
**TUGAS AKHIR – KI091391**

# IMPLEMENTASI PROGRESSIVE CACHING PADA CONTENT-CENTRIC NETWORKING UNTUK STREAMING VIDEO

**RADITE BAYU PRAKOSO**
**NRP 5110100040**

**Dosen Pembimbing I**
**Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D.**

**Dosen Pembimbing II**
**Dr. Eng. Radityo Anggoro, S.Kom., M.Sc.**

**JURUSAN TEKNIK INFORMATIKA**
**Fakultas Teknologi Informasi**
**Institut Teknologi Sepuluh Nopember**
**Surabaya 2014**


UNDERGRADUATE THESES – KI091391

# IMPLEMENTATION OF PROGRESSIVE CACHING IN CONTENT-CENTRIC NETWORKING FOR VIDEO STREAMING

RADITE BAYU PRAKOSO
NRP 5110100040

Supervisor I
Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D.

Supervisor II
Dr. Eng. Radityo Anggoro, S.Kom., M.Sc.

DEPARTMENT OF INFORMATICS
FACULTY OF INFORMATION TECHNOLOGY
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA 2014

## LEMBAR PENGESAHAN

# IMPLEMENTASI PROGRESSIVE CACHING PADA CONTENT-CENTRIC NETWORKING UNTUK STREAMING VIDEO

## TUGAS AKHIR

Diajukan Guna Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Bidang Studi Komputasi Berbasis Jaringan
Program Studi S-1 Jurusan Teknik Informatika
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember

Oleh :
**RADITE BAYU PRAKOSO**
NRP : 5110 100 040

Disetujui oleh Dosen Pembimbing Tugas Akhir :

Royyana Muslim Itjihadie, S.Kom, M.Kom., Ph.D. ..... (pembimbing 1)
NIP: 197708242006041001

Dr.Eng Radityo Anggoro, S.Kom. M.Sc. ..... (pembimbing 2)
NIP: 19841016200812100 2

**SURABAYA**
**DESEMBER 2014**

**IMPLEMENTASI PROGRESSIVE CACHING PADA CONTENT-CENTRIC NETWORKING UNTUK STREAMING VIDEO**


| | |
|---|---|
| **Nama Mahasiswa** | **: RADITE BAYU PRAKOSO** |
| **NRP** | **: 5110100040** |
| **Jurusan** | **: Teknik Informatika FTIF-ITS** |
| **Dosen Pembimbing 1** | **: Royyana Muslim Ijtihadie, S.Kom., M.Kom. Ph.D.** |
| **Dosen Pembimbing 2** | **: Dr.Eng. Radityo Anggoro, S.Kom., M.Sc.** |


## Abstrak


*Akhir-akhir ini, kebutuhan akan streaming video telah meingkat secara signifikan dan telah menjadi salah satu sumber informasi yang dipandang vital. Materi yang tersedia melalui sara streaming video antara lain peliputan berita secara langsung, video tutorial dan panduan, hiburan dan sebagainya*

*Namun, dengan arsitektur jaringan saat ini yang bisa dibilang konvensional, perkembangan teknologi yang mengarah ke streaming video telah menemui beberapa hambatan. Antara lain, server yang digunakan sebagai penyedia konten akan mengalami penurunan performa saat diakses oleh pengguna dalam jumlah banyak dan pada waktu yang bersamaan. Hal ini menjadi masalah yang cukup serius dikarenakan jumlah pengguna internet yang terus bertambah*

*Content-Centric Networking (CCN) adalah sebuah paradigma baru dalam jaringan. Pada CCN, data akan di-cache pada node yang tersebar pada jaringan. Ketika pengguna ingin menemukan data tertentu, pengguna tidak perlu tahu lokasi dari data tersebut. Karena data akan disebar*

*(cached) pada node yang terhubung dengan jaringan CCN. Secara teori, jika data dapat disebar dengan metode yang tepat, maka hal ini dapat mengurangi kerja dari server penyedia konten secara signifikan. Hal ini dimungkinkan karena server penyedia konten tidak harus mengirimkan data kembali kepada pengguna setiap kali pengguna ingin mengakses data yang sama dikarenakan data tersebut sudah mengalami proses caching pada node yang menghubungkan antara server penyedia konten dengan pengguna*

***Kata kunci: Content-Centric Networking, Progressive Caching, Video Streaming.***

**IMPLEMENTATION OF PROGRESSIVE CACHING IN CONTENT-CENTRIC NETWORKING FOR VIDEO STREAMING**


| | |
|---|---|
| **Student's Name** | **: RADITE BAYU PRAKOSO** |
| **Student's ID** | **: 5110100040** |
| **Department** | **: Teknik Informatika FTIF-ITS** |
| **First Advisor** | **: Royyana Muslim Ijtihadie, S.Kom., M.Kom. Ph.D** |
| **Second Advisor** | **: Dr.Eng. Radityo Anggoro, S.Kom., M.Sc.** |


## Abstract


*Nowadays, the need of video streaming has arisen significantly and is currently becoming one of vital source of information. Video streaming encompasses realtime news broadcasting, how-to and do-it-yourself video, tutorials, entertainment, etc.*

*However, using today's conventional network architecture, video streaming has encountered some rather difficult obstruction. Which is, video streaming server will have gradual decrease in performance when more user is accessing server's resource at the same time. This has become a serious problem considering that internet user is increasing exponentially.*

*Content-Centric Networking is a new network paradigm which primarily revolves in heavily cached data. In order to find wanted data, user needn't to know the exact location of the data. Because the data will be scattered (cached) in the node connected to the CCN network. Theoretically, if the data was to be cached appropriately across the network, it will significantly reduces server load*

*because server will not necessarily have to send data everytime any of network host requested same data because such data is likely to be cached in any of the network node according to how often the data was requested*

***Keywords: Content-Centric Networking, Progressive Caching, Video Streaming***

# KATA PENGANTAR

Alhamdulillahirabbil'alamin, segala puji bagi Allah SWT, yang telah melimpahkan rahmat dan hidayah-Nya sehingga penulis dapat menyelesaikan Tugas Akhir yang berjudul ***"Implementasi Progressive Caching pada Content-Centric Networking untuk Streaming Video".***

Salah satu kutipan paling terkenal dari Siddharta Gautama adalah "You are what you think". Awalnya, penulis dengan pesimis berpikiran bahwa penulis tidak akan pernah bisa menyelesaikan Tugas Akhir ini.

Namun, karena adanya pihak-pihak yang secara *altruist* selalu mendorong penulis baik secara moral maupun material, Tugas Akhir ini akhirnya bisa terselesaikan dengan baik.

Selesainya Tugas Akhir ini tidak lepas dari bantuan dan dukungan beberapa pihak. Sehingga pada kesempatan ini penulis mengucapkan terima kasih kepada:

1. Allah SWT yang telah melancarkan segala urusan dalam berbagai masalah yang saya hadapi.
2. Ayah, ibu, dan adik saya Dika, yang tanpa henti mendukung dan terus mendorong saya dalam pengerjaan Tugas Akhir ini.
3. Bapak Royyana selaku dosen pembimbing I yang tanpa lelah memberikan arahan dan bimbingan dalam pengerjaan Tugas Akhir ini.
4. Bapak Onggo sebagai dosen pembimbing II yang juga tanpa lelah membimbing dan mengoreksi kesalahan-kesalahan dalam pengerjaan Tugas Akhir ini.
5. Ibu Nanik selaku dosen wali yang selalu memberikan motivasi dan bantuan dalam menyelesaikan Tugas Akhir ini.
6. Teman-teman *administrator* dan sesama *user* laboratorium GCL yang telah beralih nama menjadi Alpro yaitu Khairy, Bobby, Alief, Valent, Zepry, Ubal, Yanto, Fahry yang

juga terkadang harus memotivasi penulis meskipu dengan cara yang sedikit keras

7. Teman-teman angkatan 2010 yang tidak bisa saya sebutkan satu-persatu
8. Hera, sebagai satu-satunya orang yang mungkin masih mau mendengar keluh-kesah penulis dan membantu penulis disaat penulis sudah mulai lelah dan mengalami *mental breakdown.*
9. Clarisa, Fifi, dan kawan-kawan penulis lain yang selalu dengan tepat mengajak penulis berkaraoke saat penulis sedang sibuk ataupun sedang mengalami *mental breakdown.*
10. Semua pihak yang baik secara langsung maupun tidak langsung telah membantu penulis menyelesaikan Tugas Akhir ini.

Penulis menyadari bahwa Tugas Akhir ini masih memiliki banyak kekurangan. Sehingga dengan kerendahan hati, penulis mengharapkan kritik dan saran dari pembaca untuk perbaikan ke depan.

Surabaya, Desember 2014

# DAFTAR ISI

# DAFTAR TABEL

# DAFTAR GAMBAR

# DAFTAR KODE SUMBER

# BAB I
# PENDAHULUAN

## 1.1 Latar Belakang

Pada saat ini kebutuhan akan *streaming* merupakan salah satu kebutuhan vital dari masyarakat. Kebutuhan akan informasi *realtime* yang berupa berita menjadi kebutuhan yang dirasa semakin penting dalam masyarakat. Namun, dengan paradigma jaringan konvensional pada saat ini, video *streaming* mengalami masalah yang cukup pelik. Performa dari *streaming server* akan berkurang secara drastis seiring dengan banyaknya pengguna yang mengakses *server* pada saat yang bersamaan. Hal ini menjadi masalah yang cukup serius mengingat banyaknya jumlah pengguna internet pada saat ini.

Salah satu solusi yang pernah ditawarkan adalah dengan menggunakan *cache server*. Namun, kelemahan terbesar *cache server* adalah kapasitas yang terbatas.

CCN (*Content-Centric Networking*) dapat menjadi solusi. CCN menawarkan solusi dengan membagi data menjadi *chunk* yang akan mengalami proses *caching* pada *node* yang terhubung dengan jaringan CCN. Dengan ini, diharapkan terjadi pengurangan *traffic* yang dihasilkan dari proses transfer data dari *server* ke *host* dan akan membuat kerja server menjadi lebih ringan.

## 1.2 Rumusan Masalah

Rumusan masalah yang diangkat dalam Tugas Akhir ini adalah sebagai berikut:

Bagaimana menerapkan CCN pada jaringan yang sudah ada?

Bagaimana menerapkan metode *progressive caching* pada jaringan CCN?

Bagaimana *host* dapat berkomunikasi dengan jaringan CCN dan menjalankan aplikasi *streaming*?
Fitur *video streaming* dari CCN kompatibel dengan jenis *file* apa saja?

## 1.3 Batasan Masalah

Beberapa batasan masalah yang menjadi batas pada Tugas Akhir ini adalah:

- CCN dan metode *progressive caching* hanya dijalankan pada lapisan *application*
- Implementasi Jaringan CCN menggunakan CCNx
- Mekanisme *caching* menggunakan algoritma LRU dan *Progressive Caching*
- Uji coba menggunakan 10 PC dengan 4 PC yang digunakan untuk analisa paket.
- 2 PC digunakan sebagai *content provider*
- 5 *file* dengan ekstensi .mkv, .avi, .flv, .mp3, .mp4 yang digunakan dalam *video streaming*

## 1.4 Tujuan

Tujuan dari pembuatan Tugas Akhir ini adalah:

- Dapat menghasilkan sebuah aplikasi *live streaming* beserta sebuah jaringan dengan arsitektur CCN.
- Menerapkan *progressive caching* pada jaringan CCN
- Membandingkan performa antara LRU dengan *Progressive Caching*.

## 1.5 Manfaat

Tugas Akhir ini diharapkan masyarakat dapat mengakses aplikasi *streaming* dengan performa yang terjamin.

## 1.6 Metodologi

Tahapan-tahapan yang dilakukan dalam pengerjaan Tugas Akhir ini adalah sebagai berikut:

### 1.6.1 Penyusunan proposal Tugas Akhir.

Proposal Tugas Akhir ini berisikan mengenai apa saja yang dibutuhkan, serta rumusan masalah yang ada dalam perancangan arsitektur CCN beserta implementasi klien untuk *streaming*.

### 1.6.2 Studi literatur

Tahap ini merupakan tahap pengumpulan informasi yang diperlukan untuk pengerjaan Tugas Akhir sekaligus mempelajarinya. Mulai dari pengumpulan literatur, diskusi, serta pemahaman topik Tugas Akhir di antaranya tentang:
Perancangan jaringan CCN menggunakan *node* yang terintegrasi dengan CCNx
Perancangan mekanisme alur *caching* dari video yang akan di-*streaming*

### 1.6.3 Perancangan perangkat lunak

Pada tahap ini dilakukan analisis terhadap sistem serta perancangan yang akan dibuat. Hal ini dimaksudkan untuk merumuskan suatu solusi yang tepat untuk melakukan implementasi pada sistem.

### 1.6.4 Implementasi perangkat lunak

Dalam pembuatan perangkat, digunakan beberapa teknologi untuk dapat menerapkan rancangan yang sudah ada, di antaranya:
Ubuntu
Merupakan salah satu sistem operasi Linux paling populer yang mudah dimengerti dan dioperasikan oleh orang awam
CCNx
CCNx merupakan sebuah implementasi dari sebuah pendekatan baru terhadap arsitektur komunikasi baru yang disebut *Content-Centric Networking* (CCN). CCNx dikembangkan oleh Palo Alto Research Center (PARC) dan didasarkan oleh arsitektur CCN yang juga dikembangkan oleh PARC.
Pengujian dan evaluasi

Proses pengujian dilakukan di lingkungan laboratorium Arsitektur dan Jaringan Komputer pada Gedung Teknik Informatika Institut Teknologi Sepuluh Nopember Surabaya.

### 1.6.5 Penyusunan buku Tugas Akhir.

Pada tahapan ini disusun buku yang memuat dokumentasi mengenai pembuatan serta hasil dari uji coba sistem yang telah dibuat.

## 1.7 Sistematika Penulisan Laporan Tugas Akhir

Buku Tugas Akhir ini disusun dengan sistematika penulisan sebagai berikut:

**BAB I. PENDAHULUAN**

Bab yang berisi mengenai latar belakang, tujuan, dan manfaat dari pembuatan Tugas Akhir. Selain itu permasalahan, batasan masalah, metodologi yang digunakan, dan sistematika penulisan juga merupakan bagian dari bab ini.

**BAB II. TINJAUAN PUSTAKA**

Bab ini berisi penjelasan secara detail mengenai dasar-dasar penunjang dan teori-teori yang digunakan untuk mendukung pembuatan Tugas Akhir ini.

**BAB III. PERANCANGAN PERANGKAT LUNAK**

Bab ini berisi tentang perancangan sistem, *flowchart,*dan perancangan sistem yang akan dibuat. Perancangan yang dibahas meliputi perancangan sistem, pengambilan gambar dan penghitungan jarak setelah terdeteksi oleh sensor infra merah, mengirimkan *email* yang berisi notifikasi, gambar, dan *map*.

**BAB IV. IMPLEMENTASI**

Bab ini membahas implementasi dari desain yang telah dibuat pada bab sebelumnya. Penjelasan berupa *code* yang digunakan untuk proses implementasi.

**BAB V. UJI COBA DAN EVALUASI**

Bab ini menjelaskan kemampuan perangkat lunak dengan melakukan pengujian fungsionalitas dan pengujian perfroma dalam beberapa skenario. Pengujian fungsionalitas merupakan pengujian jalannya aplikasi sesai dengan perancangan sistem pada beberapa skenario yang telah ditentukan.

**BAB VI. KESIMPULAN DAN SARAN**

Bab ini merupakan bab terakhir yang menyampaikan kesimpulan dari hasil uji coba yang dilakukan dan saran untuk pengembangan perangkat lunak ke depannya.

# BAB II
# TINJAUAN PUSTAKA

Bab ini berisi penjelasan teori-teori yang berkaitan dengan implementasi *progressive caching* pada CCN untuk keperluan *streaming* video, Penjelasan ini bertujuan untuk memberikan gambaran secara umum terhadap sistem yang dibuat.

## 2.1 Content-Centric Networking

Dasar-dasar arsitektur internet secara umum pada saat ini diciptakan pada tahun 1960-1970. Permasalahan utama dalam jaringan adalah pembagian *resource*, yakni secara *remote* menggunakan peralatan yang jumlahnya sedikit dan mahal. Model komunikasi yang dihasilkan adalah interaksi antara 2 mesin, satu mesin mengakses *resource* dan mesin lain memperbolehkan akses. Dari sana, dihasilkan sebuah protokol yang bernama Internet Protocol (IP) dimana didalam paket dari IP terdapat dua *identifier*, yang pertama adalah asal paket dan yang kedua adalah tujuan paket [1].

