DESAIN DAN IMPLEMENTASI *IN-NETWORK CACHING* PADA *CONTENT CENTRIC NETWORKING* MENGGUNAKAN CCN-LITE DENGAN SIMULATOR OMNET++

SKRIPSI

KEMINATAN TEKNIK KOMPUTER

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Ibrahim
NIM: 135150307111015

PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2018

PENGESAHAN

DESAIN DAN IMPLEMENTASI *IN-NETWORK CACHING* PADA *CONTENT CENTRIC NETWORKING* MENGGUNAKAN CCN-LITE DENGAN SIMULATOR OMNET++

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh :
Ibrahim
NIM: 135150307111015

Skripsi ini telah diuji dan dinyatakan lulus pada
15 Januari 2018
Telah diperiksa dan disetujui oleh:

Dosen Pembimbing I	Dosen Pembimbing II
Achmad Basuki, S.T., M.MG., Ph.D.	Eko Sakti Pramukantoro, S.Kom., M.Kom.
NIP: 19741118 200312 1 002	NIK: 201102 860805 1 001

Mengetahui
Ketua Jurusan Teknik Informatika

Tri Astoto Kurniawan, S.T., M.T., Ph.D.
NIP: 19710518 200312 1 001

PERNYATAAN ORISINALITAS

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah skripsi ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu perguruan tinggi, dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis disitasi dalam naskah ini dan disebutkan dalam daftar pustaka.

Apabila ternyata didalam naskah skripsi ini dapat dibuktikan terdapat unsur-unsur plagiasi, saya bersedia skripsi ini digugurkan dan gelar akademik yang telah saya peroleh (sarjana) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang, 15 Januari 2018

Ibrahim
NIM: 135150307111015

KATA PENGANTAR

Bismillah hirrohmanirrohim. Puji syukur penulis panjatkan kehadirat Tuhan Yang Maha Esa, atas berkat rahmat dan hidayat-Nya, penulis dapat menyelesaikan skripsi dengan judul "DESAIN DAN IMPLEMENTASI *IN-NETWORK CACHING* PADA *CONTENT CENTRIC NETWORKING* MENGGUNAKAN CCN-LITE DENGAN SIMULATOR OMNET++".

Dalam penyusunan dan penelitian skripsi ini tidaklah lepas dari bantuan berbagai pihak baik dalam bentuk materiil ataupun nasehat. Dalam kesempatan ini penulis ingin mengucapkan kepada:

1. Allah Yang Maha Esa atas berkah rahmat dan hidayat-Nya serta karena-Nya skripsi ini telah selesai dengan baik.
2. Ibu, Ayah, dan seluruh keluarga atas segenap do'a dan dukungan yang telah diberikan.
3. Bapak Achmad Basuki, S.T., M.MG, Ph.D. dan Bapak Eko Sakti Pramukantoro, S.Kom., M.Kom. selaku dosen pembimbing yang telah memberikan bantuan berupa ilmu yang sangat luas dan bimbingannya serta memberikan support kepada penulis untuk menyelesaikan skripsi ini.
4. Bapak Tri Astoto Kurniawan, S.T., M.T., Ph.D. selaku Ketua Jurusan Teknik Infomatika.
5. Hanif Kuncahyo Adi, S.Kom. yang telah memberikan saran dan semangat dalam pengerjaan skripsi.
6. M. Maulana dan Fadlun Akbar atas kesediaannya menjadi teman diskusi selama pengerjaan skripsi
7. Seluruh teman-teman di program studi Teknik Komputer 2013 yang telah membantu memberikan semangat untuk bisa menyelesaikan skripsi hingga akhir.
8. Seluruh pihak yang tidak dapat diucapkan satu persatu, penulis mengucapkan terima kasih atas bantuan do'a sehingga skripsi ini dapat terselesaikan.

Penulis menyadari bahwa dalam penulisan laporan skripsi ini masih banyak terdapat kekurangan baik format penulisan maupun isinya. Untuk itu penulis sangat mengharapkan kritik dan saran yang membangun. Penulis berharap skripsi ini dapat bermanfaat bagi seluruh pihak.

Malang, 15 Januari 2018

Penulis
hanyabaim@gmail.com

ABSTRAK

Ibrahim, Desain dan Implementasi In-Network Caching Pada Content Centric Networking Menggunakan CCN-Lite Dengan Simulator OMNeT++

Pembimbing: Achmad Basuki, S.T., M.MG., Ph.D. dan Eko Sakti Pramukantoro, S.Kom., M.Kom.

Content Centric Networking (CCN) adalah sebuah arsitektur yang diusulkan untuk mengatasi permasalahan pada Internet saat ini dalam hal pendistribusian data. CCN ini sudah berbasis *content centric* atau dalam melakukan pencarian data berdasarkan pada nama konten. Hal ini memberikan keuntungan pada CCN jika dibandingkan dengan arsitektur Internet yang melakukan pencarian data berdasarkan lokasi dari data (*host centric*). CCN juga mempunyai mekanisme *caching* yang berbeda dengan *caching* pada umumnya yaitu, *in-network caching*. Penelitian ini bertujuan untuk mengetahui bagaimana kinerja dari *in-network caching* pada CCN. Untuk mewujudkan hal tersebut, dilakukan desain dan implementasi *in-network caching* pada CCN menggunakan CCN-Lite dengan simulator OMNeT++. Parameter yang diuji adalah latensi pengiriman dan *cache hit ratio* (CHR). Berdasarkan dari simulasi yang telah dilakukan, diperoleh latensi pengiriman terkecil yaitu 10.69 ms. Nilai ini didapatkan dari terjadinya proses *caching*, jika dibandingkan dengan latensi pengiriman tanpa *caching* yaitu sebesar 32.45 ms. Hal tersebut membuktikan bahwa dengan adanya *caching,* latensi yang didapatkan semakin rendah. CHR tertinggi didapatkan pada protokol file CCNTLV yaitu sebesar 0.54 atau 54%, jika dibandingkan dengan protokol file CCNB sebesar 0.45 atau 45%. Hal ini membuktikan bahwa semakin tinggi nilai CHR maka semakin baik pula konten tersebut dilayani.

Kata kunci: *in-network caching*, *content centric networking*, CCN-Lite, CHR, latensi pengiriman.

ABSTRACT

Ibrahim, ***Design and Implementation In-Network Caching On Content Centric Networking Using CCN-Lite With OMNeT++'s Simulator.***

Supervisor: **Achmad Basuki, S.T., M.MG., Ph.D.** ***and*** **Eko Sakti Pramukantoro, S.Kom., M.Kom.**

Content Centric Networking (CCN) is a proposed architecture for addressing current Internet issues in terms of data distribution. CCN is already content-centric based or in searching data based on the name of the content. This provides an advantage over CCN when compared to an Internet architecture that searches data based on the location of the data (host centric). CCN also has a different caching mechanism than caching in general, in-network caching. This study aims to find out how the performance of in-network caching on CCN. To achieve this, the design and implementation of in-network caching on CCN using CCN-Lite with the OMNeT ++ simulator is performed. The parameters tested were delivery latency and cache hit ratio (CHR). Based on the simulation that has been done, obtained the smallest delivery latency is 10.69 ms. This value is obtained from the caching process, when compared with the latency of delivery without caching that is equal to 32.45 ms. It proves that with the caching, the latency gets lower. The highest CHR obtained in the CCNTLV protocol file is 0.54 or 54%, when compared with the CCNB file protocol of 0.45 or 45%. This proves that the higher the value of CHR the better the content is served.

Keywords: *in-network caching, content centric networking, CCN-Lite, CHR, delivery latency.*

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

DAFTAR LAMPIRAN

BAB 1 PENDAHULUAN

1.1 Latar Belakang

Content Centric Networking (CCN) adalah sebuah arsitektur yang diusulkan untuk mengatasi permasalahan pada Internet saat ini dalam hal pendistribusian data. CCN ini sudah berbasis *content centric* atau dalam melakukan pencarian data berdasarkan pada nama konten. Hal ini memberikan keuntungan pada CCN jika dibandingkan dengan arsitektur Internet yang melakukan pencarian data berdasarkan lokasi dari data (*host centric*). Ditambah lagi dengan adanya protokol CCNx yang mendukung dalam pencarian data konten. CCN ini dikembangkan berdasar dari konsep *Information Centric Networking* (ICN). Terdapat banyak arsitektur yang telah diusulkan berdasarkan konsep ICN seperti *Named Data Networking* (NDN), *Network of Information* (NetInf), *A Data-Oriented Network Architecture* (DONA), dan masih banyak lagi, tidak terkecuali *Content Centric Networking* (CCN) (Wang, dkk., 2011). Akan tetapi, arsitektur-arsitektur tersebut masih belum ada yang dioperasionalkan.

CCN memiliki 2 tipe paket pengiriman yaitu *Interest Packet* atau disebut paket permintaan dan *Data Packet* atau paket yang berisi konten. CCN juga memiliki 3 struktur data yaitu *Content Store* (CS), *Pending Interest Table* (PIT) dan *Forwarding Information Base* (FIB). Ketika sebuah *client* melakukan permintaan data, maka paket *Interest* akan dikirimkan menuju *router*. Pada *router* dilakukan pencarian di dalam CS, apakah *cache* dari data tersebut ada atau tidak, apabila ada maka data tersebut akan dikirimkan menuju *client* yang meminta tanpa harus menuju ke *server*. Pada CCN juga memiliki model penghapusan (*cache replacement*) dan model penyimpanan *cache* (*cache decision* atau *cache strategy*) atau biasa disebut juga dengan *in-network caching* (Jacobson, 2009).

Di dalam ilmu komputer, *cache* adalah sebuah komponen *hardware* atau *software* yang menyimpan data sementara. Tujuan dari *cache* adalah untuk meningkatkan transfer data dengan menyimpan data yang pernah diakses pada *cache* tersebut, sehingga apabila pengguna ingin mengakses data tersebut maka akses dapat dilakukan lebih cepat. *In-network caching* yang ada pada CCN berbeda dengan *caching* yang ada pada Internet saat ini. Mekanisme *caching* pada CCN menyimpan *cache* dari konten tersebut pada *Content Store* di dalam *router,* berbeda dengan mekanisme yang dilakukan pada arsitektur *Content Delivery Network* (CDN). CDN melakukan c*ache* dari konten tersebut dan ditempatkan pada server yang telah tersebar. Akan tetapi *caching* pada CDN memiliki kelemahan yaitu semakin jauh lokasi *server* dengan *client*, semakin lama pula konten tersebut sampai ke *client* (Beal, 2014).

Sebaliknya, CCN memiliki keuntungan yang menyelesaikan permasalahan dari CDN seperti meningkatkan *throughput* pada pengguna karena *cache* dari konten yang pernah diakses telah tersimpan di *router* CCN, mengurangi trafik dan latensi yang tinggi akibat permintaan data yang sama dari *client* yang berbeda (Ahir & Kumbharkar, 2012). Metode-metode dari *in-network caching* telah banyak

dilakukan dalam penelitian untuk menerapkan metode *caching* mana yang lebih baik. Hal ini dibuktikan dengan adanya penelitian yang melakukan perbandingan metode seperti (Mangili, dkk,. 2015) dan (Tortelli, dkk,. 2016), dan beberapa penelitian yang melakukan implementasi dari metode-metode yang diajukan seperti (Saino, dkk,. 2014) dan (Shibuya, dkk,. 2016). Akan tetapi, penelitian tersebut berfokus pada *caching* mana yang lebih baik.

Oleh karena itu, penelitian ini bertujuan untuk mendesain dan mengimplementasikan *in-network caching* pada arsitektur CCN menggunakan CCN-Lite, mengetahui bagaimana kinerja *in-network caching* pada arsitektur CCN di lingkungan CCN-Lite, terutama pada aspek latensi pengiriman dan *cache hit ratio* (CHR). Dalam penelitian ini, pengujian dilakukan dengan menggunakan simulator OMNeT++ dengan bantuan pustaka pendukung INET Framework.

1.2 Rumusan Masalah

Berdasarkan latar belakang yang telah dikemukakan, rumusan masalah pada penelitian ini sebagai berikut:

1. Bagaimana melakukan desain dan implementasi dari *in-network caching* pada CCN-Lite berdasarkan arsitektur CCN?
2. Bagaimana kinerja dari *in-network caching* pada CCN-Lite di lingkungan simulator OMNeT++?

1.3 Tujuan

Tujuan dari penelitian ini adalah untuk melakukan desain dan implementasi *in-network caching* pada *Content Centric Networking* menggunakan CCN-Lite dengan simulator OMNeT++ dan mengetahui bagaimana proses komunikasi dari arsitektur *content centric networking* (CCN) pada CCN-Lite.

1.4 Manfaat

Manfaat dari penelitian ini adalah mendapatkan pengetahuan tentang bagaimana mekanisme komunikasi *Content Centric Networking*, mekanisme dan kinerja dari *in-network caching* pada CCN yang dilakukan melalui simulasi, dan mendapatkan pengetahuan dasar dalam melakukan penelitian selanjutnya terkait dengan arsitektur CCN, serta dapat dijadikan referensi oleh penulis, pembaca, dan peneliti selanjutnya terkait dengan *in-network caching* pada arsitektur CCN.

1.5 Batasan Masalah

Agar permasalahan yang dirumuskan dapat lebih terfokus, maka penelitian ini dibatasi dalam hal:

1. Simulasi ini dilakukan menggunakan *workspace* protokol CCN-Lite dan simulator OMNeT++ serta bantuan INET Framework sebagai pustaka pendukung.
2. Pada simulasi ini menggunakan topologi 6 *client*, 4 *router*, 1 *server*.

3. Pengiriman paket dilakukan dengan menggunakan 2 tipe *file* yaitu CCNB dan CCNTLV.
4. Penelitian ini mengabaikan aspek keamanan jaringan.
5. Parameter pengujian yang digunakan yaitu latensi pengiriman dan *Cache Hit Ratio* (CHR).
6. Selama simulasi berlangsung, penulis tidak dapat mengubah baik skenario pengujian maupun topologi.
7. Pengaturan jalur pengiriman dari simulasi dilakukan secara statik.

1.6 Sistematika Pembahasan

Sistematika penulisan penelitian ditunjukkan untuk memberikan gambaran dan uraian dari penyusunan tugas akhir secara garis besar yang meliputi beberapa bab, sebagai berikut.

BAB I PENDAHULUAN

Menjelaskan latar belakang, rumusan masalah, batasan masalah, tujuan, manfaat dan, sistematika penulisan.

BAB II LANDASAN PUSTAKA

Menguraikan kajian pustaka dan dasar teori yang mendasari dari arstitektur *Content Centric Networking*, *In-Network Caching*, CCN-Lite, OMNeT++ dan INET Framework.

BAB III METODOLOGI

Menguraikan tentang metode dan langkah kerja yang terdiri dari studi literatur, analisis kebutuhan simulasi, desain topologi simulasi, perangkat yang digunakan, persiapan simulasi, implementasi dan pengujian, analisis hasil, serta penarikan kesimpulan dan saran.

BAB IV PERANCANGAN DAN IMPLEMENTASI SIMULASI

Memuat proses dari instalasi aplikasi yang digunakan dalam simulasi dan melakukan pengaturan topologi serta perilaku dari setiap *node*.

BAB V SIMULASI DAN ANALISIS HASIL

Memuat proses simulasi dari rancangan yang telah dibuat dan menganalisis hasil pengujian dari penelitian yang telah dilakukan.

BAB VI PENUTUP

Bab terakhir ini berisi kesimpulan dari pembahasan rumusan masalah yang ada berdasarkan analisis dan pengujian yang telah dilakukan dalam proses penelitian "DESAIN DAN IMPLEMENTASI *IN-NETWORK CACHING* PADA *CONTENT CENTRIC NETWORKING* MENGGUNAKAN CCN-LITE DENGAN SIMULATOR OMNET++".

BAB 2 LANDASAN KEPUSTAKAAN

2.1 Kajian Pustaka

Beberapa penelitian yang dijadikan referensi penulis terkait dengan konsep CCN dan *In-network caching* yaitu "*Could In-Network Caching Benefit Information-Centric Networking?*" yang ditulis oleh Sen Wang, Jun Bi, Jianping Wu, Zhaogeng Li, Wei Zhang, Xu Yang pada tahun 2011. Penelitian ini membahas tentang bagaimana menggabungkan permasalahan *in-network caching* pada ICN kedalam permasalahan *Mixed-Integer Linier Programming* dan menggunakan *cache policy Least Benefit* (LB). LB adalah sebuah *cache policy* yang digunakan dalam penelitian ini untuk menghitung *hit ratio* sama seperti yang dilakukan *Least Frequency Used* (LFU). Skema pengiriman yang digunakan adalah *Forwarding Shallow Flooding* (FSF). Hasil yang didapatkan dari penelitian ini bahwa kinerja LB lebih baik dari pada LFU ketika menggunakan skema pengiriman FSF dan mengurangi rata-rata *hops* sebesar 6.3%.

Penelitian selanjutnya yaitu "*Could End System Caching and Cooperation Replace In-Network Caching in CCN?*" yang ditulis oleh Haibo Wu, Jun Li, dan Jiang Zhi pada tahun 2015. Penelitian ini membahas tentang penerapan *End System Caching and Cooperation* (ESCC) di dalam CCN. ESCC adalah suatu skema distribusi konten yang dapat melakukan distribusi konten langsung menuju *client* seperti halnya *peer-to-peer.* Beberapa keuntungan yang diberikan dari metode ini yaitu simpel karena hanya memodifikasi data dan tidak memerlukan komponen tambahan, efisien karena memungkinkan mendapatkan kembali konten dari *client* terdekat, tahan/kuat dalam hal hilangnya konten atau kesalahan pada *client,* dan rendah daya. Penelitian ini diimplementasikan melalui simulasi menggunakan NDNSim. Hasil yang diperoleh menunjukkan bahwa skema ESCC memiliki kinerja yang lebih baik daripada *caching* secara umum karena menjadikan *caching* simpel, efisien, kuat dan rendah daya.

