ANALISIS KINERJA *ON-PATH CACHING* DAN *OFF-PATH CACHING* PADA *INFORMATION-CENTRIC NETWORKING*

SKRIPSI

KEMINATAN TEKNIK KOMPUTER

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Muhamad Rizka Maulana
NIM: 135150300111006

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

PENGESAHAN

ANALISIS KINERJA *ON-PATH CACHING* DAN *OFF-PATH CACHING* PADA *INFORMATION-CENTRIC NETWORKING*

SKRIPSI

Keminatan Teknik Komputer

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh :
Muhamad Rizka Maulana
NIM: 135150300111006

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

Dosen Pembimbing I	Dosen Pembimbing II
Achmad Basuki, S.T, M.MG, Ph.D	Kasyful Amron, S.T., M.Sc
NIP: 19741118 200312 1 002	NIP: 19750803 200312 1 003

Mengetahui
Ketua Jurusan Teknik Informatika

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

PERNYATAAN ORISINALITAS

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

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

Malang, 11 Januari 2018

Muhamad Rizka Maulana

NIM: 135150300111006

KATA PENGANTAR

Puji syukur penulis panjatkan kehadirat Tuhan Yang Maha Esa, atas berkat rahmat dan hidayat-Nya, penulis dapat menyelesaikan skripsi dengan judul "ANALISIS KINERJA *ON-PATH CACHING* DAN *OFF-PATH CACHING* PADA *INFORMATION-CENTRIC NETWORKING*".

Dalam pelaksanaan dan penulisan tugas akhir ini penulis banyak mendapatkan dukungan dari berbagai pihak baik secara morel maupun materiel. Penulis juga ingin mengucapkan terima kasih sebesar-besarnya kepada :

1. Bapak Achmad Basuki, S.T, M.MG, Ph.D selaku dosen pembimbing pertama yang telah memberikan bantuan berupa ilmu yang sangat luas dan bimbingannya kepada penulis untuk menyelesaikan skripsi ini.
2. Bapak Kasyful Amron, S.T., M.Sc selaku dosen pembimbing kedua yang juga banyak memberi ilmu dan masukan untuk penulisan laporan skripsi ini.
3. Ayah, Ibu, dan seluruh keluarga saya atas seluruh segenap do'a dan dukungan yang telah diberikan.
4. Segenap dosen dan karyawan FILKOM Universitas Brawijaya yang telah membantu dalam penyelesaian skripsi ini.
5. Teman-teman Fakultas Ilmu Komputer, khususnya di program studi Teknik Komputer 2013 dan POROS FILKOM UB, yang telah membantu memberikan semangat untuk bisa menyelesaikan skripsi hingga akhir.
6. Seluruh pihak yang tidak dapat diucapkan satu persatu, penulis mengucapkan terima kasih atas bantuan do'a sehingga skripsi ini dapat terselesaikan.

Penulis menyadari bahwa dalam penulisan laporan skripsi ini skripsi ini masih jauh dari sempurna karena keterbatasan materi dan pengetahuan yang dimiliki penulis. Untuk itu penulis sangat mengharapkan kritik dan saran dari pembaca. Akhirnya, semoga skripsi ini bermanfaat bagi seluruh pihak.

Malang, 11 Januari 2018

Penulis

Email: muhamaul@gmail.com

ABSTRAK

Muhamad Rizka Maulana, Analisis Kinerja *On-Path Caching* Dan *Off-Path Caching* Pada *Information-Centric Networking*

Pembimbing: Achmad Basuki, S.T, M.MG, Ph.D dan Kasyful Amron, S.T., M.Sc

Peningkatan lalu lintas data di Internet yang sangat tinggi tidak didukung oleh arsitektur Internet saat ini yang masih bersifat *host-centric*. Paradigma *Information-centric Networking* (ICN) berusaha untuk mengatasi masalah tersebut dengan model komunikasi yang bersifat *content-centric* atau *information-centric*. Beberapa konsep arsitektur berbasis ICN telah diusulkan seperti CCN, DONA, dan NetInf. Arsitektur baru tersebut umumnya memiliki tiga kesamaan konsep kerja, yaitu operasi *Publish-Subscribe*, *In-network Caching*, dan *Content-oriented Security*. *In-network caching* memainkan peran penting dalam meningkatkan kinerja jaringan dengan cara menyimpan konten di *cache* yang tersebar di jaringan. Terdapat 2 jenis mekanisme *in-network caching*, yaitu *on-path caching* dan *off-path caching*. Sebelum arsitektur ICN diterapkan, perlu adanya studi analisis kinerja *in-network caching* agar nantinya dapat dijadikan sebagai acuan. Penelitian ini bertujuan untuk menganalisis dan membandingkan kinerja *on-path caching* dan *off-path caching* pada topologi Biznet menggunakan simulator Icarus. Metrik kinerja yang digunakan adalah *cache hit ratio* (CHR), latensi, dan *link load*.

Hasil pengujian menunjukkan bahwa dalam aspek *cache hit ratio*, mekanisme *off-path caching* memiliki CHR 12,45% lebih tinggi dibandingkan mekanisme *on-path caching*. Sementara dalam aspek *link load*, mekanisme *on-path caching* memiliki *link load* 63,14% lebih rendah dibandingkan dengan mekanisme *off-path caching*. Dan dalam aspek latensi, mekanisme *on-path caching* memiliki latensi 42,07% lebih rendah dibandingkan dengan mekanisme *off-path caching*. Selain itu, ditemukan pula bahwa semakin besar kapasitas *cache* dan nilai α berarti semakin tinggi *cache hit ratio*, semakin rendah *link load*, serta semakin rendah latensi.

Kata kunci: *Information-centric networking*, *in-network caching*, analisis kinerja, simulator Icarus

ABSTRACT

Muhamad Rizka Maulana, Performance Analysis of On-Path Caching and Off-Path Caching in Information-Centric Networking

Supervisors: Achmad Basuki, S.T, M.MG, Ph.D and Kasyful Amron, S.T., M.Sc

The massively increasing Internet traffic is not supported by the current Internet architecture which is still based on host-centric communication. Information-centric networking (ICN) paradigm has been proposed to resolve that problem with content-centric communication. Some ICN architectures have been proposed, such as CCN, DONA, and NetInf. Those proposed architectures generally have three main concepts, which are publish-subscribe operation, in-network caching, and content-oriented security. In-network caching plays a very important role in ICN paradigm. It can improve network performance by saving content in caches that are spread in the network. There are two types of in-network caching, which are on-path caching and off-path caching. Before the ICN architecture is deployed to replace current architecture, it is necessary to conduct a performance analysis of in-network caching so that the result can be used as a reference for the deployment. This research aims to analyze and to compare on-path and off-path caching strategies on Biznet topology using Icarus simulator. Cache hit ratio, latency, and link load are used as performance metrics.

The result of this research shows that off-path caching has 12.45% higher cache hit ratio than on-path caching, and on-path caching has 63.14% lower link load and 42,07% *lower latency than off-path caching. In addition to that, it is worth noted that bigger cache capacity and α value results in higher cache hit ratio, lower link load, and lower latency.*

Keywords: Information-centric networking, in-network caching, performance analysis, Icarus simulator

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB 1 PENDAHULUAN

1.1 Latar Belakang

Teknologi Internet yang berkembang dengan pesat sejak tahun 1960-an telah menyebabkan peningkatan jumlah konten dan lalu lintas data yang sangat signifikan. Berdasarkan prediksi Cisco Visual Networking Index (2016), pada tahun 2020 lalu lintas IP secara global akan mencapai 2,3 ZB (Zetabita) per tahunnya, atau 194 EB per bulan. Lalu lintas data yang sangat besar tersebut tidak didukung oleh arsitektur komunikasi Internet saat ini yang masih berbasis pada komunikasi *host-centric* dan dirancang hanya untuk berbagi sumber daya. Paradigma *Information-Centric Networking* (ICN) diusulkan untuk mengatasi masalah tersebut dengan merancang ulang arsitektur Internet yang sebelumnya *host-centric* menjadi *information-centric* atau *content-centric*. Pada ICN, pengguna tidak peduli di mana konten itu berada, tetapi mementingkan konten apa yang ia inginkan. Ada banyak arsitektur ICN yang sudah diusulkan, antara lain *Data-Oriented Network Architecture* (DONA) (Koponen, et al., 2007), *Publish/Subscribe Internet Routing Paradigm* (PSIRP) (Tarkoma, Ain, & Visala, 2009), *Content-Centric Networking* (CCN) (Jacobson, et al., 2009), dan *Network of Information* (NetInf) (Dannewitz, et al., 2013). Namun begitu, dari arsitektur-arsitektur yang sudah diusulkan terdapat 3 kesamaan konsep, yaitu operasi *Publish-Subscribe*, *Universal Caching*, dan *Content-oriented Security* (Ghodsi, et al., 2011).