Dalam 50 tahun perkembangannya, jaringan beserta komputer dan perangkat lainnya telah menjadi murah dan ada dimana-mana. Konektivitas yang ditawarkan oleh internet yang disertai dengan biaya penyimpanan berperan besar dalam pertumbuhan *content* data yang amat besar, hingga mencapai 500 exabyte, hanya dalam tahun 2008. Orang-orang mengakses internet untuk mendapatkan konten yang diinginkan. Namun pada prakteknya, paradigma komunikasi masih didasrkan oleh lokasi [1].

Hal ini menyebabkan munculnya berbagai masalah dalam jaringan. Ketersediaan konten yang cepat dan *reliable* memerlukan mekanisme-mekanisme yang cukup rumit, membutuhkan persiapan ataupun memakan banyak *bandwidth*, sekuritas yang buruk karena lokasi *device* yang diakses terkadang tidak aman, ataupun ketergantungan akan lokasi mempersulit

proses pemetaan dalam konfigurasi ataupun implementasi dari jaringan.

Solusi yang ditawarkan adalah dengan mengganti paradigma dasar jaringan dari *dimana* (berorientasi utama kepada lokasi) kepada *apa* (berorientasi utama kepada *content*).

*Content-Centric Networking* (CCN) adalah sebuah paradigma arsitektur jaringan dimana pengguna tidak perlu tahu lokasi fisik dari sebuah *content*, tapi pengguna hanya perlu tahu *content* apa yang diinginkan [2]. Salah satu keunggulan utama dari CCN adalah efisiensi distribusi data. Pada CCN, data akan disebar pada *node-node* yang terdapat dalam jaringan dalam bentuk *cache* (*In-Network Storage*) sebagai potongan-potongan data dengan panjang data yang sama (*chunk*) [3]. *Request* pada suatu *node* untuk suatu *content* yang berasal dari *host* yang berbeda (*request aggregation*) akan dikumpulkan pada *Pending Interest Table* (PIT). Kemudian, *request* akan diteruskan pada *node* selanjutnya hingga *content* yang diinginkan dapat ditemukan. *Content* yang telah ditemukan akan diteruskan hingga ke *node* dimana *request aggregation* terjadi. Pada *node* tersebut, akan dilakukan *caching* dan *content* akan diperbanyak sejumlah *request* dari masing-masing *host* yang tadinya dikumpulkan dalam *request aggregation* [2] [4].

### 2.1.1 *Content Store*

*Content Store* merupakan salah satu elemen dari 3 elemen utama pada *node* di jaringan CCN. *Content Store* bertugas untuk menyimpan *content* yang didapatkan melalui proses *caching* pada *node* tersebut. Tidak seperti IP *packet* dimana *packet* IP hanya akan disimpan dalam *buffer* dalam waktu sementara hingga *packet* diteruskan ke *node* lainnya, paket pada CCN didesain secara khusus agar memaksimalkan penggunaan kembali *paket* (satu paket yang ter-*cache* bisa digunakan oleh banyak *node*) [1] [2]. Pada *content store, cache* dalam bentuk *chunk* (potongan data dengan ukuran yang sama) disimpan. Dalam *content store* terdapat sistem *expiration* yang secara otomatis menghapus

*content*. Sistem *expiration* didesain untuk memnyimpan *packet* selama mungkin dan seefisien mungkin (*packet* yang sering diakses memiliki kemungkinan dihapus lebih kecil daripada *packet* yang jarang diakses) dengan kapasitas penyimpanan yang terbatas [1].

### 2.1.2 *Forward Information Base*

*Forward Information Base* digunakan untuk meneruskan *interest packet* menuju sumber potensial dari data (*content provider* maupun *node* yang memiliki *content* yang diinginkan). Pada FIB dari jaringan CCN, FIB dapat mengakses *face* dalam sebuah *list* (lebih dari satu *face*). Hal ini menyebabkab FIB dapat mengakses data lebih dari satu sumber dan melakukan *query* secara paralel [1] [2].

### 2.1.3 *Pending Interest Table*

*Pending Interest Table* mencatat *interest packet* yang diteruskan secara *upstream* (menjauhi asal *interest packet*) sehingga data (*content*) yang diperoleh bisa dikirimkan *downstream* (mendekati asal *interest packet*) menuju *content requester* (*node* yang mengirimkan *interest packet*). Dalam CCN, hanya *interest packet* yang melalui proses *routing* dan saat *interest packet* mencari *content* yang diinginkan, *packet* akan meninggalkan jejak "remah roti" [1] [2]. Jika *content* yang diinginkan telah ditemukan, maka *content* akan dikirimkan kembali kepada *content requester* melalui jejak "remah roti" yang pada jaringan CCN adalah setiap entri dari PIT. Entri dari PIT suati *node* akan dihapus setelah *content* yang diminta *interest packet* yang meninggalkan jejak "remah roti" telak melewati *node* tersebut (*content* "memakan jejak roti"). Entri PIT yang tidak pernah menemui *content* yang diinginkan akan dihapus [1].

## 2.2 Progressive Caching

*Progressive Caching* adalah metode *caching* yang diusulkan oleh Jason Min Wang dan Brahim Bensaou dari Hong Kong University of Science and Technology. *Progressive*

*Caching* merupakan sebuah metode *caching* yang secara progresif melakukan *caching* pada *node-node* di jaringan CCN. Metode ini memanfaatkan fitur *expiration* dari CCN. Pada jaringan CCN, secara otomatis *content* yang paling jarang diakses akan dihapus dari *node* saat kapasitas *node* mulai penuh dan dibutuhkan ruang baru untuk menampung *cache*. Dengan memanfaatkan hal itu, maka penggandaan *cache* yang ditujukan kepada *node* dibawah lokasi *node* yang memiliki *content* yang diinginkan oleh *interest packet*, untuk *content* yang sering diakses oleh pengguna, akan meningkatkan performa jaringan, karena lokasi *content* yang sering diakses menjadi lebih dekat (*hop count* dari *node* yang menerbitkan *interest packet* dengan *node* yang memiliki *content* yang diinginkan akan berkurang) dan juga hal ini akan meningkatkan lokalitas dari *content* yang sering diakses [2].

Metode *caching* yang merupakan bawaan dari CCN, yakni metode LRU (*Least Recently Used*) hanya akan melakukan *cache* pada *node* yang mengakses *content.* Mungkin hal ini bisa terhitung efektif jika hanya 1 *node* yang mengakses *content* tertentu. Namun terkadang, ditemukan bahwa banyak *node* mengakses *content* yang sama [2]. Hal ini akan menjadi kurang efekti karena akan terjadi penggandaan *content* pada *node* yang mengakses *content* yang sama. Untuk itu, metode *progressive caching* akan membantu meningkatkan performa *caching*.

## 2.3 VLC Media Player

VLC Media Player adalah perangkat lunak *open source* yang digunakan untuk memutar media, baik video maupun audio dan juga sebagai *streaming server*. VLC dikembangkan oleh VideoLAN. VLC dapat memutar berkas audio dan video dengan berbagai macam metode kompresi maupun sebagai protokol yang digunakan untuk *streaming* video [5]. Proyek VideoLAN diawali dari proyek akademis pada tahun 1996. VLC merupakan singkatan dari VideoLAN Client. VLC pertama kali dikembangkan di Ecole Centrale, Paris. VLC dikembangkan oleh para kontributor yang tersebar di seluruh dunia dan

dikoordinasikan oleh VideoLAN, sebuah organisasi *non-profit*. VLC merupakan *packet-based media player*. VLC dapat memutar hampir semua format *file* multimedia. VLC dapat digunakan hampir di seluruh *platform* seperti Windows, OS X, iOS, Linux, Android, BSD, BeOS, OS/2, Solaris, Syllable dan QNX [5].

VLC dapat dieksekusi melalui *terminal* pada sistem operasi Linux. Pada Kode Sumber 2.1 dijelaskan cara eksekusi VLC melalui *terminal*

```
vlc [path dari file yang akan diakses]
```

**Kode Sumber 2.1 Perintah eksekusi VLC**

Pada CCNx, perintah eksekusi *file* untuk mengakses *content* pada repositori CCNx menggunakan VLC adalah sebagai berikut:

```
vlc ccnx:///[path dari file yang akan diakses]
```

**Kode Sumber 2.2 Perintah eksekusi VLC pada CCNx**

## 2.4 CCNx

CCNx merupakan sebuah *platform* CCN yang dikembangkan oleh PARC (Palo Alto Research Center). CCNx menggunakan 2 jenis *messages* untuk mentransfer data. Sebuah *request message* yang disebut *Interest* dan sebuah *response message* yang terenkapsulasi dan disebut *Content Object.* Sebagai bentuk implementasi dari CCN, CCNx memiliki berbagai tambahan fitur untuk menunjang operasi dari CCNx. Antara lain, plugin untuk Wireshark yang dapat menampilkan paket CCN pada saat *packet capture* dan plugin VLC untuk mengimplementasikan proses *streaming* pada CCNx [3].

CCNx berawal dari presentasi Van Jacobson, seorang peneliti yang berasal dari Palo Alto Research Center, mengenai paradigma jaringan baru di Google. Proyek CCN secara resmi diluncurkan oleh Palo Alto Research Center pada 2007. Pada 2009, Palo Alto Research Center mengumumkan bahwa proyek CCNx telah menyertakan spesifikasi operasi CCNx pada situs

web dari CCNx, yakni ccnx.org pada September 2009. Desain awal CCNx dijelaskan pada sebuah *paper* yang dipublikasikan pada International Conference on Emerging Networking Experiments (CoNEXT) pada September 2009. [1].

CCNx merupakan protokol yang menyediakan layanan transmisi data yang *location independent* (tidak memiliki ketergantungan terhadap lokasi) untuk *data packet* yang memiliki *name*. Beberapa layanan mencakup meneruskan *packet* secara *multihop* (lebih dari satu *node*), pengontrolan aliran data, mekanisme pengantaran data menggunakan metode *multicast* yang menggunakan penyimpanan *buffer* pada *node* yang tersedia dan kemampuan meneruskan *packet* secara *multipath* (melalui lebih dari 1 jalur) yang bebas *loop*. Aplikasi menjalankan protokol CCNx melalui lapisan komunikasi tingkat rendah yang dapat mentransmisikan paket. Tidak ada batasan mengenai sifat dari lapisan komunikasi tingkat bawah. Lapisan komunikasi yang digunakan mungkin lapisan *physical, transport* ataupun *network* pada jaringan. Sebagai contoh, aplikasi akan menjalankan CCNx protokol diatas UDP untuk mengambil keuntungan dari konektivitas pada protokol IP. Karena penamaan *content* tidak tergantung dari lokasi, maka *content* bisa disimpan dimana saja dalam jaringan, hal ini merupakan bentuk layanan distribusi *content* yang bisa dibilang efektif [6].

CCNx mendukung banyak aplikasi jaringan. Sewajarnya, CCNx secara otomatis mendukung aplikasi yang menggunakan *content* yang tersimpan pada *node*, seperti distribusi video maupun berkas dokumen. Namun, CCNx juga mendukung komunikasi secara *real-time* dan protokol *discovery* (protokol yang digunakan untuk mencari *node* yang belum terdaftar pada jaringan CCNx) dan bisa digunakan untuk berkomunikasi antara *host* menggunakan koneksi TCP. Protokol mendukung berbagai jenis aplikasi dengan menyerahkan mekanisme *naming* (penamaan/pemberian pengenal) pada *content* kepada aplikasi yang bersangkutan. Protokol CCNx dirancang untuk komunikasi *end to end* (antara peminta *content* dengan penyedia *content*) [6].

Secara umum, terdapat dua jenis *message* (data yang berlalu-lalang dalam jaringan CCNx) yakni *Interest Message* dan *Content Object*.

### 2.4.1 *Interest Message*

*Interest Message* merupakan *packet* yang digunakan CCNx untuk meminta data berdasarkan *name* (sebuah kumpulan *string* yang berlaku sebagai pengenal dari masing-masing *content*, *name* bersifat unik). Sebuah *interest message packet* dapat mengenali *content* yang akan diminta secara spesifik. Selain itu, sebuah *interest message* dapat menyediakan sebuah *name* dan persyaratan lain untuk membatasi data mana saja yang dapat diterima [6].

### 2.4.2 *Content Object*

*Content Object* adalah sebuah *packet* yang digunakan sebagai sarana untuk mensuplai data berdasarkan *name*. Sebuah *content object message* tidak hanya mengandung data yang diminta oleh *interest message* tapi juga mengandung *name* yang berlaku sebagai pengenal, penanda kriptografis, dan pengenal bagi penanda (*signer*) beserta informasi lain mengenai proses penandaan (*signing*) [6].

## 2.5 Wireshark

Wireshark adalah sebuah perangkat lunak *packet analyzer open-source*. Wireshark umumnya digunakan untuk analisis jaringan, *troubleshooting*, Wireshark merupakan perangkat lunak *cross platform*. Wireshark dapat dijalankan pada berbagai *platform*, seperti sistem operasi berbasis Unix dan Windows [6].

Wireshark, yang pada saat itu bernama Ethereal, dikembangkan oleh Gerald Combs, lulusan Universitas Missouri-Kansas City. Gerald bekerja di sebuah perusahan ISP kecil. Pada saat itu, sekitar tahun 90an, perangkat lunak *protocol analysis* berharga sangat mahal dan tidak cocok dengan *platform* utama perusahaan. Lalu, Gerald memulai menulis Ethereal dan merilis versi pertama pada 1998 [7].

Berikut ini adalah fitur-fitur dari Wireshark:

- Data dapat ditangkap "*from the wire*" (saat data melintasi jaringan) melalui jaringan secara langsung ataupun membaca *file* dari *packet* yang telah ditangkap sebelumnya
- Penangkapan data dan pembacaan data secara langsug dapat dilakukan melalui berbagai jenis jaringan
- Data dari *packet* jaringan yang tertangkap dapat dijelajahi melalui GUI yang disediakan oleh Wireshark maupun melalui *terminal*
- *Plugins* dapat dikembangkan untuk menganalisa protokol-protokol baru
- Panggilan VoIP pada jaringan dapat ditangkap. Jika panggilan melalui proses *encoding* yang kompatibel, maka panggilan VoIP dapat diputar ulang.

## 2.6 C

Bahasa pemrograman C merupakan bahasa pemrograman yang dikembangkan oleh Dennis Ritchie antara tahun 1969 hingga tahun 1973 di Laboratorium AT&T Bell. Bahasa pemrograman C merupakan salah satu bahasa pemrograman yang paling banyak digunakan sepanjang waktu dan *compiler* C sudah disertakan di sebagian besar sistem operasi. Beberapa bahasa pemrograman lain, secara langsung maupun tak langsung meminjam fitur-fitur dari bahasa pemrograman C. Bahkan beberapa memiliki kesamaan yang cukup banyak jika dilihat secara struktural maupun sintaksis.

## 2.7 Ubuntu

Ubuntu adalah sebuah sistem operasi Linux berbasis Debian yang menggunakan Unity sebagai *desktop environment*. Ubuntu dinamai berdasarkan filosofi dari Afrika Selatan yakni filosofi kemanusiaan., yang bisa diartikan sebagai sebuah

kepercayaan ikatan universal yang menghubungkan semua manusia. [8]

Pengembangan Ubuntu dipimpin oleh Canonical Ltd. yang dipimpin oleh pengusaha Afrika Selatan yang bernama Mark Shuttleworth. Perusahaan ini berbasis di Britania Raya. Canonical mendapatkan keuntungan dari dukungan teknis dan layanan lain yang terkait dengan Ubuntu. Proyek Ubuntu ini didasarkan dari prinsip pengembangan *Open-Source* dimana orang-orang dapat menggunakan perangkat lunak secara gratis, mempelajari cara kerjanya, dan menyebarkannya [9].