Penelitian selanjutnya yaitu "*Multi-Objective In-Network Caching Strategies*" yang ditulis oleh Liang Wang dari *Department of Computer Science,* Universitas Helsinki, Finland pada tahun 2013. Dalam penelitian ini membahas bagaimana menggunakan model optimasi dan desain heuristik untuk mempelajari dan mengembangkan strategi *caching* baru dengan beberapa tujuan untuk *cache* jaringan serta mengevaluasi kinerjanya dalam bentuk yang realistis. Pada penelitian ini, sebuah strategi *caching* terdiri dari 3 kebijakan yaitu konten mana yang akan di *cache* (*admission policy*), konten mana yang akan diganti (*replacement policy*) dan, dimana konten di cache dalam jaringan *cache* kolaboratif (*cooperation policy*). Strategi *caching* yang digunakan dalam jaringan *Sprint* ada 3 yaitu *ALL, Cachedbit* dan, *NbSC*. *ALL* adalah strategi dasar, *admission policy* yaitu menyimpan semuanya dan *replacement policy* yaitu LRU. *Cachedbit* menambahkan probabilistik *caching* pada paket yang melewati *router* berdasarkan dari *ALL*. *NbSC* menambahkan *cooperation policy* berdasarkan *Cachedbit* agar dapat mencari konten yang hilang pada *node* sebelah.

Parameter pengujian yang digunakan ada 3 yaitu *hit rate, footprint reduction,* dan *NbSC* dengan berbeda radius pencarian. Dari pengujian yang telah dilakukan berdasar pada 3 parameter pengujian didapatkan hasil bahwa strategi NbSC lebih baik daripada *ALL* dan *Cachedbit*. Semakin tinggi *cache size* semakin tinggi *hit rate*, semakin besar *cache size* semakin tinggi pula *footprint reduction* dan, semakin meningkat radius pencarian, *hit rate* tetap pada level yang sama, tetapi *footprint reduction* menurun drastis. Ini menandakan bahwa ketika radius pencarian kecil maka trafik baik pada intra maupun inter-ISP, namun ketika radius pencarian sangat besar, hal tersebut akan memberatkan trafik intra-ISP karena terlalu banyak permintaan pada jaringan.

Penelitian selanjutnya adalah "*Performance analysis of in-network caching for content-centric networking*" yang ditulis oleh Yusung Kim dan Ikjun Yeom dari *Department of computer engineering*, *Sung Kyun Kwan University*, Suwon, Republic of Korea pada tahun 2013. Dalam penelitian ini membahas bagaimana analisis kinerja dari *in-network caching* pada *Named Data Networking* (NDN). Pada penelitian ini, pengujian pertama yang dilakukan adalah pengujian terhadap ketidakefisienan dari LRU sebagai *simple cache replacement* yang terdapat pada setiap *router* yang dilalui oleh jalur data. Di dalam LRU, digunakan 2 mekanisme *in-network caching* untuk popularitas konten jangka panjang pada jaringan AS. Yang pertama adalah *single-path caching*, dimana permintaan diharapkan untuk mencari konten melalui jalur terpendek menuju penyedia konten. Pada mekanisme lain adalah *Network-Wide Caching*, dimana konten dapat diperoleh dari *router* mana saja di dalam jaringan, tidak terkecuali data yang telah di *cache*.

Dari kedua mekanisme, agar model penempatan konten bekerja dengan optimal, maka digunakan *Mixed Integer Program* (MIP), dengan mem-pertimbangkan mulai dari *link cost* pada internal dan eksternal, ukuran *cache*, *link capacity,* dan popularitas konten. Dari pengujian yang telah dilakukan, didapatkan hasil bahwa MIP meningkatkan performa jaringan dibandingkan dengan LRU. Terutama pada *Network-Wide Caching* yang lebih unggul dari *Single-Path Caching* pada keseluruhan jaringan.

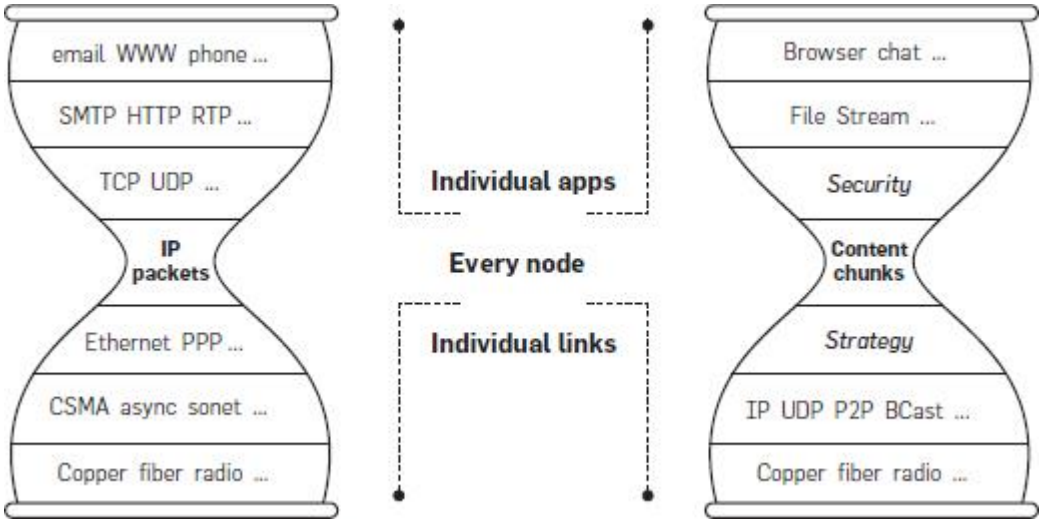
2.2 Dasar Teori

2.2.1 *Content Centric Networking* (CCN)

Content Centric Networking atau biasa disingkat CCN adalah suatu arsitektur baru di dalam dunia jaringan. CCN ini dikembangkan berdasar dari konsep *Information Centric Networking* (ICN). ICN adalah sebuah terobosan baru yang ditujukan untuk mengembangkan arsitektur dari Internet yang jauh dari paradigma *host-centric* berdasarkan prinsip *end-to-end*, menjadi arsitektur jaringan yang berfokus "*named information*" (atau *Content*). Arsitektur ini diharapkan memiliki manfaat yaitu pengingkatan efisiensi, skalabilitas yang lebih baik dengan informasi atau permintaan *bandwidth* dan ketahanan yang lebih baik dalam skenario komunikasi (Jacobson, 2009).

Ketika arsitektur ICN muncul, banyak sekali arsitektur baru yang berdasarkan pada konsep ICN seperti NDN, NetInf, DONA, dan masih banyak lagi, tidak terkecuali CCN (Wang, et al., 2011). CCN merupakan pendekatan alternatif untuk arsitektur jaringan berbasis pada prinsip bahwa jaringan komunikasi harus memungkinkan pengguna untuk fokus pada data yang dia butuhkan, lokasi fisik, dan dari mana data yang akan diambil. CCN memungkinkan *caching* konten untuk mengurangi kemacetan dan meningkatkan kecepatan pengiriman, konfigurasi sederhana perangkat jaringan, dan keamanan yang dibangun ke dalam jaringan di tingkat data (Ahir & Kumbharkar, 2012).

Tujuan dari CCN adalah untuk memberikan jaringan yang lebih aman, fleksibel dan terskala sehingga dapat mengatasi kebutuhan modern Internet untuk distribusi konten yang aman pada skala besar untuk beragam perangkat (Jacobson, dkk,. 2009). CCN juga menyediakan model keamanan dimana yang diamankan adalah potongan-potongan dari konten bukan koneksinya. Hal ini memberikan fleksibilitas dengan menggunakan nama, bukan alamat IP. Selain itu, nama dan konten yang aman berada di *cache* yang didistribusikan secara otomatis sesuai permintaan (Ghali, dkk,. 2016). Ketika pengguna meminta sebuah konten, CCN memberikan nama konten untuk pengguna dari *cache* terdekat, melintasi hop jaringan lebih sedikit, menghilangkan permintaan berlebihan, dan mengurangi konsumsi sumber daya secara keseluruhan (PARC Company, 2010).

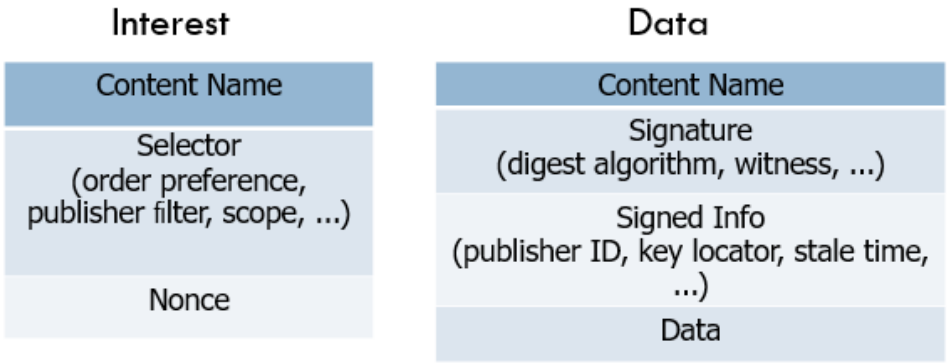


Gambar 2.1 Perbedaan layer pada TCP/IP dan CCN
Sumber: (Jacobson, dkk., 2009)

Pada Gambar 2.1 adalah perbedaan layer yang terjadi pada protokol TCP/IP dengan layer pada CCN. Dalam pengiriman paket, layer TCP berfokus pada IP dari konten yang dituju, berbeda dengan layer CCN yang berfokus pada *content chunks.* Beberapa perbedaan model komunikasi TCP/IP dengan CCN adalah sebagai berikut:

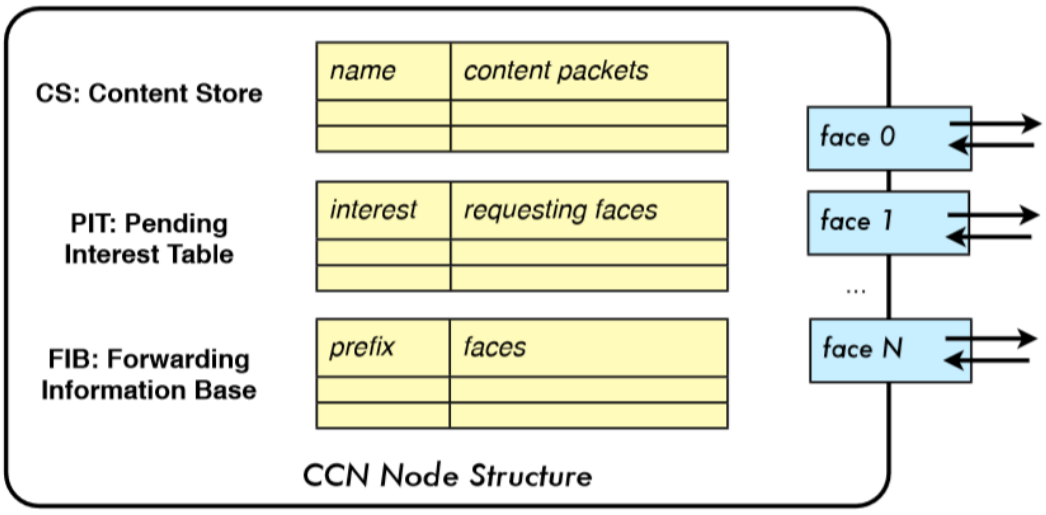
- Model komunikasi *receiver-drive*: Penerima/pengguna mendapatkan informasi dengan mengirimkan pesan Interest.

- Skema penamaan konten yang hirarki: CCN tidak mengutamakan *host* tertentu, melainkan *Content Object* itu sendiri. Konten diberi nama yang hirarkis seperti nama URL. *Interest Packet* dikirimkan dengan melakukan *longest-prefix matching* pada pemilihan jalur *forwarding*.
- Arsitektur *Cache* dan *Forward*: Setiap router CCN dapat menyimpan data *cache* agar dapat digunakan untuk permintaan yang sama di waktu yang berbeda.



Gambar 2.2 *Interest Packet* dan Data *Packet*
Sumber: (Jacobson, dkk,. 2009)

Pada Gambar 2.2, komunikasi pada CCN dilakukan dengan menggunakan dua tipe paket yang berbeda, yaitu paket *Interest* dan paket Data. Paket Interest adalah sebuah paket permintaan yang dikirimkan menggunakan nama data. Paket Data adalah paket hasil permintaan dari paket *Interest* yang disebut juga dengan *Content Object*. Secara umum, *Content Object* terdiri atas potongan-potongan data yang disebut dengan *Chunks* (Ahir & Kumbharkar, 2012).



Gambar 2.3 Struktur data CCN
Sumber: (Mahadevan, 2014)

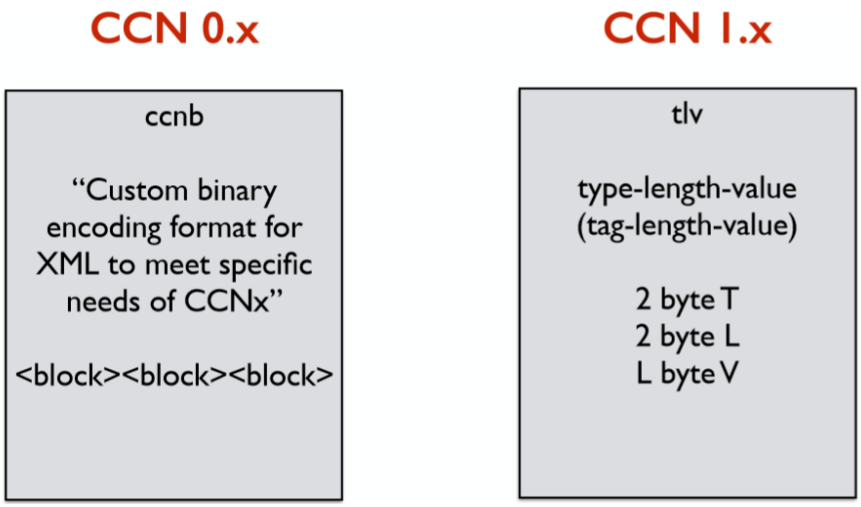
Seperti pada Gambar 2.3, CCN memiliki tiga struktur data yaitu *Content Store* (CS), *Pending Interest Table* (PIT), dan *Forwarding Information Base* (FIB). Beberapa istilah yang muncul di dalam konsep CCN seperti CS , *faces* (penyebutan lain dari *interface*), CCNx (protokol yang digunakan pada CCN) (Jacobson, dkk,. 2009). Penjelasan struktur data yang ada pada CCN adalah sebagai berikut:

1. *Content Store* (CS): Sebuah tempat penyimpanan sementara data *cache* yang didapat dari penerimaan paket data. *Cache* yang disimpan di CS bersifat sementara, artinya sewaktu-waktu dihapus. Ada 4 strategi penghapusan yang digunakan di CS yaitu LRU (*Least Recently Used*), LFU (*Least Frequency Used*), FIFO (*First In First Out*), dan *Random*. LRU akan menyimpan data yang masih baru dipakai dan akan menghapus data yang jarang dipakai. LFU akan melihat dari penggunaan data tersebut, seberapa sering data *cache* itu diambil, jika ada yang jarang diambil maka akan dihapus. *Random*, akan menghapus data *cache* yang ada di CS tanpa melihat seberapa sering data tersebut diminta atau seberapa besar *cache* tersebut.
2. *Pending Interest Table* (PIT): Struktur data yang menyimpan *Interest* beserta *face* yang meminta. Ketika data yang dicari tidak ditemukan pada CS maka, PIT akan menyimpan nama data beserta dengan asal *face* yang meminta sebelum diteruskan menuju FIB. Apabila ada *faces* lain meminta data yang sama, maka *Interest* dari *face* tersebut akan di drop dan ditambah ke PIT.
3. *Forwarding Information Base* (FIB): Struktur data yang sama dengan tabel IP *routing*. Meneruskan *Interest* menuju *next-hop* dari *router* awal. Proses yang digunakan dalam pencarian jalur adalah *longest-prefix matching*.

2.2.2 Protokol CCNx Dan Perubahan

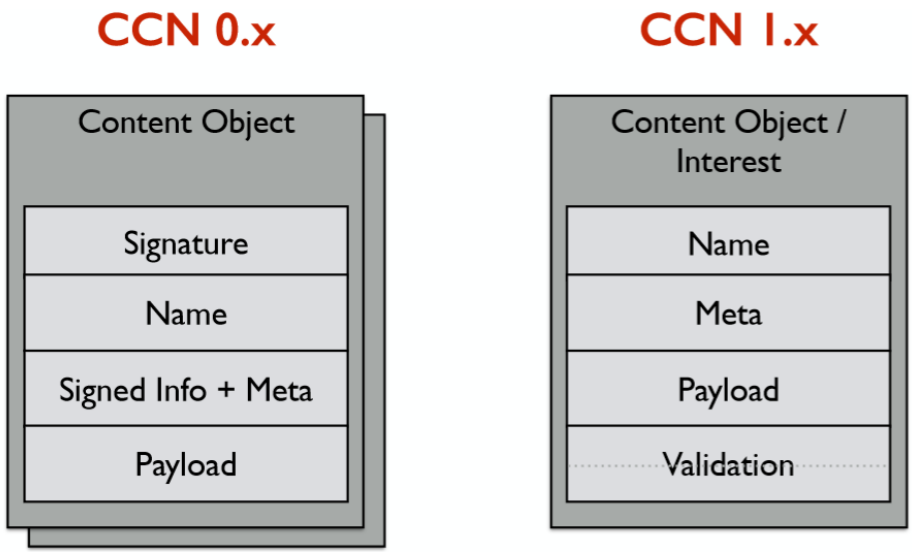
CCN mempunyai protokol yang digunakan dalam berkomunikasi saat ini yaitu CCNx 1.x. Protokol ini adalah bentuk pengembangan dari protokol yang sebelumnya yaitu CCNx 0.x yang dikembangkan oleh perusahaan PARC. Perusahaan ini berasal dari Palo Alto, California dan telah didirikan pada tahun 1970 oleh Alan Kay dan Jack Goldman. PARC adalah sebuah perusahaan yang melakukan riset dan pengembangan serta kontribusi pada teknologi informasi dan sistem *hardware*. Banyak sekali perubahan dari CCNx 0.x ke 1.x seperti perubahan format paket pengiriman, pengamanan data, *payload* pada *Interest*, dan lain-lain.