Universal caching atau *in-network caching* adalah metode penyimpanan data sementara di dalam jaringan dan merupakan salah satu topik penelitian ICN yang cukup aktif dilakukan. Metode *caching* sudah diakui sebagai salah satu cara yang efektif untuk mengatasi masalah *scalability* pada sistem berskala besar dan berpengaruh langsung terhadap penyebaran konten. Saat metode *caching* diterapkan di dalam jaringan, akan terjadi penurunan lalu lintas data dan latensi pengiriman karena konten akan tersebar. Hal ini tentunya sangat bermanfaat bagi *Internet Service Provider* (ISP) dan bagi konsumen, karena lalu lintas data yang keluar dari ISP akan sangat berkurang dan konten akan lebih dekat dengan konsumen. Namun, menurut Wang (2015), metode *caching* yang sudah dipakai pada arsitektur Internet saat ini tidak banyak membantu dalam konteks arsitektur ICN karena tidak tersebar di dalam jaringan. Meskipun metode *caching* secara umum sudah diteliti secara ekstensif, perbedaan arsitektur ICN dan Internet saat ini menjadikan metode *caching* harus diteliti kembali agar dapat bekerja dengan baik pada arsitektur ICN (Rossi & Rossini, 2012). Oleh karena itu, penelitian pada bidang *in-network caching* berfokus pada analisis dan perancangan strategi *caching* yang efektif agar meningkatkan kinerja sistem (Wang, 2015).

Secara umum, mekanisme *in-network caching* dapat dibedakan menjadi dua jenis, yaitu *on-path caching* dan *off-path caching*. Dalam mekanisme *on-path caching*, konten akan disimpan sementara di jalur atau *node* yang dilalui oleh

paket Data. Hal ini tentunya akan menimbulkan *redundancy* karena konten akan disimpan di banyak *node*. Sedangkan dalam mekanisme *off-path caching*, konten akan disimpan pada *node* yang dipilih berdasarkan aturan yang sudah ditentukan sebelumnya, misalnya dengan metode Hash-routing (Saino, Psaras, & Pavlou, 2013). Cara tersebut hampir memastikan adanya penemuan konten karena *node* yang bertanggung jawab untuk menyimpannya sudah dipetakan dengan jelas menggunakan fungsi *hash*. Namun, di sisi lain, mekanisme *off-path caching* tersebut akan meningkatkan *cost* tambahan untuk koordinasi antar-*node* dan *re-routing* paket.

Dengan meningkatnya lalu lintas Internet global setiap tahun dan visi ICN sebagai arsitektur Internet masa depan, diperlukan analisis kinerja *in-network caching* agar nantinya dapat dijadikan sebagai acuan saat arsitektur ICN akan diterapkan. Penelitian yang dilakukan oleh Gupta, Shailendra, dan Girish (2016) menganalisis kinerja beberapa mekanisme *content placement policy* dan *content replacement policy* pada arsitektur ICN pada topologi Rocketfuel, GEANT, dan TISCALI menggunakan simulator Icarus. Parameter pengujian pada penelitian tersebut adalah *cache hit ratio* dan *relative hops*. Penelitian lain yang berkaitan dengan uji kinerja *in-network caching* dilakukan oleh Kuliesius dan Paulauskas (2015), yang mengevaluasi kinerja *on-path* dan *off-path* caching pada topologi LITNET. Penelitian tersebut menggunakan simulator Icarus dan parameter pengujian yang dipilih adalah *cache hit ratio* dan *link load* serta analisis hasil pengujian tidak dilakukan secara mendalam.

Penelitian ini bertujuan untuk menganalisis dan membandingkan kinerja mekanisme *on-path* dan *off-path caching*, khususnya dalam aspek *cache hit ratio*, *link load*, dan latensi. Pengujian kinerja dilakukan melalui sebuah simulasi pada simulator Icarus menggunakan topologi jaringan salah satu ISP di Indonesia, Biznet.

1.2 Rumusan Masalah

Berdasarkan latar belakang di atas, maka rumusan masalah penelitian adalah sebagai berikut:

1. Bagaimana perbandingan kinerja mekanisme *on-path* dan *off-path caching*?
2. Manakah strategi *in-network caching* yang cocok dengan topologi seperti Biznet berdasarkan aspek *cache hit ratio*, *link load*, dan latensi?

1.3 Tujuan

Penelitian ini bertujuan untuk menganalisis dan membandingkan kinerja *in-network caching* pada topologi Biznet menggunakan simulator Icarus. Simulator Icarus adalah simulator *caching* yang didesain khusus untuk arsitektur ICN.

1.4 Manfaat

Manfaat dari penelitian ini adalah sebagai berikut:

1. Memberikan penjelasan tentang bagaimana cara menguji kinerja *in-network caching* pada simulator Icarus.
2. Memberi gambaran tentang perbandingan kinerja *on-path* dan *off-path caching* pada topologi Biznet.
3. Dapat dijadikan sebagai referensi dalam penelitian yang berkaitan *in-network caching*.
4. Dapat dijadikan sebagai dasar pengambilan keputusan untuk menentukan strategi *in-network caching* yang akan dipakai saat arsitektur ICN diterapkan untuk menggantikan arsitektur Internet saat ini.

1.5 Batasan Masalah

Agar penelitian ini lebih terarah, maka batasan-batasan masalah dalam penelitian ini adalah sebagai berikut:

1. Pengujian kinerja *in-network caching* dilakukan melalui simulasi pada simulator Icarus.
2. Strategi *in-network caching* yang diuji antara lain Leave Copy Everywhere, ProbCache, Random, Hash-routing Symmetry, Hash-routing Asymmetry, dan Hash-routing Multicast.
3. Parameter yang diujikan antara lain *cache hit ratio*, *link load*, dan latensi.
4. Topologi pengujian yang digunakan adalah topologi *existing* Biznet untuk Pulau Jawa dan Bali.
5. Strategi *content replacement* yang digunakan adalah Least Recently Used.

1.6 Sistematika Pembahasan

Bagian ini berisi penjelasan singkat masing-masing bab dari sistematika pembahasan penelitian. Bab-bab tersebut antara lain:

BAB 1 PENDAHULUAN

Bab ini mencakup latar belakang yang merupakan alasan penulisan, rumusan masalah sebagai inti permasalahan, batasan masalah sebagai batas cakupan bahasan, tujuan dari penelitian, manfaat yang diperoleh dari penelitian, dan sistematika penulisan laporan penelitian.

BAB 2 LANDASAN KEPUSTAKAAAN

Bab landasan kepustakaan menjelaskan tentang teori – teori yang di pakai dalam penelitian, temuan dan bahan penelitian yang mungkin sebelumnya diperoleh dari beberapa referensi yang menunjang penelitian dalam penulisan skripsi.

BAB 3 METODOLOGI

Bab Metodologi berisi langkah-langkah dalam melakukan penelitian. Langkah-langkah tersebut seperti analisis kebutuhan, implementasi, pengujian, serta analisis hasil dibahas secara umum.

BAB 4 IMPLEMENTASI DAN PENGUJIAN

Bab ini berisi persiapan simulasi yang dimulai dengan pembuatan topologi pengujian sampai dengan konfigurasi simulasi dan dilanjutkan dengan cara melakukan pengujian berdasarkan rancangan yang telah dibuat.

BAB 5 ANALISIS DAN PEMBAHASAN

Bab ini menyajikan data hasil pengujian beserta analisis dari dari hasil pengujian tersebut. Proses analisis dilakukan terhadap mekanisme *on-path caching* dan *off-path caching*, lalu membandingkan kinerja dari kedua mekanisme tersebut.

BAB 6 PENUTUP

Pada Bab Penutup dipaparkan kesimpulan dari pelaksanaan penelitian. Kesimpulan ini dibuat berdasarkan hasil analisis terhadap hasil pengujian. Saran-saran juga diberikan agar hasil dari penelitian ini dapat diperbaiki dan disempurnakan apabila penelitian ini dikembangkan kemudian.

BAB 2 LANDASAN KEPUSTAKAAN

Bab ini berisi landasan kepustakaan yang terdiri dari kajian terhadap penelitian-penelitian terkait dan dasar teori yang mendukung dalam melakukan penelitian.

2.1 Penelitian Terkait

Penelitian yang dilakukan oleh Kalla dan Sharma (2016) bertujuan untuk mengevaluasi kinerja *on-path*, *off-path,* dan *edge caching* serta mengusulkan sebuah mekanisme *caching* baru, yaitu EDOP (Edge Off-path) *caching*. Pengujian dilakukan sebanyak sepuluh kali pada enam topologi riil, yaitu Abilene, Geant, Germany50, India35, Exodus US, dan Ebone Europe dengan metrik kinerja *hit ratio, average retrieval delay, unique contents cached* (banyaknya konten yang berbeda di dalam *cache*), dan *percentage of external traffic* (proposi lalu lintas data yang melewati *links* eksternal). Berdasarkan hasil penelitian tersebut, kinerja *off-path caching* lebih baik dibandingkan dengan *on-path caching* dalam seluruh metric kinerja yang diuji. Namun peneliti tersebut tidak menyebutkan simulator apa yang dipakai karena mengembangkan simulator sendiri.

