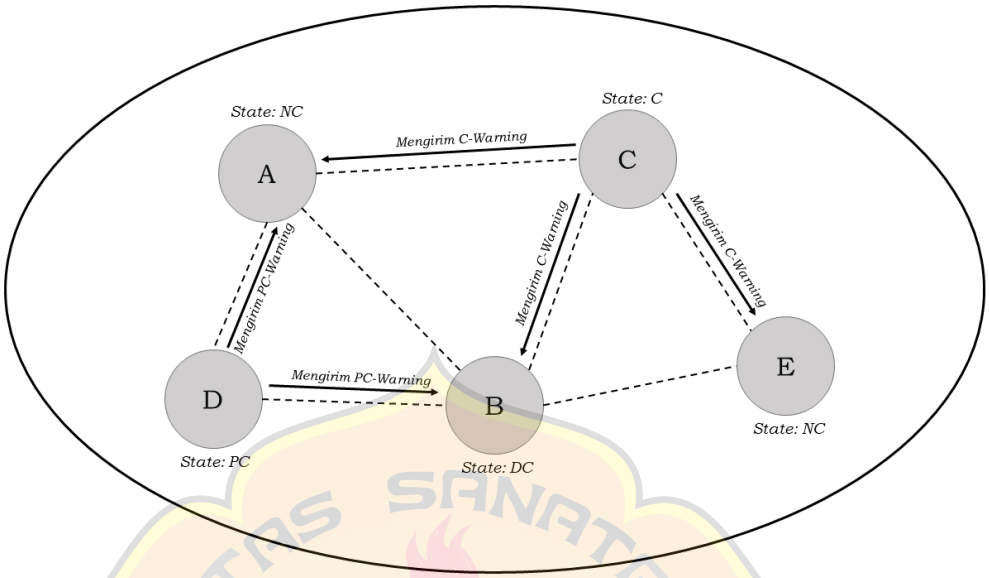
Pada *reinforcement learning*, tujuan yang ingin dicapai adalah memaksimalkan total *reward* yang diterima setiap waktu. Pada *Q-Learning* nilai *reward* yang diterima mengindikasikan evaluasi terhadap suatu *action* yang diterapkan. Jika *action* tersebut mengubah kondisi *environment* ke *state* yang baik, maka *reward* yang diterima akan bernilai positif. Sebaliknya, jika *action* tersebut mengubah kondisi *environment* ke *state* yang tidak baik, maka *reward* yang diterima akan bernilai negatif.

Pada penelitian ini, *reward function* yang diterapkan dapat dilihat pada Tabel 3.2. *Reward function* yang diterapkan berdasarkan uji coba penulis untuk mendapatkan hasil yang terbaik.

Tabel 3. 2. *Reward*

Transition from state S	*To state* ***S′***			
	C	**NC**	**PC**	**DC**
C	-2	+2	-	+1
NC	-2	+2	-1	-
PC	-2	+2	-1	+0.5
DC	-2	+2	-1	-0.5

3.2.Implementasi Mekanisme yang Kooperatif Pada *Q-Learning*



Gambar 3. 3. Skema algoritma *Q-Learning* dengan menerapkan kerja sama antara setiap *node*.

Kerja sama antara setiap *node* dalam mencegah terjadinya kongesti sangat dibutuhkan. Pada penelitian ini, kerja sama antara *node* diwujudkan dengan cara:

1) Saat suatu *node* berada pada *state "Congested*", maka *node* tersebut wajib untuk memperingatkan *node* lain di sekitarnya tentang keadaannya saat ini. *Node* yang menerima peringatan tersebut wajib untuk menurunkan limit jumlah replikasi dengan mengalikan limit tersebut dengan nilai 0.05.
2) Saat suatu *node* berada pada *state "Propective-Congested*", maka *node* tersebut wajib untuk memperingatkan *node* lain di sekitarnya tentang keadaannya saat ini. *Node* yang menerima peringatan tersebut wajib untuk menurunkan limit jumlah replikasi dengan mengalikan limit tersebut dengan nilai 0.1.

Pada Gambar 3.3 terlihat bahwa saat *node* C mengalami *state "congested"* maka *node* C akan mengirimkan peringatan kongesti (*Congested Warning*) ke *node* disekitarnya, dan *node* D yang sedang berada di *state "prospective-congested"* juga mengirimkan peringatan peningkatan kongesti

(*Prospective-Congested Warning*) ke *node* sekitarnya. *Node* lain yang menerima peringatan ini akan melakukan suatu tindakan penurunan limit pengiriman sesuai dengan peringatan yang diterima olehnya.

3.3. Pseudocode

Beberapa rancangan pseudocode yang digunakan pada penelitian ini adalah:

Process Event

Require: *nrofDrops* = 0
Require: *nrofReps* = 0
Require: *otherNrofDrops* = 0
Require: *otherNrofReps* = 0
Require: *CV* = 0;
Require: *stateLastTimeUpdate* = 0;
Require: *stateUpdateInterval*;

if *event* = Drop A Message Because Buffer is Full **then**
    nrofDrops++;
if *event* = A Message Received **then**
    nrofReps++;
if *event* = Contact With Peer (When Connection Down) **then**
    otherNrofDrops = *otherNrofDrops* + *peer.nrofDrops*;
    otherNrofReps = *otherNrofReps* + *peer.nrofReps*;
if *event* = Update **then**
    if (*stateLastTimeUpdate* = 0 || (Current Time − *stateLastTimeUpdate*) >= *stateUpdateInterval*) **then**
        drop = *nrofDrops* + *otherNrofDrops*;
        reps = *nrofReps* + *otherNrofReps* + $\sum_{m \in stored\ messages\ m}(\boldsymbol{hops}(\boldsymbol{m}) - \mathbf{1})$;
        Reset (*nrofDrops, nrofReps, otherNrofDrops, otherNrofReps*)

13: *CV'* = $\boldsymbol{\alpha}$ * (*drop/reps*) + (1 − $\boldsymbol{\alpha}$) * *CV*;

14: Do the *Q-Learning* based on new CV' and old CV

15: *CV = CV';*

16: *stateLastTimeUpdate = CurrentTime*

Cooperative Q-Learning Algorithm

Require: Initialize the Q-table entry $\boldsymbol{Q(s,a)}$ to zero for each *state* and *action*

Input: *CV, CV'*

The procedure:

1. Initialize the $\boldsymbol{s}$
2. **Repeat** (for each step)
3. Choose action $\boldsymbol{a}$ from state $\boldsymbol{s}$ depends on the action restriction policy
4. Apply the action $\boldsymbol{a}$ and observe $\boldsymbol{s'}$ and reward $\boldsymbol{r}$
5. Update Q-table for the $\boldsymbol{s, a}$