Pada Gambar 2.4 adalah perubahan paket *encoding* CCNx dari v.0 menjadi v.1. Perubahan ini merubah paket yang awalnya menggunakan CCNb menjadi TLV. Alasan mengapa paket ini diubah adalah CCNb fleksibel tetapi rumit, bergantung pada struktur meta, dan sedikit efisien. Sedangkan pada TLV mudah untuk diuraikan, mudah dimengerti serta efisien dalam penguraian data.



Gambar 2.4 Bagian paket *encoding* pada protokol CCNx
Sumber: (Mahadevan, 2014)

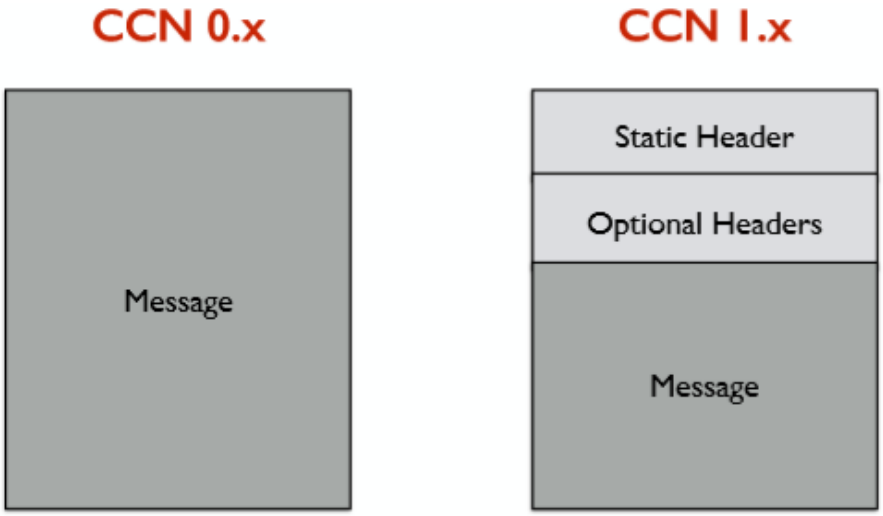
Pada Gambar 2.5 adalah perubahan susunan paket baik pada *Interest* maupun *Content Object.* Perubahan ini menggantikan fitur *signature* dengan *validation* sebagai pengamanan data. Alasan mengapa paket ini diubah adalah dengan menempatkan nama konten diawal, maka akan mempercepat *parsing* konten, validasi yang terpisah dengan metadata pada bagian akhir sebagai modular keamanan data, dan penggabungan format paket untuk menyederhanakan paket dan mempercepat *parsing* paket.



Gambar 2.5 Perubahan susunan pesan pada CCNx

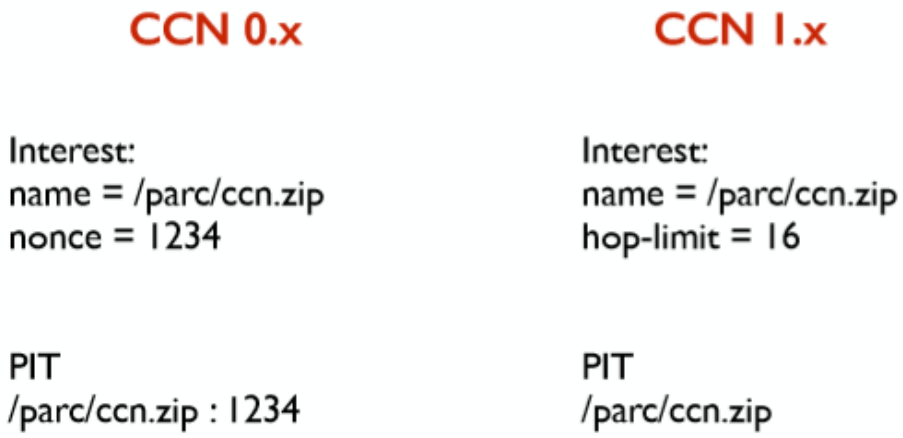
Pada Gambar 2.6 adalah perubahan dari format paket pesan pada protokol CCNx. Perubahan yang dilakukan adalah menambahkan *static header* dan *optional headers* pada CCNx 1.x. Adapun alasan dari penambahan *static header* adalah untuk mempermudah dan mempercepat dalam *parsing* paket, berisi kebutuhan umum yang dibutuhkan oleh paket, serta memungkinkan adanya perubahan versi pada paket. Pada *optional headers*, penambahan ini bertujuan untuk

memungkinkan elemen-elemen pada jaringan dilakukan penambahan atau perubahan informasi.



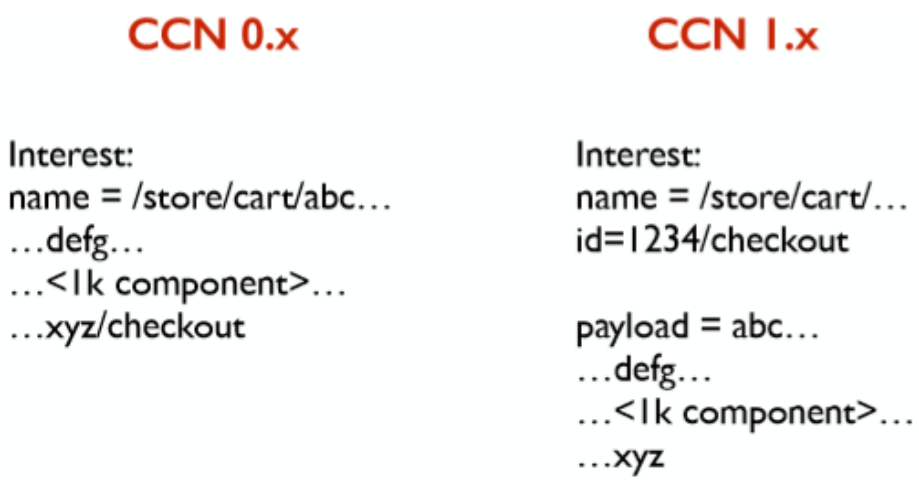
Gambar 2.6 Perubahan format paket pesan

Pada Gambar 2.7 adalah perubahan paket *looping* yang dilakukan pada saat pengiriman pesan sering berhenti untuk mencari data. Perubahan ini terdiri dari 3 alasan yaitu untuk mengurangi *overhead* dimana *nonce* digantikan dengan jumlah *hop* yang terbatas dan keseluruhan dari paket perulangan diatur oleh PIT.



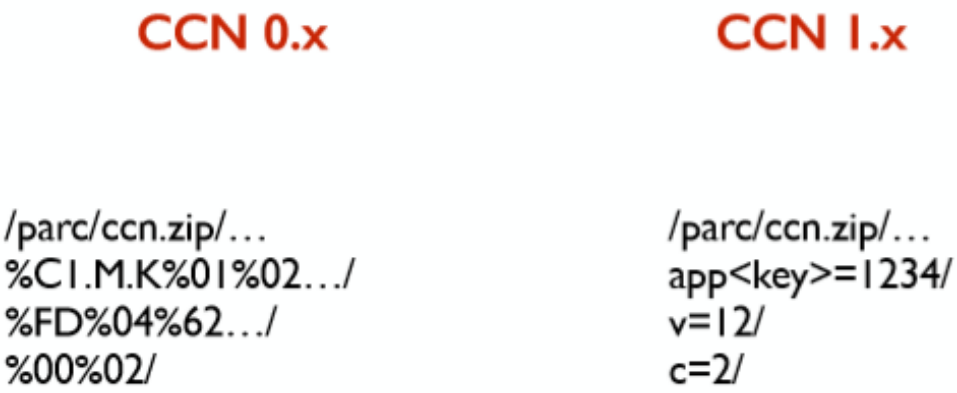
Gambar 2.7 Perubahan paket *looping*

Gambar 2.8 adalah perubahan yang terjadi pada paket *Interest* dimana pada CCNx 1.x ditambahkan fungsi *payload.* Adapun alasan perubahan ini adalah untuk mengurangi proses pada *router* dimana tidak memerlukan *parsing* menggunakan nama yang panjang pada setiap waktu, tidak menghabiskan kapasitas yang tersedia pada *router* dan, menurunkan trafik yang terlalu tinggi.



Gambar 2.8 Perubahan paket *Interest*

Gambar 2.9 adalah perubahan yang terjadi pada label berbasis nama di dalam CCNx. Adapun alasan mengapa perubahan ini dilakukan adalah agar paket dapat dibaca atau diketahui oleh manusia (*human readable*) dan memungkinkan sturktur elemen jaringan dalam membuat pilihan.



Gambar 2.9 Perubahan label berbasis nama pada paket

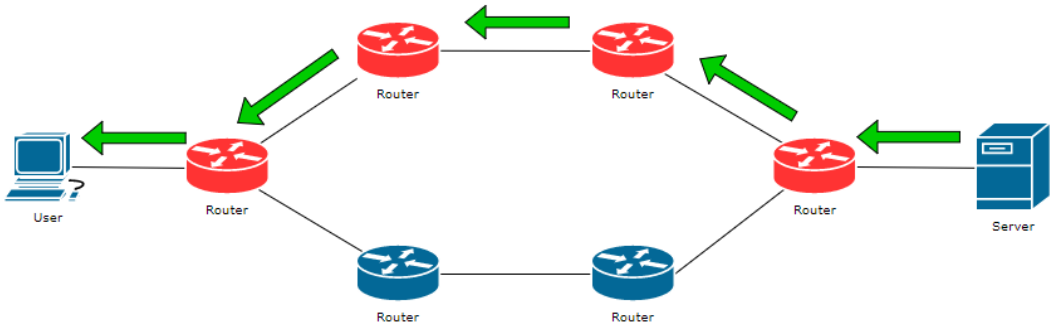
2.2.3 *In-Network Caching*

In-network caching adalah suatu mekanisme penyimpanan data *cache* yang ada di dalam arsitektur *centric network*. *In-network caching* ini memiliki tujuan meningkatkan efisiensi jaringan dan kinerja dari distribusi konten dengan menyimpan *cache* dari konten tersebut kedalam penyimpanan *cache* (Yuemei Xu, 2016). Tidak hanya itu, penyimpanan *cache* ini juga bertujuan untuk mengurangi trafik dan latensi yang tinggi akibat permintaan data yang sama dari pengguna yang berbeda. Berbeda dengan penyimpanan *cache* pada *Content Delivery Network* (CDN) yang apabila *cache* disimpan pada sebuah server dan jika letak dari server berdekatan dengan pengguna, maka akan lebih cepat untuk pengiriman datanya (Kim & Yeom, 2013). Pada *in-network caching*, apabila *cache* tersebut didapatkan dari sebuah *node*, maka disebut dengan *Cache Hit*, dan jika *cache* tersebut tidak dididapatkan maka disebut dengan *Cache Miss*. *In-network caching* dibagi menjadi dua jenis strategi yaitu *On-path caching* dan *Off-path caching*.

On-path caching adalah strategi dimana konten yang disimpan atau yang akan dicari masih dalam jalur yang dilalui oleh paket data. Di dalam mekanisme *on-path caching* terdapat beberapa jenis strategi yang dapat digunakan yaitu:

1. *Leave Copy Everywhere* (LCE)

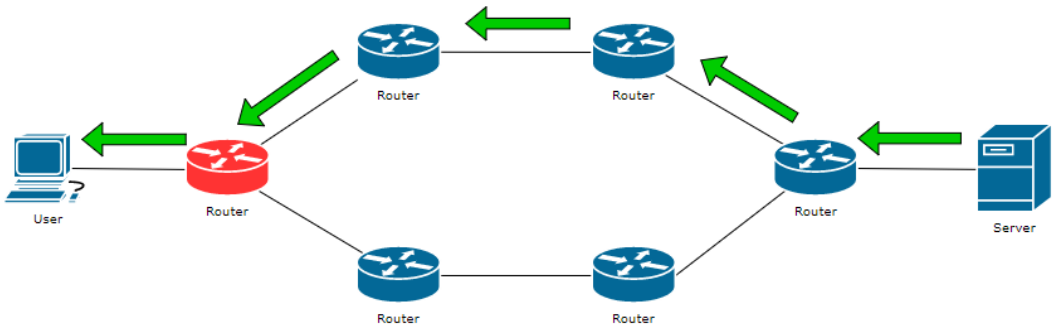
Pada Gambar 2.10, strategi LCE akan menyimpan replika konten pada setiap *node* yang dilalui oleh paket data menuju pengguna. Strategi ini sangat cocok digunakan pada jaringan dengan trafik tinggi.



Gambar 2.10 Penyimpanan *cache* pada LCE

2. *Leave Copy Down* (LCD)

Gambar 2.11 adalah proses penyimpanan *cache* yang dilakukan LCD. Setiap terjadi *cache hit*, maka konten akan direplikasi pada *node* satu level di bawahnya dalam *cache* hirarki. Dengan kata lain, semakin popular konten tersebut maka *cache* tersebut akan disimpan sedekat mungkin di *cache store*.



Gambar 2.11 Penyimpanan *cache* pada LCD

3. *Random*

Konten akan disimpan pada satu *node* yang dipilih secara acak di jalur pengirimannya.

4. *ProbCache*

Strategi ini akan menyimpan dari replika konten pada satu *node* terbaik disepanjang jalur pengiriman berdasarkan perhitungan probabilitas dari beberapa parameter.

Off-path caching atau nama lain dari *longest path* adalah mekanisme dimana konten yang disimpan atau dicari tidak berada di dalam jalur yang dilalui oleh paket data tetapi berdasarkan aturan yang telah ditentukan (Saucez, et al., 2012).

Di dalam *in-network caching* juga terdapat model penghapusan *cache* atau disebut juga dengan *cache replacement.* Ketika c*ache* yang sudah penuh harus menghapus beberapa kontennya agar dapat menampung konten-konten baru. Berikut adalah beberapa strategi *content replacement*:

1. *Least Recently Used* (LRU)

   Strategi LRU menghapus konten yang paling lawas dipakai agar konten baru dapat disimpan. Dengan demikian, konten yang baru-baru ini dipakai terhindar dari penghapusan.

2. *Least Frequently Used* (LFU)

   Strategi LFU menghitung berapa kali sebuah konten dipakai. Konten dengan jumlah pemakaian paling sedikit akan dihapus dari *cache*, sehingga menyisakan konten dengan tingkat pemakaian yang tinggi.

3. *First In First Out* (FIFO)

   Pada saat ada konten baru datang, strategi FIFO akan menghapus konten di dalam *cache* yang paling pertama masuk *cache* di antara konten yang lain, seperti sebuah antrian.

4. Random

   Strategi Random akan menghapus konten di dalam *cache* secara acak saat konten baru datang. Strategi ini tidak melihat apakah konten tersebut paling sering digunakan atau paling jarang digunakan.

2.2.4 CCN-Lite

Dalam penelitian ini, CCN-Lite digunakan sebagai ruang kerja berdasarkan dari implementasi CCN. CCN-Lite adalah sebuah *workspace* implementasi protokol jaringan dari *Content Centric Networking* yang bertujuan untuk mengurangi pengembangan yang memiliki kompleksitas tinggi dengan menyediakan implementasi *prototype* yang ringan pada protokol baik CCNx dan NDN, dimana dapat dilakukan pada lingkungan emulasi dan simulasi. CCN-Lite dikembangkan dan diurus oleh grup jaringan komputer dari *Mathematics and Computer Science Dept* di *University of Basel* (CCN-Lite, 2011).

CCN-Lite terdiri dari sekitar 2000 baris kode C, dimana interoperabilitas antara forwarder CCNx yang lengkap dan *node* pada CCN-lite dijamin. CCN-lite juga mendukung protokol CCNx dan NDN. Namun, karena kesederhanaan simulator ini, hanya komponen utama yang direproduksi, seperti ccnb dan TLV *encoding variants*, dasar struktur data CCN, *longest prefix matching*, dll. Pada saat yang sama, aspek-aspek lain yang tidak tercakup, seperti fungsi kripto, filter pengecualian, konektivitas TCP, *Server SYNC*, dan sebagainya (CCN-Lite, 2011).

Sebagai patokan interoperabilitas, CCN-lite mendukung paket fragmentasi dan mendeteksi paket hilang saat menjalankan protokol CCNx melewati diatas *Ethernet*. Selain itu, kode CCN-lite adalah bersifat *portable*, sama seperti kode yang berjalan tidak berubah pada pengguna dan *kernel space* dari kedua arsitektur x86 dan ARM, atau pada OMNeT ++ simulasi *Platform* (menggunakan INET Framework). Namun di dalam beberapa kasus pada CCN-Lite, hanya struktur yang sangat dasar disediakan, sementara semua bagian sisanya perlu dilaksanakan dari awal, seperti management *Content Store*, *application layer*, dan lain sebagainya (Tortelli, et al., 2016).

2.2.5 OMNeT++

Dalam penelitian ini, implementasi CCN dilakukan menggunakan simulator. Simulator yang digunakan adalah OMNeT++. Adapun alasan menggunakan simulator OMNeT++ karena CCN-Lite dapat dijalankan dengan simulasi dan CCN-Lite juga menyediakan *workspace* yang dapat dilakukan dengan bantuan simulator OMNeT++. Simulator OMNeT++ adalah sebuah *framework* simulasi *discrete-event* yang bertipe *object-oriented* yang dibuat oleh Andras Varga dari *Technical University of Budapest, Department of Telecommunications* (BME-HIT) (OMNET, 2016). OMNeT++ memiliki sebuah arsitektur yang umum, sehingga OMNeT++ dapat digunakan dalam banyak bidang permasalahan jaringan, antara lain:

- Permodelan dari jaringan komunikasi dengan kabel dan nirkabel.
- Permodelan protokol.
- Permodelan dan *queueing network*.
- Permodelan dari *multiprocessors* dan *system* perangkat keras terdistribusi lainnya.
- Memvalidasi arsitektur dari perangkat keras.
- Mengevaluasi performa dari perangkat lunak.