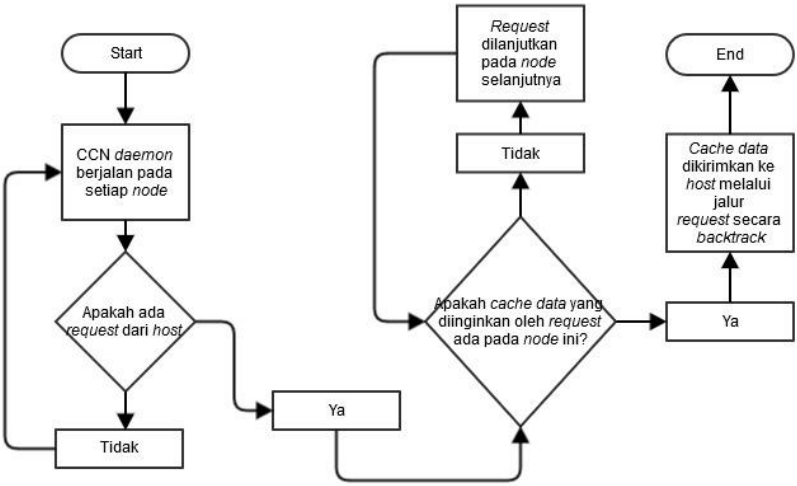
# BAB III
# DESAIN DAN IMPLEMENTASI

Perancangan merupakan bagian penting dari pembuatan perangkat lunak yang berupa perancanaan-perancanaan secara teknis mengenai sistem yang akan dibuat. Bab ini secara khusus akan menjelaskan perancangan sistem yang dibuat dalam Tugas Akhir ini. Penjelasan mencakup deskripsi umum aplikasi hingga perancangan proses, alur sistem.

## 3.1 Deskripsi Sistem Secara Umum

Tugas Akhir ini akan membuat sebuah klien pada perangkat untuk mengakses *streaming* menggunakan arsitektur jaringan CCN. *Host* akan meminta video dengan *name* tertentu. Kemudian, perangkat akan mengirimkan paket *interest* ke dalam jaringan CCN.

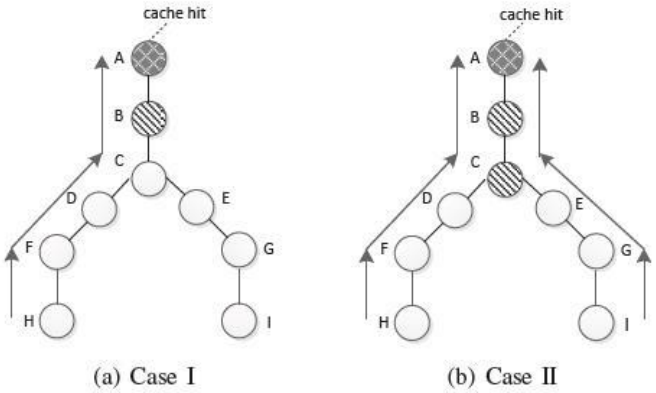
Pada jaringan CCN, *node* yang menerima paket *interest* akan mengecek apakah *node* tersebut memiliki konten yang sesuai dengan yang diinginkan oleh paket *interest* pada *content store*. Jika *node* tersebut memiliki konten yang diinginkan, maka secara langsung *node* tersebut akan mengirimkan konten menuju asal dari paket *interest*. Jika tidak, maka *node* tersebut akan mengecek *Pending Interest Table* (PIT) jika ada permintaan untuk konten yang sama. Jika ada, maka PIT akan mengumpulkan semua permintaan yang serupa dan melanjutkan permintaan ke *node* selanjutnya. Jika *node* menerima konten yang diinginkan, maka *node* akan menyebarkan konten berdasarkan semua *host* yang meminta konten tersebut berdasarkan data pada PIT




**Gambar 3.1 Diagram alir proses CCN**

Gambar 3.1 menunjukkan diagram alir dari proses kerja dari sistem *caching* CCNx. Awalnya, CCN *daemon* pada setiap *node* harus sudah berjalan. Jika pada salah satu *host* dari jaringan CCN melakukan *request*, maka *request* tersebut akan dilanjutkan pada *node* terdekat. Setelah itu, akan dilakukan pengecekan apakah pada *node* tersebut terdapat *cache data* yang diinginkan oleh *host*. Jika ada, maka *cache data* akan dikirimkan kembali kepada *host* yang mengirimkan *request*. Pencarian jalur menuju *host* yang melakukan *request* dilakukan melalui proses *backtracking* dari jalur *request* yang dikirimkan *host.*Jika *cache data* yang dimaksud tidak terdapat pada *node* ini, maka *request* akan diteruskan ke *node* lainnya.

Secara singkat, *Progressive Caching* adalah proses *caching* yang secara teori dapat membantu meningkatkan *content hit rate* pada jaringan CCN dengan secara aktif melakukan *cache* pada *node* yang terletak di bawah *node* yang mengalami *content hit*. Hal ini akan meningkatkan lokalitas *content* dan mengurangi *hop count* dari *content* yang sering diakses oleh pengguna.




**Gambar 3.2 Mekanisme *caching* pada *Progressive Caching***

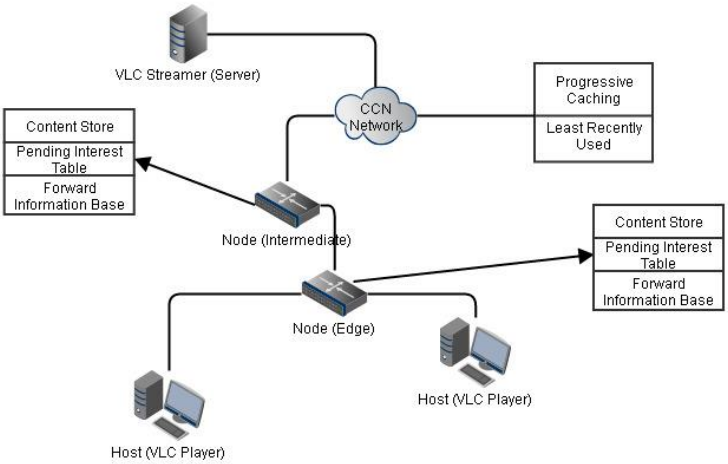
Pada Gambar 3.1, *progressive caching* dapat dicapai dengan *node* yang mengalami *content hit* akan melakukan proses *caching* dari *content* yang mengalami *hit* pada *node* dibawahnya. Pada *Progressive Caching*, terdapat 2 jenis *node.* Yakni, *edge node* dan *intermediate node*. *Edge node* merupakan *node* yang berbatasan langsung dengan pengguna. Sedangkan *intermediate node* merupakan *node* yang menghubungkan antara *node* yang mengalami *content hit* dengan *edge node* [2]. Mekanisme *caching* pada *intermediate node* dijelaskan pada *pseudocode* yang terdapat pada Gambar 3.2 dibawah ini.

```
Caches are organized into multiple queues: Q[K]
Initially L ← 0
The current request matches chunk p;
IF p is already in cache space
        Remove it from current queue Q[H_p];
        Append it to tail of Q[H'_p] where H'_p = L + c_p;
IF p is nor in the cache space
        IF eviction is needed:
                Let L ← arg min Q[k] ≠ Ø;
                        k ∈ K
                Evict data chunk q from the head of Q[L]
                i.e., the queue with H_q = L;
        append p to the tail of Q[H_p], where H_p = L + c_p
```

**Gambar 3.3 *Pseudocode* mekanisme *caching* pada *intermediate node***

## 3.2 Arsitektur Umum Sistem

Rancangan arsitektur sistem untuk *progressive caching* dan LRU ditunjukkan seperti pada Gambar 3.4



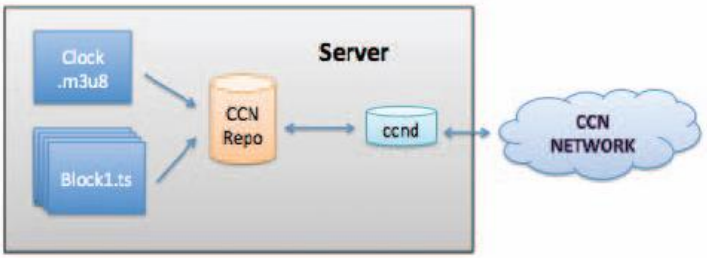

**Gambar 3.4 Arsitektur umum jaringan CCN**

Sebelum membahas lebih jauh mengenai arsitektur jaringan untuk CCN, ada 2 istilah utama yang perlu dipahami.

- *ccnd*
  *ccnd* adalah sebuah CCN *routing daemon*. Masing masing *node* dari CCN harus menjalankan *ccnd*
- *ccn-repo*
  Sebuah aplikasi berbasis CCN yang digunakan sebagai sarana penyimpanan data [4]

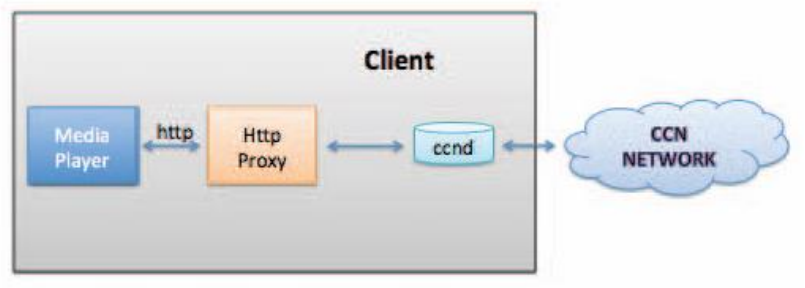
Pada penerapannya, arsitektur CCN untuk *live streaming* memiliki dua elemen utama dari CCN yakni pada *client-end* dan -*server-end*

a. *Server-end architecture*
   Sama seperti HTTP *live streaming* pada umumnya, masukan video dipecah menjadi beberapa bagian dan akan dibangkitkan berkas indeks sebagai daftar putar yang digunakan untuk menyusun kembali masukan video yang telah dipecah [4]



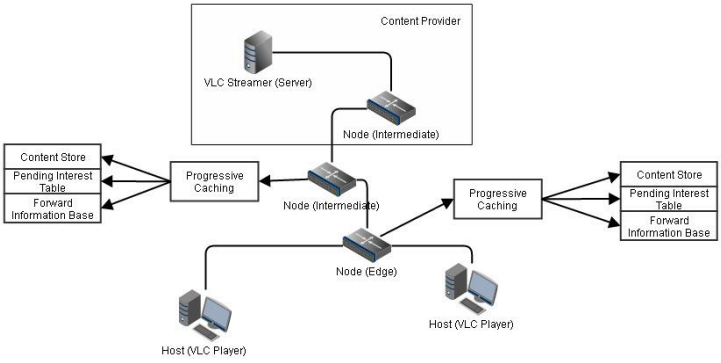

**Gambar 3.5 Arsitektur *server-end***

b. *Client-end architecture*
   Implementasi pada sisi klien dijalankan pada *host* yang berbasis perangkat lunak. Pada perangkat lunak disertakan juga *media player* yang mendukung HTTP *live streaming*. HTTP *proxy* akan menerjemahkan permintaan HTTP menjadi permintaan CCN. Lalu, *ccnd* akan meneruskan permintaan yang berbentuk paket *interest*. Kemudian, *proxy* akan menerjemahkan respon CCN menjadi respon HTTP yang akan dikirim kembali ke *media player* [4].




**Gambar 3.6 Arsitektur *client-end***

*Progressive caching* akan dijalankan di masing-masing *node* pada jaringan CCN. Implementasi dilakukan dengan mengubah mekanisme pengiriman *content* pada *Content Store, Forward Information Base,* dan *Pending Interest Table*. *Progressive caching* akan memodifikasi pengiriman *content* dengan menambahkan pengiriman *content* yang sama pada *node* yang terletak di bawah *node* yang mengirimkan *content object*. Program akan membaca informasi dari *interest packet* mengenai jalur pengiriman *content object* yang didapatkan melalui entri *interest packet* yang melewati FIB pada masing-masing *node* di jaringan CCN.




**Gambar 3.7 Desain implementasi *progressive caching***

# BAB IV
# IMPLEMENTASI

Pada bab ini akan dibahas mengenai implementasi yang dilakukan berdasarkan rancangan yang telah dijabarkan pada bab sebelumnya. Sebelum penjelasan implementasi akan ditunjukkan terlebih dahulu lingkungan untuk melakukan implementasi.

## 4.1 Lingkungan Implementasi

Dalam merancang perangkat lunak ini digunakan beberapa perangkat pendukung pengembangan sebagai berikut.

### 4.1.1 Lingkungan Implementasi Perangkat Keras

Perangkat keras yang digunakan dalam pengembangan sistem adalah komputer dan perangkat Raspberry Pi. Spesifikasi dari perangkat tersebut adalah sebagai berikut:

- Komputer jinjing Acer Aspire 4741 Intel Core i3-350M 2.26 GHz 3MB L3 cache, DDR3 RAM 4 GB
- 10 buah PC Intel Core 2 Duo RAM 2 GB HD 320GB LAN Card 100 MBps

### 4.1.2 Lingkungan Implementasi Perangkat Lunak

Spesifikasi perangkat lunak yang digunakan dalam pengembangnan sistem adalah sebagau berikut:

- Microsoft Windows 7 Ultimate 64 bit sebagai sistem operasi.
- Ubuntu 14.04 Trusty Tahr sebagai sistem operasi untuk implementasi CCNx
- Modul CCNx
- Wireshark 1.8.6 sebagai *Network Analyzer*

## 4.2 Implementasi CCN

Pada subbab ini akan dijelaskan bagaimana implementasi CCN pada perangkat keras dan perangkat lunak

### 4.2.1 Implementasi CCN pada Perangkat Keras

- Proses pemasangan dependensi

  Untuk menunjang CCN agar dapat berjalan dengan baik pada perangkat keras, diperlukan beberapa dependensi yang terlebih dahulu harus terpasang pada perangkat keras. Dependensi tersebut antara lain:

  1. libssl-dev
  2. openssl
  3. expat
  4. libexpat-dev
  5. libpcap
  6. libxml2
  7. ant
  8. JDK 1.7

  Proses pemasangan dependensi tersebut dapat dilakukan seperti ini :

```
sudo apt-get install [nama dependensi]
```

**Kode Sumber 4.1 Perintah instalasi dependensi**

Jika semua dependensi telah terinstall, selanjutnya adalah proses instalasi CCNx. CCNx bisa diunduh di http://www.ccnx.org/software-download-information-request/download-releases/. Setelah itu, ekstrak *source code* yang telah diunduh dengan menggunakan perintah berikut:

```
tar –jxvf ccnx-0.8.2.tar.gz
```

**Kode Sumber 4.2 Perintah ekstraksi *file* CCNx**

Setelah proses ekstraksi selesai, masuk ke dalam *folder* yang telah diekstrak dengan menggunakan perintah berikut

```
cd ccnx-0.8.2
```

**Kode Sumber 4.3 Perintah pindah direktori**

Setelah berhasil masuk ke dalam *folder* ccnx-0.8.2, jalankan perintah berikut untuk menyelesaikan proses instalasi CCNx

```
./configure
make
make install
```

**Kode Sumber 4.4 Perintah instalasi CCNx**

Dependensi selanjutnya yang harus terinstal adalah vlc, libvlc-dev dan libvlccore-dev. Instalasi dapat dilakukan dengan perintah berikut

```
sudo apt-get install vlc
sudo apt-get install libvlc-dev
sudo apt-get install libvlccore-dev
```

**Kode Sumber 4.5 Perintah instalasi dependensi VLC**

- Konfigurasi CCNx

  Langkah pertama dalam konfigurasi CCNx adalah menjalankan *daemon* dari CCNx yang juga dikenal sebagai CCND. Untuk menjalankan CCND, masuk ke dalam *folder* ccnx-0.8.2 dan ketikkan perintah berikut

```
bin/ccndstart
```

**Kode Sumber 4.6 Perintah menjalankan *daemon* CCNx**

Langkah selanjutnya adalah menentukan letak *folder* untuk menempatkan *cache* pada CCNx. Untuk itu, ketik perintah berikut pada *terminal*

```
export CCNR_DIRECTORY=[path direktori]
```

**Kode Sumber 4.7 Perintah ekspor direktori repositori CCNx**

Setelah itu, jalankan repositori dari CCNx dengan mengetik perintah berikut

```
bin/ccnr
```