Jeon, Lee, dan Song (2013) dalam penelitiannya yang berjudul "*On-Path Caching in Information-Centric Networking*" berfokus pada evaluasi kinerja *on-path caching*. Beberapa strategi *content placement* (Always, FIX(P), Leave Copy Down, WAVE) dan *content replacement* (least recently used, least-frequently used, *locality-based*) diujikan untuk melihat pengaruh kedua strategi tersebut terhadap kinerja jaringan. Pengujian dilakukan pada topologi *tree* dengan 127 *node* internal dan 128 *node leaf* menggunakan simulator yang tidak disebutkan namanya. Selain itu, metrik kinerja yang diujikan hanya *cache hit ratio* dan *average hop counts* (rerata jumlah *hop* yang diperlukan untuk mendapatkan sebuah konten).

Penelitian oleh Rossini dan Rossi (2011) yang berjudul "*A dive into the caching performance of Content Centric Networking*" berfokus pada evaluasi kinerja strategi *content replacement* (LRU dan Random) pada arsitektur *Content Centric Networking*. Penelitian dilakukan pada topologi GEANT dan torus 4x4 menggunakan simulator ccnSim. Kelebihan penelitian tersebut adalah penyesuaian lingkungan simulasi dengan kondisi riil Internet. Namun evaluasi kinerja hanya terfokus pada strategi *content replacement*, tanpa memperhatikan strategi *content placement*.

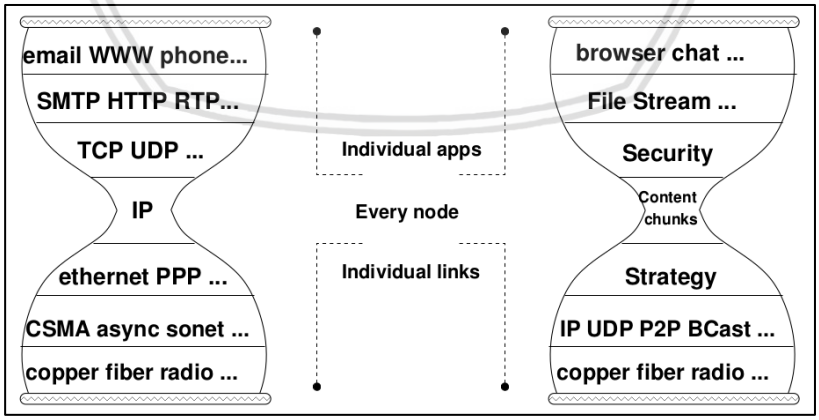
Dari kajian terhadap penelitian terkait yang telah dijelaskan di atas, maka penelitian ini akan berfokus pada analisis dan perbandingan kinerja *on-path* dan *off-path caching* dalam aspek *cache hit ratio*, *link load*, dan latensi. Penelitian dilakukan pada simulator Icarus dengan lingkungan simulasi yang disesuaikan mendekati kondisi riil Internet.

2.2 Dasar Teori

Dasar teori membahas teori-teori yang mendukung penelitian. Teori-teori tersebut antara lain teori *information-centric networking*, *in-network caching*, popularitas konten, simulator Icarus, metrik kinerja *in-network caching*, dan Graph Modelling Language (GML).

2.2.1 Information-centric Networking

Information-centric Networking (ICN) adalah paradigma baru dalam arsitektur Internet yang diajukan untuk menggantikan arsitektur Internet saat ini. Arsitektur Internet saat ini bersifat *host-centric* yang mana pada saat suatu komputer ingin meminta informasi, maka komputer tersebut harus mengetahui di mana informasi itu berada sebelum mengirimkan permintaan informasi. Lokasi dari komputer tujuan tersebut direpresentasikan dengan alamat IP. Selain itu, arsitektur Internet saat ini dirancang untuk berbagi sumber daya seperti pada awal pengembangannya di tahun 1960-1970, di mana pada saat itu jumlah perangkat terbatas dan mahal sehingga harus ada mekanisme pembagian sumber daya. Hal tersebut tentunya tidak sesuai dengan keadaan saat ini di mana jumlah perangkat sangat banyak dan lalu lintas Internet global sangat tinggi serta meningkat tiap tahunnya. Beberapa arsitektur ICN yang sudah diusulkan antara lain DONA (Koponen, et al., 2007), PSIRP (Tarkoma, Ain, & Visala, 2009), CCN (Jacobson, et al., 2009), dan NetInf (Dannewitz, et al., 2013). Terdapat 3 kesamaan konsep dari arsitektur-arsitektur yang sudah diusulkan, yaitu operasi *Publish-Subscribe*, *Universal Caching*, dan *Content-oriented Security*. Namun, arsitektur yang cukup menjanjikan untuk mengatasi masalah-masalah pada arsitektur saat ini dan pantas untuk menjadi dasar arsitektur ICN adalah *Content-centric Networking* (CCN) yang diusulkan oleh Van Jacobson, et al. (2009). Pada penelitian ini, arsitektur ICN yang dijadikan sebagai acuan adalah CCN.



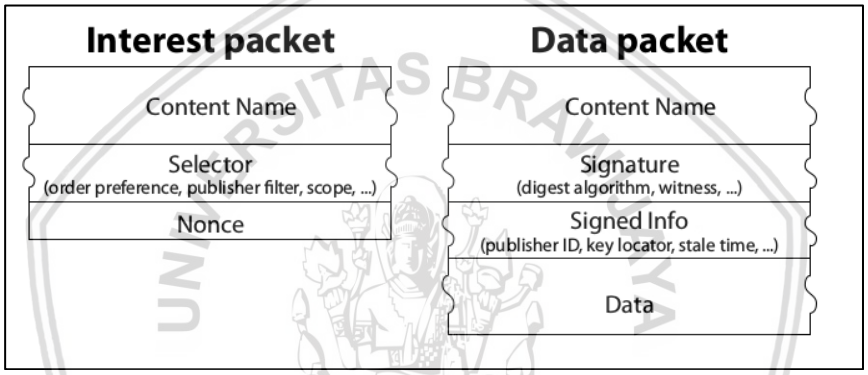
Gambar 2.1 Perbandingan layer Internet saat ini dengan ICN
Sumber: Van Jacobson, et al. (2009)

Gambar 2.1 adalah perbandingan layer pada Internet dan ICN. Pada gambar tersebut, komunikasi pada Internet saat ini yang berbasis IP yang bersifat *host-centric* diganti dengan berbasis *content chunks* yang bersifat *content-centric*.

Pengguna tidak peduli di mana konten itu berada, tetapi lebih mementingkan konten apa yang ia inginkan. Oleh karena itu, komunikasi pada ICN bersifat *consumer-driven*, yang berarti bahwa komunikasi terjadi pada saat pengguna menginginkan suatu informasi. Model komunikasi seperti ini mirip dengan model komunikasi pada HTTP yang bersifat *Request-Response*.

2.2.1.1 *Jenis-jenis Paket ICN*

Pada arsitektur ICN terdapat dua jenis paket, yaitu Interest dan Data. Paket Interest digunakan untuk meminta suatu konten dengan cara mengirimkannya secara *broadcast* pada jaringan. Saat suatu *node* menerima paket Interest dan memiliki konten yang diinginkan oleh paket Interest, maka *node* tersebut akan mengirimkan paket Data yang berisi konten yang diminta ke *node* yang membuat paket Interest. Gambar 2.2 adalah struktur paket Interest dan paket Data pada arsitektur ICN.