Bahasa pemrograman simulasi untuk OMNeT++ menggunakan bahasa NED, yaitu bahasa pemrograman yang digunakan untuk mendeskripsikan topologi jaringan. Dengan menggunakan NED, deskripsi jaringan dapat berisikan komponen penyusun jaringan yang bersifat moduler. Untuk pengembangan model dilakukan dengan objek dan bahasa pemrograman C++. Pada OMNeT++ dimungkinkan juga penggunaan bahasa Java, namun untuk bahasa ini harus menggunakan komponen tambahan (OMNET, 2016).

OMNeT++ juga menyediakan infrastruktur dan *tools* untuk memprogram simulasi sendiri. Pemrograman OMNeT++ bersifat object-oriented dan bersifat hirarki. Objek-objek yang besar dibuat dengan cara menyusun objek-objek yang lebih kecil. Objek yang paling kecil disebut *simple module*, akan memutuskan algoritma yang akan digunakan dalam simulasi tersebut. Berbagai tipe objek pada OMNeT++ adalah sebagai berikut:

1. *Module* (*Simple Module and Compound Module*) adalah objek yang telah dibuat, diprogram dan disusun. *Compound Module* adalah sebuah modul yang dibuat dengan cara menggabungkan beberapa *Simple Module*.
2. *Gate* adalah pintu keluar / masuk nya *message*. Setiap *module* hanya bisa berinteraksi dengan *module* lainnya melalui *gate*.
3. *Message* adalah komunikasi yang dilakukan antar *module*. *Message* ini adalah konsep inti dari simulasi OMNeT++. Sebuah *module* bisa mengirimkan *message* pada *module* lain atau dirinya sendiri (*self message*).
4. *Connection* adalah jalur tempat dimana message mengalir. Di dalam *connection dapat* mendefinisikan parameter atau variabel yang berkaitan dengan koneksi, misalnya hambatan udara, *datarate* dan lain sebagainya (OMNET, 2016).

2.2.6 INET Framework

INET Framework adalah sebuah *library* model *open-source* yang digunakan pada simulasi OMNeT++. Framework ini menyediakan protokol, agen dan model tentang jaringan komunikasi. INET sangat berguna untuk mendesain dan memvalidasi protokol baru, atau menemukan skenario yang baru (INET, 2015).

INET dibangun berdasar pada konsep modul berkomunikasi dengan *message passing*. Agen dan protokol jaringan diwakili oleh komponen di dalamnya, dimana dapat dengan bebas dikombinasikan dalam bentuk *host*, *router*, *switch*, dan perangkat jaringan lainnya (INET, 2015). Komponen baru dapat dikelola oleh pengguna, dan untuk komponen yang telah dibuat dapat dimofikasi dan mudah dimengerti oleh pengguna lain.

INET Framework memiliki banyak model untuk berbagai *library* pada Internet (TCP, UDP, IPv4, IPv6, dll), *wired*, dan *link layer* dengan protokol nirkabel (Ethernet, PPP, IEEE802.11, dll), mendukung untuk mobilitas, protokol MANET, DiffServ, MPLS dengan mensinyalkan LDP dan RSVP-TE, beberapa model aplikasi, dan banyak lagi protokol dan komponen lainnya.

2.2.7 *Cache Hit Ratio* (CHR)

Cache Hit Ratio adalah perbandingan jumlah permintaan yang berhasil dilayani (cache hit) oleh *cache router* dengan total jumlah permintaan yang dikirimkan. Persamaan untuk menghitung CHR adalah:

$$CHR = N(s)/N(T) \quad (2.1)$$

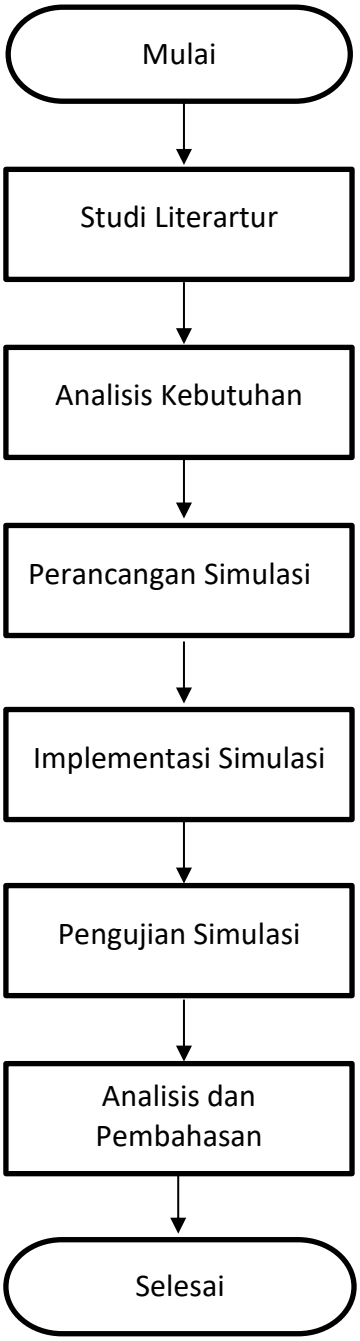
N(s) adalah jumlah permintaan berhasil dilayani (*cache hit*) dan N(T) adalah total permintaan yang dikirimkan. Nilai dari CHR berkisar antara 0 dan 1, dengan 0 berarti tidak ada permintaan yang berhasil dilayani *cache* dan 1 berarti semua permintaan berhasil dilayani oleh *cache*.

2.2.8 Latensi Pengiriman

Latensi pengiriman di dalam CCN-Lite adalah rata-rata waktu yang dibutuhkan suatu *node* untuk mendapatkan sebuah konten. Terhitung dari *chunk* pertama dikirimkan oleh *client* sampai *chunk* terakhir diterima oleh *client* tersebut sesuai dengan jumlah chunk yang diminta.

BAB 3 METODOLOGI

Pada bagian ini menjelaskan langkah-langkah yang dilakukan dalam pengerjaan skripsi yaitu studi literatur, penyusunan dasar teori, analisis kebutuhan dan perancangan simulasi, implementasi dan pengujian, serta analisis dan pembahasan dari simulasi yang telah dilakukan. Kesimpulan dan saran disertakan sebagai cacatan hasil dari penelitian ini dan kemungkinan pengembangan penelitian selanjutnya. Diagram alir dari pengerjaan penelitian ini disajikan seperti pada Gambar 3.1.



Gambar 3.1 Diagram alir metodologi penelitian

3.1 Studi Literatur

Studi literatur yaitu mempelajari mengenai penjelasan dasar teori yang digunakan untuk menunjang penelitian ini. Dasar-dasar teori tersebut diperoleh dari buku, jurnal, *e-book*, dan dokumentasi *project*. Dasar-dasar teori yang digunakan pada penelitian ini, antara lain:

1. *Content Centric Networking* (CCN)

   Penjelasan dasar mengenai arsitektur jaringan dimana data yang dikirimkan berupa sebuah konten.

2. Protokol CCNx

   Penjelasan dasar mengenai protokol jaringan pada arsitektur CCN dan perubahan yang terjadi pada protokol tersebut.

3. *In-Network Caching*

   Penjelasan mengenai bagaimana mekanisme penyimpan *cache* yang terdapat di dalam arsitektur *Centric Network.*

4. CCN-Lite

   Penjelasan mengenai *tools* yang bertujuan untuk mengurangi pengembangan yang memiliki kompleksitas tinggi dengan menyediakan implementasi *prototype* yang ringan pada protokol.

5. OMNeT++

   Penjelasan dasar mengenai simulator dapat digunakan dalam bidang permasalahan jaringan.

6. INET Framework

   Penjelasan mengenai sebuah *library* model *open-source* yang digunakan pada simulator OMNeT++.

7. *Cache Hit Ratio* (CHR)

   Penjelasan mengenai bagaimana perhitungan *cache hit ratio* yang ada di dalam lingkungan *Centric Networking*.

8. Latensi Pengiriman

   Penjelasan mengenai definisi latensi yang digunakan di dalam simulasi

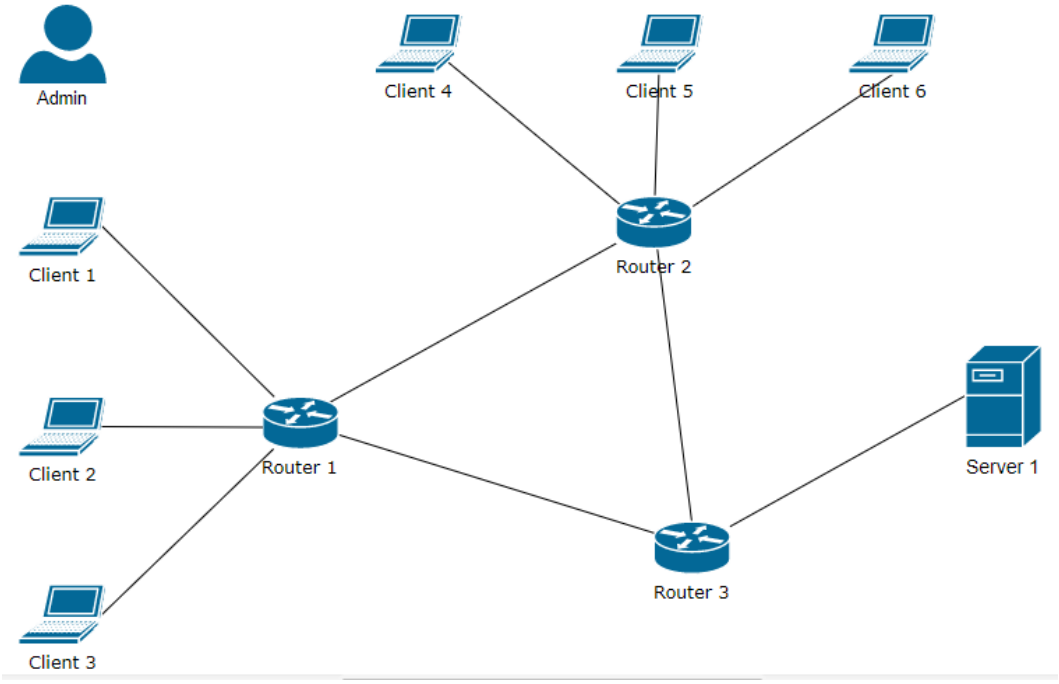
3.2 Analisis Kebutuhan Simulasi

Analisis kebutuhan diperlukan untuk mengetahui apa saja yang dibutuhkan dalam membangun lingkungan simulasi dalam penelitian ini. Dalam simulasi ini, perlu adanya topologi jaringan agar pengujian *in-network caching* dapat dilakukan. Topologi ini harus memiliki *node* yang berperan sebagai penyedia konten asli atau *content source*, *cache router* yang meneruskan paket baik paket *interest* ataupun paket data dan dapat menyimpan konten sementara, *requester* yang berperan sebagai peminta dan penerima konten serta, *node* yang

menginisialisasi *node-node* yang ada pada topologi agar dapat berjalan sesuai dengan skenario yang telah diberikan. Skenario permintaan simulasi juga diperlukan untuk tercapainya tujuan dari penelitian ini. Skenario permintaan harus terdiri dari waktu dimulai permintaan, nama konten yang diminta, berapa jumlah chunk yang diminta, dan siapa yang meminta.

Dari analisis kebutuhan simulasi yang telah dituliskan, dibutuhkan identifikasi terhadap desain topologi simulasi, perangkat lunak yang digunakan, dan perangkat keras yang digunakan.

3.2.1 Desain Topologi Simulasi



Gambar 3.2 Desain topologi pengujian

Gambar 3.2 adalah desain dari topologi simulasi yang digunakan. Topologi ini dibangun dengan tujuan untuk mengetahui proses pengiriman data dari 2 jalur yang berbeda. Topologi ini terdiri atas 1 *server*, 3 *router*, dan 6 *client*. Agar topologi ini dapat berjalan, pengaturan *cache router, content source,* dan *requester* diatur di dalam CCN-Lite. Model aliran data yang digunakan adalah *client-server*. Admin akan melakukan proses inisialisasi kepada semua node yang ada pada topologi. Ketika proses selesai, maka admin akan mengirimkan perintah kepada client agar client segera melakukan permintaan konten. Paket interest dikirimkan menuju router terdekat. Router akan memeriksa apakah konten yang diminta terdapat pada *Content Store*, jika ada maka konten akan dikirimkan kembali menuju client. Apabila tidak ditemukan maka paket akan diteruskan menuju router selanjutnya hingga menuju kepada server.

3.2.2 Perangkat Lunak yang Digunakan

Perangkat lunak utama yang digunakan dalam penelitian ini adalah sebagai berikut:

Sistem Operasi : Windows 10 Enterprise 64-bit.

Virtual Machine : VMware Workstation 12.0.

Virtual OS : Ubuntu 14.04.

Simulator dan Library : OMNeT++ 4.5, CCN-Lite v.0.3, INET Framework 2.4.

3.2.3 Perangkat Keras yang Digunakan

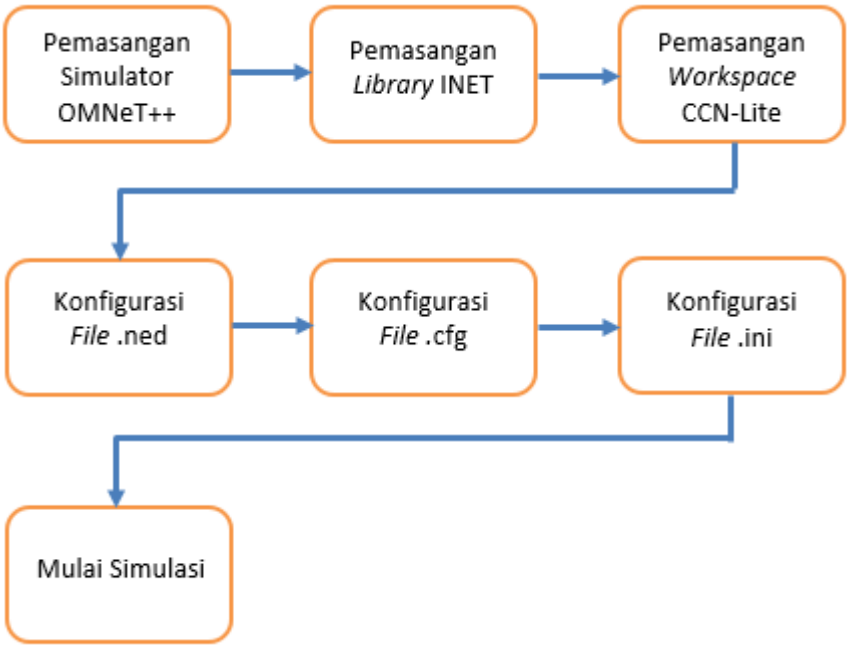
Agar perangkat lunak yang digunakan untuk pengujian dapat berjalan dengan baik, maka pengujian ini dilakukan pada sebuah laptop dengan spesifikasi:

Processor : Intel Core i5-4210U CPU @ 1.70GHz 2.40GHz.

RAM : 8 GB DDR3L.

Hard Drive : 750 GB dan 240GB (SSD).

3.3 Persiapan Simulasi



Gambar 3.3 Step diagram implementasi

Agar simulasi dapat dijalankan, maka perlu dilakukan instalasi perangkat lunak yang digunakan dalam penelitian. Adapun perangkat lunak yang disediakan adalah OMNeT++ sebagai simulator, CCN-Lite sebagai *workspace* dari *Content Centric Networking*, dan INET Framework sebagai *library* yang di *import* kedalam OMNeT++. Langkah-langkah untuk melakukan implementasi digambarkan pada Gambar 3.3. Langkah pertama hingga langkah ketiga bertujuan agar implementasi *in-network caching* dapat dilakukan. Langkah keempat bertujuan untuk

melakukan perancangan topologi, mulai dari jumlah *client, router*, dan server, mengatur latensi dan *datarate,* serta mengatur hubungan antar *node*. Langkah kelima bertujuan untuk mengatur skenario pada setiap *node.* Langkah keenam bertujuan agar *file* .cfg yang telah dikonfigurasi dapat dieksekusi pada simulasi.

3.4 Implementasi dan Pengujian

Pada tahap ini dilakukan implementasi berdasarkan perancangan yang telah dilakukan. Implementasi ini berupa melakukan perancangan topologi pengujian yang ada di dalam CCN-Lite pada *file* .ned, mengkonfigurasi setiap *node* yang ada pada topologi dalam *file* .cfg, serta melakukan konfigurasi pada *file* omnetpp.ini untuk agar file konfigurasi dapat dieksekusi pada simulasi.

Setelah tahap implementasi selesai, dilakukan pengujian untuk mengetahui bagaimana implementasi dan kinerja dari *in-network caching* pada CCN-Lite di lingkungan OMNeT++. Pengujian dilakukan dengan menjalankan simulasi CCN pada OMNeT++ berdasarkan *workspace* dari CCN-Lite. Pengujian dilakukan untuk mendapatkan hasil latensi dari awal pengiriman *chunk* hingga konten didapatkan oleh *client* dan untuk mendapatkan nilai CHR pada *client*. Agar tujuan tersebut tercapai maka, dilakukan penjadwalan waktu pengiriman permintaan seperti pada Tabel 3.1.

Dalam Tabel 3.1, waktu permintaan dilakukan sebanyak 6 kali. Tipe file protokol yang digunakan ada 2 yaitu CCNB (movie1) atau protokol CCNx 0.x dan CCNTLV (movie2) atau protokol CCNx 1.x. Pada setiap waktu permintaan, dilakukan oleh 2 client yang berbeda. Data permintaan yang dilakukan berupa chunk. Dari setiap waktu permintaan yang dilakukan memiliki tujuan tersendiri.