**Kode Sumber 4.8 Perintah menjalankan repositori CCNx**

Untuk menghubungkan *node* CCN dengan *node* lain, harus ada proses penambahan repositori dari *node* lain. Untuk melakukan hal ini, ketik perintah berikut pada *terminal.*

```
bin/ccndc add ccnx:[path dari repositori
yang ingin ditambahkan] udp [alamat IP
dari node yang ingin ditambahkan]
```

**Kode Sumber 4.9 Perintah menghubungkan *node* pada repositori CCNx**

Untuk menambahkan *file* ke dalam repositori CCN, ketikkan perintah ini pada *terminal*

```
ccnputfile ccnx:[path dan nama file yang
ingin ditambahkan pada repositori CCN]
[path dari file yang ingin ditambahkan]
Contoh: ccnputfile
ccnx:/ccnx.org/video/tes1.flv
/home/user/Videos/tes1.flv
```

**Kode Sumber 4.10 Perintah menambahkan *file* pada repositori CCNx**

Setelah itu, VLC Media Player memerlukan *plugin* khusus CCN agar dapat memutar video dari repositori CCN. Untuk itu, masuklah ke dalam *folder* apps/vlc pada *folder* ccnx-0.8.2. Kemudian, jalankan *file* Makefile.Linux untuk membuat *plugin file* CCN untuk VLC Media Player dengan mengetik perintah sebagai berikut

```
make –f Makefile.Linux
```

**Kode Sumber 4.11 Perintah *makefile* untuk *plugin* VLC *streaming* pada CCNx**

Pada *folder* apps/vlc, akan muncul 2 *file* baru, yakni libaccess_ccn_plugin.o dan libaccess_ccn_plugin.so. Pindahkan *file* libaccess_ccn_plugin.so ke *folder plugin* VLC Media Player dengan mengetik perintah berikut

```
cp [path dari libaccess_ccn_plugin.so]
[path folder plugin VLC]
Contoh: cp /home/user/Downloads/ccnx-
0.8.2/apps/vlc/libaccess_ccn_plugin.so
/usr/lib/vlc/plugins/codec/
```

**Kode Sumber 4.12 Perintah salin file *plugin* dari direktori apps/vlc pada CCNx ke direktori *library plugin* VLC**

Untuk mengakses video pada repositori CCN, ketikkan perintah berikut di *terminal*. Pastikan garis miring setelah ccnx: pada perintah di bawah tertulis sebanyak 3 kali

```
vlc ccnx:///[path repositori dari video
yang akan diakses]
Contoh: vlc
ccnx:///ccnx.org/video/tes1.flv
```

**Kode Sumber 4.13 Perintah *streaming* VLC pada CCNx**

- Konfigurasi Wireshark

  Proses instalasi Wireshark dimulai dengan mengunduh *tarball* dari www.wireshark.org. Pastikan versi yang diunduh adalah Wireshark versi 1.8.6. Setelah diunduh ekstrak *tarball* dengan menggunakan perintah berikut

```
tar –jxvf wireshark wireshark-
1.8.6.tar.bz2
```

**Kode Sumber 4.14 Perintah ekstraksi direktori Wireshark**

Setelah itu, salin *flie plugin* dari ccnx-0.8.2/apps/wireshark/ccn ke *folder plugin* Wireshark dengan perintah berikut

```
cp –r [path direktori plugin CCN] [path
direktori plugin Wireshark]
Contoh: cp –r /home/user/Downloads/ccnx-
0.8.2/apps/wireshark/ccn
/home/user/Downloads/wireshark-
1.8.6/plugins
```

**Kode Sumber 4.15 Perintah salin *file plugin* CCN pada direktori *plugin* Wireshark**

Setelah itu, salin *patch* pada direktori ccnx-0.8.2/apps/wireshark pada *folder* wireshark-1.8.6 dengan perintah berikut

```
cp [path direktori patch CCN untuk
Wireshark] [path direktori Wireshark]
Contoh: cp /home/user/Downloads/ccnx-
0.8.2/apps/wireshark/wireshark-1.8.6.patch
/home/user/Downloads/wireshark-1.8.6
```

**Kode Sumber 4.16 Perintah salin *patch* pada Wireshark**

Kemudian, jalankan *patch* pada direktori wireshark-1.8.6 dengan mengetik perintah berikut

```
patch –p1 < wireshark-1.8.6.patch
```

**Kode Sumber 4.17 Perintah eksekusi *patch* Wireshark**

Setelah itu, jalankan *file* autogen.sh pada direktori wireshark-1.8.6 dengan mengetik perintah berikut

```
./autogen.sh
```

**Kode Sumber 4.18 Perintah eksekusi *file* autogen.sh**

Setelah itu, akhiri proses instalasi dengan mengetik perintah berikut

```
./configure
make
make install
```

**Kode Sumber 4.19 Perintah instalasi Wireshark**

Untuk menjalankan Wireshark, ketik perintah berikut

```
sudo ldconfig
wireshark
```

**Kode Sumber 4.20 Perintah menjalankan Wireshark**

### 4.2.2 Implementasi CCN pada perangkat lunak

Implementasi kode sumber untuk *progressive caching* pada masing-masing *node* di CCNx adalah pada *daemon* dari masing-masing *node*. Kode sumber yang mengatur mengenai mekanisme kerja *daemon* adalah **ccnd.c**. Pada kode sumber ini, dilakukan implementasi agar *content* tidak hanya dikirimkan pada *head of the queue*, tapi juga dikirim kepada *node* yang terletak di bawah dari *node* yang mengirimkan *content object*. Hal ini dicapai dengan menggantai parameter *face* pada kode sumber menjadi alamat dari *node* yang terletak di bawah *node* yang mengirimkan *content object*. Adapun implementasi pada kode sumber dapat dilihat sebagai berikut :

```
/**
 * Queue a ContentObject to be sent on a face.
 */
static int
face_send_queue_insert(struct ccnd_handle *h,
                       struct face *face, struct content_entry
*content)
{
    int ans;
    int delay;
    int n;
    enum cq_delay_class c;
    enum cq_delay_class k;
    struct content_queue *q;

    if (face == NULL || content == NULL || (face->flags &
CCN_FACE_NOSEND) != 0)
        return(-1);
    c = choose_content_delay_class(h, face->faceid, content-
>flags);
    if (face->q[c] == NULL)
        face->q[c] = content_queue_create(h, face, c);
    q = face->q[c];
    if (q == NULL)
        return(-1);
    /* Check the other queues first, it might be in one of them */
    for (k = 0; k < CCN_CQ_N; k++) {
        if (k != c && face->q[k] != NULL) {
            ans = ccn_indexbuf_member(face->q[k]->send_queue,
content->accession);
            if (ans >= 0) {
                if (h->debug & 8)
                    ccnd_debug_content(h, __LINE__,
"content_otherq", face,
                                       content);
                return(ans);
            }
        }
    }
    n = q->send_queue->n;
    ans = ccn_indexbuf_set_insert(q->send_queue, content-
>accession);
    if (n != q->send_queue->n)
        content->refs++;
    if (q->sender == NULL) {
        delay = randomize_content_delay(h, q);
        q->ready = q->send_queue->n;
        q->sender = ccn_schedule_event(h->sched, delay,
```

```
                                        content_sender, q, face-
>faceid);
        if (h->debug & 8)
            ccnd_msg(h, "face %u q %d delay %d usec", face-
>faceid, c, delay);
    }
    return (ans);
}
```

**Kode Sumber 4.21 Kode Sumber pada ccnd.c yang digunakan untuk mengirimkan *content object* menuju *node* yang mengirimkan *interest packet***

Untuk menghasilkan sebuah *face*, dijalankan salah satu proses *daemon* dari CCNx, yakni ccndc. Adapun *syntax* dari ccndc akan dijelaskan sebagai berikut

```
ccndc [-v] (create|destroy) (udp|tcp) host [port
[flags [mcastttl [mcastif]]]]
```

**Kode Sumber 4.22 *Syntax* dari ccndc untuk menciptakan *face***

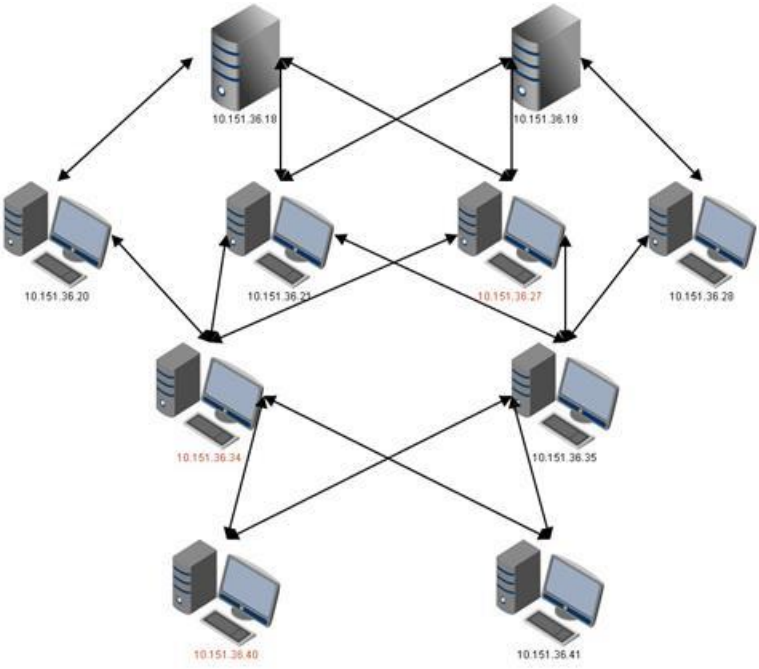
Kode Sumber 4.22 menjelaskan tentang ccndc dan parameter yang digunakan untuk menciptakan *face* baru. Adapun ccndc akan dijalankan pada setiap *node* dengan parameter *host* yang diisi adalah alamat IP dari *node* yang terletak dibawahnya. Setelah itu, didapatkan *face* yang diinginkan dan kemudian *faceid* dari *face* yang baru saja diciptakan digunakan sebagai parameter masukan dari fungsi *content_sender* yang dijalankan pada Kode Sumber 4.21

# BAB V
# UJI COBA dan EVALUASI

Pada bab ini akan dijelaskan uji dari segi fungsionalitas dan performa daris sistem yang telah dirancang. Uji coba fungsionalitas dan performa akan dibagi ke dalam beberapa scenario uji coba.

## 5.1 Lingkungan Uji Coba

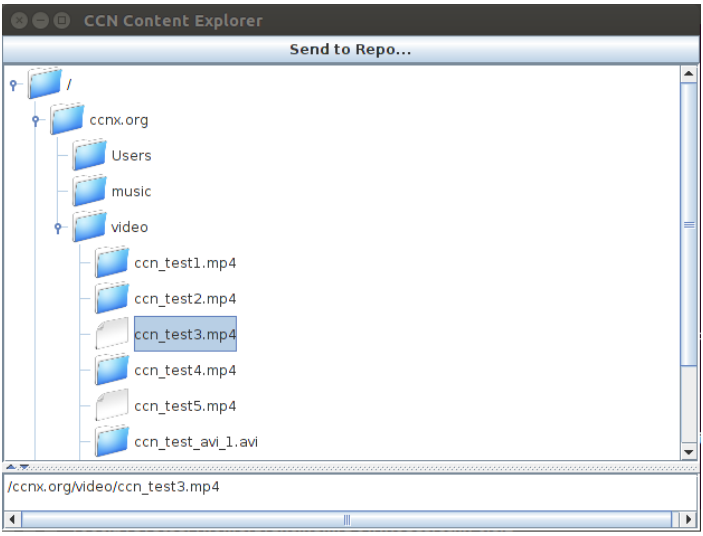
Pada subbab ini, dijelaskan mengenai gambaran dari lingkungan yang digunakan sebagai uji coba sistem. Uji coba akan dilakukan pada Laboratorium Arsitektur dan Jaring\an Komputer, Teknik Informatika ITS. Uji coba dilakukan dengan menggunakan 10 PC yang terhubung dengan jaringan CCN dengan topologi pada gambar di bawah ini. *Node* dengan warna biru merupakan *node* yang didalamnya terpasang Wireshark untuk keperluan analisis paket. Pada Gambar 5.1 ditampilkan arsitektur jaringan pada Lab AJK yang digunakan untuk implementasi CCNx. *Node* dengan warna *font* merah pada nomor IP merupakan *node* yang digunakan untuk uji coba. Pada *node* ini telah terpasang Wireshark yang akan digunakan untuk menjalankan *packet capture*.




**Gambar 5.1 Topologi jaringan uji coba**

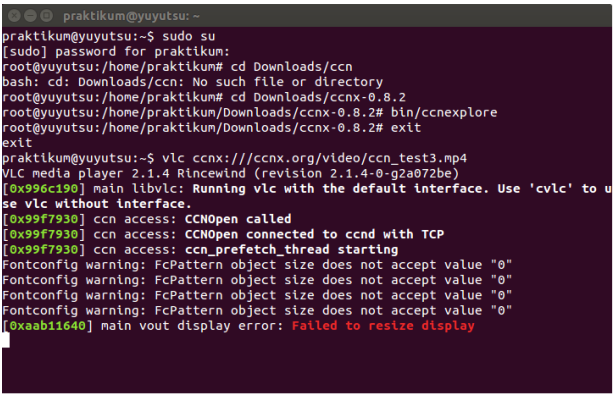
## 5.2 Uji Coba Fungsionalitas Sistem

Uji coba fungsionalitas merupakan pengujian terhadap jalannya fungsi utama dari sistem ini, yakni *streaming* video melalui jaringan CCN. Sebelum itu, harus dipastikan bahwa semua *service* dari CCNx (**ccnd** dan **ccnr**) harus dijalankan terlebih dahulu. Dilakukan pengecekan terhadap konten repositori CCNx menggunakan perintah bin/ccnexplore




**Gambar 5.2 *File* yang tersimpan di repositori CCNx**

Selanjutnya, *streaming* video di CCNx dijalankan melalui aplikasi VLC. Gambar 5.2 dan 5.3 dibawah akan menunjukkan hasil *streaming* melalui jaringan CCN. Eksekusi *streaming* pada *terminal* dilakukan dengan menjalankan aplikasi VLC dengan menggunakan parameter alamat URI pada direktori di repositori CCNx. Pada Gambar 5.2, terlihat eksekusi *streaming* dari *terminal*. Pada Gambar 5.3, merupakan hasil *streaming* pada jaringan CCN:




**Gambar 5.3 Eksekusi *streaming* pada CCNx menggunakan VLC Player**


**Gambar 5.4 Video hasil *streaming* pada CCNx**

## 5.3 Uji coba Performa

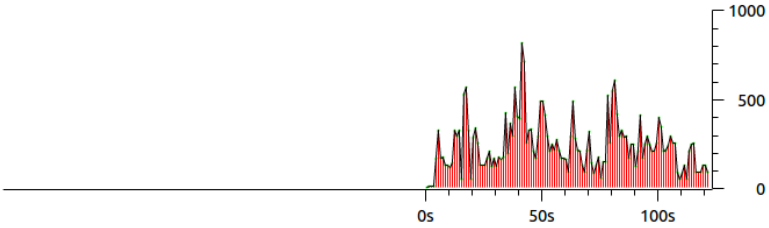
Uji coba performa pertama dilakukan untuk mengamati peningkatan performa video *streaming* menggunakan CCNx yang dapat diamati dari perbandingan jumlah *traffic* pada saat sebelum *caching* dan sesudah *caching*. Pada uji coba *streaming* kedua, *traffic* jaringan seharusnya lebih sedikit bila dibandingkan dengan uji coba *streaming* pertama. Karena pada uji coba kedua, telah terjadi proses *caching* pada *node* antara *content provider* dengan *host*. Uji coba dilakukan dengan menggunakan 5 jenis ekstensi *file* yakni avi, mkv, flv, mp3 dan mp4. Performa dihitung dengan membandingkan jumlah paket yang tertangkap saat uji coba.