➔ $\boldsymbol{Q_n^*(s,a) = (1-\eta_n)Q_{n-1}^*(s,a) + \eta_n(r + \gamma V_{n-1}^*(s'))}$

6. **if** ($\boldsymbol{s'}$ = *Congested*) **then**
7. Broadcast *CongestionWarning* to neighbors
8. **else if** ($\boldsymbol{s'}$ = *Prospective-Congested*) **then**
9. Broadcast *ProspectiveCongestionWarning* to neighbors
10. $\boldsymbol{s' \rightarrow s}$
11. **Until** the end of simulation

Broadcast Congestion Warning Algorithm

Require: *id* ← *message id* (*Congested Prefix + node id*)

Require: *ttl* ← 5 mins

Require: *host_id*

1. Create *CW*(*id_host*, *id*)

2. Set *CWttl* = *ttl*
3. Broadcast CW to connected neighbors

Broadcast Prospective-Congestion Warning

Require: *id* ← *message id* (*Prospective-Congested Prefix* + *node id*)

Require: *ttl* ← 5 mins

Require: *host_id*

1. Create *PCW*(*id_host*, *id*)
2. Set *PCWttl* = *ttl*
3. Broadcast *P*CW to connected neighbors

***Warning Detected* - Receiver Node**

Require: *limit* → limit of sending message for a connection

1. **if** (receive CW) **then**
2. *limit* = *limit* * 0.05
3. **else if** (receive PCW) **then**
4. *limit* = *limit* * 0.1

Delivery Limitation Process Event

Require: *ConLimit(Connection, limit)* → Map of delivery limitation for each connection

Require: *limit* → limit value that increased/decreased by AIMD mechanism

Input: *con* → A connection which connected with this node

1. **if** *event* = Connection Up **then**
2. Add *con* and *limit* to *ConLimit*
3. **if** *event* = Start Transfer to A Connection **then**
4. **if** *ConLimit* contains *con* **then**

```
5.          Try to transfer message to con
6.          if Transfer Started then
7.              remainingLimit = get the limit of con from ConLimit
8.              remainingLimit = remainingLimit − 1
9.              if remainingLimit ≠ 0 then
10.                 Set limit for con in ConLimit as remainingLimit
11.             else
12.                 Remove con and its limit from ConLimit
13. if event = Connection Down then
14.     Remove con and its limit from ConLimit
```

BAB IV
PENGUJIAN DAN ANALISIS

4.1 Skenario Simulasi

Skenario simulasi ini dilakukan dengan menggunakan protokol *Epidemic Routing*[9], yang mekanisme pengirimannya yaitu dengan mengirimkan replikasi pesan ke *node* manapun yang ditemui. Selain itu, juga menggunakan dua macam pergerakan, yaitu:

- *Haggle-3*[6] yang merupakan pergerakan manusia didalam sebuah ruangan.
- *Random Waypoint* yang merupakan pergerakan yang dilakukan secara acak.

Dua pergerakan tersebut digunakan untuk mengamati bagaimana kinerja *Congestion Control* ini pada lingkungan yang memiliki *populer node-unpopuler node* dan lingkungan yang acak sehingga *node-node* yang ada pada lingkungan tersebut memiliki kemungkinan yang sama dan pandangan tentang lingkungan yang kurang lebih sama.

4.2 Parameter Simulasi

Parameter yang digunakan, adalah sebagai berikut :

Tabel 4. 1. Parameter Simulasi

Parameter	**Skenario 1**	**Skenario 2**
Waktu Simulasi	274883 detik	274883 detik
Protokol *Routing*	*Epidemic Routing*	*Epidemic Routing*
Model Pergerakan *Node*	*Haggle-3*	*RandomWaypoint*
Transmit Range	150	150
Transmit Speed	250K	250K
TTL *(Time To Live)* Pesan	1440 menit	60 menit
Jumlah *Node*	41	41
CQLCC.CongestionTH	0.1	0.5
CQLCC.NonCongestionTH	0.00001	0.1
CQLCC.ai	1	1

CQLCC.md	0.2	0.2
CQLCC.K	2	2
CQLCC.StateUpdateInterval	1200 detik	1200 detik
CQLCC.Alpha	0.9	0.9

4.3 Matriks Unjuk Kerja

Terdapat beberapa matriks unjuk kerja untuk membuktikan kinerja dari algoritma *Double Q-Learning* dalam mengatasi *congestion*, diantaranya sebagai berikut:

1. ***Delivery Ratio***

   Delivery Ratio adalah rasio rata-rata antara jumlah dari pesan yang sampai di node-node tujuan dan jumlah pesan yang dibuat dan ditujukan kepada node-node tersebut.

$$delivery\ ratio = \frac{number\ of\ received\ messages}{number\ of\ created\ messages} \times 100\%$$

Rumus 4. 1. *Delivery Ratio*[3]

2. ***End-to-end Latency***

   End-to-end Latency adalah rata-rata interval waktu untuk mengirimkan pesan ke tujuannya.

$$end - to - end\ latency = \frac{\sum_{i=1}^{number\ of\ messages\ received} (t_i - t_c)}{number\ of\ messages\ received}$$

Rumus 4. 2. *End-to-end Latency*[3]

Dengan keterangan :

t_i = Waktu saat pesan *i* sampai ditujuannya

t_c = Waktu pesan ditransmisikan pada tempat awalnya

3. *Goodput*

Goodput adalah jumlah dari pesan yang diterima dibagi dengan jumlah dari pesan yang ditransferkan (termasuk transfer-transfer yang tidak berhasil.).

$$goodput = \frac{number\ of\ received\ messages}{number\ of\ relayed\ messages} \times 100\%$$

Rumus 4. 3. *Goodput*[3]

4. *Overhead Ratio*

Overhead merupakan beban atau *cost* (biaya) yang berlebihan, yang digunakan saat mengirim pesan. *Overhead* didefinisikan sebagai jumlah pesan yang disebarkan dikurangi dengan jumlah pesan yang terkirim, kemudian dibagi dengan jumlah pesan yang terkirim.

$$overhead = \frac{number\ of\ relayed\ messages - number\ of\ delivered\ messages}{number\ of\ delivered\ messages} \times 100\%$$