Table 3.1 Jadwal waktu pengiriman permintaan

Waktu Permintaan dimulai (/s)	Tipe File Permintaan	Nama Client Yang Meminta	Jumlah Chunk Permintaan
1	CCNB (movie1)	*Client 1 & Client 4*	100 & 100
2	CCNTLV (movie2)	*Client 2 & Client 5*	75 & 75
3	CCNB (movie1)	*Client 3 & Client 6*	75 & 60
4	CCNTLV (movie2)	*Client 1 & Client 6*	115 & 115
5	CCNB (movie1)	*Client 2 & Client 5*	60 & 75
6	CCNTLV (movie2)	*Client 3 & Client 4*	60 & 60

Waktu ke-1 dilakukan dengan meminta *file* CCNB oleh *Client 1* dan *Client 4* dengan jumlah *chunk* permintaan setiap *client* adalah 100 atau *file* utuh. Tujuan dari permintaan ini adalah untuk mengetahui apakah permintaan dari kedua *client* akan dilayani atau tidak, apakah terdapat *caching* pada permintaan pertama kali, siapa yang melayani data yang diminta.

Waktu ke-2 dilakukan dengan meminta *file* CCNTLV oleh Client 2 dan 5 dengan jumlah *chunk* permintaan setiap *client* 75. Tujuan dari permintaan ini sama dengan waktu ke-1 dan apakah permintaan ini akan dilayani apabila jumlah chunk yang diminta hanya sebagian dari total *chunk* pada *file* tersebut.

Pada waktu ke-3 dilakukan dengan meminta *file* CCNB oleh *Client 3* dan 6 namun jumlah *chunk* yang diminta berbeda yaitu 75 & 60. Tujuan pada waktu ini adalah untuk mengetahui apakah permintaan akan dilayani ketika kedua *client* meminta dengan jumlah *chunk* yang berbeda, apakah terjadi *caching*, siapa yang melayani permintaan data.

Waktu ke-4 dilakukan dengan meminta *file* CCNTLV oleh *Client 1* dan Client 6 dengan jumlah *chunk* permintaan setiap *client* yaitu 115 atau *file* utuh. Tujuan dari pengiriman ini adalah untuk mengetahui apakah permintaan ini dilayani atau tidak, apakah terjadi *caching* atau tidak, siapa yang melayani permintaan tersebut.

Waktu ke-5 dilakukan dengan meminta *file* CCNB oleh *Client 2* dan Client 5 dengan jumlah chunk yang berbeda setiap *client* yaitu 60 dan 75. Tujuan dari pengiriman sama dengan tujuan dari waktu permintaan ke-3.

Waktu ke-6 dilakukan dengan meminta file CCNTLV oleh client 3 dan Client 4 dengan jumlah *chunk* permintaan setiap *client* adalah 60. Tujuan dari pengiriman sama dengan tujuan dari waktu permintaan ke-5.

3.5 Analisis Hasil Pengujian

Berdasarkan simulasi yang telah dijalankan, dilakukan analisis terhadap kinerja *in-network caching.* Analisis dilakukan dengan menghitung rata-rata pada latensi pengiriman mulai dari *chunk* pertama dikirimkan hingga *chunk* terakhir diterima oleh *client* yang meminta dan menghitung *Cache Hit Ratio* setiap *client* pada *file* permintaan yang berbeda. Sehingga dari analisis tersebut dapat diketahui bagaimana kinerja *in-network caching* yang ada pada CCN-Lite berdasarkan arsitektur CCN.

3.6 Penarikan Kesimpulan dan Saran

Berdasarkan dari terbentuknya latar belakang, rumusan masalah yang didapat, tujuan dari penelitian, dasar teori yang digunakan, perancangan dan implementasi simulasi, serta pengujian simulasi dan analisis hasil yang telah dilakukan, maka diambil kesimpulan. Isi kesimpulan disusun berdasarkan hasil implementasi *in-network caching* pada arsitektur CCN menggunakan *workspace* CCN-Lite di lingkungan OMNeT++. Dari kesimpulan tersebut, didapatkan saran yang diharapkan dapat menjadi acuan untuk penelitian selanjutnya.

BAB 4 PERANCANGAN DAN IMPLEMENTASI SIMULASI

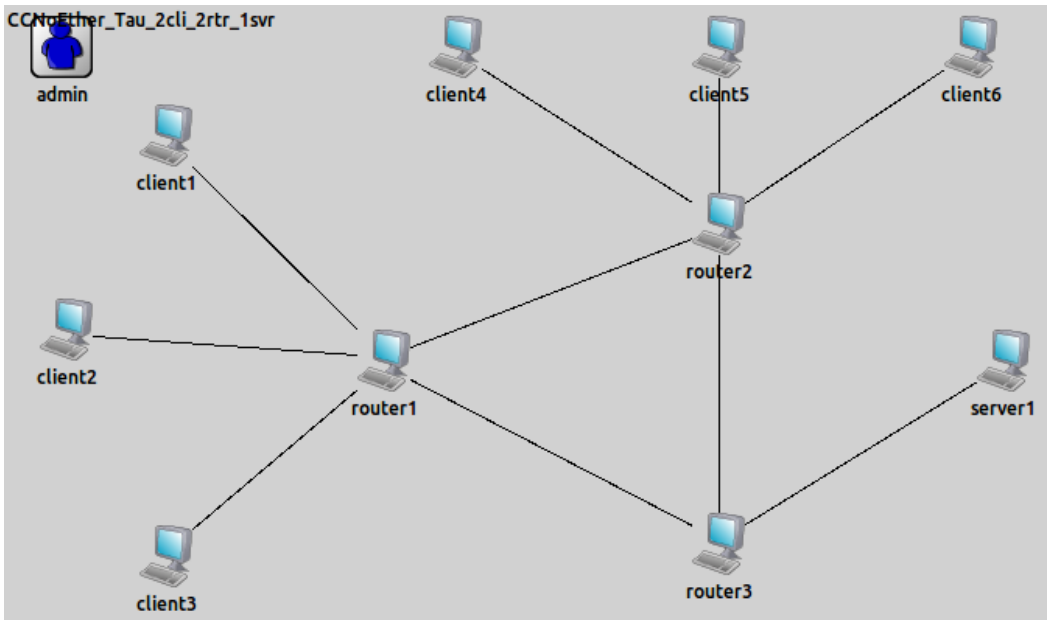
4.1 Perancangan

Pada bab ini menjelaskan perancangan yang dilakukan agar proses implementasi dapat dijalankan. Proses perancangan yang dilakukan adalah perancangan topologi, alur dari simulasi dan, spesifikasi simulasi. Setelah proses perancangan selesai dilakukan, maka dilanjutkan dengan proses implementasi berdasarkan dari perancangan yang telah dilakukan.

4.1.1 Perancangan Topologi Pengujian

4.1.1.1 Node

Agar simulasi dapat dilakukan, perlu adanya kebutuhan *node* yang ditempatkan pada sebuah topologi. Adapun kebutuhan *node* pada topologi simulasi yang digunakan adalah 1 *server*, 3 *router* dan, 6 *client*. Pemberian 1 server dilakukan dengan alasan sebagai penyedia konten asli dan sebagai persyaratan minimal dalam sebuah simulasi jaringan. Pemberian 3 router dilakukan dengan alasan untuk mengetahui bagaimana proses *caching* terjadi dan mengetahui mekanisme *caching* yang digunakan. Pemberian 6 client dengan menempatkan 3 client pada bagian yang berbeda dilakukan dengan alasan untuk mengetahui bagaimana dan dari mana konten didapat. Penempatan setiap *node* ditunjukkan seperti pada Gambar 4.1.



Gambar 4.1 Topologi simulasi

4.1.1.2 Links

Berdasarkan pada Gambar 4.1, setiap *node* yang terdapat pada topologi simulasi juga memiliki peran masing-masing. Penjelasan peran ditunjukkan pada Tabel 4.1. *Node* yang bertugas untuk menginisialisasi *node-node* yang lain adalah

admin. Adapun alasan admin tidak terhubung dengan node yang lain karena admin di dalam simulasi dianggap sebagai manusia yang memberikan perintah terhadap *node-node* yang lain. *Node* yang beperan sebagai *content source* untuk menyimpan konten asli adalah server. Node yang berperan sebagai *cache router* untuk menyimpan data konten sementara dan meneruskan paket baik paket *Interest* maupun Data adalah *router*. 3 *router* ditempatkan dan dihubungkan dalam bentuk segitiga. Penempatan *router* ini diharapkan agar strategi *caching* dapat terlihat dan terjadi dan ketika terjadi proses *cache replacement* pada *router* awal, permintaan konten tidak langsung menuju *content source* untuk melainkan menuju *router* selanjutnya. Node terakhir yang berperan sebagai peminta dan penerima konten yaitu *client*. 6 *client* dibagi menjadi 2 bagian yaitu 3 *client* pada *router* 1 dan 3 *client* pada *router* 2. Penempatan ini diharapkan agar proses *caching* dapat terjadi ketika permintaan dilakukan oleh *client* berbeda lokasi dan jalur awal permintaan. *Router 1* terhubung dengan router 2 dan router

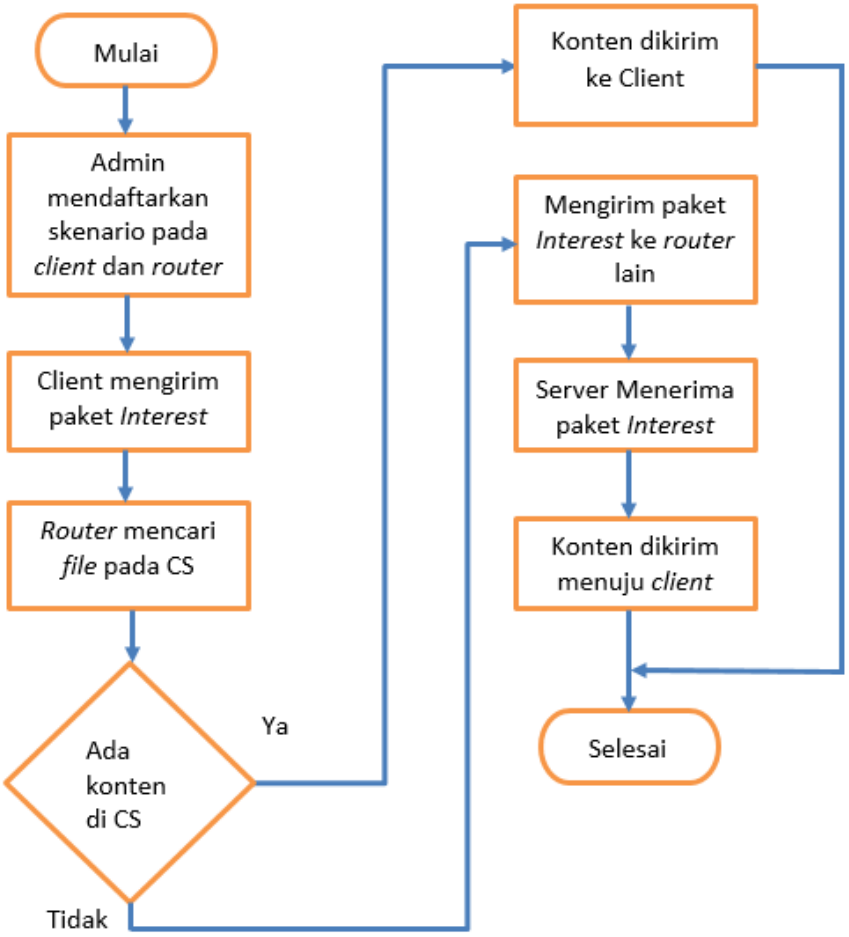
Tabel 4.1 Penjelasan node pada topologi simulasi

Admin	Sebuah *node* yang meng-inisialisasikan *client*, *router* dan server untuk mengirimkan paket *Interest* atau paket Data, meneruskan paket hingga mendaftarkan FIB pada setiap *node*.
Client	Sebuah *node* yang mengirimkan paket *Interest*.
Router	*Node* yang akan meneruskan permintaan paket *Interest* dan paket Data baik dari *client* dan server.
Server	*Node* yang menyimpan konten asli.

4.1.2 Alur Simulasi

Gambar 4.2 adalah diagram alir simulasi yang dilakukan. Proses ini menggambarkan bagaimana inisialisasi awal yang dilakukan oleh oleh admin terhadap *client, router,* dan *server*, bagaimana proses permintaan konten hingga proses pengiriman konten yang dilakukan baik oleh *server* maupun *router* sebagai *cache router*.

Pertama kali admin melakukan perdaftaran skenario yang berisikan nama konten yang diminta, konten yang dilalui dan tipe konten yang disimpan. Ketika telah mendaftarkan pada semua *node*, admin mengirimkan sebuah perintah pada *client* untuk meminta *file* sesuai dengan waktu yang telah ditentukan. *Client* mengirimkan paket *Interest* menuju *router.* Di dalam *router*, paket *Interest* diperiksa apakah konten yang diminta terdapat di dalam *Content Store* (CS), jika ada maka konten dikirimkan kembali menuju *client* yang meminta, apabila tidak maka paket *Interest* dikirimkan menuju *router* selanjutnya dan memeriksa apakah ada konten yang diinginkan hingga menuju ke server. Ketika server telah menerima paket *Interest*, maka server mengirimkan konten menuju *client* yang meminta melalui *router* yang dilalui sejak awal pengiriman paket.



Gambar 4.2 Diagram alir simulasi

4.1.3 Spesifikasi Simulasi

Dalam hal ini, perlu adanya pula spesifikasi simulasi yang dilakukan agar simulasi dapat dijalankan. Topologi yang digunakan adalah topologi 1 server, 3 router, 6 client sesuai dengan topologi pada Gambar 4.1. Data yang diujikan adalah latensi pengiriman dan *Cache Hit Ratio* (CHR). Dalam simulasi ini, konten yang diminta sebanyak 2 file untuk setiap *client,* dengan asumsi bahwa semua *client* dapat menerima semua konten yang diminta. Dua *file* tersebut yaitu CCNB movie1 dan CCNTLV movie2 dengan total *chunk* setiap *file* yaitu 100 dan 115. *Datarate* yang digunakan sebesar 10Mbps. Nilai ini digunakan karena nilai ini adalah nilai terkecil yang dapat digunakan dalam simulasi. Delay atau latensi diberikan sebesar 5000 µs atau 5ms.

Table 4.2 Spesifikasi simulasi

Parameter	Nilai
Topologi	6 *Client*, 3 *Router*, 1 *Server*
Data Yang Diuji	Latensi pengiriman dan *Cache Hit Ratio* (CHR)
Delay	5000 µs atau 5 ms

Datarate	10 Mbps
Max Cache Bytes	524288000 Bytes
Max Cache Slot	100
Jumlah Skenario Pengujian	6 kali permintaan
Tipe File Konten	CCNB dan CCNTLV
Penghubung Antar *Node*	*Fast Ethernet*

Max cache bytes adalah jumlah kapasitas *cache* yang dapat ditampung oleh setiap *node.* Nilai yang digunakan adalah sebesar lebih kurang 54 MB. Nilai ini adalah nilai *default* dari CCN-Lite. *Max cache slot* adalah jumlah slot *cache* yang dapat disimpan pada *cache router.* Dalam simulasi ini, slot *cache* yang digunakan adalah sebesar 100 slot dengan asumsi bahwa ketika *cache* penuh maka akan terjadi *cache replacement.* Spesifikasi simulasi tersebut ditunjukkan pada Tabel 4.2.

4.2 Implementasi Perangkat Lunak Simulasi

Pada bab ini menjelaskan apa saja yang dibutuhkan dalam instalasi CCN-Lite, apa hubungan dari CCN-Lite dengan OMNeT++ dan INET Framework serta bagaimana implementasi dari CCN-Lite pada OMNeT++. Sebelum menjalankan simulasi ini, terdapat 3 perangkat lunak yang perlu disiapkan yaitu CCN-Lite v.3.0, OMNeT++ v.4.5 dan INET Framework v.2.4. Perangkat lunak ini memiliki tugas masing-masing, seperti CCN-Lite sebagai *workspace* dari implementasi CCN, OMNeT++ sebagai simulator, dan INET Framework sebagai *library* dari OMNeT++. Dalam simulasi ini, CCN-Lite sangat bergantung pada INET karena bagian jaringan seperti *client, router, server* dan admin diambil dari *library* ini.

Instalasi pertama yaitu simulator OMNeT++ versi 4.5. Aplikasi ini dapat di unduh pada situs http://omnetpp.org dengan mengikuti proses instalasi seperti pada bagian lampiran. Selanjutnya adalah INET *Framework* versi 2.4 yang dapat di unduh di situs http://inet.omnetpp.org/Download dengan mengikuti proses *import* seperti pada bagian lampiran. Instalasi ketiga adalah CCN-Lite v.3.0 yang dapat diunduh pada situs https://github.com/cn-uofbasel/ccn-lite. Proses instalasi CCN-Lite dapat dilakukan seperti pada bagian lampiran.

4.2.1 Konfigurasi Simulasi

Pada bagian ini dilakukan perancangan topologi simulasi pada *file* .ned, mengkonfigurasi *client, router* dan *server* pada *file* .cfg serta mengkonfigurasi *file* omnetpp.ini agar *file* .cfg yang telah diatur dapat dieksekusi disaat simulasi berlangsung. Pengaturan *file* tersebut dilakukan pada *workspace* CCN-Lite yang berada di dalam simulator OMNeT++.

4.2.2 Konfigurasi Topologi Simulasi

Dalam melakukan implementasi topologi jaringan ini tidak hanya dengan menempatkan *node*, tetapi juga mengatur konfigurasi agar setiap n*ode* dapat terhubung mengacu pada perancangan topologi yang telah dibuat. Para Tabel 4.3, penghubung antar *node* yang digunakan adalah *Fast Ethernet* dengan *delay* 5 ms dan *datarate* 10 Mbps.

Table 4.3 Potongan kode pengaturan *channel, delay,* dan *datarate*

	CCNoEther_Tau_6cli_3rtr_1svr.ned
	.....
13	types:
14	channel fastEthernet extends DatarateChannel
15	{
15	delay = 5000us; //delay = 5ms;
17	datarate = 10Mbps;
18	}
	.....