**Tabel 5.1 Spesifikasi *file* uji coba dengan ekstensi .flv**

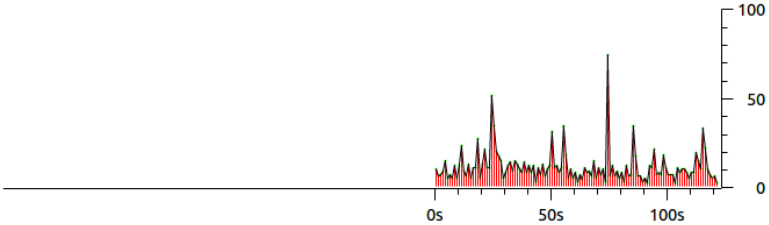
| | |
|---|---|
| Nama *file* | Ccn_test_flv_1.flv |
| Ukuran | 23.7 MB |
| Durasi *file* | 02.16 |
| Lama tes | 2 Menit |
| Parameter filter paket grafik | Black Line : ip.dst==alamat host<br>Red Impulse : udp.port==9695<br>Green Dot : Protokol ccn |
| Sumbu X | Durasi penangkapan paket |
| Sumbu Y | Jumlah paket |

Tabel 5.1 diatas menjelaskan tentang spesifikasi *file* yang akan digunakan untuk uji coba beserta keterangan dari *graph* yang digunakan sebagai alat bantu untuk menjelaskan data secara terperinci. *File* yang digunakan untuk uji coba adalah *file* dengan ekstensi .flv

10.151.36.27



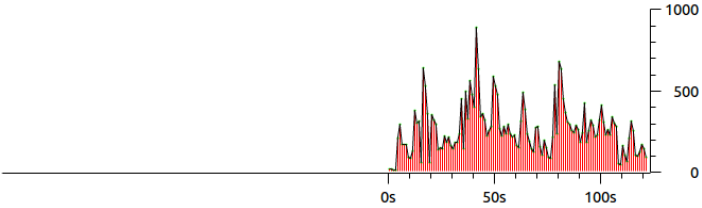

**Gambar 5.5 *Traffic* paket pada *node* 10.151.36.27 pada percobaan pertama**



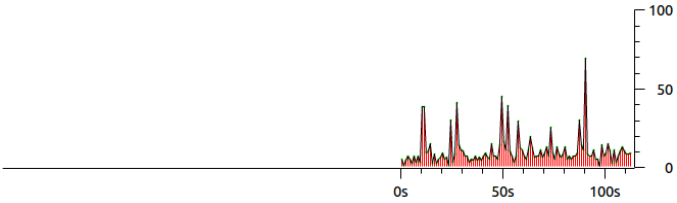

**Gambar 5.6 *Traffic* paket pada *node* 10.151.36.27 pada percobaan kedua**

Dari Gambar 5.5 dan 5.6, ditemukan bahwa terjadi perbedaan *traffic* yang cukup signifikan antara percobaan pertama dan percobaan kedua. Percobaan kedua menghasilkan *traffic* yang lebih sedikit daripada percobaan pertama pada *node* 10.151.36.27.

10.151.36.34



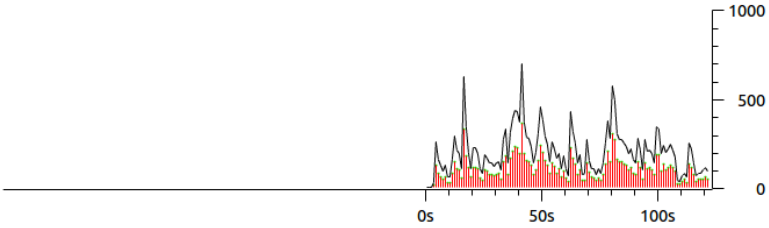

**Gambar 5.7 Traffic paket pada node 10.151.36.34 pada percobaan pertama**



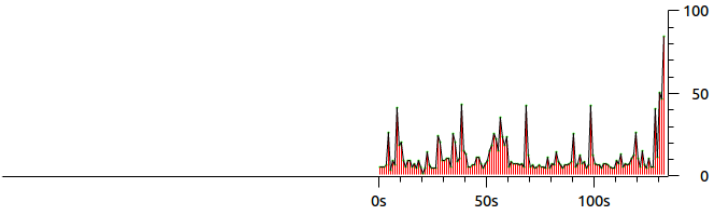

**Gambar 5.8 *Traffic* paket pada *node* 10.151.36.34 pada percobaan kedua**

Dari Gambar 5.7 dan 5.8, ditemukan bahwa terjadi perbedaan *traffic* yang cukup signifikan antara percobaan pertama dan percobaan kedua. Percobaan kedua menghasilkan *traffic* yang lebih sedikit daripada percobaan pertama pada *node* 10.151.36.34.

10.151.36.40




**Gambar 5.9 *Traffic* paket pada *node* 10.151.36.40 pada percobaan pertama**




**Gambar 5.10 *Traffic* paket pada *node* 10.151.36.27 pada percobaan kedua**

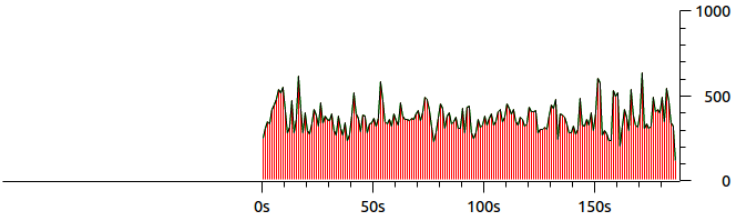
Dari Gambar 5.9 dan 5.10, ditemukan bahwa terjadi perbedaan *traffic* yang cukup signifikan antara percobaan pertama dan percobaan kedua pada *node* 10.151.36.40. Percobaan kedua menghasilkan *traffic* yang lebih sedikit daripada percobaan pertama Dapat disimpulkan bahwa *streaming* video pada jaringan CCN menggunakan *file* dengan .flv telah berhasil

**Tabel 5.2 Spesifikasi *file* uji coba dengan ekstensi .avi**

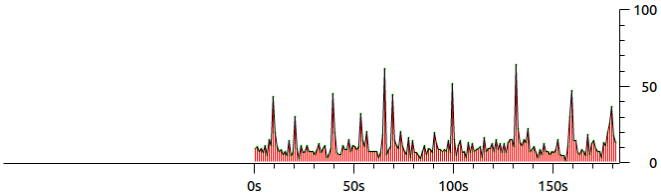
| | |
|---|---|
| Nama *file* | ccn_test_avi_1.avi |
| Ukuran | 73.5 MB |
| Durasi *file* | 04:00 |
| Lama tes | 3 Menit |
| Parameter filter paket grafik | Black Line : ip.dst==alamat host<br>Red Impulse : udp.port==9695<br>Green Dot : Protokol ccn |
| Sumbu X | Durasi penangkapan paket |
| Sumbu Y | Jumlah paket |

Tabel 5.2 diatas menjelaskan tentang spesifikasi *file* yang akan digunakan untuk uji coba beserta keterangan dari *graph* yang digunakan sebagai alat bantu untuk menjelaskan data secara terperinci. *File* yang digunakan untuk uji coba adalah *file* dengan ekstensi .avi

10.151.36.27



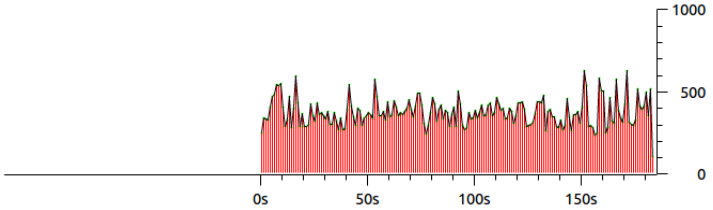

**Gambar 5.11 *Traffic* paket pada *node* 10.151.36.27 pada percobaan pertama**



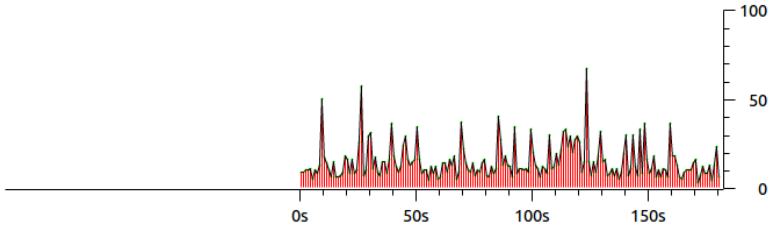

**Gambar 5.12 *Traffic* paket pada *node* 10.151.36.27 pada percobaan kedua**

Dari Gambar 5.11 dan 5.12, ditemukan bahwa terjadi perbedaan *traffic* yang cukup signifikan antara percobaan pertama dan percobaan kedua. Percobaan kedua menghasilkan *traffic* yang lebih sedikit daripada percobaan pertama pada *node* 10.151.36.27.

10.151.36.34




**Gambar 5.13 *Traffic* paket pada *node* 10.151.36.34 pada percobaan pertama**

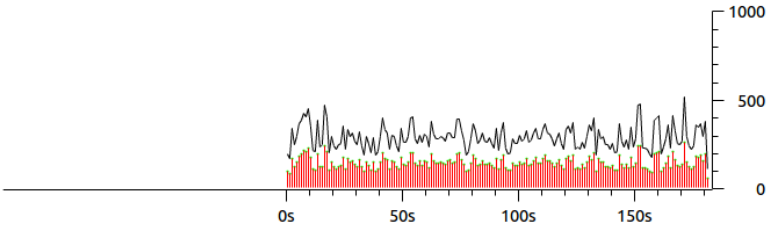



**Gambar 5.14 *Traffic* paket pada *node* 10.151.36.34 pada percobaan kedua**

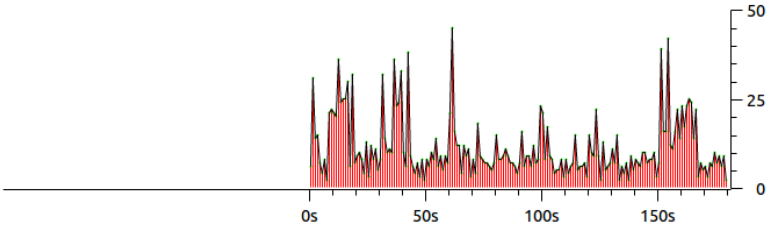
Dari Gambar 5.13 dan 5.14, ditemukan bahwa terjadi perbedaan *traffic* yang cukup signifikan antara percobaan pertama

dan percobaan kedua. Percobaan kedua menghasilkan *traffic* yang lebih sedikit daripada percobaan pertama pada *node* 10.151.36.34.

10.151.36.40




**Gambar 5.15 *Traffic* paket pada *node* 10.151.36.40 pada percobaan pertama**




**Gambar 5.16 *Traffic* paket pada *node* 10.151.36.40 pada percobaan pertama**

Dari Gambar 5.15 dan 5.16, ditemukan bahwa terjadi perbedaan *traffic* yang cukup signifikan antara percobaan pertama dan percobaan kedua. Percobaan kedua menghasilkan *traffic* yang lebih sedikit daripada percobaan pertama Dapat disimpulkan bahwa *streaming* video pada jaringan CCN menggunakan *file* dengan .avi telah berhasil.

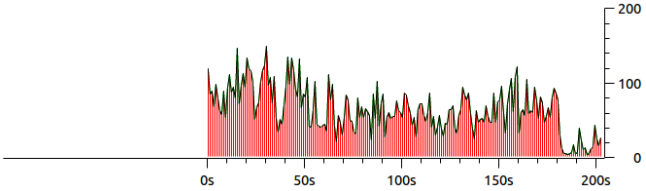
**Tabel 5.3 Spesifikasi *file* uji coba dengan ekstensi .mkv**

| | |
|---|---|
| Nama *file* | ccn_test_mkv_1.mkv |
| Ukuran | 72.0 MB |
| Durasi *file* | 23:20 |

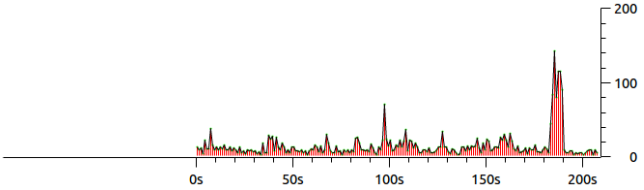
| | |
|---|---|
| Lama tes | 2 Menit |
| Parameter filter paket grafik | Black Line : ip.dst==alamat host<br>Red Impulse : udp.port==9695<br>Green Dot : Protokol ccn |
| Sumbu X | Durasi penangkapan paket |
| Sumbu Y | Jumlah paket |

Tabel 5.3 diatas menjelaskan tentang spesifikasi *file* yang akan digunakan untuk uji coba beserta keterangan dari *graph* yang digunakan sebagai alat bantu untuk menjelaskan data secara terperinci. *File* yang digunakan untuk uji coba adalah *file* dengan ekstensi .mkv

10.151.36.27



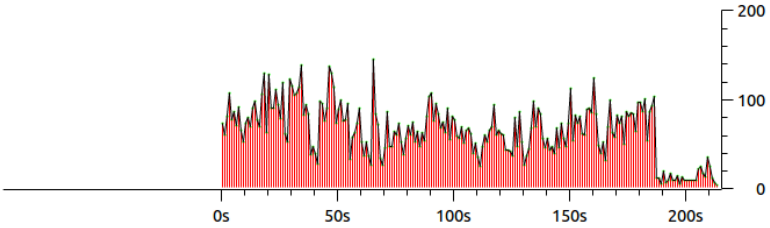

**Gambar 5.17 *Traffic* paket pada *node* 10.151.36.27 pada percobaan pertama**



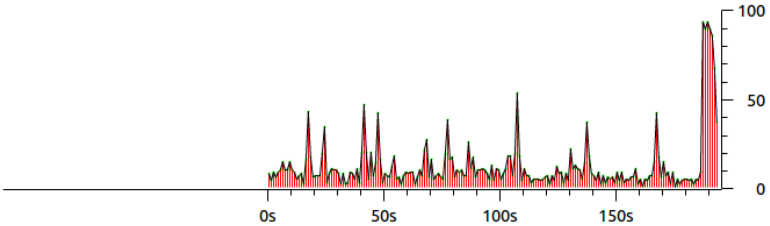

**Gambar 5.18 *Traffic* paket pada *node* 10.151.36.27 pada percobaan kedua**

Dari Gambar 5.17 dan 5.18, ditemukan bahwa terjadi perbedaan *traffic* yang cukup signifikan antara percobaan pertama dan percobaan kedua. Percobaan kedua menghasilkan *traffic* yang lebih sedikit daripada percobaan pertama pada *node* 10.151.36.27.

10.151.36.34



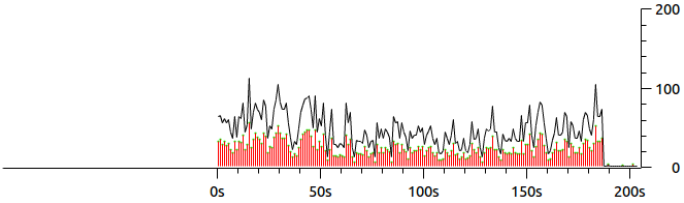

**Gambar 5.19 *Traffic* paket pada *node* 10.151.36.34 pada percobaan pertama**



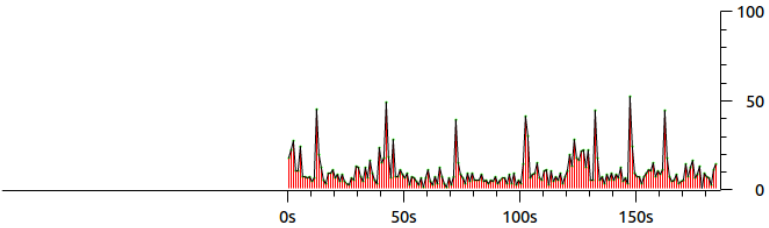

**Gambar 5.20 *Traffic* paket pada *node* 10.151.36.34 pada percobaan kedua**

Dari Gambar 5.19 dan 5.20, ditemukan bahwa terjadi perbedaan *traffic* yang cukup signifikan antara percobaan pertama dan percobaan kedua. Percobaan kedua menghasilkan *traffic* yang lebih sedikit daripada percobaan pertama pada *node* 10.151.36.34.

10.151.36.40




**Gambar 5.21 *Traffic* paket pada *node* 10.151.36.40 pada percobaan pertama**




**Gambar 5.22 *Traffic* paket pada *node* 10.151.36.40 pada percobaan kedua**

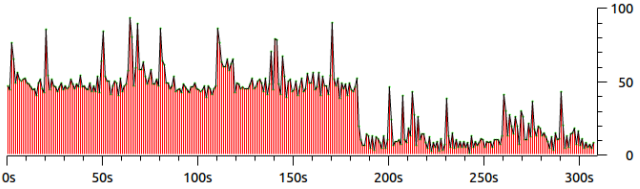
Dari Gambar 5.21 dan 5.22, ditemukan bahwa terjadi perbedaan *traffic* yang cukup signifikan antara percobaan pertama dan percobaan kedua pada *node* 10.151.36.40. Percobaan kedua menghasilkan *traffic* yang lebih sedikit daripada percobaan pertama. Dapat disimpulkan bahwa *streaming* video pada jaringan CCN menggunakan *file* dengan ekstensi .mkv telah berhasil.