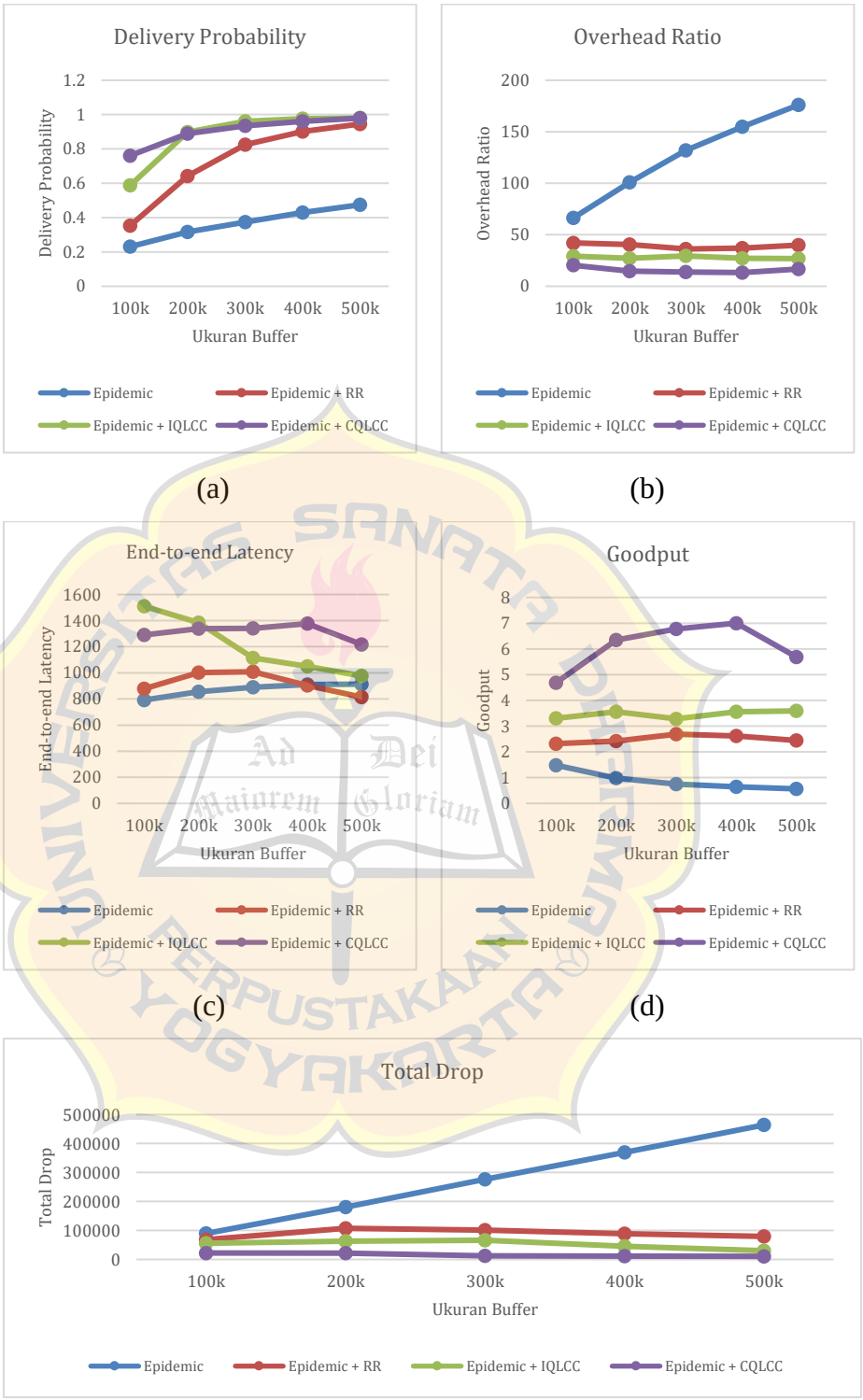
Rumus 4. 4. *Overhead Ratio*

4.4 Analisis Hasil Pengujian

Untuk melakukan evaluasi terhadap unjuk kerja CQLCC dalam menangani kongesti, maka dilakukan simulasi dan pengujian dengan menggunakan rancangan skenario yang telah dijelaskan diatas. Data didapatkan dari *report* yang dihasilkan saat simulasi berlangsung dan kemudian menjadi bahan untuk dilakukan analisis.

4.4.1 Hasil Pengujian Dengan Menggunakan Pergerakan *Random Waypoint*

Unjuk kerja dari algoritma CQLCC dalam mengendalikan kongesti di pergerakan *Random Waypoint* dapat dilihat pada gambar di bawah ini:

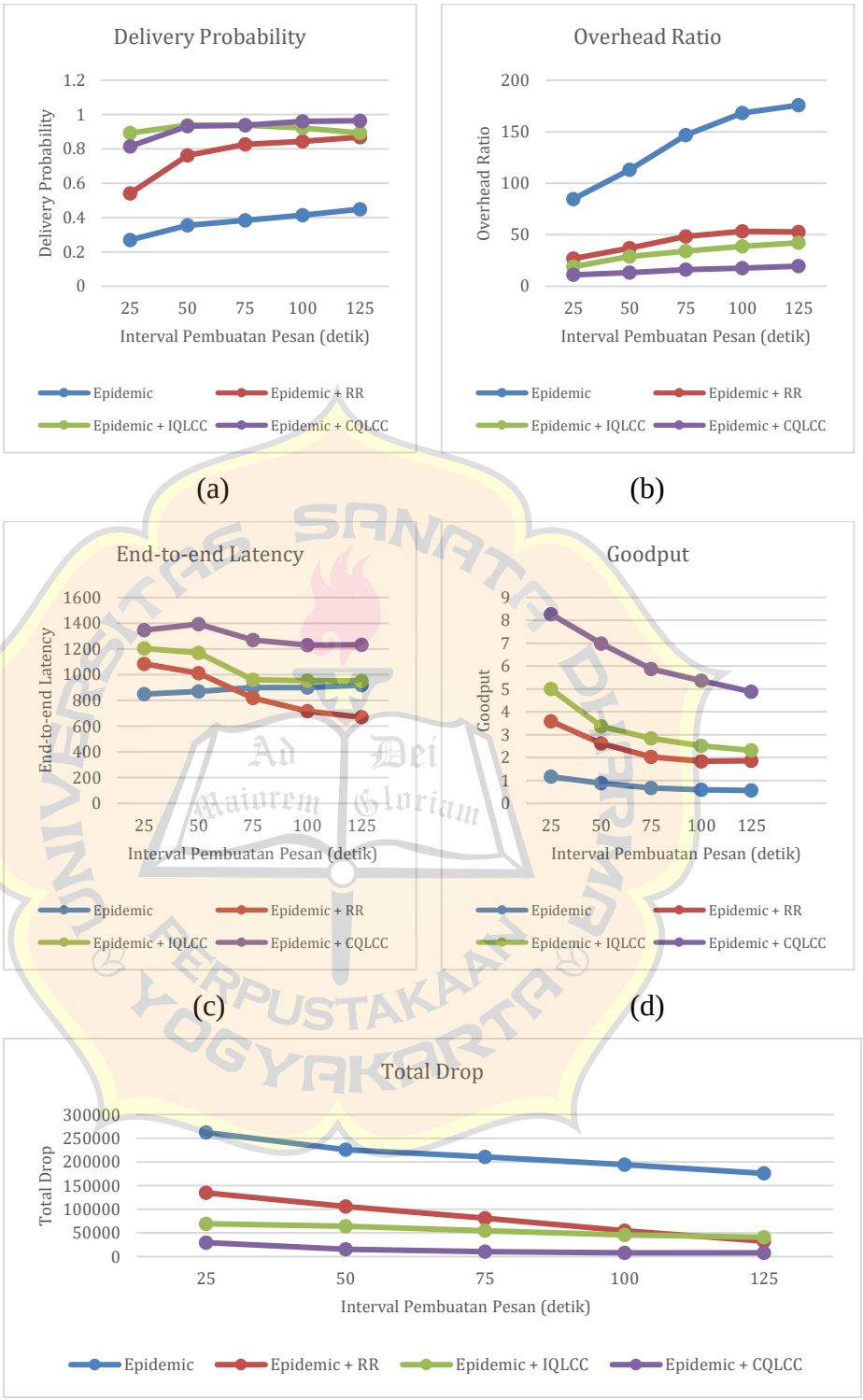


Gambar 4. 1. Grafik *delivery probability*, *overhead*, *end-to-end latency*, *goodput* dan *total drop* pada pergerakan *Random Waypoint* berdasarkan kenaikan *buffer*.

Terlihat pada Gambar 4.1(a), semakin besarnya ukuran suatu *buffer*, maka nilai *delivery probability* pun juga semakin tinggi. Namun, dikarenakan probabilitas kontak antara setiap *node* di pergerakan *Random Waypoint* cenderung sama, maka pandangan dari setiap *node* terhadap kondisi jaringan juga semakin sama antara satu dengan yang lainnya. Karena itu, meskipun IQLCC hanya menukarkan informasi CV dengan *node* tetangga tanpa adanya suatu tindakan kerja sama, tetapi pada ukuran *buffer* yang semakin besar, IQLCC dapat lebih cepat menyamai hasil *delivery probability* CQLCC, seperti yang terlihat pada Gambar 4.1(a). Sedangkan, RR yang tidak memiliki sistem pembelajaran, membutuhkan *buffer* yang lebih besar agar bisa menyamai kinerja keduanya.

Dengan menerapkan CQLCC yang membuat setiap *node* mampu bekerja sama untuk mencapai kondisi jaringan yang optimal, kemungkinan terjadinya kongesti di jaringan semakin kecil. Hal ini dibuktikan dengan hasil *overhead*, *goodput* dan *total drop* pada Gambar 4.1(b)(d)(e) yang lebih baik dibandingkan dengan IQLCC dan RR. Dengan jumlah replikasi pesan yang lebih sedikit dibandingkan kedua pengendali kongesti yang lain, CQLCC mampu menghasilkan tingkat *delivery probability* yang lebih baik. Namun, seperti yang terlihat pada Gambar 4.1(c), CQLCC dan IQLCC memiliki *end-to-end latency* yang lebih besar dibandingkan yang lainnya. Hal ini dikarenakan ukuran *buffer* semakin besar, tetapi CQLCC dan IQLCC tetap membatasi penyebaran pesan yang ada.

Meskipun pada awalnya CQLCC memiliki *end-to-end latency* yang lebih baik dari pada IQLCC, tetapi semakin besarnya *buffer*, IQLCC cenderung memiliki *delay* yang lebih baik. Hal ini dikarenakan setiap kali ada *node* yang berada di kondisi *congested*, maka *node* lain langsung ikut menurunkan batas pesan yang dapat dikirimkan, berbeda dengan IQLCC yang hanya membatasi pengiriman di dirinya sendiri.



Gambar 4. 2. Grafik *delivery probability*, *overhead*, *end-to-end latency*, *goodput* dan *total drop* pada pergerakan *Random Waypoint* berdasarkan kenaikan interval pembuatan pesan.

Pada Gambar 4.2(a) terlihat bahwa semakin besarnya interval pembuatan pesan, maka *delivery probability* juga semakin tinggi. Hal tersebut karena semakin sedikit pesan yang dibuat, maka semakin banyak replikasi pesan yang dapat disimpan oleh *buffer* suatu *node*. Akan tetapi, pada awalnya CQLCC memiliki *delivery* probability yang lebih rendah dari IQLCC. Ini dikarenakan semakin banyak pesan yang dibuat dengan ukuran *buffer* setiap *node* yang tetap sama, maka semakin sedikit replikasi setiap pesan yang dapat disebarkan didalam jaringan akibat dari terbatasnya *buffer* milik *node*.

Seperti yang kita ketahui, bahwa pada Jaringan Oportunistik, pesan disebarkan dengan jumlah replikasi yang banyak untuk meningkatkan keberhasilan pengiriman. CQLCC cenderung lebih menekan penyebaran pesan di jaringan dibandingkan dengan IQLCC, sedangkan penyebaran replikasi pesan di jaringan pun sudah terbatasi akibat banyaknya pesan yang dibuat. Karenanya, IQLCC pada awalnya cenderung memiliki *delivery probability* yang lebih baik akibat performanya dalam menekan penyebaran pesan yang tidak sebesar CQLCC. Namun, disaat jumlah pesan yang dibuat semakin sedikit, *delivery probability* yang dihasilkan CQLCC semakin meningkat dan melebihi performa dari IQLCC maupun RR. Hal ini diakibatkan jumlah pesan yang semakin sedikit dibuat, maka jumlah replikasi pesan yang dapat disimpan oleh *buffer* setiap *node* semakin besar kemungkinannya. CQLCC yang selalu menekan penyebaran pesan seoptimal mungkin, mampu lebih mencegah terjadinya kongesti yang dapat menurunkan tingkat keberhasilan pengiriman. Karena itu, *delivery probability* CQLCC cenderung naik dan menjadi lebih baik dibandingkan dengan kedua kendali kongesti yang lain.