Pada Tabel 4.4 dibawah adalah bagian dari konfigurasi topologi node. *Display* menunjukkan lokasi dari *node* tersebut berada. Untuk *Ethg* berfungi sebagai berapa jumlah *node* yang terhubung pada satu *node*.

Table 4.4 Potongan kode pengaturan letak *node* dan *port ethernet*

	CCNoEther_Tau_6cli_3rtr_1svr.ned
	.....
20	submodules:
21	
22	admin: CcnAdmin {
23	@display("p=38,28");
24	}
25	client1: CcnMacNode {
26	parameters:
27	@display("p=105,85");
28	gates:
29	ethg[1];
30	}
31	client2: CcnMacNode {
32	parameters:
33	@display("p=41,210");
34	gates:
35	ethg[1];
36	}
37	client3: CcnMacNode {
38	parameters:
39	@display("p=105,356");
40	gates:
41	ethg[1];
42	}
43	router1: CcnMacNode {
44	parameters:
45	@display("p=245,230");
46	gates:
47	ethg[5];

```
48          }
49          router2: CcnMacNode {
50              parameters:
51                    @display("p=460,142");
52              gates:
53                  ethg[5];
54          }
55          router3: CcnMacNode {
56              parameters:
57                    @display("p=460,348");
58              gates:
59                  ethg[3];
60          }
61          server1: CcnMacNode {
62              parameters:
63                    @display("p=643,230");
64              gates:
65                  ethg[1];
66          }
67          client4: CcnMacNode {
68              parameters:
69                    @display("p=291,28");
70              gates:
71                  ethg[1];
72          }
73          client5: CcnMacNode {
74              parameters:
75                    @display("p=460,28");
76              gates:
77                  ethg[1];
78          }
79          client6: CcnMacNode {
80              parameters:
81                    @display("p=622,28");
82              gates:
83                  ethg[1];
84          }
    .....
```

Pada Tabel 4.5, agar dapat saling terhubung membutuhkan konfigurasi pengaturan jalur untuk setiap *node*. *Client* yang terhubung dengan *router* dituliskan dengan *port* yang disambungkan. Tujuannya agar pengiriman data dapat dilakukan saat simulasi dijalankan.

Tabel 4.5 Potongan kode pengaturan hubungan antar *node*

```
    CCNoEther_Tau_6cli_3rtr_1svr.ned
    .....
85  connections:
86
87      client1.ethg[0] <--> fastEthernet <--> router1.ethg[0];
88      client2.ethg[0] <--> fastEthernet <--> router1.ethg[1];
89      client3.ethg[0] <--> fastEthernet <--> router1.ethg[2];
90      client4.ethg[0] <--> fastEthernet <--> router2.ethg[0];
91      client5.ethg[0] <--> fastEthernet <--> router2.ethg[1];
92      client6.ethg[0] <--> fastEthernet <--> router2.ethg[2];
93      router1.ethg[3] <--> fastEthernet <--> router3.ethg[0];
```

```
94      router3.ethg[1] <--> fastEthernet <--> server1.ethg[0];
95      router2.ethg[3] <--> fastEthernet <--> router3.ethg[2];
96      router2.ethg[4] <--> fastEthernet <--> router1.ethg[4];
    .....
```

4.2.3 Implementasi Mekanisme *In-Network Caching* Pada *Node*

Dalam melakukan implementasi *in-network caching*, diperlukan konfigurasi simulasi pada *client*, *router*, *server* dan *file* omnetpp.ini yang berada di dalam *workspace* CCN-Lite. Pengaturan perilaku dari setiap *node* diatur pada *file* .cfg. Konfigurasi ini dilakukan untuk mengatur *file* apa yang akan diminta oleh *client*, kemudian penentuan jalur pengiriman paket dan *file* apa saja yang berada di dalam *server* serta penempatan *file* konfigurasi tiap *node* pada omnetpp.ini.

4.2.3.1 Konfigurasi Client, Router dan, Server

Konfigurasi awal dimulai dengan melakukan konfigurasi pada *client* dengan jumlah 6 *node*, *router* yang berjumlah 3 *node*, dan 1 *server*. Konfigurasi ini dilakukan pada tipe *file* .cfg yang ada di dalam *workspace* CCN-Lite. Berikut adalah penjelasan dari kode di dalam *file* .cfg:

- *eInterestMode*: Mode yang digunakan untuk meminta *file* yang dinginkan. *ContentName* digunakan sebagai nama *file* yang akan diminta, *StartChunk* digunakan sebagai angka dimulainya permintaan *chunk*, *ChunksCount* digunakan sebagai jumlah *chunk* yang akan diminta, dan *RequestTime* digunakan sebagai waktu pengiriman paket *Interest* dimulai.
- *ePreCacheMode*: Mode yang digunakan untuk menyimpan *cache* di dalam server. Sebagian besar isi dari mode ini hampir sama dengan *eInterestMode*, hanya yang membedakan pada mode ini adalah adanya *UpdateTime* yang digunakan untuk meng-*update* isi dari konten yang akan diminta dan *ChunkCount* yang menandakan jumlah *chunk* dari setiap tipe *file*.
- *eFwdRulesMode*: M*ode* ini digunakan sebagai *Forwarding Information Base* (FIB). Isi dari mode ini yaitu *ContentPrefix* digunakan sebagai konten yang akan dikirimkan, *NextHop* digunakan sebagai arah tujuan selanjutnya dari konten yang akan dikirimkan, *AccessFrom* digunakan sebagai tempat saat konten berada di*router*, dan *UpdateTime.*
- *eCommentsMode*: Mode ini digunakan untuk memberikan komentar.

```
[eInterestMode]
ContentName = ccnb:/b3c/wowmom/movie1  , StartChunk = 0 , ChunksCount = 100 , RequestTime = 1/*s*/
ContentName = ndntlv:/b3c/wowmom/movie2  , StartChunk = 0 , ChunksCount = 16 , RequestTime = 2/*s*/

[ePreCacheMode]


[eFwdRulesMode]
ContentPrefix = ccnb:/b3c/wowmom , NextHop = router1.eth[0] , AccessFrom = client1.eth[0] , UpdateTime = 0/*s*/
ContentPrefix = ndntlv:/b3c/wowmom , NextHop = router1.eth[0] , AccessFrom = client1.eth[0] , UpdateTime = 0/*s*/

[eCommentsMode]
--------------------------------------------------------------------------------
comments go here
```

Gambar 4.3 Contoh bagian konfigurasi *client*

Pada konfigurasi setiap *client,* yang dibutuhkan oleh *client* adalah nama konten, waktu pengiriman paket *Interest*, jumlah dari *chunk* yang diminta dan kemana jalur dari paket ini akan dikirimkan. Oleh karena itu mode yang digunakan adalah *eInterestMode* dan *eFwdRulesMode.* Dalam *eInterestMode* harus terdapat *ContentName, StartChunk, ChunksCount,* dan *RequestTime.* Sedangkan di dalam *eFwdRulesMode* harus terdapat *ContentPrefix, NextHop, AccessFrom,* dan *UpdateTime* seperti yang ada pada Gambar 4.3.

Dalam simulasi ini digunakan 2 tipe *file* yaitu CCNB dan CCNTLV dengan perilaku yang berbeda di setiap *client* seperti waktu permintaan dilakukan dan berapa. Pada konsep CCN yang sebenarnya tipe permintaan ini sama hanya yang membedakan adalah *header* pada setiap *file*. Akan tetapi, di dalam CCN-Lite tidak ada perbedaan signifikan yang terjadi ketika file tersebut diminta.

```
[eInterestMode]

[ePreCacheMode]

[eFwdRulesMode]
ContentPrefix = ccnb:/b3c/wowmom , NextHop = router3.eth[0] , AccessFrom = router1.eth[3] , UpdateTime = 0/*s*/
ContentPrefix = ndntlv:/b3c/wowmom , NextHop = router3.eth[0] , AccessFrom = router1.eth[3] , UpdateTime = 0/*s*/

[eCommentsMode]
--------------------------------------------------------------------------------
comments go here
```

Gambar 4.4 Contoh bagian konfigurasi *router*

Pada konfigurasi *router,* mode yang digunakan adalah *eFwdRulesMode*, mode untuk mengatur jalur dari permintaan file yang diminta oleh *client*. Pada mode *eFwdRulesMode* harus terdapat *ContentPrefix, NextHop, AccessFrom,* dan *UpdateTime*. Penulisan kode konfigurasi untuk *router* dituliskan seperti pada Gambar 4.4.

```
[eInterestMode]

[ePreCacheMode]
ContentName = ccnb:/b3c/wowmom/movie1   , StartChunk = 0 , ChunksCount = 100 , UpdateTime = 0/*s*/
ContentName = ndntlv:/b3c/wowmom/movie2   , StartChunk = 0 , ChunksCount = 110 , UpdateTime = 0/*s*/

[eFwdRulesMode]

[eCommentsMode]
--------------------------------------------------------------------------------
comments go here
```

Gambar 4.5 Contoh bagian konfigurasi *server*

Ketika proses konfigurasi *client* dan *router* telah dilakukan, dilanjutkan menuju *file* konfigurasi *server*. Pada *server*, mode yang digunakan adalah *ePreCacheMode*. Mode ini melakukan *pre-load* konten dalam *cache* pada waktu yang ditentukan. Di dalam mode ini harus terdapat *ContentName, StartChunk, ChunksCount* dan *UpdateTime.* Untuk *ChunksCount* yang ada pada konfigurasi *server* berguna sebagai jumlah *chunk* dari setiap *file* yang dimiliki. Penulisan kode konfigurasi pada *server* dituliskan seperti pada Gambar 4.5.

4.2.3.2 Konfigurasi File Omnetpp.ini

Setelah konfigurasi dari setiap *node* telah diatur, maka dilanjutkan dengan melakukan konfigurasi pada *file* omnetpp.ini. *File* omnetpp.ini adalah sebuah *file* yang digunakan untuk mengatur agar bagian-bagian yang telah dikonfigurasi dapat dipanggil ketika dijalankan pada simulasi. Pada tahap ini, konfigurasi omnetpp.ini dilakukan dengan menambahkan *file* .cfg yang telah dikonfigurasi sedemikian rupa agar dapat dijalankan pada simulasi. Di dalam konfigurasi ini berisikan baris kode seperti pada Table 4.6 yang dijelaskan sebagai berikut:

- *Network* : Topologi yang digunakan dalam pengujian.
- *Description* : Deskripsi dari topologi pengujian.
- *Default Debug Level* : Debug level yang digunakan pada pengujian yaitu level 4.
- *Min Transmition Pace* : Minimal *pace* transmisi pada paket CCN.
- *Max Cache Slots* : *Cache storage* yang ada pada setiap *node*.
- *Max Cache Bytes* : Maksimal ukuran *cache* yang dapat disimpan.
- *CCN Core Version* : Versi dari *tools* CCN-Lite yang digunakan.
- *CCN Scenario of File* : *File* skenario CCN yang diatur pada setiap node yang digunakan dalam simulasi. Skenario konfigurasi *client*, *router* dan *server* dituliskan di dalam konfigurasi ini.

Table 4.6 *File* Konfigurasi omnetpp.ini

	Omnetpp.ini
1	[Config CCNoEther_6c_3r_1s]
2	
3	network = unibas.ccnlite.topology.CCNoEther_Tau_6cli_3rtr_1svr
4	description = "Example CCN over Ethernet Tau topology with 6 clients, 3 routers, 1 servers. Scenario setup
5	taken from clientX.cfg files"
6	*## topology/scenario settings*
7	*.defaultDebugLevel = 4
8	*.auxDebug = **true**
9	
10	*## per node settings*
11	**.debugLevel = 4
12	**.minTxPace = 100ms
13	**.maxCacheSlots = 100
14	**.maxCacheBytes = 524288000Bytes
15	**.ccnCoreVersion = "CCN Lite v0.3.0"
16	*.client1.net.ccnScenarioFile = "client1_ccn.cfg"
17	*.client2.net.ccnScenarioFile = "client2_ccn.cfg"
18	*.client3.net.ccnScenarioFile = "client3_ccn.cfg"
19	*.client4.net.ccnScenarioFile = "client4_ccn.cfg"
20	*.client5.net.ccnScenarioFile = "client5_ccn.cfg"
21	*.client6.net.ccnScenarioFile = "client6_ccn.cfg"
22	*.router1.net.ccnScenarioFile = "router1_ccn.cfg"
23	*.router2.net.ccnScenarioFile = "router2_ccn.cfg"

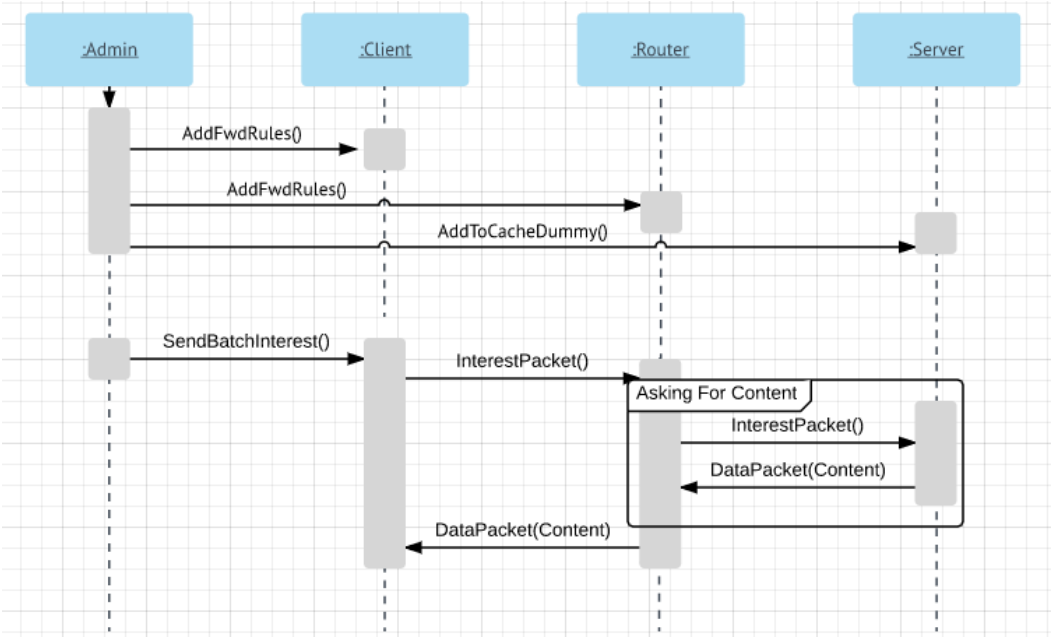
```
24 *.router3.net.ccnScenarioFile = "router3_ccn.cfg"
25 *.server1.net.ccnScenarioFile = "server1_ccn.cfg"
26
```

Dalam Konfigurasi ini, *maxCacheSlots* diberikan adalah sebanyak 100 slot dengan tujuan untuk mengetahui apakah semua *chunk* akan di *cache* oleh *cache router*. *MaxCacheBytes* diberikan sebesar 524288000 Bytes atau lebih kurang sebesar 54 MB. Nilai ini diberikan berdasarkan *default* dari CCN-Lite.

BAB 5 SIMULASI DAN ANALISIS HASIL

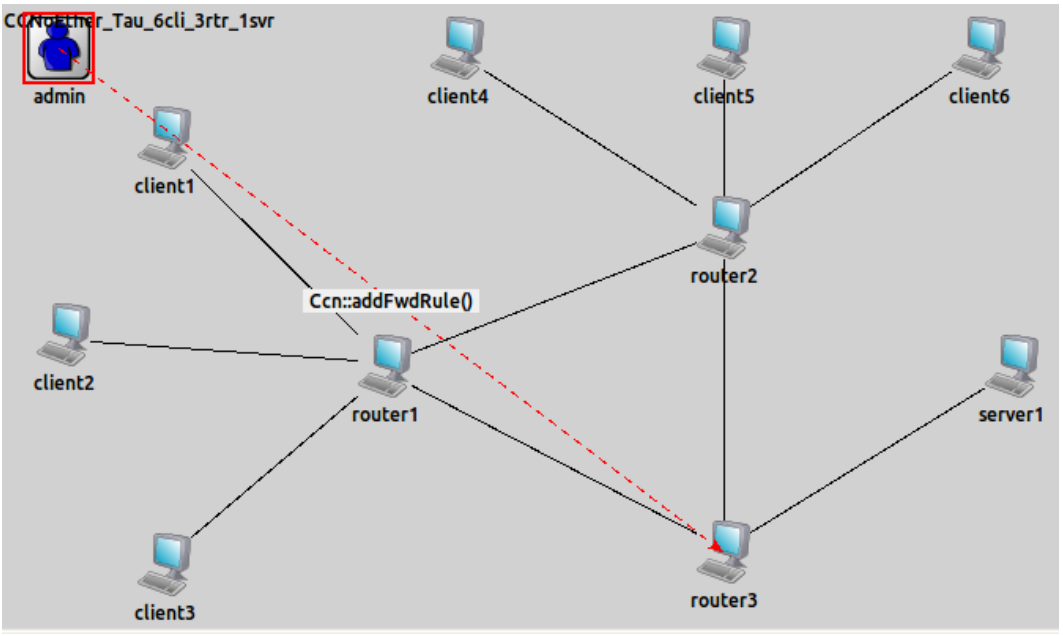
5.1 Simulasi

Pada tahap ini, dilakukan pengujian untuk mendapatkan hasil dari simulasi pada OMNeT++. Simulasi ini dilakukan sebanyak 1 kali dengan jadwal waktu pengiriman yang telah dibuat merujuk pada Tabel 3.2 . Tujuan dari simulasi ini adalah untuk mengetahui bagaimana komunikasi CCN yang terjadi di dalam CCN-Lite, bagaimana mekanisme dan kinerja *in-network caching* serta untuk mengetahui latensi dan CHR dari simulasi. Alur dari simulasi ini ditunjukkan seperti pada Gambar 5.1 dalam bentuk *sequence diagram*. Admin memberikan perintah inisialisasi kepada *node,* maka dilanjutkan dengan proses permintaan *file*. Pada saat *Interest* berada di dalam router, router tersebut akan melakukan pencarian konten pada CS, apabila ditemukan data yang diminta maka data tersebut akan segera dikirimkan menuju *client* yang meminta, apabila tidak ditemukan maka akan diteruskan menuju *router* lain hingga menuju *server*. Proses inisialisasi yang terjadi dari Gambar 5.2 hingga 5.4 merujuk pada Gambar 5.1.