**Tabel 5.4 Spesifikasi *file* uji coba dengan ekstensi .mp3**

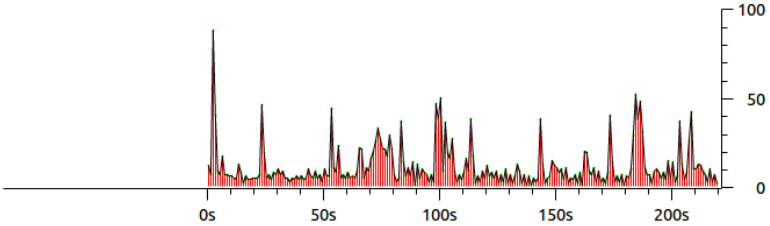
| | |
|---|---|
| Nama *file* | ccn_test_mp3_1.mp3 |
| Ukuran | 7.2 MB |
| Durasi *file* | 03:44 |
| Lama tes | 2 Menit |
| Parameter filter paket grafik | Black Line : ip.dst==alamat host<br>Red Impulse : udp.port==9695<br>Green Dot : Protokol ccn |
| Sumbu X | Durasi penangkapan paket |
| Sumbu Y | Jumlah paket |

Tabel 5.4 diatas menjelaskan tentang spesifikasi *file* yang akan digunakan untuk uji coba beserta keterangan dari *graph* yang digunakan sebagai alat bantu untuk menjelaskan data secara terperinci. *File* yang digunakan untuk uji coba adalah *file* dengan ekstensi .mp3

10.151.36.27



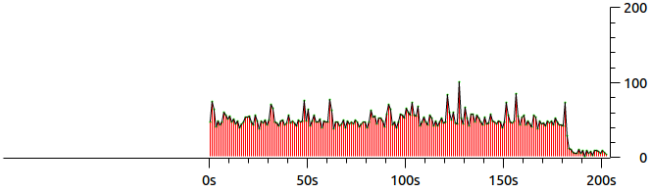

**Gambar 5.23 *Traffic* paket pada *node* 10.151.36.27 pada percobaan pertama**



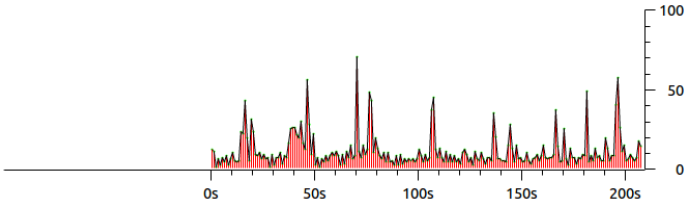

**Gambar 5.24 *Traffic* paket pada *node* 10.151.36.27 pada percobaan kedua**

Dari Gambar 5.23 dan 5.24, ditemukan bahwa terjadi perbedaan *traffic* yang cukup signifikan antara percobaan pertama dan percobaan kedua. Percobaan kedua menghasilkan *traffic* yang lebih sedikit daripada percobaan pertama pada *node* 10.151.36.27.

10.151.36.34



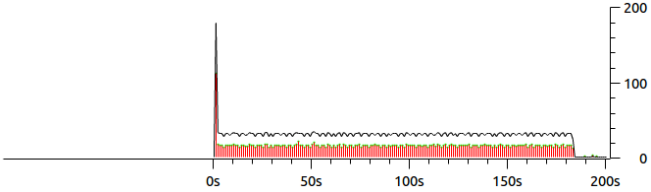

**Gambar 5.25 *Traffic* paket pada *node* 10.151.36.34 pada percobaan pertama**



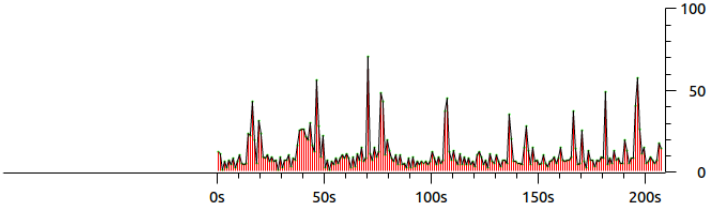

**Gambar 5.26 *Traffic* paket pada *node* 10.151.36.34 pada percobaan kedua**

Dari Gambar 5.25 dan 5.26, ditemukan bahwa terjadi perbedaan *traffic* yang cukup signifikan antara percobaan pertama dan percobaan kedua. Percobaan kedua menghasilkan *traffic* yang lebih sedikit daripada percobaan pertama pada *node* 10.151.36.34.

10.151.36.40




**Gambar 5.27 *Traffic* paket pada *node* 10.151.36.40 pada percobaan pertama**




**Gambar 5.28 *Traffic* paket pada *node* 10.151.36.40 pada percobaan pertama**

Dari Gambar 5.15 dan 5.16, ditemukan bahwa terjadi perbedaan *traffic* yang cukup signifikan antara percobaan pertama dan percobaan kedua pada *node* 10.151.36.40. Percobaan kedua

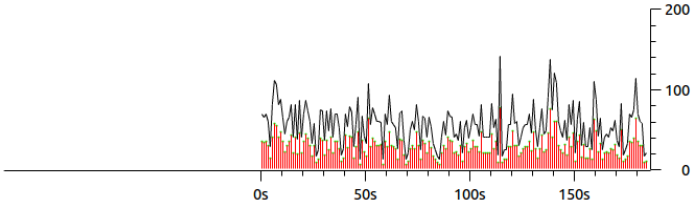
menghasilkan *traffic* yang lebih sedikit daripada percobaan pertama. Dapat disimpulkan bahwa *streaming* video pada jaringan CCN menggunakan *file* dengan ekstensi .mp3 telah berhasil

**Tabel 5.5 Spesifikasi *file* uji coba dengan ekstensi .mp4**

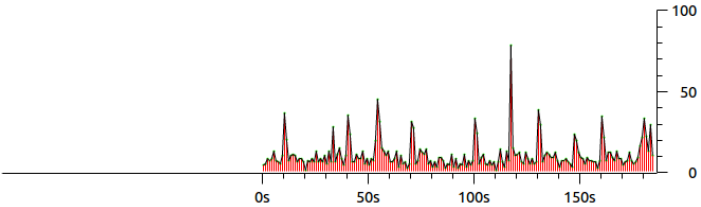
| | |
|---|---|
| Nama *file* | ccn_test_mp4_1.mp4 |
| Ukuran | 15.8 MB |
| Durasi *file* | 04:55 |
| Lama tes | 3 Menit |
| Parameter filter paket grafik | Black Line : ip.dst==alamat host<br>Red Impulse : udp.port==9695<br>Green Dot : Protokol ccn |
| Sumbu X | Durasi penangkapan paket |
| Sumbu Y | Jumlah paket |

Tabel 5.5 diatas menjelaskan tentang spesifikasi *file* yang akan digunakan untuk uji coba beserta keterangan dari *graph* yang digunakan sebagai alat bantu untuk menjelaskan data secara terperinci. *File* yang digunakan untuk uji coba adalah *file* dengan ekstensi .mp4

10.151.36.27



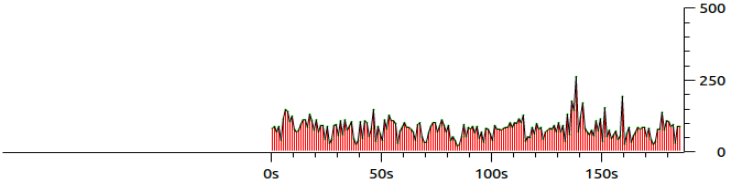

**Gambar 5.29 *Traffic* paket pada *node* 10.151.36.27 pada percobaan pertama**



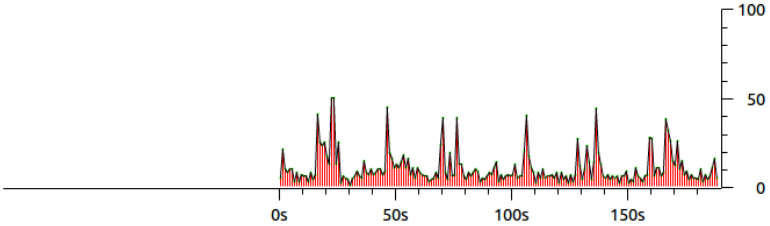

**Gambar 5.30 *Traffic* paket pada *node* 10.151.36.27 pada percobaan kedua**

Dari Gambar 5.29 dan 5.30, ditemukan bahwa terjadi perbedaan *traffic* yang cukup signifikan antara percobaan pertama dan percobaan kedua. Percobaan kedua menghasilkan *traffic* yang lebih sedikit daripada percobaan pertama pada *node* 10.151.36.27

10.151.36.34



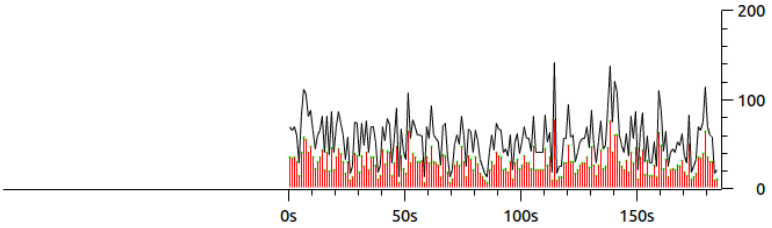

**Gambar 5.31 *Traffic* paket pada *node* 10.151.36.34 pada percobaan pertama**



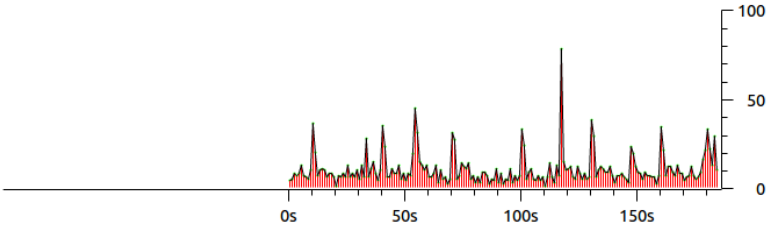

**Gambar 5.32 *Traffic* paket pada *node* 10.151.36.34 pada percobaan kedua**

Dari Gambar 5.31 dan 5.32, ditemukan bahwa terjadi perbedaan *traffic* yang cukup signifikan antara percobaan pertama dan percobaan kedua. Percobaan kedua menghasilkan *traffic* yang lebih sedikit daripada percobaan pertama pada *node* 10.151.36.34.

10.151.36.40




**Gambar 5.33 *Traffic* paket pada *node* 10.151.36.40 pada percobaan pertama**




**Gambar 5.34 *Traffic* paket pada *node* 10.151.36.40 pada percobaan kedua**

Dari hasil uji coba pertama dengan menggunakan 5 jenis ekstensi *file* diatas dan diuji pada 3 *node* pada jaringan CCN, telah ditemukan bahwa proses *caching* dalam CCN telah berjalan dengan baik. Hal ini bisa diamati dari *traffic* paket yang berkurang saat percobaan kedua. Hal ini disebabkan oleh proses *caching* yang berjalan setelah percobaan pertama selesai.
Uji coba performa kedua dilakukan untuk mengamati perbedaan performa antara algoritma *caching* LRU (*Least Recently Used*)

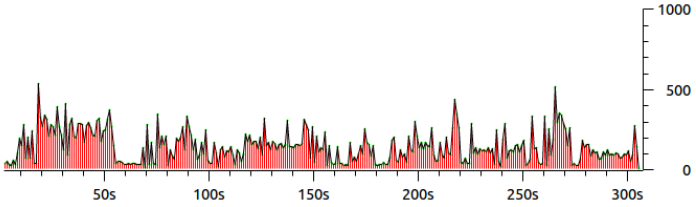
yang secara *default* terdapat pada CCNx dengan algoritma *progressive caching*.

**Tabel 5.6 Spesifikasi *file* untuk uji coba dengan ekstensi .mkv**

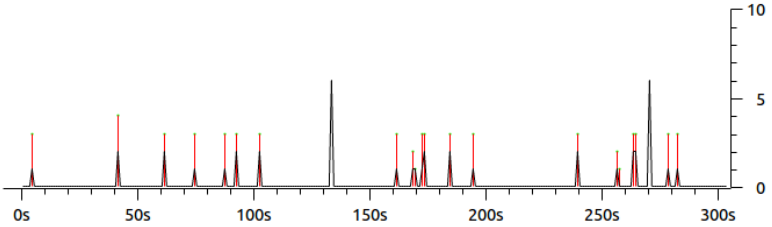
| | |
|---|---|
| Nama *file* | By5.KH.mkv |
| Ukuran | 449 MB |
| Durasi *file* | 1:02:45 |
| Lama tes | 5 Menit |
| Parameter filter paket grafik | Black Line : ip.dst==alamat host<br>Red Impulse : udp.port==9695<br>Green Dot : Protokol ccn |
| Sumbu X | Durasi penangkapan paket |
| Sumbu Y | Jumlah paket |

Tabel 5.6 diatas menjelaskan tentang spesifikasi *file* yang akan digunakan untuk uji coba beserta keterangan dari *graph* yang digunakan sebagai alat bantu untuk menjelaskan data secara terperinci yang akan digunakan untuk membandingkan performa antara LRU (*Least Recently Used*) dan *Progressive Caching*. *File* yang digunakan untuk uji coba adalah *file* dengan ekstensi .mkv

10.151.36.27

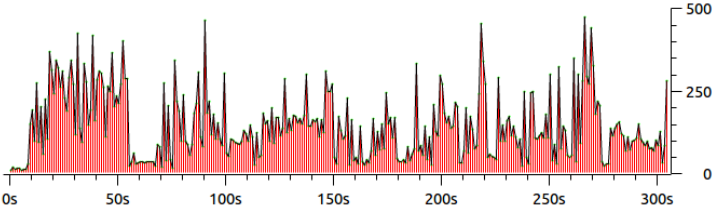



**Gambar 5.35 *Traffic* paket pada *node* 10.151.36.27 pada percobaan pertama dengan menggunakan LRU**

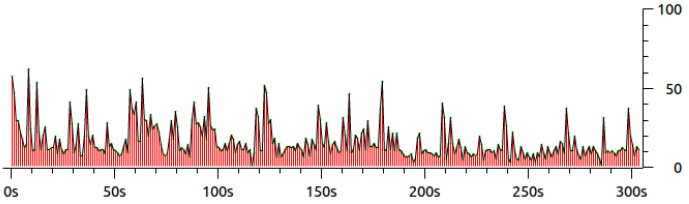



**Gambar 5.36 *Traffic* paket pada *node* 10.151.36.27 pada percobaan kedua dengan menggunakan LRU**

Dari Gambar 5.35 dan 5.39, ditemukan bahwa terjadi perbedaan *traffic* yang cukup signifikan antara percobaan pertama dan percobaan kedua. Percobaan kedua menghasilkan *traffic* yang lebih sedikit daripada percobaan pertama




**Gambar 5.37 *Traffic* paket pada *node* 10.151.36.27 pada percobaan pertama dengan menggunakan *Progressive Cache***

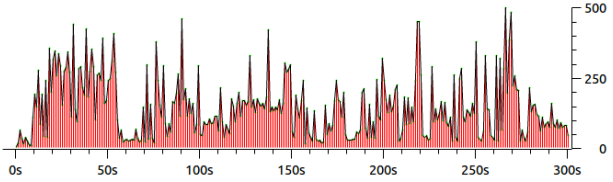



**Gambar 5.38 *Traffic* paket pada *node* 10.151.36.27 pada percobaan pertama dengan menggunakan *Progressive Cache***

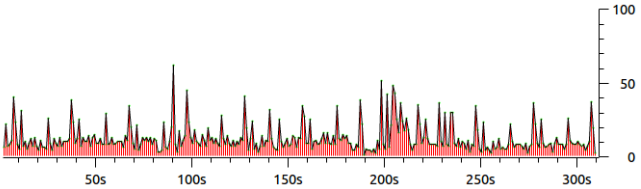
Dari Gambar 5.37 dan 5.38, ditemukan bahwa terjadi perbedaan *traffic* yang cukup signifikan antara percobaan pertama dan percobaan kedua. Percobaan kedua menghasilkan *traffic* yang lebih sedikit daripada percobaan pertama. Tidak ada perbedaan

jumlah *traffic* yang dapat diamati antara algoritma LRU dan algoritma *Progressive Caching*