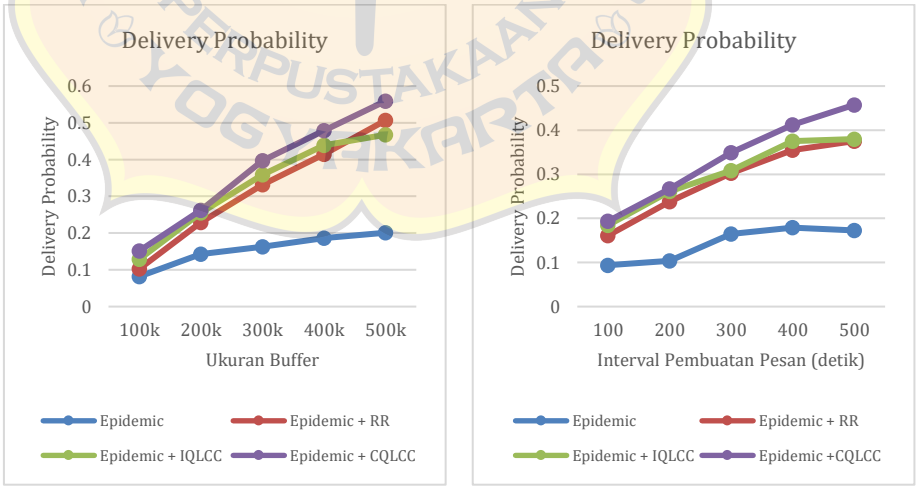
Selain itu, CQLCC juga mampu menghasilkan *total drop* yang lebih kecil dibandingkan yang lain, seperti yang diperlihatkan pada Gambar 4.2(e). Hal ini dapat terjadi karena tingginya pembatasan penyebaran pesan yang dilakukan oleh CQLCC, membuat jumlah

replikasi yang tidak berlebihan di jaringan dan juga kongesti yang jarang terjadi. Akibatnya, jumlah pesan yang dibuang pun juga semakin menurun.

Meskipun hasil *delivery probability* dan *total drop* seluruh algoritma kendali kongesti cenderung semakin bagus, tetapi hasil *overhead* dan *goodput* cenderung semakin tidak bagus setiap kali interval pembuatan pesan dinaikkan. Karena, semakin banyaknya kemungkinan jumlah replikasi pesan yang dapat disimpan oleh suatu *node*. Akan tetapi, seperti yang tampak pada Gambar 4.2(b)(d), CQLCC tetap menghasilkan nilai *overhead* dan *goodput* yang lebih baik dibandingkan yang lain akibat kerja sama antara setiap *node* dalam menurunkan penyebaran pesan setiap kali kongesti terjadi.

Berbeda dengan hasil CQLCC yang baik pada *delivery probability*, *overhead*, *goodput* dan *total drop*, pada *end-to-end latency* dipergerakan ini, CQLCC tetap memiliki *delay* yang lebih besar dibandingkan yang lainnya, karena penyebaran pesan yang lebih dibatasi.

4.4.2 Hasil Pengujian Dengan Menggunakan Pergerakan *Haggle*-3

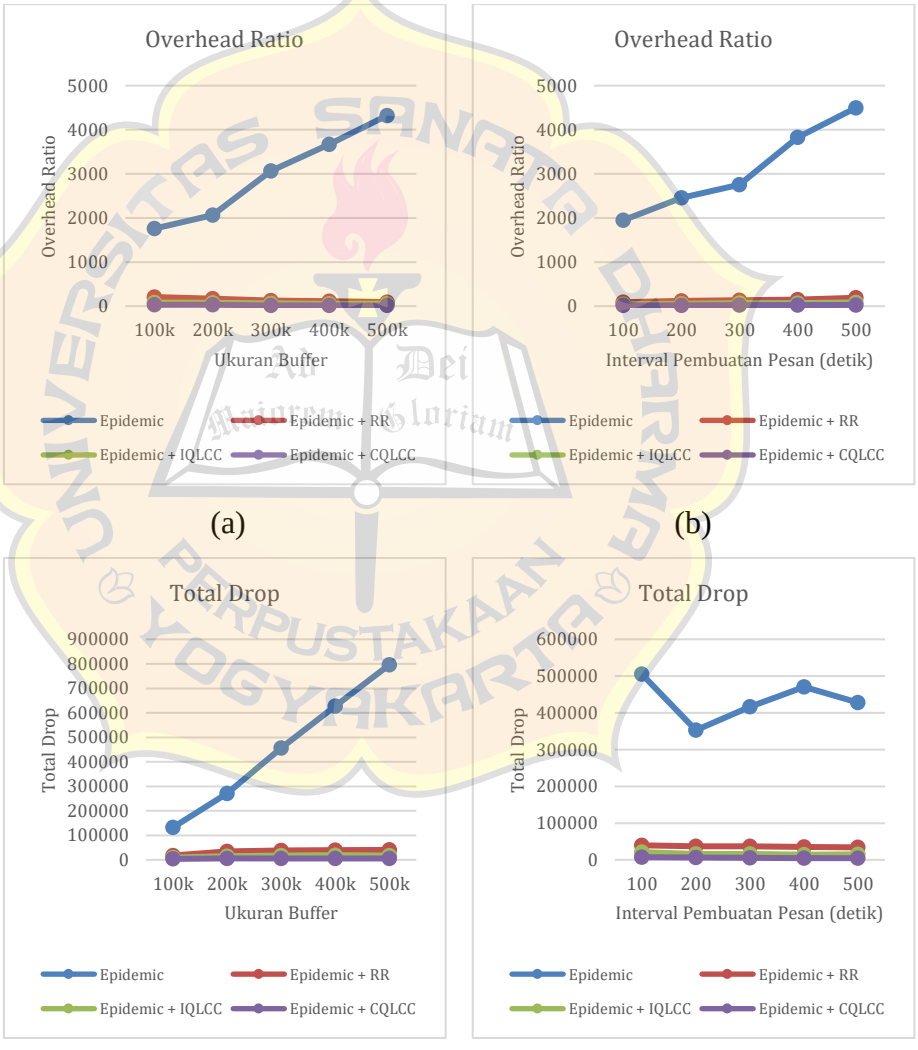


(a) (b)

Gambar 4. 3. Grafik *delivery Probability* pada pergerakan *Haggle-3*.