Gambar 2.2 Struktur paket pada ICN
Sumber: Van Jacobson, et al. (2009)

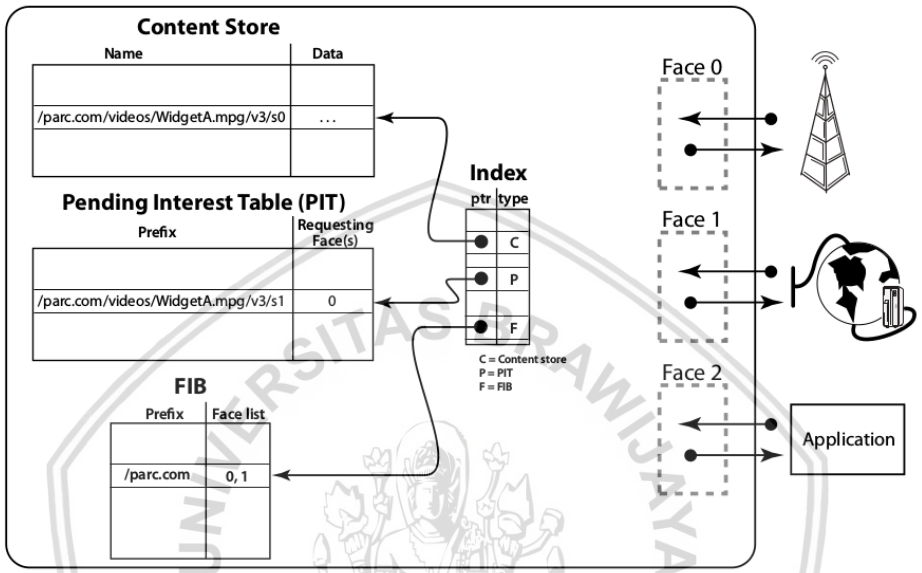
Struktur paket Interest terdiri atas Content Name, Selector, dan Nonce. Content Name berisi nama konten yang diminta oleh pengguna. Tiap *node* yang menerima paket Interest akan memeriksa apakah ia memiliki konten sesuai dengan Content Name. Selector bersifat opsional dan digunakan untuk menentukan kriteria tambahan pada konten yang akan diminta. Dan Nonce berisi angka acak untuk menghindari adanya duplikasi paket Interest. Apabila menerima paket Interest lagi dengan Nonce yang sama, maka *node* akan mengabaikan paket Interest tersebut.

Sedangkan struktur paket Data adalah Content Name yang berisi nama konten yang diminta oleh pengguna, Signature yang berisi *signature* dari sebuah konten, Signed Info yang berisi informasi tambahan untuk memverifikasi Signature dari paket Data dan untuk memastikan bahwa Signature pada paket Data milik *content publisher* tertentu, dan Data yang berisi konten yang diminta oleh pengguna.

Pada penulisan penelitian ini, penggunaan istilah paket Interest dapat dipertukarkan dengan paket *request* atau permintaan, dan paket Data dapat dipertukarkan dengan paket *content* atau konten.

2.2.1.2 *Forwarding Model Pada ICN*

Forwarding model pada ICN memiliki tiga struktur utama, yaitu *Content Store*, *Pending Interest Table* (PIT), dan *Forwarding Information Base* (FIB). Content Store ialah tempat penyimpanan konten sementara berdasarkan strategi *caching* yang diterapkan, PIT berfungsi untuk mencatat setiap paket Interest yang masuk beserta dari *face* mana paket Interest tersebut datang, sedangkan FIB berfungsi untuk meneruskan paket Interest ke *node* lain.



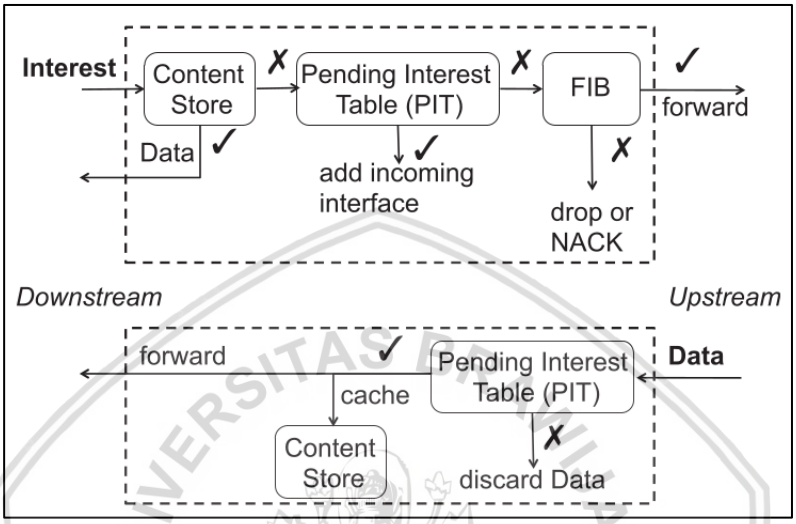
Gambar 2.3 Struktur *forwarding model* pada sebuah *node*
Sumber: Van Jacobson, et al. (2009)

Gambar 2.3 adalah struktur *forwarding model* suatu *node* pada arsitektur ICN. Saat suatu paket Interest masuk ke suatu *node* melalui sebuah *face*, maka *node* tersebut pertama kali akan memeriksa apakah ia memiliki konten di Content Store yang sesuai dengan Content Name di paket Interest. Apabila konten ditemukan, maka konten tersebut akan diteruskan ke *face* asal dari paket Interest tersebut.

Apabila konten tidak ditemukan di Content Store, maka paket Interest akan diperiksa di PIT. Jika di PIT sudah ada paket Interest untuk konten yang sama, maka *face* asal paket Interest tersebut akan ditambahkan pada PIT dan paket Interest akan dihapus karena sudah ada paket Interest untuk konten yang sama.

Jika di PIT tidak ada catatan paket Interest untuk konten yang sama, maka paket Interest akan diperiksa di FIB apakah dimungkinkan untuk diteruskan ke *node* lain berdasarkan tabel *forwarding* di FIB. *Prefix* paket Interest akan dicocokkan dengan *prefix* di FIB dan bila cocok akan diteruskan ke *node* lain atau langsung ke *content source* lalu *face* asal Interest akan ditambahkan di PIT. Bila *prefix* tidak ditemukan di FIB, maka paket Interest akan dihapus. Catatan paket Interest di PIT yang terlalu lama tidak mendapatkan Data akan kedaluwarsa dan dihapus dari PIT.

Saat suatu *node* menerima paket Data, *node* tersebut akan memeriksa apakah ia memiliki catatan untuk paket Data tersebut di PIT miliknya. Jika cocok, *node* akan menyimpan Data tersebut di *Content Store* dan meneruskannya ke *face* yang tercantum di PIT lalu menghapus *face* tersebut setelah diteruskan. Bila tidak ada paket Interest untuk Data tersebut di dalam PIT, maka paket Data akan dihapus. Gambar 2.4 adalah alur pemrosesan paket Interest dan paket Data pada arsitektur ICN.



Gambar 2.4 Pemrosesan paket Interest dan paket Data
Sumber: Yi, et al. (2013)

2.2.2 *In-network Caching*

Secara umum, *cache* adalah sebuah media penyimpanan sementara. Dalam arsitektur sistem komputer, *cache* adalah media penyimpanan berkecepatan tinggi dengan kapasitas kecil untuk menyimpan perintah-perintah kerja CPU. Sedangkan dalam bidang web, *cache* digunakan untuk menyimpan konten-konten web agar dapat diakses kembali oleh pengguna secara cepat. Penggunaan *cache* di dalam web dapat menghemat *bandwidth* jaringan, mengurangi latensi ke *end client*, dan mengurangi beban server (Danzig, Hall, & Schwartz, 1993).

Arsitektur ICN juga memanfaatkan *cache* untuk meningkatkan kinerja jaringan. Dalam arsitektur ICN, *universal caching* atau *in-network caching* adalah mekanisme penyimpanan data sementara di dalam jaringan dan merupakan bagian yang tidak terpisahkan dari arsitektur ICN. *In-network caching* bertujuan untuk menurunkan lalu lintas data dan latensi pengiriman dengan cara menempatkan *cache* di dalam jaringan. Berbeda dengan arsitektur *Content Delivery Network* (CDN) dan Peer-to-Peer (P2P) yang menempatkan *cache* dekat dengan pengguna, mekanisme *in-network caching* mencoba untuk menyebar *cache* di seluruh jaringan (*cache* menyatu dengan *router*). Apabila suatu replika konten ditemukan pada suatu *cache*, maka peristiwa tersebut dinamakan *cache hit*. Sebaliknya, jika replika konten tidak ditemukan, dinamakan *cache miss*. Secara umum, terdapat dua mekanisme yang saling melengkapi dalam *in-*

network caching, yaitu *content placement* dan *content replacement*. *Content placement* mengatur bagaimana cara menyimpan konten dalam *cache* tersebar di jaringan, sedangkan *content replacement* mengatur penghapusan konten saat *cache* penuh. Mekanisme *content placement* masih dapat dikategorikan menjadi dua, yaitu *on-path caching* dan *off-path caching*. Strategi-strategi pada *on-path caching* tidak bisa diterapkan sebagai *off-path caching*, dan begitu pula sebaliknya, strategi-strategi pada *off-path caching* tidak bisa diterapkan sebagai *on-path caching*.