10.151.36.34

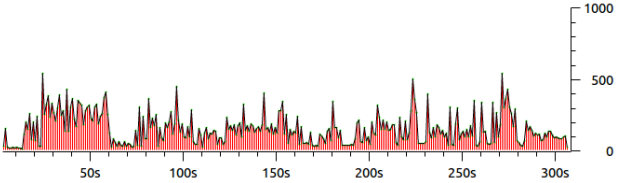



**Gambar 5.39 *Traffic* paket pada *node* 10.151.36.34 pada percobaan pertama dengan menggunakan LRU**

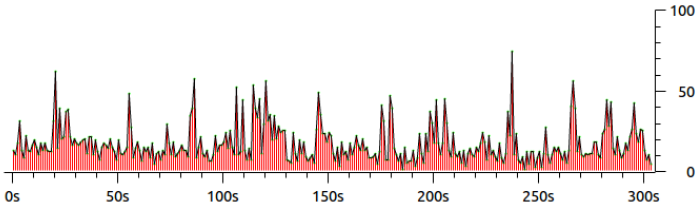



**Gambar 5.40 *Traffic* paket pada *node* 10.151.36.34 pada percobaan kedua dengan menggunakan LRU**

Dari Gambar 5.39 dan 5.40, ditemukan bahwa terjadi perbedaan *traffic* yang cukup signifikan antara percobaan pertama dan percobaan kedua. Percobaan kedua menghasilkan *traffic* yang lebih sedikit daripada percobaan pertama



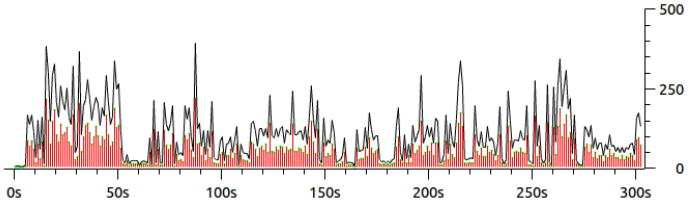

**Gambar 5.41 *Traffic* paket pada *node* 10.151.36.34 pada percobaan pertama dengan menggunakan *Progressive Cache***



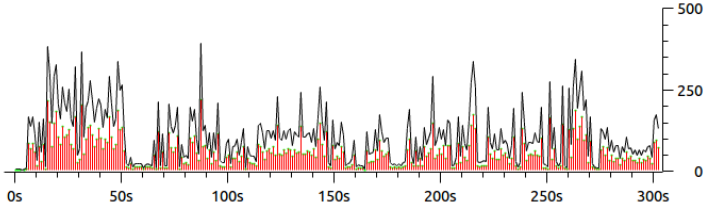

**Gambar 5.42 *Traffic* paket pada *node* 10.151.36.34 pada percobaan kedua dengan menggunakan *Progressive Cache***

Dari Gambar 5.41 dan 5.42, ditemukan bahwa terjadi perbedaan *traffic* yang cukup signifikan antara percobaan pertama dan percobaan kedua. Percobaan kedua menghasilkan *traffic* yang lebih sedikit daripada percobaan pertama. Tidak ada perbedaan jumlah *traffic* yang dapat diamati antara algoritma LRU dan algoritma *Progressive Caching*

10.151.36.40

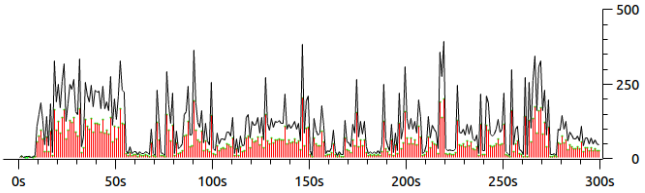



**Gambar 5.43 *Traffic* paket pada *node* 10.151.36.40 pada percobaan pertama dengan menggunakan LRU**

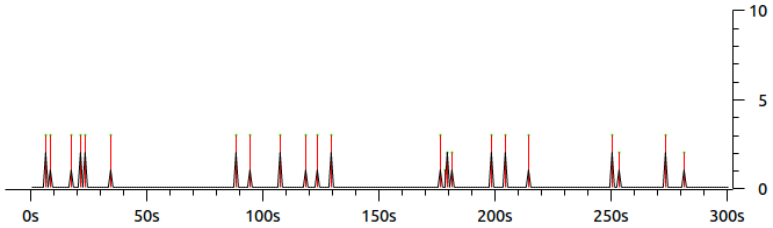



**Gambar 5.44 *Traffic* paket pada *node* 10.151.36.40 pada percobaan kedua dengan menggunakan LRU**

Dari Gambar 5.43 dan 5.44, ditemukan bahwa terjadi perbedaan *traffic* yang cukup signifikan antara percobaan pertama dan percobaan kedua. Percobaan kedua menghasilkan *traffic* yang lebih sedikit daripada percobaan pertama.




**Gambar 5.45 *Traffic* paket pada *node* 10.151.36.40 pada percobaan pertama dengan menggunakan *Progressive Cache***




**Gambar 5.46 *Traffic* paket pada *node* 10.151.36.40 pada percobaan kedua dengan menggunakan *Progressive Cache***

Dari Gambar 5.45 dan 5.46, ditemukan bahwa terjadi perbedaan *traffic* yang cukup signifikan antara percobaan pertama dan percobaan kedua. Percobaan kedua menghasilkan *traffic* yang lebih sedikit daripada percobaan pertama. Tidak ada perbedaan jumlah *traffic* yang dapat diamati antara algoritma LRU dan algoritma *Progressive Caching*

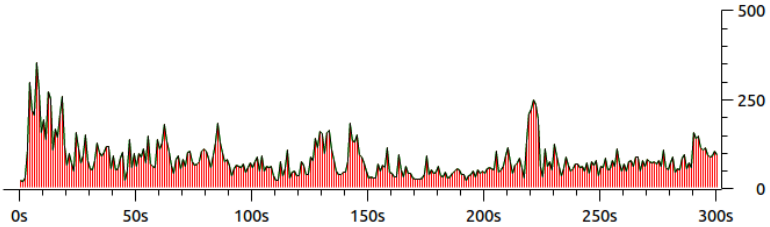
**Tabel 5.7 Spesifikasi *file* untuk uji coba dengan ekstensi .mkv**

| | |
|---|---|
| Nama *file* | Arrow.S01E01.480p.HDTV.x264-ChameE.mkv |
| Ukuran | 133 MB |
| Durasi *file* | 42:04 |

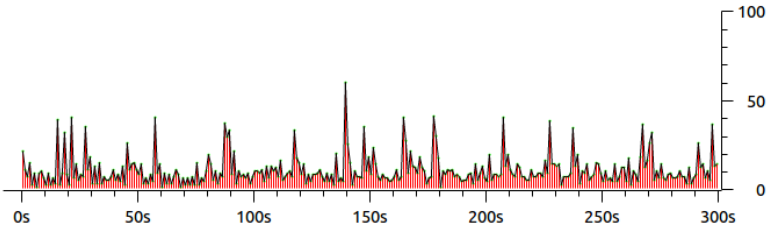
| | |
|---|---|
| Lama tes | 5 Menit |
| Parameter filter paket grafik | Black Line : ip.dst==alamat host<br>Red Impulse : udp.port==9695<br>Green Dot : Protokol ccn |
| Sumbu X | Durasi penangkapan paket |
| Sumbu Y | Jumlah paket |

Tabel 5.7 diatas menjelaskan tentang spesifikasi *file* yang akan digunakan untuk uji coba beserta keterangan dari *graph* yang digunakan sebagai alat bantu untuk menjelaskan data secara terperinci yang akan digunakan untuk membandingkan performa antara LRU (*Least Recently Used*) dan *Progressive Caching*. *File* yang digunakan untuk uji coba adalah *file* dengan ekstensi .mkv

10.151.36.27

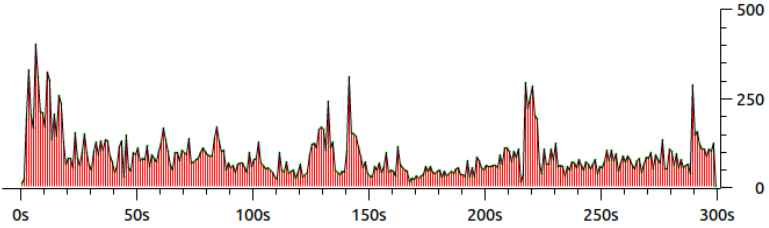



**Gambar 5.47 *Traffic* paket pada *node* 10.151.36.27 pada percobaan pertama dengan menggunakan LRU**

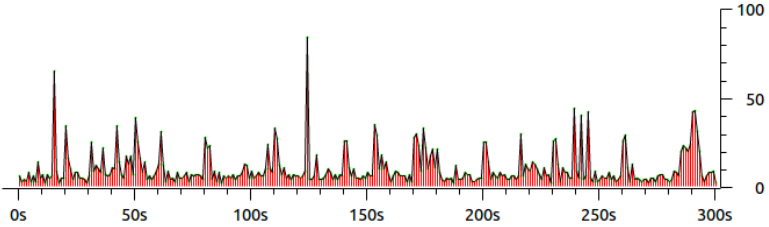



**Gambar 5.48 *Traffic* paket pada *node* 10.151.36.27 pada percobaan kedua dengan menggunakan LRU**

Dari Gambar 5.47 dan 5.48, ditemukan bahwa terjadi perbedaan *traffic* yang cukup signifikan antara percobaan pertama dan percobaan kedua. Percobaan kedua menghasilkan *traffic* yang lebih sedikit daripada percobaan pertama.



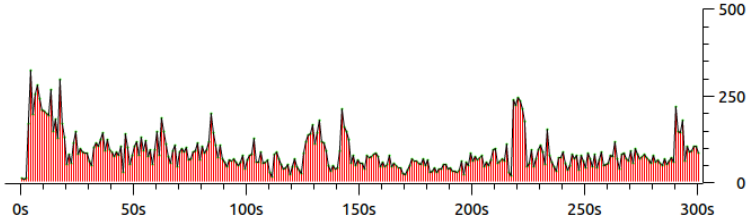

**Gambar 5.49 *Traffic* paket pada *node* 10.151.36.27 pada percobaan pertama dengan menggunakan *Progressive Cache***



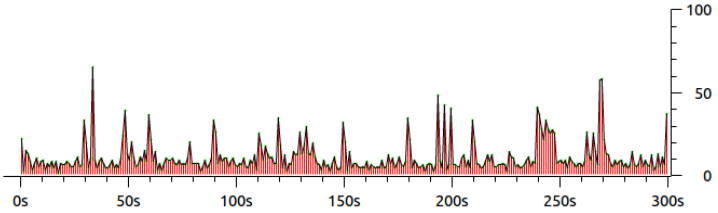

**Gambar 5.50 *Traffic* paket pada *node* 10.151.36.27 pada percobaan kedua dengan menggunakan *Progressive Cache***

Dari Gambar 5.49 dan 5.50, ditemukan bahwa terjadi perbedaan *traffic* yang cukup signifikan antara percobaan pertama dan percobaan kedua. Percobaan kedua menghasilkan *traffic* yang lebih sedikit daripada percobaan pertama. Tidak ada perbedaan jumlah *traffic* yang dapat diamati antara algoritma LRU dan algoritma *Progressive Caching*

10.151.36.34

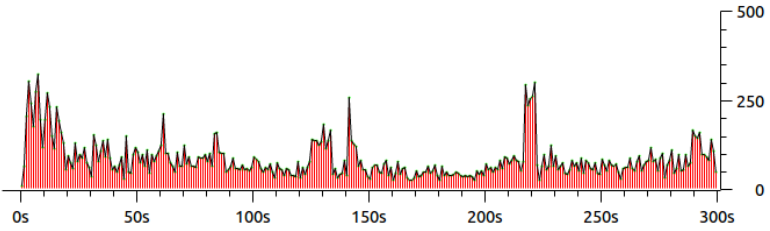



**Gambar 5.51 *Traffic* paket pada *node* 10.151.36.34 pada percobaan pertama dengan menggunakan LRU**

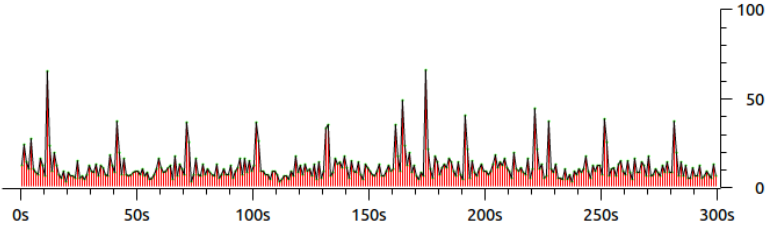



**Gambar 5.52 *Traffic* paket pada *node* 10.151.36.34 pada percobaan kedua dengan menggunakan LRU**

Dari Gambar 5.51 dan 5.52, ditemukan bahwa terjadi perbedaan *traffic* yang cukup signifikan antara percobaan pertama dan percobaan kedua. Percobaan kedua menghasilkan *traffic* yang lebih sedikit daripada percobaan pertama.



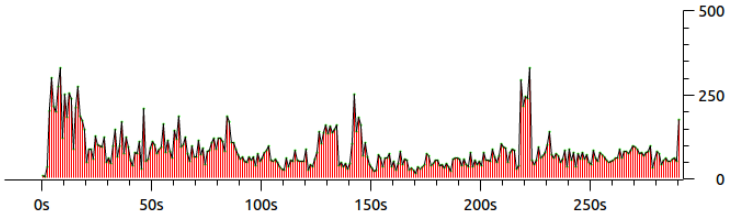

**Gambar 5.53 *Traffic* paket pada *node* 10.151.36.34 pada percobaan pertama dengan menggunakan *Progressive Cache***



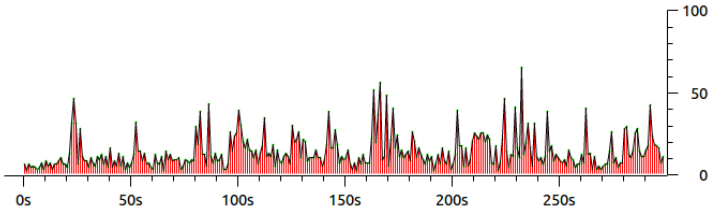

**Gambar 5.54 *Traffic* paket pada *node* 10.151.36.34 pada percobaan kedua dengan menggunakan *Progressive Cache***

Dari Gambar 5.53 dan 5.54, ditemukan bahwa terjadi perbedaan *traffic* yang cukup signifikan antara percobaan pertama dan percobaan kedua. Percobaan kedua menghasilkan *traffic* yang lebih sedikit daripada percobaan pertama. Tidak ada perbedaan jumlah *traffic* yang dapat diamati antara algoritma LRU dan algoritma *Progressive Caching*

10.151.36.40

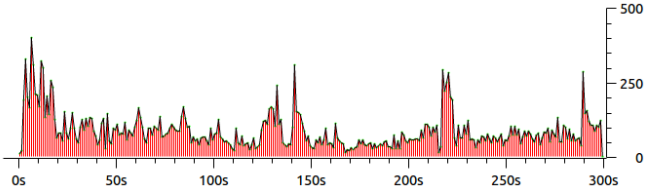



**Gambar 5.55 *Traffic* paket pada *node* 10.151.36.40 pada percobaan pertama dengan menggunakan LRU**

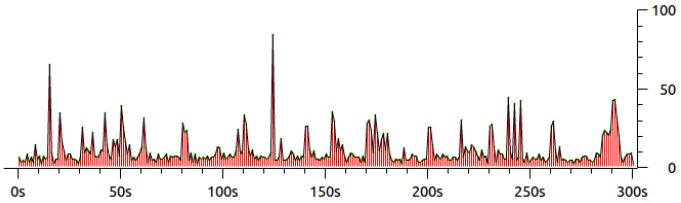



**Gambar 5.56 *Traffic* paket pada *node* 10.151.36.40 pada percobaan kedua dengan menggunakan LRU**

Dari Gambar 5.55 dan 5.56, ditemukan bahwa terjadi perbedaan *traffic* yang cukup signifikan antara percobaan pertama dan percobaan kedua. Percobaan kedua menghasilkan *traffic* yang lebih sedikit daripada percobaan pertama.