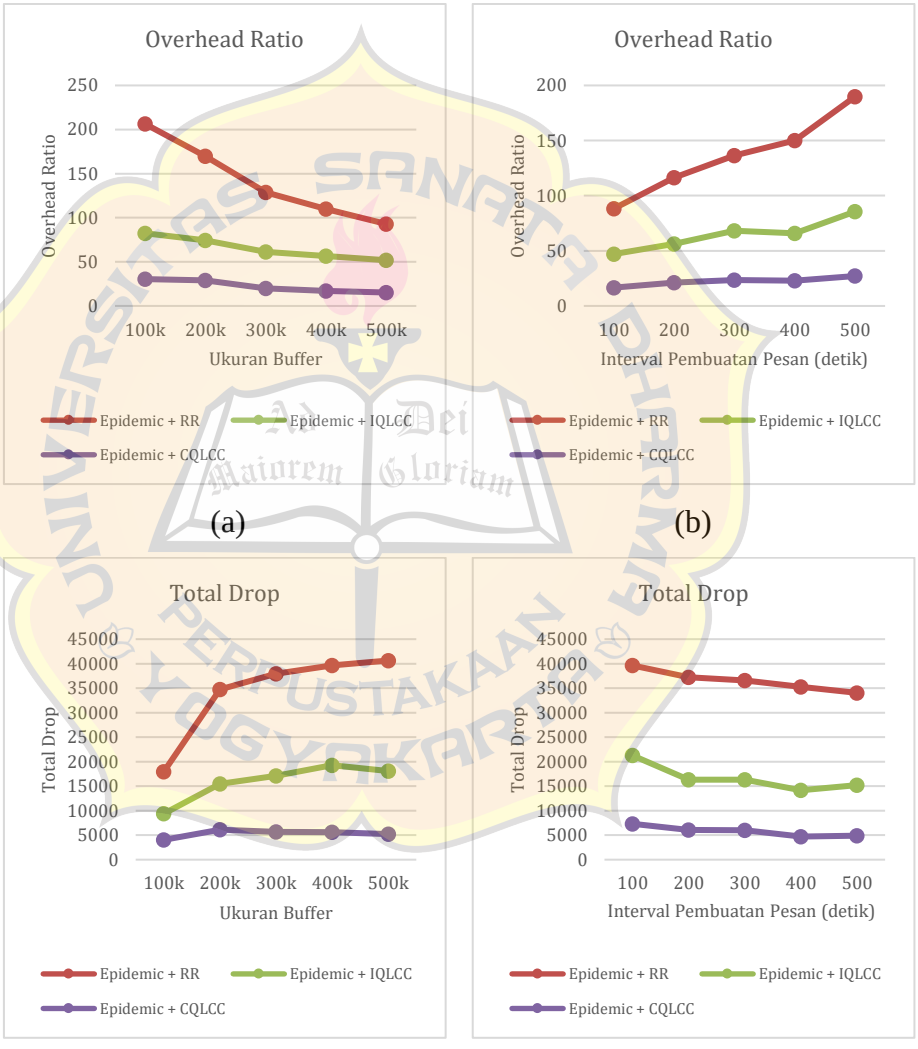
Pada pergerakan *Haggle-3* performa *delivery probability* yang dihasilkan lebih rendah dibandingkan dengan *Random Waypoint*. Akan

tetapi, *delivery probability* CQLCC cenderung selalu lebih baik dibandingkan yang lain, baik berdasarkan kenaikan *buffer* maupun interval pembuatan pesan. Karena pada pergerakan *Haggle-3*, probabilitas kontak antara setiap *node* berbeda-beda. Ada *node* yang sangat sering kontak dengan *node* yang lain dan ada juga *node* yang sangat jarang kontak dengan *node* yang lain. Akibatnya, ada *node* yang memiliki pengamatan yang sesuai dengan kondisi jaringan, dan ada juga *node* yang memiliki pengamatan yang tidak sesuai dengan kondisi jaringan.



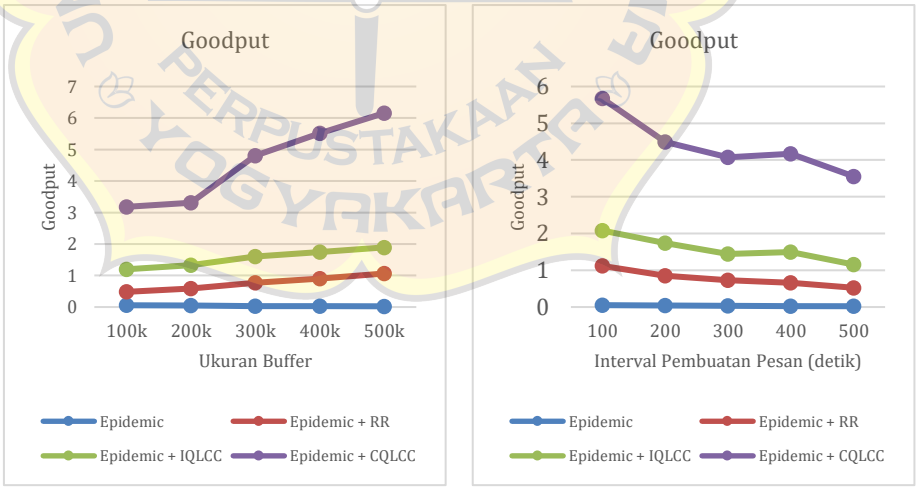
Gambar 4. 4. Grafik o*verhead* dan *total drop* pada pergerakan *Haggle-3.*

Pada *Random Waypoint* pengamatan setiap *node* tentang kondisi jaringan cenderung sama, sehingga kompak dalam mengambil suatu tindakan. Namun, pada *Haggle-3*, pengamatan *node* yang berbeda-beda tentang kondisi jaringan akan membuat setiap *node* cenderung mengambil tindakan yang berbeda-beda juga. Akibatnya, unjuk kerja dari CQLCC yang memiliki sistem kerja sama antara setiap *node*, terlihat sangat menonjol dibandingkan IQLCC dan RR pada pergerakan ini.



Gambar 4. 5. Grafik *overhead* dan *total drop* yang dihasilkan *Epidemic*+RR, *Epidemic*+IQLCC, *Epidemic*+CQLCC pada pergerakan *Haggle-3*.

Pada Gambar 4.4, *overhead* dan *total drop* pada *epidemic* dengan ketiga kendali kongesti terlihat jauh lebih baik dari pada *Epidemic* tanpa kendali kongesti. Namun, jika kita lihat lebih baik pada Gambar 4.5, CQLCC memiliki kinerja yang lebih unggul dan stabil dibandingkan dengan IQLCC dan RR. Terutama, pada hasil *overhead* yang terlihat sangat menonjol dibandingkan kedua algoritma kendali kongesti yang lain. Pada *Random Waypoint*, karena adanya probabilitas kontak yang sama antara setiap *node*, perbedaan hasil *overhead* (Gambar 4.1(b), 4.2(b)) yang dihasilkan oleh RR, IQLCC dan CQLCC tidak sebesar perbedaan pada pergerakan *Haggle-3*. Karena pandangan *node* yang berbeda-beda di pergerakan ini, RR dan IQLCC memiliki nilai *overhead* yang jauh lebih besar dibandingkan dengan *overhead* yang dihasilkan pada pergerakan *Random Waypoint*. Tetapi, karena adanya pemberitahuan kongesti kepada *node* tetangga dan adanya kerja sama untuk segera melakukan pembatasan pengiriman pesan, CQLCC mampu menghasilkan hasil *overhead* pada pergerakan *Haggle-3* yang tidak begitu jauh dengan hasil *overhead* CQLCC pada pergerakan *Random Waypoint*.