2.2.2.1 *Content Placement: On-path Caching*

Dalam *on-path caching*, konten akan disimpan secara sementara di jalur atau *node* yang dilalui oleh paket Data. Hal ini tentunya akan menimbulkan *redundancy* karena konten akan disimpan di *node* yang dilaluinya. Namun *on-path caching* juga dapat meningkatkan peluang ditemukannya konten walaupun bersifat oportunistis. Jenis-jenis strategi *on-path caching* sebagai berikut:

1. Leave Copy Everywhere

Strategi Leave Copy Everywhere (LCE) akan menyimpan replika konten pada setiap *cache* yang dilalui oleh paket Data menuju *receiver*. Meskipun hal tersebut akan menyebabkan *redundancy* konten di dalam jaringan, tetapi strategi ini cocok untuk jaringan dengan lalu lintas data yang tinggi. Strategi ini adalah strategi *default* dari arsitektur CCN yang diusulkan Van Jacobson.

2. Random

Strategi Random akan menyimpan suatu konten pada satu *cache* yang dipilih secara acak di jalur pengiriman paket Data.

3. ProbCache

Strategi ProbCache diusulkan oleh Psaras, et al. (2012). Strategi ini akan menyimpan suatu replika konten pada satu *node* terbaik di sepanjang jalur pengiriman berdasarkan persamaan probabilitas. Tujuan dari ProbCache adalah mengurangi *cache reduncancy* sehingga konten yang disimpan di jaringan *cache* lebih bervariasi.

Strategi ProbCache mengasumsikan paket *request* memiliki *header* tambahan bernama Time Since Inception (TSI) sedangkan paket *content* memiliki *header* tambahan TSI dan Time Since Birth (TSB). TSI dan TSB memiliki sifat yang sama seperti Time To Live (TTL) *field* pada format datagram IP, tetapi TSI dan TSB akan bertambah satu tiap melewati satu *router*. Dengan kata lain, TSI dan TSB mengindikasikan jumlah hop yang dilalui oleh paket *request* atau *content*. Saat sebuah *content source* menerima paket *request*, *content source* akan memindahkan nilai TSI paket *request* ke TSI paket *content* yang akan dikirim. Nilai TSI pada paket *content* tersebut tidak berubah sampai paket tiba di *receiver*. Strategi ProbCache juga mengasumsikan konten akan tersimpan di dalam suatu jalur *cache* dalam jangka waktu tertentu. Lamanya suatu konten disimpan dalam jalur *cache* disebut *Target Time Window* (T_{tw}). Besarnya T_{tw} adalah 10 detik.

Nilai TSI, TSB, dan T_{tw} di atas digunakan dalam persamaan 2.1 untuk menentukan di *cache* mana sebuah konten akan disimpan.

$$ProbCache\ x\ = \frac{\sum_{i=1}^{c-x-1} N_i}{T_{tw} N_x} \times \frac{x}{c} \qquad (2.1)$$

Berikut adalah keterangan dari persamaan 2.1 di atas:

- N_i : Kapasitas sebuah *cache*
- N_x : Rerata kapasitas *cache* sepanjang jalur yang dilewati paket
- T_{tw}: *Target Time Window*. Lamanya suatu konten disimpan dalam jalur *cache*. Dalam ProbCache, T_{tw} bernilai 10 detik.
- x : *Time Since Inception* (TSI). Banyaknya *hop* yang dilalui oleh paket *request* dari *receiver* ke *content source*.
- c : *Time Since Birth* (TSB). Banyaknya *hop* yang dilalui oleh paket *content* dari *content source* ke *receiver*.

2.2.2.2 *Content Placement: Off-path Caching*

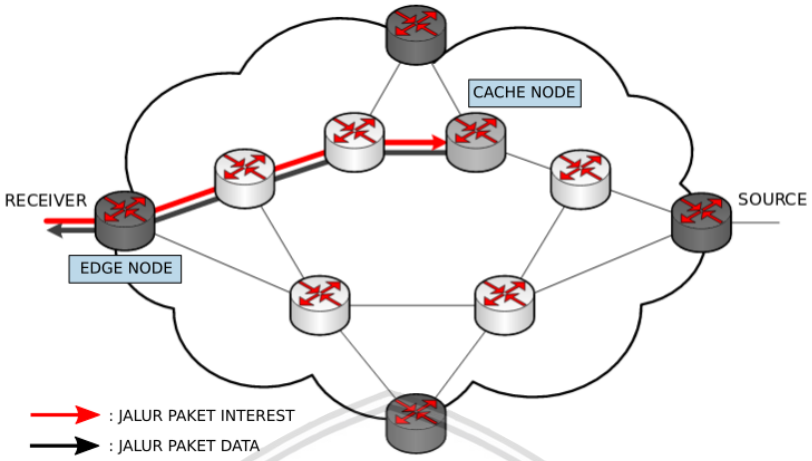
Mekanisme *off-path caching* akan menyimpan konten pada *node* yang dipilih berdasarkan aturan yang sudah ditentukan sebelumnya. Metode Hash-routing dapat digunakan untuk memilih *node* mana yang akan digunakan sebagai *cache* berdasarkan fungsi *hash* (Saino, et al., 2013). Dengan mekanisme *off-path caching* berbasis Hash-routing, sebuah paket Interest atau konten akan diteruskan ke suatu *cache* berdasarkan fungsi *hash* untuk meminta atau menyimpan konten.

Mekanisme *off-path caching* yang berbasis Hash-routing memiliki dua entitas *node*, yaitu *edge node* dan *cache node*. *Edge node* berada di dekat *receiver* serta bertugas untuk melakukan komputasi fungsi *hash* dan meneruskan paket Interest dan konten ke *cache node*. Sedangkan *cache node* bertugas untuk menyimpan konten. Jenis-jenis strategi *off-path caching* yang berbasis Hash-routing sebagai berikut:

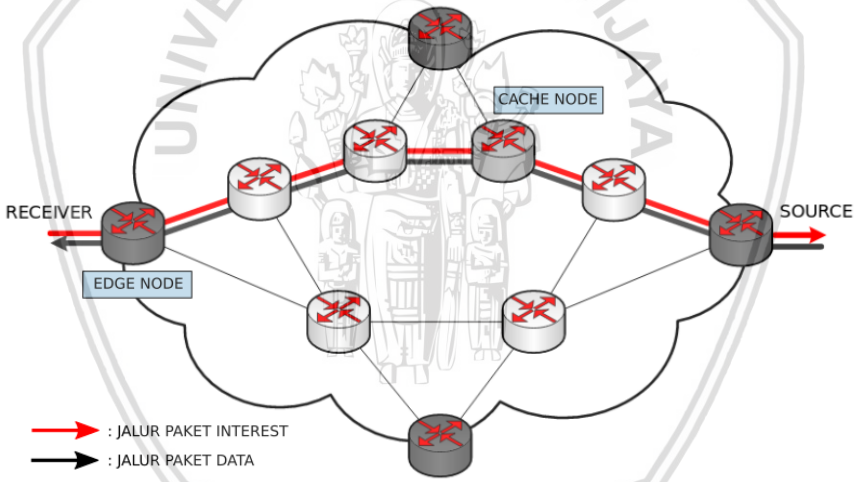
1. Hash-routing Symmetry

Hash-routing Symmetry adalah strategi *off-path caching* di mana jalur pengiriman yang dilalui oleh paket Data sama dengan jalur yang dilalui oleh paket Interest. Saat menerima paket Interest dari *receiver*, *edge node* akan melakukan komputasi fungsi *hash* untuk memetakan *cache node* lalu meneruskan paket Interest ke *cache node* tujuan. Apabila memiliki konten yang diminta (*cache hit*), *cache node* tujuan akan mengirim paket Data ke peminta melalui jalur yang dilalui oleh paket Interest. Apabila tidak memiliki konten yang diminta (*cache miss*), *cache node* tujuan akan meneruskan paket Interest ke *content source*. *Content source* akan mengirimkan paket Data ke *receiver* melalui jalur yang dilewati paket Interest dan *cache node* menyimpan replika konten. Apabila *content source* tidak memiliki konten yang diminta, maka paket Interest

akan ditolak. Gambar 2.5 dan Gambar 2.6 mengilustrasikan Hash-routing Symmetry apabila terjadi *cache hit* dan *cache miss*.