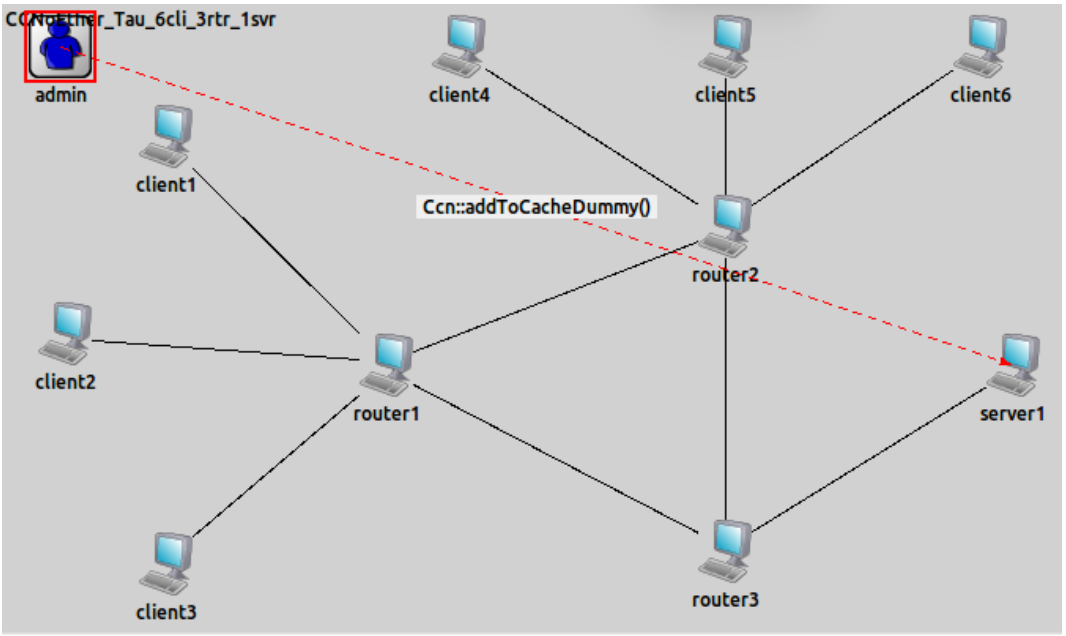
Gambar 5.1 *Sequance diagram* simulasi

Saat simulasi dimulai, pertama kali admin melakukan proses *addFwdRule* atau proses inisialisasi skenario pada setiap *client* dan *router*. Pada setiap *client,* proses ini dilakukan sebanyak 2 kali karena pada setiap *client* meminta 2 *file* berbeda dan pada setiap *router*, proses ini juga dilakukan sebanyak 2 kali karena setiap *router* dilalui 2 tipe *file* yang berbeda. Proses *addFwdRule* ditunjukkan seperti pada Gambar 5.2.



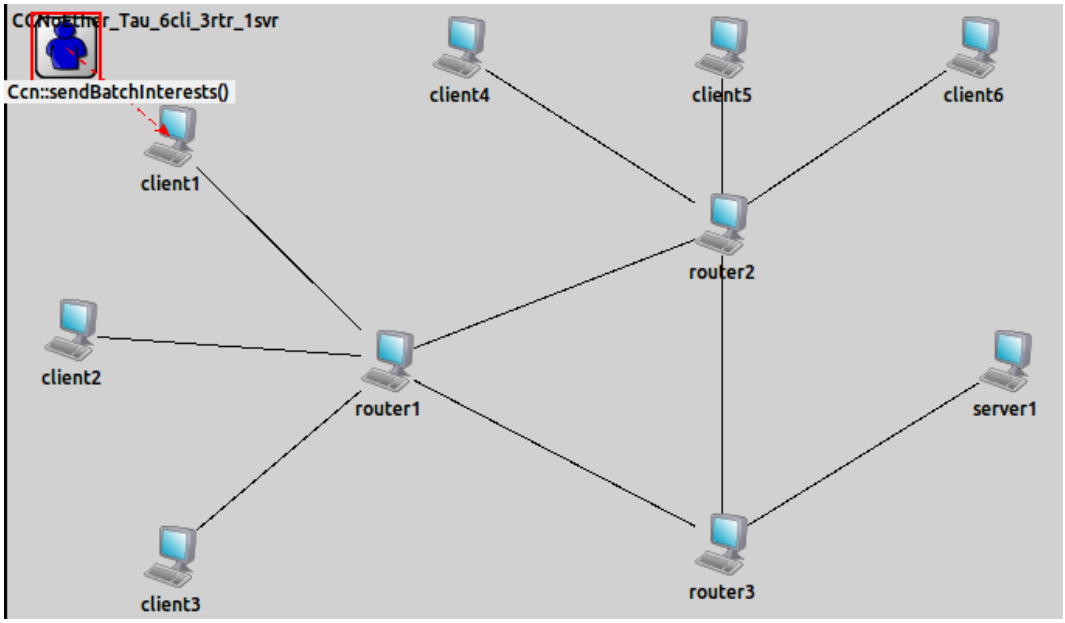
Gambar 5.2 Proses *addFwdRule* pada setiap *client* dan *router*

Ketika proses pertama telah dilakukan ke semua *client* dan router, maka proses selanjutnya adalah proses *addToCacheDummy* yang dilakukan oleh admin kepada *server* sebanyak 2 kali. Alasan proses ini dilakukan sebanyak 2 kali adalah karena *server* menyimpan konten sebanyak 2 *file* yaitu CCNB dan CCNTLV. Tujuan dari proses ini adalah untuk mendaftarkan *file* kedalam *server* agar konten dapat diambil ketika *client* meminta. Proses ini ditunjukkan seperti pada Gambar 5.3.



Gambar 5.3 Proses *addToCacheDummy* pada *server*

Ketika proses inisialisasi, maka dilanjutkan dengan admin memberikan proses *sendBatchInterest* pada *client* seperti pada Gambar 5.4. Proses ini memberikan perintah kepada *client* untuk memulai meminta file sesuai dari waktu permintaan yang telah dibuat pada Tabel 3.2.



Gambar 5.4 Proses *sendBatchInterest* oleh admin kepada *client*

5.2 Pengujian Simulasi

Pengujian simulasi dilakukan berdasarkan skenario penjadwalan yang telah dibuat merujuk pada Tabel 3.1. Tujuan dari pengujian ini adalah untuk mengetahui bagaimana implementasi dan kinerja dari *in-network caching* pada CCN-Lite dalam lingkungan simulator OMNeT++. Parameter yang diujikan adalah latensi pengiriman dan *cache hit ratio* (CHR). Dalam simulasi ini, data akhir dari hasil latensi pengiriman dan *cache hit ratio* didapatkan ketika simulasi telah berakhir.

5.3 Analisis Hasil Simulasi

Berdasarkan hasil simulasi yang telah dilakukan merujuk pada jadwal waktu permintaan pada Tabel 3.2, CCN-Lite dapat mendesain dan mengimplementasikan *in-network caching* dengan baik berdasarkan topologi yang telah dirancang. Bekerjanya *in-network caching* pada simulasi ini dikarenakan adanya pemberian *cache slot* dan *cache size* pada konfigurasi omnetpp.ini. Proses *caching* terjadi pada waktu permintaan ke 2 sampai waktu permintaan ke 6. Hal ini dibuktikan pada Gambar 5.5 grafik hasil latensi pengiriman.

Untuk *cache replacement* yang digunakan dalam simulasi ini yaitu *Least Recently Used* (LRU). Proses penghapusan ini terjadi pada waktu permintaan ke-5 pada saat *Client 5* meminta data CCNB. Ketika paket *Interest* berada pada router 2, hanya chunk ke 1 sampai 60 yang diberikan langsung menuju *client 2,* untuk chunk 61 sampai 75 didapatkan dari router 3. Hal ini menandakan bahwa tidak semua *chunk* dilayani oleh *router* terdekat. Untuk *cache decision* yang digunakan

dalam simulasi ini adalah *LCE* (*Leave Copy Everywhere*). Strategi ini dibuktikan pada waktu permintaan ke-3 oleh *Client* 6 yang mendapatkan data secara penuh dari *router* 3 dan waktu permintaan ke-5 oleh *Client 5* yang mendapatkan data konten dari *router 3* pada chunk 61 sampai 75. Menurut (Chiocchetti, et al., 2013), LCE dan LRU adalah pasangan model penyimpanan dan penghapusan yang sering digunakan dalam proposal ICN atau *centric network* lainnya.

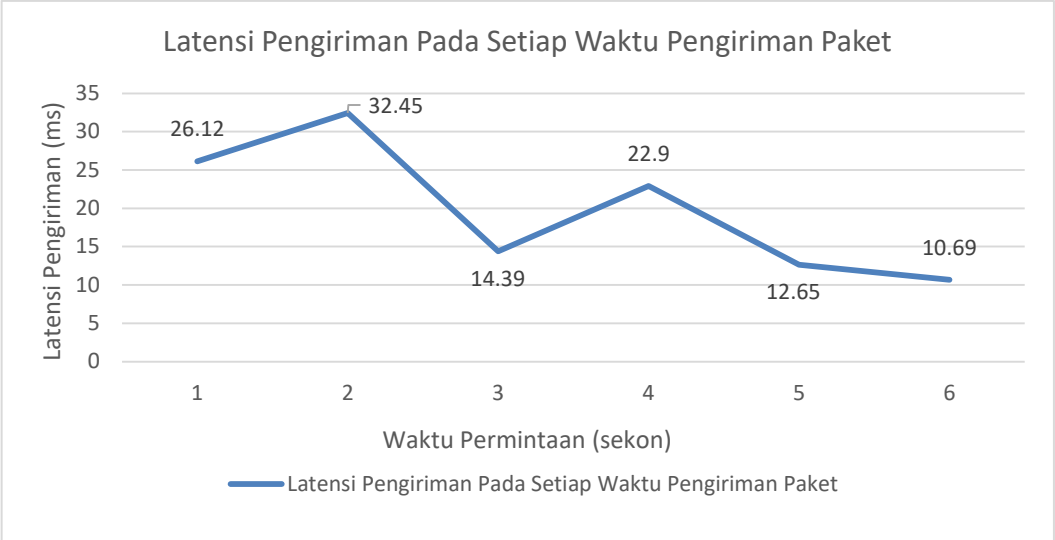
Akan tetapi, CCN-Lite didapatkan beberapa kekurangan dalam melakukan implementasi *in-network caching* seperti pengaturan jalur pengiriman dilakukan secara statik, kemudian pada *cache slot* dan *cache size* yang diatur pada konfigurasi omnetpp.ini, selama diberikan nilai tidak sama dengan 0 maka *caching* akan tetap terjadi. Jadi, dapat diambil kesimpulan bahwa CCN-Lite mampu melakukan implemetasi *in-network caching* pada arsitektur CCN dengan baik dengan beberapa kekurangan.

5.3.1 Analisis Latensi Pengiriman

Pada Tabel 5.1, latensi tertinggi didapatkan pada waktu permintaan ke-2 yaitu sebesar 32.45 ms. Nilai ini diperoleh karena pada permintaan tersebut belum terjadi *caching* dan menyebabkan pengambilan data diambil menuju server. Latensi terendah didapatkan pada waktu permintaan ke-6 yaitu sebesar 10.69 ms. Nilai ini diperoleh karena data didapatkan dari *router* terdekat atau terjadi proses *caching*. Pada waktu ke-4 terjadi proses *caching* dan *non-caching*. Proses *caching* terjadi dari permintaan *chunk* 1 hingga 75 dan untuk chunk 76 hingga 115 dilakukan proses non-caching atau meminta data menuju server. Dari waktu inilah didapatkan nilai sebesar 22.9 ms. Jadi, dapat diambil kesimpulan bahwa dengan adanya proses *caching,* latensi pengiriman yang diperoleh semakin rendah dari pada tanpa menggunakan *caching*.

Tabel 5.1 Tabel hasil latensi pengiriman

Data Hasil Latensi Pengiriman Pada Setiap Skenario		
Waktu permintaan (/s)	Nama Client	Latensi Pengiriman (/ms)
1	Client 1 dan Client 4	26.12
2	Client 2 dan Client 5	32.25
3	Client 3 dan Client 6	14.39
4	Client 1 dan Client 6	22.9
5	Client 2 dan Client 5	12.65
6	Client 3 dan Client4	10.39



Gambar 5.5 Grafik latensi pengiriman paket

5.3.2 Analisis *Cache Hit Ratio* (CHR)

Tabel 5.2 adalah tabel dari perhitungan *Cache Hit Ratio* ketika meminta file CCNB movie1. Dalam tabel tersebut didapatkan hasil CHR sebesar 0.45 atau 45% dari total konten yang diminta. *client2* dan *client4* mendapatkan nilai CHR 0 karena *client* tersebut tidak mengirimkan paket *Interest* sehingga permintaan tidak dilayani baik oleh *router* maupun *server*. Sedangkan *client1* mendapatkan nilai CHR 0 karena permintaan dilayani oleh *server* bukan *cache router*.

Table 5.2 Hasil perhitungan CHR pada permintaan CCNB

Data Hasil *Cache Hit Ratio* Dari Permintaan CCNB			
Nama *Client*	*Chunk* yang diminta	*Cache Hit*	CHR
Client 1	100	0	0
Client 2	60	0	0
Client 3	75	75	1
Client 4	100	0	0
Client 5	75	75	1
Client 6	60	60	1
Jumlah	470	210	
N(Hit) / N(Request)			0.45

Pada Tabel 5.3, didapatkan hasil *cache hit ratio* pada permintaan *file* movie2 protokol CCNTLV yaitu sebesar 0.54 atau 54%. Dalam tabel 5.3, client2 dan client5 mendapatkan nilai CHR 0 karena paket *Interest* yang dikirimkan dilayani oleh *server* dan bukan *cache router* dan permintaan yang dilakukan adalah permintaan pertama kali pada *file* movie2. Pada client1 dan client6, *cache hit* yang didapatkan sebanyak 75. Nilai ini diperoleh karena pada chunk ke 1 sampai 75 data konten di dapat dari *cache router*, sedangkan pada chunk ke 76 sampai 115 data konten didapatkan dari *server*.

Table 5.3 Hasil perhitungan CHR pada permintaan CCNTLV

Data Hasil *Cache Hit Ratio* Dari Permintaan CCNTLV			
Nama *Client*	*Chunk* yang diminta	*Cache Hit*	CHR
Client 1	115	75	0.65
Client 2	75	0	0
Client 3	60	60	1
Client 4	60	60	1
Client 5	75	0	0
Client 6	115	75	0.65
Jumlah	500	270	
N(Hit) / N(Request)			0.54

Jadi, dapat diambil kesimpulan bahwa file protokol CCNTLV dilayani dengan baik oleh *cache router* maupun *server*. Hal ini dibuktikan dengan nilai CHR dari CCNTLV sebesar 0.54.

BAB 6 PENUTUP

6.1 Kesimpulan

Berdasarkan hasil dari perancangan, simulasi serta analisis dari desain dan implementasi *In-network Caching* pada *Content Centric Networking* menggunakan CCN-Lite dengan simulator OMNeT++, dapat diambil kesimpulan sebagai berikut:

1. Berdasarkan dari simulasi yang telah dilakukan, CCN-Lite dapat mendesain dan mengimplementasikan *in-network caching* pada arsitektur CCN dengan baik dengan berdasar pada topologi pengujian 1 *server*, 3 *router* dan, 6 *client* serta skenario pengujian yang telah dibuat. Bekerjanya *in-network caching* pada simulasi ini dikarenakan adanya pemberian *cache slot* dan *cache size* pada konfigurasi omnetpp.ini. Untuk *cache replacement* dan *cache decision* yang digunakan adalah *Least Recently Used* (LRU)/ *Leave Copy Everywhere* (LCE). Jadi, dapat diambil kesimpulan bahwa CCN-Lite mampu melakukan implemetasi *in-network caching* pada arsitektur CCN dengan beberapa kekurangan.
2. Berdasarkan kinerja dari *in-network caching* yang telah dilakukan, latensi yang didapatkan ketika menggunakan *caching* yaitu 10.69ms. Nilai ini lebih rendah dari pada latensi ketika tanpa menggunakan *caching* sebesar 32.45ms. Jadi, ketika menggunakan *caching* latensi yang didapat semakin rendah dari pada tanpa menggunakan *caching*.
3. Dari segi CHR setiap konten, didapatkan hasil bahwa nilai CHR dari protokol CCNTLV sebesar 0.54 atau 54% dan nilai CHR dari protokol CCNB yaitu sebesar 0.45 atau 45% dari total *chunk* yang diminta. Hal ini menunjukkan bahwa protokol CCNTLV lebih baik dalam hal pelayanan konten yang diminta dari pada CCNB. Semakin tinggi nilai CHR yang didapat, semakin baik pula konten tersebut dilayani.

Berdasarkan dari simulasi yang telah dilakukan, didapatkan bahwa CCN-Lite memiliki kelemahan dalam melakukan implementasi CCN yaitu pengaturan jalur pengiriman dilakukan secara statik, kemudian pada *cache slot* dan *cache size* yang diatur pada konfigurasi .ini, selama diberikan nilai tidak sama dengan 0 maka *caching* akan tetap terjadi.

6.2 Saran

Saran yang dapat diberikan untuk pengembangan penelitian selanjutnya adalah sebagai berikut:

1. Dapat menerapkan *cache decision* dan *cache replacement* selain LCE dan LRU dengan pengerjaan secara berkelompok.
2. Dapat menerapkan *in-network caching* pada implementasi sesungguhnya secara berkelompok.

DAFTAR PUSTAKA

Ahir, Deepali Damodar. & Kumbharkar, Prashant. B., 2012. *Content Centric Networking and its Applications,* India: Jurnal of Global Research in Computer Science.