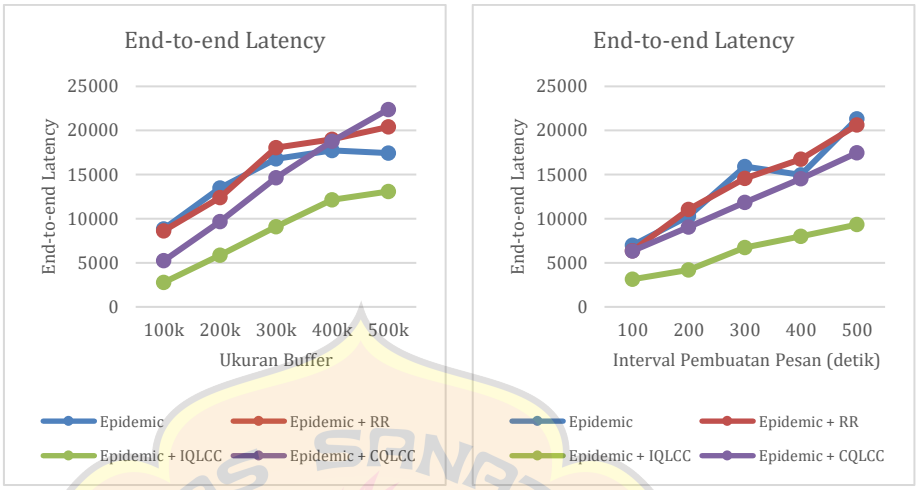


(a) (b)

Gambar 4. 6. Grafik *goodput* pada pergerakan *Haggle-3*.

Karena sistem kerja sama yang baik antara setiap *node*, pada pergerakan *Haggle-3*, hasil *goodput* CQLCC tetap jauh lebih baik dibandingkan yang lain, seperti yang ditunjukkan pada Gambar 4.6.

Bahkan, disaat ukuran *buffer* semakin besar, CQLCC menghasilkan kenaikan *goodput* yang cukup signifikan.



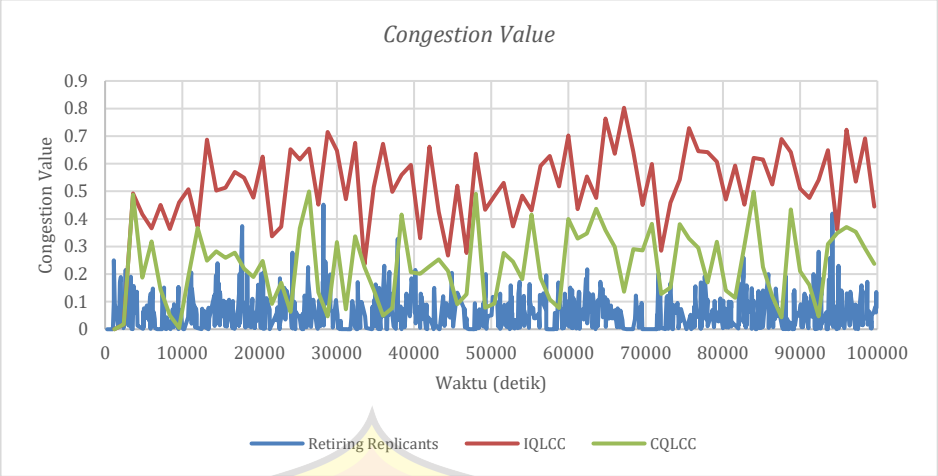
(a) (b)

Gambar 4. 7. Grafik *end-to-end latency* pada pergerakan *Haggle-3*.

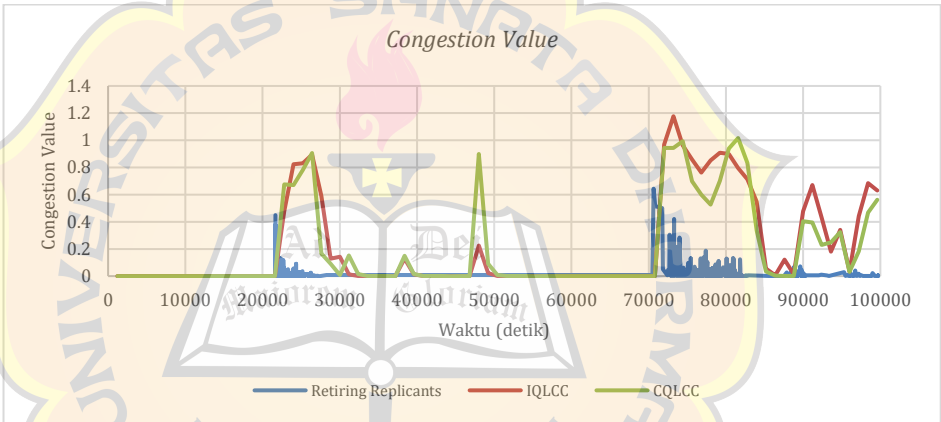
Sedangkan pada *end-to-end latency*, CQLCC memiliki *delay* yang lebih lama dibandingkan IQLCC karena penyebaran pesan yang lebih dibatasi. Namun, seperti yang dapat kita lihat pada Gambar 4.7(a), meskipun dengan penyebaran pesan yang dibatasi, *delay* yang dihasilkan oleh CQLCC saat ukuran *buffer* semakin kecil, jauh lebih baik dibandingkan RR dan *routing epidemic* yang asli. Hal ini menandakan, bahwa semakin kongestinya suatu jaringan, kerja sama antara setiap *node* yang memiliki perbedaan pandangan semakin dibutuhkan untuk meningkatkan kinerja jaringan itu sendiri. Bahkan, pada Gambar 4.7(b), terlihat bahwa *delay* yang dihasil olehkan CQLCC, selalu cenderung lebih baik dibandingkan dengan RR dan *routing epidemic* yang asli.

4.4.3 Hasil *Congestion Value*

CV atau nilai kongesti merupakan parameter yang mengindikasikan kondisi kongesti di jaringan dalam penelitian ini. Nilai CV yang stabil tentu akan mempengaruhi kinerja jaringan juga.



Gambar 4. 8. Grafik CV pada pergerakan *Random Waypoint*.



Gambar 4. 9. Grafik CV pada pergerakan *Haggle-3*.

Pada pergerakan *Random Waypoint* maupun pada pergerakan *Haggle-3* pada Gambar 4.8 dan Gambar 4.9, CV yang dihasilkan oleh RR memiliki nilai yang lebih rendah dari pada CQLCC dan IQLCC. Tetapi, fluktuasi CV yang terjadi pada RR sangatlah sering. Hal ini dikarenakan pada algoritma RR, nilai CV diperbaharui setiap adanya kontak dengan *node* lain. Sedangkan, CQLCC dan IQLCC memperbaharui CV dalam interval waktu tertentu.