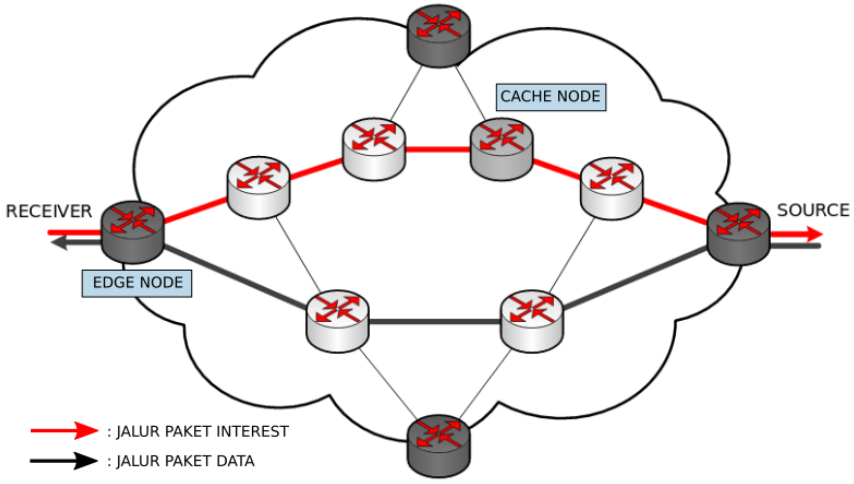
Gambar 2.5 *Cache hit* terjadi di *cache node*
Sumber: Lorenzo Saino (2013)



Gambar 2.6 *Cache miss* pada Hash-routing Symmetry
Sumber: Lorenzo Saino (2013)

2. Hash-routing Asymmetry

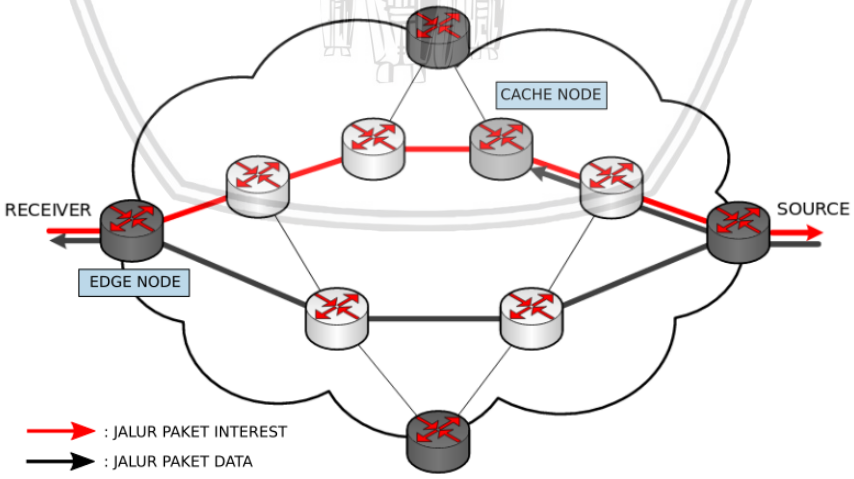
Perbedaan Hash-routing Symmetry dan Hash-routing Asymmetry hanya terletak pada bagaimana *edge node* di dekat *content source* mengirimkan paket Data ke *receiver* saat terjadi *cache miss*. Pada strategi Hash-routing Asymmetry, *Edge node* di dekat *content source* akan menentukan jalur terpendek menuju *receiver* dan paket Data akan dikirim melalui jalur terpendek tersebut. Paket Data tidak akan disimpan pada *cache node*. Namun, apabila pada jalur terpendek tersebut melewati *cache node* tujuan (yang sebelumnya terjadi *cache miss*), maka konten tersebut akan disimpan di *cache node* tersebut. Gambar 2.7 mengilustrasikan Hash-routing Asymmetry saat terjadi *cache hit* dan *cache miss*.



Gambar 2.7 *Cache miss* pada Hash-routing Asymmetry
Sumber: Lorenzo Saino (2013)

3. Hash-routing Multicast

Hash-routing Multicast merupakan kombinasi dari Hash-routing Symmetry dan Hash-routing Asymmetry dengan memanfaatkan keuntungan dari kedua strategi tersebut. Hash-routing Multicast dapat menyimpan replika konten sekaligus mengirimkannya melalui jalur terpendek. Saat terjadi *cache miss* dan *content source* harus bertanggung jawab untuk membalas paket Interest, *edge node* di dekatnya akan mengirimkan replika konten ke *cache node* dan *receiver* melalui jalur terpendek. Saat terjadi *cache hit*, Hash-routing Multicast memiliki perilaku yang sama dengan kedua strategi Hash-routing sebelumnya. Gambar 2.8 mengilustrasikan Hash-routing Multicast saat terjadi *cache hit* dan *cache miss*.



Gambar 2.8 *Cache miss* pada Hash-routing Multicast
Sumber: Lorenzo Saino (2013)

Meskipun memiliki karakter yang cukup berbeda, tetapi ketiga strategi *off-path caching* di atas akan memiliki karakter yang sama apabila *cache node* hasil fungsi *hash* terletak di jalur pengiriman konten saat terjadi *cache miss*.

2.2.2.3 *Content Replacement*

Cache yang sudah penuh harus menghapus beberapa kontennya agar dapat menampung konten-konten baru. Berikut adalah beberapa strategi *content replacement.*

1. Least Recently Used (LRU)

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

2. Least Frequently Used (LFU)

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

3. First In First Out (FIFO)

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

4. Random

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

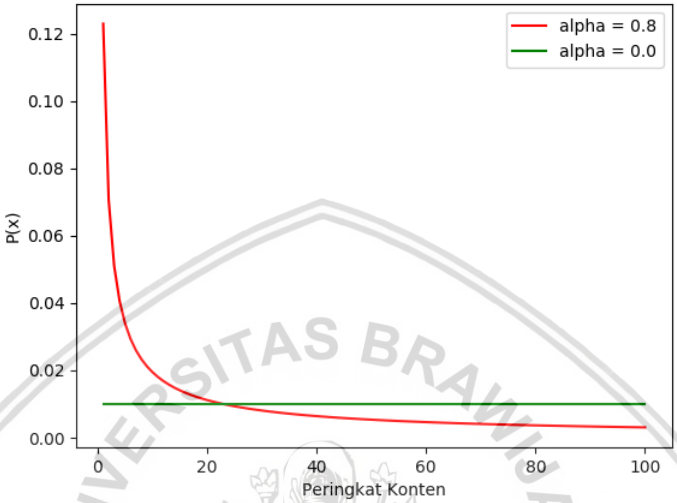
2.2.3 Popularitas Konten

Popularitas konten adalah jumlah permintaan untuk suatu konten yang mana semakin tinggi jumlah permintaan maka konten tersebut semakin populer. Studi tentang popularitas atau frekuensi kemunculan suatu objek pertama kali dilakukan oleh Auerbach (1913) dan selanjutnya oleh Zipf (1949) (Newman, 2006). Dalam bidang ilmu komputer, studi yang sama juga pernah dilakukan oleh Breslau, et al. (1999) yang menyatakan bahwa permintaan halaman web mengikuti distribusi *Zipf-like* atau *power law*. Persamaan 2.2 adalah persamaan peluang permintaan konten dengan peringkat popularitas ke-i dalam distribusi *Zipf-like*.

$$P\ i\ = \frac{C}{i^{\alpha}} \qquad (2.2)$$

Pada persamaan 2.2, C adalah $\sum_{i=1}^{n} \frac{1}{i^{\alpha}}^{-1}$ dari sejumlah n konten dan α adalah parameter *skewness* dan mengindikasikan derajat konsentrasi permintaan (Laoutaris, Syntila, & Stavrakakis, 2004). Umumnya nilai α berkisar antara 0 dan 1. Nilai α yang semakin mendekati 1 berarti semakin sedikit konten yang menguasai mayoritas permintaan, sedangkan nilai α yang semakin mendekati 0 berarti semakin banyak konten yang memiliki popularitas yang hampir sama. Pada penelitian ini, distribusi popularitas konten diasumsikan mengikuti distribusi *Zipf-like*. Dengan demikian, permintaan suatu konten akan

mengikuti distribusi tersebut. Gambar 2.9 adalah contoh grafik distribusi peluang *Zipf-like* untuk 100 konten yang telah diperingkatkan dari 1 sampai 100. Dari gambar tersebut, grafik dengan nilai α = 0.0 memiliki sebaran peluang yang merata, di mana semua konten memiliki peluang permintaan 0,01. Pada grafik dengan nilai α = 0,8, hanya beberapa konten pada peringkat atas yang memiliki peluang permintaan cukup tinggi, sedangkan mayoritas konten pada peringkat bawah memiliki peluang yang hampir sama.



Gambar 2.9 Grafik distribusi *Zipf-like*

2.2.4 Simulator Icarus

Icarus (Saino, Psaras, & Pavlou, 2014) merupakan sebuah simulator *caching* berbasis Python yang dapat digunakan untuk mengevaluasi strategi *caching* tanpa terikat oleh arsitektur ICN tertentu. Icarus juga menyediakan *tool* untuk pemodelan dalam bidang penelitian *caching*. Icarus dirancang agar memiliki *scalability* dan *extensibility*. *Scalability* berarti bahwa Icarus dapat mendukung simulasi jutaan permintaan konten dan *extensibility* berarti Icarus mudah diadopsi dan dikembangkan lebih lanjut agar dapat mendukung penelitian di bidang caching. Icarus membutuhkan pustaka-pustaka Python pendukung seperti FNSS, Matplotlib, NetworkX, Numpy, dan Scipy agar dapat melakukan simulasi *in-network caching* dengan baik. Icarus juga memerlukan topologi jaringan yang dibuat dalam format GraphML atau GML beserta penentuan lokasi *cache* dan *content source* di topologi yang dibuat.