Amadeo, M., Campolo, C., Molinaro, A. & Ruggeri, G., 2013. Content-centric wireless networking: A survey. *Elsevier,* Issue 12, pp. 1-13.

Beal, V., 2014. *webopedia.* [Online] Tersedia di: http://www.webopedia.com/TERM/C/CDN.html [Diakses pada 9 Maret 2017].

CCN-Lite, 2011. *CCN-Lite.* [Online] Tersedia di: http://www.ccn-lite.net [Diakses pada 1 Maret 2017].

Chiocchetti, R., Rossi, D. & Rossini, G., 2013. *ccnSim: an Highly Scalable CCN Simulator.* Paris, RESCOM.

Daras, P., Semertzidis, T., Makris, L. & Strintzis, M. G., 2010. *Similarity Content Search in Content Centric Networks.* Firenze, Association for Computer Machinery.

Ghali, C., Tsudik, G. & Wood, C. A., 2016. *Network Names is Content-Centric Networking.* Kyoto, Association for Computing Machinery.

INET, 2015. *Inet Framework.* [Online] Tersedia di: https://inet.omnetpp.org/Introduction.html [Diakses pada 14 Maret 2017].

Jacobson, V., 2009. *A Description of Content Centric Networking (CCN).* German: Parc Company.

Jacobson, V. Smetters, Diana K., Briggs, Nicholas H., Plass, Michael F., Stewart, Paul, Thorton, James D., Braynard, Rebecca L., 2009. *VoCCN : Voice-over Content-Centric Networks.* Rome, Association for Computing Machinery.

Jacobson, V. Smetters, Diana K., Thorton, James D., Plass, Michael F., Briggs, Nicholas H., Braynard, Rebecca L., 2009. *Networking Named Content.* Rome, Association for Computing Machinery.

Kim, Y. & Yeom, I., 2013. Performance analysis of in-network caching for content-centric networking. *Computer Networks,* 57(7), pp. 2465-2482.

Mangili, M., Martignon, F. & Capone, A., 2015. Performance analysis of Content-Centric and Content-Delivery network with evolving object popularity. Issue 7, pp. 1-19.

Nakamura, R. & Ohsaki, H., 2016. *Performance Analysis of CCN on Arbitary Network Topology,* Hyogo: IEEE.

OMNET, 2016. *OMNet++.* [Online] Tersedia di: https://omnetpp.org/index [Diakses pada 12 Maret 2017].

PARC Company, 2010. *Named Data Networking,* California: PARC.

Perino, D. & Varvello, M., 2011. *A Reality Check for Content Centric Networking.* Toronto, Association for Computer Machinery.

Rossini, G. & Rossi, D., 2014. *Coupling CAching and Forwarding : Benefits, Analysis, and Implementation.* Paris, Association for Computing Machinery.

Saino, L., Psaras, I. & Pavlou, G., 2014. *Icarus - a caching simulator for Information Centric Networking,* Lisbon: Internasional Conference on Science and Technology.

Saucez, D., Barakat, C., Kalla, A. & Turletti, T., 2012. *Off-Path Caching in CCN.* France, CCNx Conference.

Sena, P., Ishimori, A., Carvalho, I. & Abelem, A., 2016. *Cache-Aware Interest Routing: Impact Analysis on Cache Decision Strategies in Content Centric Networking.* Valparaiso, Association for Computer Machinery.

Shibuya, A., Hayamizu, Y. & Yamamoto, M., 2016. Cache Decision Policy for Breadcrumbs in CCN. *IEEE,* Issue 5, pp. 1-6.

Tortelli, M., Rossi, D., Boggia, G., Grieco, L.A., 2016. *ICN software tools: Survey and Cross-comparison,* Italy: SIMPAT.

Wang, L., 2013. Multi-Objective In-Network Caching Strategies. *IEEE.*

Wang, L., Bayhan, S. & Kangasharju, J., 2013. *Cooperation Policies for Efficient In-Network Caching.* Hong Kong, Association for Computer Machinery.

Wang, S. Bi, Jun, Wu, Jianping, Li, Zhaogeng, Zhang, Wei, Yang, Xu, 2011. *Could In-Network Caching Benefit Information Centric Networking?.* Bangkok, Association for Computing Machinery.

Wu, Haibo, Li, Jun, Zhi, Jiang, 2015. *Could End System Caching and Cooperation Replace In-Network Caching in CCN?.* London, Association for Computer Machinery.

Xu, Y., Ci, S., Li, Y., Lin, T., Li, G., 2016. Design and evaluation of coordinated in-network caching model for content centric networking. *Computer Networks,* 110(7), pp. 266-283.

Zhang, G. & Xu, Z., 2015. Combing CCN with network coding: And architectural perspective. *Elsevier,* 0(6), pp. 1-12.

LAMPIRAN A INSTALASI OMNET++

OMNeT ++ (*Objective Modular Network Testbed in C ++*) adalah sebuah *framework* simulasi yang berbasis C++ yang dapat digunakan untuk membangun simulasi jaringan. OMNeT++ dapat digunakan secara gratis untuk simulasi nonkomersial seperti pada institusi akademis dan untuk pengajaran. Dalam simulasi ini, OMNeT++ digunakan sebagai simulator untuk menjalankan simulasi berdasarkan dari *workspace* CCN-Lite. Langkah-langkah instalasi simulator adalah sebagai berikut:

1. Langkah pertama adalah mengunduh *file* OMNeT++ pada alamat http://omnetpp.org. Pilih dan unduh *file* OMNeT++ versi 4.5.
2. Setelah *file* terunduh, buka terminal dan tuliskan *command* untuk mengekstrak *file*:

```
$ tar xvfz omnetpp-4.5.0-src.tgz
```

   Perintah `tar` digunakan untuk mengekstrak *file* berbentuk .rar/.tgz. Kemudian maksud dari perintah opsional `xvfz` adalah X untuk mengekstrak *file* dari berkas, V untuk menampilkan proses ekstraksi, F untuk argumen yang mengikuti nama *file* dan, Z untuk metode dekompresi.
3. Masuk ke direktori OMNeT++ dengan perintah :

```
$ cd omnetpp-4.5
$ . setenv
```

   Perintah `setenv` digunakan untuk memeriksa dimana folder omnetpp-4.5 disimpan setelah diekstrak.
4. Setelah mengetahui dimana tempatkan folder omnetpp-4.5, ditahap ini dilakukan penentuan jalur agar IDE dapat dikenali oleh sistem

```
$ export PATH=$HOME/omnetpp-4.5.0/bin:$PATH
```

   Apabila perintah diatas ingin disimpan secara permanen, maka lakukan edit pada *file* `.bashrc` dengan perintah

```
$ gedit ~/.bashrc
```

5. Lakukan pembaharuan sistem dengan perintah :

```
$ sudo apt-get update
```

6. Tahap ini adalah bagian yang diperlukan oleh IDE OMNeT++ agar dapat dijalankan dengan baik dan sistem dapat bekerja dengan selayaknya.

```
$ sudo apt-get install build-essential gcc g++ bison flex perl
qt5-default tcl-dev tk-dev libxml2-dev zlib1g-dev default-jre
doxygen graphviz libwebkitgtk-3.0-0
```

Penjelasan dari paket yang diinstal seperti pada Tabel 4.1 dibawah adalah sebagai berikut:

Table Penjelasan *Command*

`Build Essential , gcc , g++ , bison , flex , perl`	Paket ini dibutuhkan untuk meng-*compile* model dari simulasi OMNeT++ dan juga sebagai *tools*.
`Tcl-dev , tk-dev`	Dibutuhkan oleh bagian dari simulasi Tkenv *runtime.*
`Libxml2-dev , zlib1g-dev`	Sebagai xml *parser* yang dibutuhkan untuk membaca *file* .xml.
`Default-jre`	Untuk menginstal java *runtime* agar dapat menjalankan simulasi IDE berbasis *Eclipse.*
`Doxygen , graphviz`	Paket ini digunakan untuk dokumentasi dari fitur NED yang ada pada IDE.

7. Ketika instalasi paket telah selesai, dilanjutkan dengan melakukan instalasi paket MPI (*Message Passing Interfaces*) :

```
$ sudo apt-get install openmpi-bin libopenmpi-dev
```

Gunanya untuk membantu waktu eksekusi simulasi pemrosesan pararel.

8. Setelah semua bagian telah terpasang, maka dilanjutkan dengan konfigurasi omnetpp dengan perintah :

```
$ cd omnetpp-4.5
$ ./configure
```

Perintah ini digunakan untuk mendeteksi semua perangkat lunak yang telah dipasang dan dikonfigurasi oleh sistem. Hasil dari konfigurasi diatas akan dituliskan kedalam *file* bernama `Makefile.inc`.

9. Setelah konfigurasi telah dilakukan maka tahap terakhir adalah mem-*compile file* OMNeT++ dengan perintah `make`. Perintah ini akan memakan waktu sekitar 30 menit dan perintah ini dilakukan di dalam direktori omnetpp-4.5.

```
$ make
```

10. Untuk menjalankan IDE ini dapat memanggil dari *command line* dengan perintah :

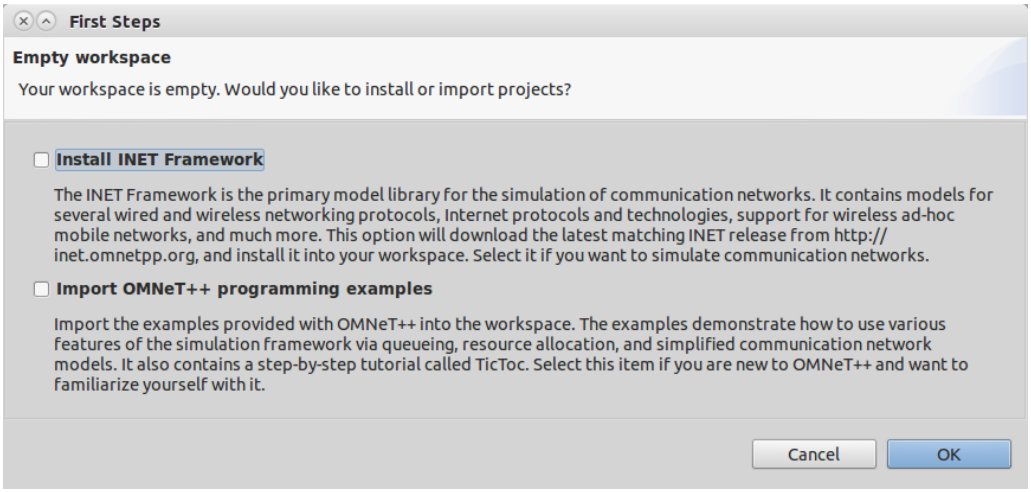
```
$ omnetpp
```

Atau dapat dilakukan dengan cara memasang ikon *desktop* atau *menu item*.

```
$ make install-menu-item
```

```
$ make install-dekstop-icon
```

Pada saat membuka IDE ini akan muncul *windows first steps* dimana menampilkan pilihan instalasi INET secara otomatis menggunakan koneksi Internet dan *import* contoh program OMNeT++, karena di dalam simulasi ini menggunakan INET yang telah ditentukan dan juga untuk meringankan jalannya program ketika melakukan *build project* maka kedua pilihan tidak di centang seperti pada dibawah.



Tampilan *First Steps* Pada OMNeT++

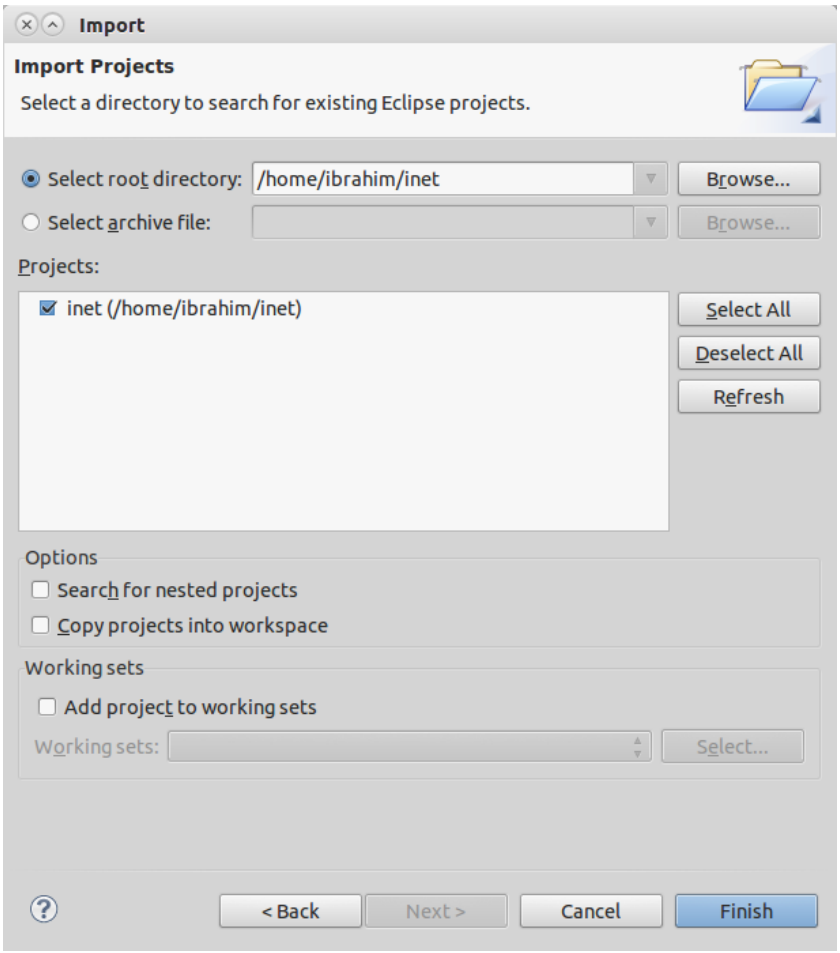
LAMPIRAN B INSTALASI INET FRAMEWORK

INET *Framework* adalah sebuah *library* model open-source yang digunakan pada simulator OMNeT++. Dalam penetian ini, INET *Framework* sangat dibutuhkan oleh CCN-Lite karena bagian-bagian node yang dipakai diambil dari *library* ini. Agar dapat dijalankan, perlu langkah-langkah instalasi INET yang sebagai berikut:

1. Langkah pertama adalah mengunduh dari website http://inet.omnetpp.org/Download , kemudian pilih inet 2.4.0.
2. Buka terminal dan tulis perintah untuk melakukan ekstrak berkas INET

```
$ tar xvfz inet-2.4.0-src.tgz
```

3. Buka IDE OMNeT++ dan lakukan *import project* melalui *File* -> *Import* -> *General* -> *Existing Projects to the Workspace* -> pilih "*select root directory*", *browse* -> cari folder INET yang telah di ekstrak.



Tampilan Ketika Melakukan *Import Workspace*

Akan muncul *workspace* INET, *klik kanan dan pilih build project.*

LAMPIRAN C INSTALASI CCN-LITE

CCN-Lite adalah sebuah projek yang merepresentasikan protokol CCN dan NDN dimana projek ini dapat digunakan berbagai *platform* seperti UNIX, Linux kernel, OMNeT++, Android, Arduino (Uno dan AtMega328, KiB RAM), RFduino (32KiB RAM) dan Docker. Dalam penelitian ini penulis menggunakan simulator OMNeT++ untuk mengetahui bagaimana kelengkapan dari CCN-Lite apabila dibandingkan dengan arsitektur CCN yang sebenarnya. Terdapat dua langkah untuk mengambil berkas CCN-Lite yaitu dengan mengunduh berkas secara *offline* atau menggunakan perintah `gitclone` dari github. Penulis melakukan cara kedua yaitu dengan perintah `gitclone`, alasan memilih cara kedua karena ketika ada *bug* atau *error* maka yang pertama kali diperbaiki adalah yang berada pada github. Berikut adalah langkah-langkah instalasi CCN-Lite:

1. Langkah pertama adalah memasang openSSL.

```
$ sudo apt-get install libssl-dev
```

OpenSSL sendiri adalah sebuah *toolkit* kriptografi yang mengimplementasikan *Secure Socket Layer* dan *Transport Layer Security* serta terkait dengan protokol jaringan standar kriptografi yang dibutuhkan oleh keduanya.

2. Lakukan `gitclone` CCN-Lite.

```
$ gitclone https://github.com/cn-uofbasel/ccn-lite
```

Perintah `gitclone` ini digunakan untuk meng-*copy file* yang berasal dari repositori Git.

3. Lakukan penyesuaian variabel dari CCN-Lite dan untuk penentuan jalur agar dapat dikenali oleh sistem.

```
$ export CCNL_HOME="`pwd`/ccn-lite"
$ export PATH=$PATH:"$CCNL_HOME/bin"
```

Agar perintah diatas menjadi permanen, dapat ditambahkan juga kedalam *file* .bashrc.

4. Kemudian masuk ke direktori CCN-Lite untuk melakukan *build*

```
$ cd $CCNL_HOME/src
$ make clean all
```

Perintah `make clean all` digunakan untuk menghapus semua bentuk temporary dan membentuk ulang *file* dari awal.

5. Setelah melakukan langkah diatas, tutup dan buka kembali terminal dan masuk kembali ke direktori ccn-lite/src untuk mem-*build* sebuah berkas bernama ccn-lite-omnet.

```
$ cd $CCNL_HOME/src
$ make ccn-lite-omnet
```

Berkas ini terdapat di dalam directori ccn-lite/src.

6. Buka IDE OMNeT++ untuk melakukan import project dengan cara File -> Import -> General -> Existing projects into workspace -> pilih select archive *file* (karena *file* bentuk .tgz) -> browse dan pilih ccn-lite-omnet.tgz. Tidak lupa untuk mengaktifkan checkbox di dalam kolom project.
7. Muncul *workspace* ccn-lite, kemudian klik kanan -> Index -> Rebuild. Ini digunakan untuk mengatur kembali seperti awal apabila terdapat konfigurasi yang tidak beraturan.
8. Setelah selesai lakukan klik kanan -> build project. Ketika proses *build* selesai, maka project siap untuk dijalankan.