Jika kontak antara setiap *node* sangat sering terjadi, maka terkadang nilai CV akan diperbaharui dengan sangat cepat, yang dapat mengakibatkan terkadang ada nilai CV yang sangat rendah, namun diwaktu berikutnya bisa terdapat nilai CV yang sangat tinggi. Akibatnya, kinerja jaringan menjadi kurang optimal.

Pada pergerakan *Random Waypoint*, meskipun IQLCC dan CQLCC memiliki nilai CV yang lebih tinggi dari pada RR, dengan adanya interval waktu untuk memperbaharui CV, maka CV tidak berfluktuasi dengan cepat, dan pengamatan pada kondisi jaringan pun menjadi lebih baik. Akibatnya, kinerja jaringan dalam menangani kongesti pun menjadi lebih optimal.

Pada Gambar 4.8, terlihat bahwa CQLCC memiliki CV yang lebih rendah lagi dibandingkan dengan IQLCC. Hal ini dikarenakan sistem kerja sama antara setiap *node* dalam membatasi penyebaran pesan yang mampu membuat kongesti jarang terjadi. Sehingga, pesan yang dibuang pun dapat semakin berkurang dan nilai CV dapat menjadi lebih rendah.

Berbeda dengan pergerakan *Random Waypoint*, pada pergerakan *Haggle-3* yang ditunjuk pada Gambar 4.9 memperlihatkan bahwa terjadi kenaikan dan penurunan nilai CV yang sangat tajam pada CQLCC dan IQLCC. Selain itu, hasil CV antara CQLCC dan IQLCC pun terlihat tidak jauh berbeda. Hal tersebut diakibatkan pada *Random Waypoint* kondisi jaringan lebih stabil akibat probabilitas kontak setiap *node* yang sama, berbeda dengan *Haggle* yang memiliki probabilitas kontak yang berbeda-beda antara setiap *node*-nya. Namun, karena fluktuasi yang tidak terjadi secara cepat dan keadaan yang lebih stabil, CQLCC dan IQLCC mampu menghasil unjuk kerja yang lebih baik dalam menangani kongesti ketika kongesti CQLCC mampu bekerja lebih baik lagi akibat adanya kerja sama dan pemberitahuan kongesti antara setiap *node* dengan *node* tetangganya.

BAB V
KESIMPULAN DAN SARAN

5.1.Kesimpulan

Sebagai suatu nilai yang mengindikasikan terjadinya kongesti, CV yang sangat cepat berfluktuasi dapat mengakibat kinerja yang kurang optimal. Karena itu, disaat setiap *node* diberikan kemampuan untuk mempelajari kondisi lingkungannya, CV yang diamati dalam suatu interval waktu mengakibatkan CV tidak berfluktuasi dengan cepat, yang membuat kinerja jaringan semakin stabil dan optimal.

CQLCC terbukti mampu mengendalikan kongesti dan membatasi penyebaran pesan dengan lebih optimal dibandingkan RR dan IQLCC. Namun, pada pergerakan *Random Waypoint* CQLCC memberikan *delay* yang lebih besar dibandingkan dengan yang lain. Sedangkan, pada pergerakan *Haggle* yang memiliki probabilitas kontak yang sangat berbeda-beda pada setiap *node*-nya, CQLCC menunjukkan performa yang lebih menonjol dalam menekan kongesti dan *delay* yang lebih baik dibandingkan di pergerakan *Random Waypoint*. Karena pada *Random Waypoint*, meski tidak adanya kerja sama, probabilitas kontak yang sama membuat keputusan yang diambil oleh setiap *node* dapat menjadi lebih kompak dibandingkan pada pergerakan *Haggle-3*. Karena itu, pada pergerakan *Haggle-3* yang memiliki *node* yang mengetahui kondisi jaringan dengan baik dan *node* yang tidak mengetahui kondisi jaringan dengan baik, kerja sama sangat diperlukan untuk meningkatkan kinerja yang optimal.

5.2.Saran

Pada penelitian selanjutnya, sebaiknya *state-space* yang dimiliki oleh *Q-Learning* dapat bersifat *continues*. Sehingga, kondisi dari CV dapat diamati dengan lebih efektif. Selain itu, untuk sistem kerja sama antara setiap *node* dapat lebih ditingkatkan lagi dengan saling menukarkan aturan (*policy, Q-Value*) satu sama lain, sehingga pada pergerakan manusia yang memiliki *node*

populer dan *node* yang tidak populer, dapat memiliki pandangan tentang aturan yang lebih konvergen antara satu sama lain.

DAFTAR PUSTAKA

[1] S. Burleigh *et al.*, "Delay-tolerant networking: An approach to interplanetary internet," *IEEE Commun. Mag.*, vol. 41, no. 6, pp. 128–136, Jun. 2003, doi: 10.1109/MCOM.2003.1204759.

[2] Institute of Electrical and Electronics Engineers., *2010 proceedings, IEEE INFOCOM : San Diego, California, USA : 15-19 March 2010.* IEEE Xplore, 2010.

[3] A. P. Silva, K. Obraczka, S. Burleigh, J. M. S. Nogueira, and C. M. Hirata, "A congestion control framework for delay- and disruption tolerant networks," *Ad Hoc Networks*, vol. 91, Aug. 2019, doi: 10.1016/j.adhoc.2019.101880.

[4] M. Tan, "Multi-Agent Reinforcement Learning: Independent vs. Cooperative A g e n ts."

[5] A. Keränen, J. Ott, and T. Kärkkäinen, "The ONE simulator for DTN protocol evaluation," 2009. doi: 10.4108/ICST.SIMUTOOLS2009.5674.

[6] James Scott, Richard Gass, Jon Crowcroft, Pan Hui, Christophe Diot, and Augustin Chaintreau, "Haggle Dataset," 2009. https://crawdad.org/cambridge/haggle/20090529/

[7] V. François-lavet *et al.*, "An Introduction to Deep Reinforcement Learning. (arXiv:1811.12560v1 [cs.LG]) http://arxiv.org/abs/1811.12560," *Found. trends Mach. Learn.*, vol. II, no. 3–4, pp. 1–140, 2018, doi: 10.1561/2200000071.Vincent.

[8] *GLOBECOM 2011-2011 IEEE Global Communications Conference.* IEEE, 2011.

[9] A. Vahdat and D. Becker, "Epidemic Routing for Partially-Connected Ad Hoc Networks."

LAMPIRAN

https://github.com/christinnetjung/ONE_CooperativeQL