Proses simulasi *in-network caching* di Icarus terdiri dari empat tahap, yaitu:

1. *Scenario generation*

Pada tahap pertama ini, Icarus akan mengatur topologi jaringan yang siap pakai dan *random event generator* untuk simulasi.

2. *Experiment orchestration*

Di tahap kedua ini, Icarus akan membaca parameter pengujian pada berkas konfigurasi dan menerapkan parameter tersebut pada pengujian.

3. *Experiment execution*

Setelah semua parameter pengujian telah diterapkan, maka simulasi dijalankan pada tahap ini. Pada tahap ini, simulasi dijalankan berdasarkan kombinasi skenario seluruh parameter. Semakin banyak jumlah *item* parameter yang diujikan, maka proses simulasi membutuhkan waktu yang lebih lama.

4. Results collection

Di tahap terakhir ini, hasil pengujian akan dikumpulkan yang nantinya bisa diproses lebih lanjut. Hasil dari proses ini juga digunakan untuk membuat grafik hasil simulasi secara otomatis setelah proses simulasi selesai.

Sampai saat ini, Icarus telah mendukung banyak strategi *content placement*, di antaranya adalah LCE, LCD, Bernoulli *random caching*, *random choice caching*, ProbCache, *centrality-based caching*, dan Hash-routing. Selain itu, *content replacement* yang didukung oleh Icarus antara lain Least Recently Used (LRU), Least Frequently Used (LFU), First In First Out (FIFO), dan Random. Strategi *content placement* dan *content replacement* masih dapat bertambah karena kemudahan dalam implementasi strategi *in-network caching* baru di Icarus.

2.2.5 Metrik Kinerja *In-network Caching*

Agar dapat mengetahui kinerja dari *in-network caching*, maka perlu ditentukan metrik atau kriteria kinerja *in-network caching*. Metrik tersebut antara lain *productivity*, *utilization*, dan *responsiveness*. *Productivity* diukur berdasarkan jumlah permintaan yang berhasil dilayani oleh *cache* atau disebut juga *cache hit ratio*. *Utilization* dapat diukur berdasarkan pemakaian suatu *link* untuk mengirimkan paket-paket atau dapat disebut *link load*. Sedangkan *responsiveness* dapat diukur berdasarkan seberapa cepat suatu *receiver* mendapatkan konten yang diinginkan sejak mengirimkan permintaan atau disebut juga latensi.

2.2.5.1 *Cache Hit Ratio*

Cache Hit Ratio (CHR) adalah perbandingan jumlah permintaan yang berhasil dilayani (*cache hit*) oleh *cache* dengan total permintaan yang dikirimkan. Persamaan 2.3 adalah persamaan untuk menghitung CHR.

$$CHR = \frac{n(hit)}{n(total)} \tag{2.3}$$

Pada persamaan 2.3, $n(hit)$ adalah jumlah permintaan yang berhasil dilayani oleh *cache* (*cache hit*) dan $n(total)$ dalah total permintaan yang dikirimkan. Nilai CHR berkisar antara 0 dan 1, dengan 0 berarti tidak ada permintaan yang berhasil dilayani *cache* dan 1 berarti semua permintaan berhasil dilayani *cache*. Nilai CHR yang semakin mendekati 0 berarti semakin sedikit permintaan yang berhasil dilayani oleh *cache*, sedangkan Nilai CHR yang semakin mendekati 1 berarti semakin banyak permintaan yang berhasil dilayani oleh *cache*.

2.2.5.2 *Latensi*

Latensi dalam simulator Icarus adalah waktu yang dibutuhkan suatu *node* untuk mendapatkan sebuah konten sejak node tersebut mengirimkan permintaan sampai mendapatkan konten. Semakin rendah latensi berarti proses pengiriman konten semakin cepat. Sebaliknya, semakin tinggi latensi berarti semakin lambat proses pengiriman konten.

2.2.5.3 *Link Load*

Link load adalah beban kerja yang dialami suatu *link* karena lalu lintas data yang terjadi di dalamnya. Dalam penelitian ini, *link Load* yang dianalisis adalah yang terdapat dalam keseluruhan topologi. Semakin tinggi *link load* suatu jaringan berarti semakin tinggi lalu lintas data yang melalui jaringan tersebut, yang berarti pula sumber daya yang terpakai tinggi. Sebaliknya, semakin rendah *link load* suatu jaringan berarti semakin rendah lalu lintas data yang melalui jaringan tersebut, yang juga berarti sumber daya yang terpakai rendah.

2.2.6 Graph Modelling Language (GML)

Graph Modelling Language (GML) adalah sebuah format berkas yang digunakan untuk merepresentasikan struktur suatu graf/topologi (Himsolt, 2010). Sebuah berkas GML berisi pernyataan *node* suatu graf dan *links* yang menghubungkan tiap *node*. Keunggulan format GML dibandingkan dengan format berkas graf yang lain adalah *portability*, sintaksis sederhana, fleksibilitas, dan *extensibility*.

Sebagai contoh, sebuah topologi memiliki 2 *node* yang terhubung. Berkas GML untuk topologi tersebut seperti pada Tabel 2.1

Tabel 2.1 Contoh berkas GML untuk topologi 2 *node*

Topologi 2 *node*

```
1  graph [
2      node [
3          id 0
4          label "Node 0"
5      ]
6
7      node [
8          id 1
9          label "Node 1"
10     ]
11
12     edge [
13         source 0
14         target 1
15         id "e0"
16     ]
17
   ]
```

BAB 3 METODOLOGI

Pada bab ini akan ini akan dijelaskan langkah-langkah yang ditempuh dalam melakukan penelitian. Langkah-langkah tersebut antara lain:

1. Studi literatur
2. Analisis kebutuhan simulasi
3. Persiapan simulasi
4. Implementasi dan pengujian
5. Analisis hasil pengujian
6. Penarikan kesimpulan.

3.1 Studi Literatur

Studi literatur dilakukan untuk mengkaji teori-teori yang mendukung penelitian analisis kinerja *in-network caching* pada arsitektur ICN. Teori-teori tersebut meliputi:

1. *Information-centric Networking*

   Untuk mengetahui dasar arsitektur ICN, perbedaanya dengan arsitektur Internet saat ini, dan cara kerja ICN.

2. *In-network Caching*

   Untuk mengetahui teori *in-network caching*, perbedaan antara *on-path caching* dan *off-path caching*, serta strategi-strategi *in-network caching* yang telah diusulkan oleh para peneliti.

3. Popularitas Konten

   Untuk memahami keterkaitan antara popularitas konten dengan distribusi *Zipf-like*.

4. Simulator Icarus

   Untuk mengetahui cara kerja simulator Icarus dan bagaimana simulator tersebut dapat digunakan untuk menguji kinerja *in-network caching*.

5. Metrik kinerja *In-network Caching*

   Untuk mengetahui metrik apa saja yang dapat digunakan untuk mengetahui kinerja *in-network caching*.

6. Graph Modelling Language

   Untuk mengetahui teori Graph Modelling Language (GML) dan cara membuat berkas GML dari suatu topologi.