**Gambar 5.57 *Traffic* paket pada *node* 10.151.36.40 pada percobaan pertama dengan menggunakan *Progressive Caching***




**Gambar 5.58 *Traffic* paket pada *node* 10.151.36.40 pada percobaan kedua dengan menggunakan *Progressive Caching***

Dari Gambar 5.57 dan 5.58, ditemukan bahwa terjadi perbedaan *traffic* yang cukup signifikan antara percobaan pertama dan percobaan kedua. Percobaan kedua menghasilkan *traffic* yang lebih sedikit daripada percobaan pertama.

**Tabel 5.8 Rata-rata jumlah paket per detik**

| | Percobaan ke- | 10.151.36.27 | 10.151.36.34 | 10.151.36.40 |
|---|---|---|---|---|
| LRU | Pertama | 80,986 | 84,542 | 84,479 |
| | Kedua | 10,705 | 10,611 | 13,527 |
| Progressive Caching | Pertama | 84,815 | 83,302 | 87,654 |
| | Kedua | 15,084 | 11,378 | 10,307 |

Tabel 5.8 menjelaskan jumlah rata-rata paket per detik yang didapatkan melalui *packet capture* pada masing-masing *host* pada uji coba

**Tabel 5.9 Total jumlah paket**

| | Percobaan ke- | 10.151.36.27 | 10.151.36.34 | 10.151.36.40 |
|---|---|---|---|---|
| LRU | Pertama | 24.357 | 25.433 | 25.360 |
| | Kedua | 3.209 | 3.180 | 4.039 |
| Progressive Caching | Pertama | 25.545 | 24.937 | 26.218 |
| | Kedua | 4.541 | 3.408 | 3.095 |

Tabel 5.9 menjelaskan jumlah total paket yang didapatkan melalui *packet capture* pada masing-masing *host* pada proses uji coba

Pada hasil perbandingan antara algoritma LRU dengan algoritma *progressive caching*, terlihat bahwa LRU memiliki performa yang lebih baik pada *node* yang dekat (memiliki *hop count* yang relatif rendah dengan *content provider*), yakni pada *node* 10.151.36.27 dan *node* 10.151.36.34. Sedangkan, *progressive caching* mengalami peningkatan performa pada *node* yang terletak jauh (memiliki *hop count* yang lebih tinggi dengan *content provider*), yakni pada *node* 10.151.36.40

# BAB VI
# KESIMPULAN DAN SARAN

Bab ini membahas mengenai kesimpulan yang dapat diambil dari hasil uji coba yang telah dilakukan sebagai jawaban dari rumusan masalah yang dikemukakan. Selain kesimpulan, juga terdapat saran yang ditujukan untuk pengembangan penelitian lebih lanjut.

## 6.1 Kesimpulan

Dari hasil uji coba yang telah dilakukan terhadap implementasi jaringan CCN dengan menggunakan *progressive caching*, kesimpulan yang dicapai adalah sebagai berikut:

1. Pengimplementasian mekanisme *caching* pada jaringan CCN yang berjalan dengan baik terbukti secara signifikan mengurangi *traffic* antara *host* dengan *content provider*
2. *Host* dapat berkomunikasi dengan *content provider* pada jaringan CCN melalui protokol UDP yang digunakan sebagai media transmisi *content*.
3. CCN diterapkan pada jaringan yang sudah ada dengan memasang CCNx pada masing-masing *node* yang akan dijadikan sebagai jaringan CCN
4. *Progressive caching* diterapkan dengan memodifikasi kode sumber **ccnd.c** pada CCNx. Modifikasi dilakukan dengan menambahkan *face* baru pada kode sumber.
5. Pada hasil uji coba, CCNx kompatibel dengan ekstensi *file* .avi, .mkv, .flv, .mp3, dan .mp4
6. LRU memiliki performa yang lebih baik daripada *progressive caching* pada saat *edge node* merupakan *node* yang memiliki nilai *hop count* yang rendah (memiliki posisi yang dekat dengan *content provider*). Sedangkan *progressive caching* memiliki performa yang lebih baik daripada LRU pada *edge node* yang memiliki nilai *hop*

*count* yang tinggi (memiliki posisi yang jauh dari *content provider*).

## 6.2 Saran

Saran yang diberikan untuk pengembangan Implementasi Progressive Caching pada Content-Centric Networking untuk Streaming Video adalah :

1. Diperlukan perangkat uji coba yang lebih banyak guna mengamati perbedaan performa antara LRU dan *progressive caching.*
2. Diperlukan *node* yang lebih banyak pada jaringan CCNx guna mengamati perbedaan performa antara LRU dengan *progressive caching*.

# LAMPIRAN

```
static void
update_ex_index(struct ccnd_handle *h, int staletime,
ccn_cookie c)
{
    struct ccn_nametree *e = NULL;
    struct ccny *y = NULL;

    e = h->ex_index;
    y = ccn_nametree_lookup(e, NULL, staletime);
    if (c == 0) {
        if (y != NULL) {
            ccny_remove(e, y);
            ccny_destroy(e, &y);
        }
    }
    else {
        if (y == NULL) {
            y = ccny_create(nrand48(h->seed), 0);
            /* Our compare action only uses keylen */
            ccny_set_key_fields(y, NULL, staletime);
            if (e->n >= e->limit)
                ccn_nametree_grow(e);
            ccny_enroll(e, y);
            if (ccny_cookie(y) == 0) abort();
        }
        ccny_set_info(y, c);
    }
}

/**
 *  Enter content into the content expiry queue according to
its staletime
 */
static void
content_enqueuex(struct ccnd_handle *h, struct content_entry
*content)
{
    struct content_entry *next = NULL;
    struct content_entry *prev = NULL;
    struct ccny *y = NULL;
    int tts;

    tts = content->staletime;
    if (content->nextx != NULL || content->accession == 0 ||
tts < 0) abort();
```

```
    prev = h->headx->prevx;
    if (prev->staletime > tts) {
        y = ccn_nametree_look_le(h->ex_index, NULL, tts);
        if (y == NULL)
            prev = h->headx;
        else
            prev = content_from_accession(h, ccny_info(y));
        // if prev is NULL, we forgot to remove an entry
    }
    if (prev->nextx->staletime <= tts && prev->nextx != h-
>headx) abort();
    if (prev->staletime > tts) {
        // Oops, this should not happen.  Revert to slow-but-
sure.
        ccnd_msg(h, "Err, break at ccnd.c:%d to debug this",
__LINE__);
        for (prev = h->headx->prevx; prev->staletime > tts;)
            prev = prev->prevx;
    }
    next = prev->nextx;
    content->nextx = next;
    content->prevx = prev;
    next->prevx = prev->nextx = content;
    if (next != h->headx)
        update_ex_index(h, content->staletime, content-
>accession);
    else if (prev != h->headx && prev->staletime < tts)
        update_ex_index(h, prev->staletime, prev->accession);
}
```

**Kode Sumber 7.1 Kode sumber yang digunakan untuk mencari *content* yang paling jarang diakses**

```
static int
remove_content(struct ccnd_handle *h, struct content_entry
*content)
{
    struct ccny *y = NULL;

    if (content == NULL)
        return(-1);
    y = ccny_from_cookie(h->content_tree, content->accession);
    if (y == NULL)
        return(-1);
    if (content->refs != 0)
        ccnd_debug_content(h, __LINE__,
"remove_queued_content", NULL, content);
```

```
    else if (h->debug & 4)
        ccnd_debug_content(h, __LINE__, "remove", NULL,
content);
    ccny_remove(h->content_tree, y);
    content = NULL;
    ccny_destroy(h->content_tree, &y); /* releases content as
well */
    return(0);
}
```

**Kode Sumber 7.2 Kode sumber untuk melakukan penghapusan *content***

```
struct pit_face_item *
send_interest(struct ccnd_handle *h, struct interest_entry *ie,
              struct pit_face_item *x, struct pit_face_item *p)
{
    struct face *face = NULL;
    struct ccn_charbuf *c = h->send_interest_scratch;
    const intmax_t default_life = CCN_INTEREST_LIFETIME_SEC <<
12;
    intmax_t lifetime = default_life;
    ccn_wrappedtime delta;
    size_t noncesize;

    face = face_from_faceid(h, p->faceid);
    if (face == NULL)
        return(p);
    h->interest_faceid = x->faceid; /* relevant if p is face 0
*/
    p = pfi_copy_nonce(h, ie, p, x);
    delta = x->expiry - x->renewed;
    lifetime = (intmax_t)delta * 4096 / WTHZ;
    /* clip lifetime against various limits here */
    lifetime = (((lifetime + 511) >> 9) << 9); /* round up -
1/8 sec */
    p->renewed = h->wtnow;
    p->expiry = h->wtnow + (lifetime * WTHZ / 4096);
    ccn_charbuf_reset(c);
    if (lifetime != default_life)
        ccnb_append_tagged_binary_number(c,
CCN_DTAG_InterestLifetime, lifetime);
    noncesize = p->pfi_flags & CCND_PFI_NONCESZ;
    if (noncesize != 0)
        ccnb_append_tagged_blob(c, CCN_DTAG_Nonce, p->nonce,
noncesize);
    ccnb_element_end(c);
```

```
    h->interests_sent += 1;
    if ((p->pfi_flags & CCND_PFI_UPENDING) == 0) {
        p->pfi_flags |= CCND_PFI_UPENDING;
        face->outstanding_interests += 1;
    }
    p->pfi_flags &= ~(CCND_PFI_SENDUPST | CCND_PFI_UPHUNGRY);
    ccnd_meter_bump(h, face->meter[FM_INTO], 1);
    stuff_and_send(h, face, ie->interest_msg, ie->size - 1, c-
>buf, c->length, (h->debug & 2) ? "interest_to" : NULL,
__LINE__);
    return(p);
}
```

**Kode Sumber 7.3 Kode sumber untuk menyalurkan paket *interest***

```
static int
content_sender(struct ccn_schedule *sched,
    void *clienth,
    struct ccn_scheduled_event *ev,
    int flags)
{
    int i, j;
    int delay;
    int nsec;
    int burst_nsec;
    int burst_max;
    struct ccnd_handle *h = clienth;
    struct content_entry *content = NULL;
    unsigned faceid = ev->evint;
    struct face *face = NULL;
    struct content_queue *q = ev->evdata;
    (void)sched;

    if ((flags & CCN_SCHEDULE_CANCEL) != 0)
        goto Bail;
    face = face_from_faceid(h, faceid);
    if (face == NULL)
        goto Bail;
    if (q->send_queue == NULL)
        goto Bail;
    if ((face->flags & CCN_FACE_NOSEND) != 0)
        goto Bail;
    /* Send the content at the head of the queue */
    if (q->ready > q->send_queue->n ||
        (q->ready == 0 && q->nrun >= 12 && q->nrun < 120))
        q->ready = q->send_queue->n;
```

```
    nsec = 0;
    burst_nsec = q->burst_nsec;
    burst_max = 2;
    if (q->ready < burst_max)
        burst_max = q->ready;
    if (burst_max == 0)
        q->nrun = 0;
    for (i = 0; i < burst_max && nsec < 1000000; i++) {
        content = content_from_accession(h, q->send_queue-
>buf[i]);
        if (content == NULL)
            q->nrun = 0;
        else {
            send_content(h, face, content);
            content->refs--;
            /* face may have vanished, bail out if it did */
            if (face_from_faceid(h, faceid) == NULL)
                goto Bail;
            nsec += burst_nsec * (unsigned)((content->size +
1023) / 1024);
            q->nrun++;
        }
    }
    if (q->ready < i) abort();
    q->ready -= i;
    /* Update queue */
    for (j = 0; i < q->send_queue->n; i++, j++)
        q->send_queue->buf[j] = q->send_queue->buf[i];
    q->send_queue->n = j;
    /* Do a poll before going on to allow others to preempt
send. */
    delay = (nsec + 499) / 1000 + 1;
    if (q->ready > 0) {
        if (h->debug & 8)
            ccnd_msg(h, "face %u ready %u delay %i nrun %u",
                     faceid, q->ready, delay, q->nrun, face-
>surplus);
        return(delay);
    }
    q->ready = j;
    if (q->nrun >= 12 && q->nrun < 120) {
        /* We seem to be a preferred provider, forgo the
randomized delay */
        if (j == 0)
            delay += burst_nsec / 50;
        if (h->debug & 8)
            ccnd_msg(h, "face %u ready %u delay %i nrun %u
surplus %u",
```

```
                    (unsigned)ev->evint, q->ready, delay, q-
>nrun, face->surplus);
        return(delay);
    }
    /* Determine when to run again */
    for (i = 0; i < q->send_queue->n; i++) {
        content = content_from_accession(h, q->send_queue-
>buf[i]);
        if (content != NULL) {
            q->nrun = 0;
            delay = randomize_content_delay(h, q);
            if (h->debug & 8)
                ccnd_msg(h, "face %u queued %u delay %i",
                         (unsigned)ev->evint, q->ready, delay);
            return(delay);
        }
    }
    q->send_queue->n = q->ready = 0;
Bail:
    q->sender = NULL;
    return(0);
}
```

**Kode Sumber 7.4 Kode sumber yang digunakan untuk mengirim *ContentObject***

# DAFTAR PUSTAKA


[1] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, H. N. Briggs dan R. L. Braynard, "Networking Named Content," *International Conference on Emerging Networking Experiments*, pp. 1-12, 2009.

[2] B. B. Jason Min Wang, "Progressive Caching in CCN," dalam *Global Communication Conference (GLOBECOM) 2012 - Next Generation Networking and Internet Symposium*, 2012.

[3] P. Mahadevan, "CCNx 1.0 Tutorial," 2014.

[4] H. Xu, Z. Chen, R. Chen dan C. Junwei, "Live Streaming with Content Centric Networking," dalam *Third International Conference on Networking and Distributed Computing*, 2012.

[5] VideoLAN, "http://videolan.org," [Online]. Available: https://videolan.org/vlc/features.php. [Diakses 15 Maret 2014].

[6] Palo Alto Research Center, "ccnx.org," [Online]. Available: https://www.ccnx.org/releases/latest/doc/technical/. [Diakses 2014 April 16].

[7] G. Combs, "Wireshark," 16 March 2014. [Online]. Available: http://www.wireshark.org. [Diakses 16 Maret 2014].

[8] Protocog, "protocog.com," [Online]. Available: http://www.protocog.com/gerald_combs_interview.html. [Diakses 11 April 2014].

[9] AskUbuntu, "askubuntu.com," [Online]. Available: http://askubuntu.com/questions/42724/whats-the-meaning-of-the-ubuntu-logo-where-does-it-come-from. [Diakses 21 April 2014].

[10] Ubuntu, “ubuntu.com,” [Online]. Available: http://www.ubuntu.com/about. [Diakses 21 April 2014].

# BIODATA PENULIS

Radite Bayu Prakoso, lahir pada 10 Maret 1993, merupakan anak sulung dari 2 bersaudara. Bersekolah di SDN Sawunggaling 1 Surabaya pada tahun 1998-2004, kemudian melanjutkan pendidikan di SMPN 1 Sidoarjo pada tahun 2004-2007 dan mengenyam pendidikan SMA di SMAN 4 Sidoarjo. Entah secara beruntung atau memang terhitung cerdas, akhirnya berhasil melanjutkan pendidikan tinggi di Institut Teknologi Sepuluh Nopember, Surabaya pada tahun 2010. Selama berkuliah, penulis juga mengikuti beberapa organisasi seperti HMTC, dan Kepemanduan. Penulis juga lumayan sering mengikuti berbagai kegiatan kampus. Seperti, LKMM Pra-TD, LKMM TD dan PP LKMM. Penulis juga terkadang mengisi dalam acara-acara pelatihan sebagai panitia, fasilitator maupun pemandu. Hobi penulis adalah bermain game, *browsing*, membaca novel, dan *travelling*. Dan jangan lupa, ngoding. Karena semua mahasiswa Teknik Informatika diwajibkan memasukkan ngoding sebagai daftar hobinya, meskipun terpaksa ataupun akhirnya tidak bisa.