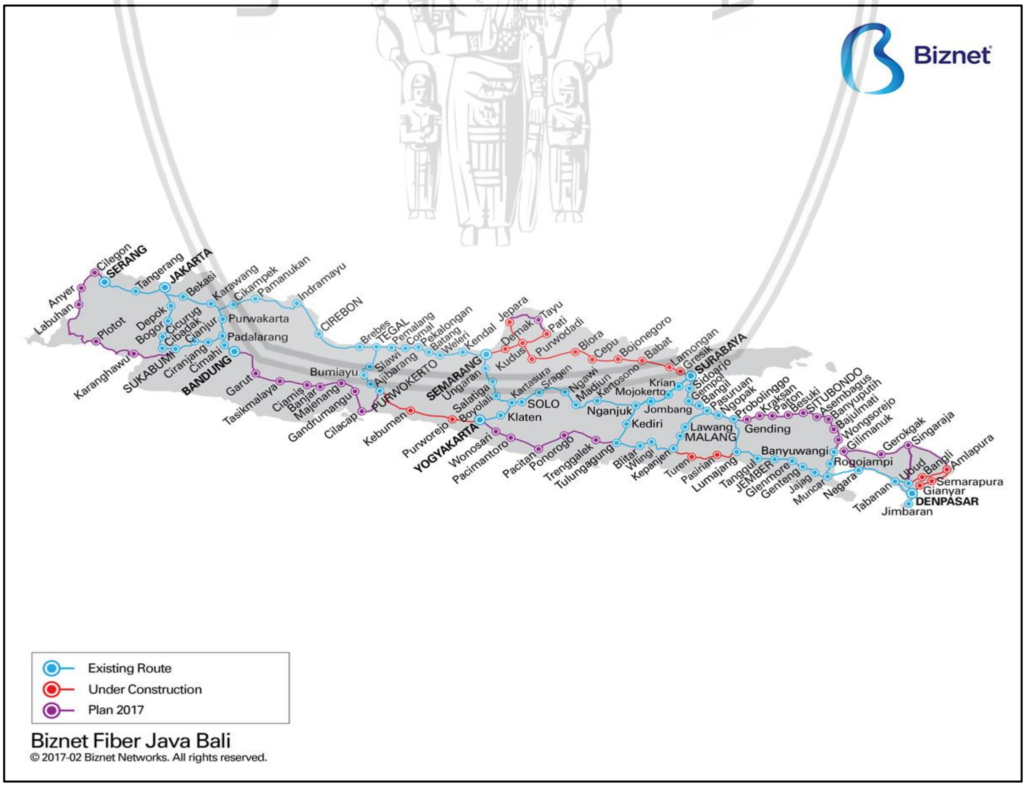
3.2 Analisis Kebutuhan Simulasi

Analisis kebutuhan diperlukan untuk menganalisis apa saja yang dibutuhkan dalam membangun lingkungan pengujian dalam penelitian ini. Dalam simulasi pengujian kinerja *in-network caching*, diperlukan sebuah topologi jaringan agar dapat menjalankan proses *caching*. Topologi tersebut harus memiliki *node* yang berperan sebagai *content source* sebagai penyedia konten, *cache router* untuk menyimpan konten sementara dan meneruskan paket, *receiver* sebagai penerima konten, dan *router* biasa untuk meneruskan paket-paket. Strategi *in-network caching* akan dapat diterapkan pada topologi yang memiliki *node* dengan peran-peran tersebut.

Dari analisis kebutuhan tersebut diperlukan identifikasi terhadap desain topologi, perangkat keras dan perangkat lunak yang digunakan, strategi *in-network caching* yang diuji, serta pengujian dan analisis yang dilakukan.

3.2.1 Desain Topologi Jaringan

Desain topologi jaringan memberikan gambaran tentang lingkungan pengujian yang disimulasikan pada penelitian ini. Topologi yang digunakan pada penelitian ini adalah topologi *existing* jaringan Biznet untuk pulau Jawa dan Bali. Pemilihan topologi jaringan Biznet tersebut agar proses simulasi dapat dijalankan pada topologi riil dan memiliki hasil yang mendekati riil. Gambar 3.1 merupakan gambar topologi jaringan ISP Biznet.



Gambar 3.1 Topologi jaringan ISP Biznet

Sumber: http://biznetnetworks.com

Topologi *existing* Biznet memiliki Topologi tersebut memiliki 76 *node* dengan derajat *node* tertinggi yaitu 3 dan derajat *node* terendah yaitu 1. *Node* tersebut yaitu Serang, Tangerang, Jakarta, Depok, Bogor, Cicurug, Cibadak, Sukabumi, Cianjur, Ciranjang, Cimahi, Padalarang, Bandung, Purwakarta, Karawang, Bekasi, Cikampek, Pamanukan, Indramayu, Cirebon, Brebes, Tegal, Slawi, Bumiayu, Ajibarang, Purwokerto, Pemalang, Comal, Pekalongan, Batang, Weleri, Kendal, Semarang, Ungaran, Salatiga, Boyolali, Kartasura, Klaten, Yogyakarta, Solo, Sragen, Ngawi, Madiun, Kertosono, Nganjuk, Kediri, Tulungagung, Blitas, Wlingi, Kepanjen, Malang, Lawang, Pasuruan, Bangil, Gempol, Sidoarjo, Surabaya, Krian, Mojokerto, Jombang, Ngopak, Probolinggo, Lumajang, Tanggul, Jember, Glenmore, Genteng, Jajag, Muncar, Rogojampi, Banyuwangi, Negara, Tabanan, Denpasar, Ubud, dan Jimbaran.

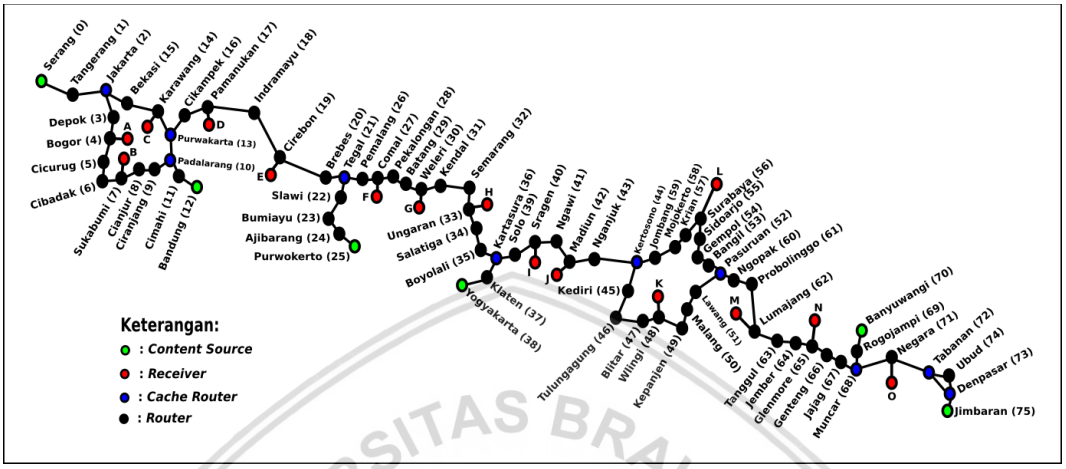
Topologi tersebut harus dibuat dalam sebuah berkas berformat GML dan lalu dipilih beberapa *node* yang berfungsi sebagai *content source*, *receiver*, *cache router*, dan *router* biasa agar dapat diproses oleh simulator Icarus. Pemilihan node tersebut didasarkan pada karakteristik graf topologi jaringan Biznet. Untuk *content source*, dipilih *node* berderajat 1 dengan asumsi bahwa sebagai penyedia utama konten, *content source* cukup menerima permintaan konten satu kali yang nantinya konten tersebut dapat disebar di dalam jaringan melalui *cache*. *Node* dengan derajat 1 pada topologi Biznet adalah *node* Serang, Bandung, Purwokerto, Yogyakarta, Banyuwangi, dan Jimbaran.

Pemilihan *node* sebagai *cache router,* yang bertugas untuk menyimpan konten sekaligus meneruskan paket yang diterimanya, didasarkan pada karakteristik *node* yang memiliki derajat paling tinggi, yaitu 3. Dengan derajat node yang tinggi, *cache router* diasumsikan dapat menerima dan meneruskan paket permintaan dan konten dari sumber yang lebih banyak. *Node* tersebut antara lain *node* Jakarta, Cimahi, Purwakarta, Tegal, Kartasura, Kertosono, Pasuruan, Muncar, Denpasar, dan Ubud.

Selain dari ke-76 *node* pada topologi *existing* Biznet, ditambahkan pula *node* yang berfungsi sebagai *receiver*. *Receiver* bertugas untuk membuat permintaan dan menerima konten. *Receiver* tersebut dihubungkankan dengan suatu *node* topologi Biznet berdasarkan jarak ke *content source* atau *cache router*. Jarak tersebut berkisar antara 1 sampai 5 *node* dari *content source* atau *cache router*. *Receiver* tersebut antara lain *receiver* A yang terhubung dengan *node* Bogor, *receiver* B yang terhubung dengan *node* Sukabumi, *receiver* C yang terhubung dengan *node* Karawang, *receiver* D yang terhubung dengan *node* Pamanukan, *receiver* E yang terhubung dengan *node* Cirebon, *receiver* F yang terhubung dengan *node* Comal, *receiver* G yang terhubung dengan *node* Weleri, *receiver* H yang terhubung dengan *node* Ungaran, *receiver* I yang terhubung dengan *node* Sragen, receiver J yang terhubung dengan *node* Madiun, *receiver* K yang terhubung dengan *node* Wlingi, *receiver* L yang terhubung dengan *node* Surabaya, *receiver* M yang terhubung dengan *node* Lumajang, *receiver* N yang terhubung dengan *node* Glenmore, dan *receiver* O yang terhubung dengan *node*

Negara. Meskipun penambahan *node receiver* menyebabkan bertambahnya derajat beberapa *node*, *cache router* tetap pada *node* yang telah dipilih.

Node yang tidak bertugas sebagai *content source*, *cache router*, maupun *receiver* akan bertindak sebagai *router* biasa untuk meneruskan paket ke *node* selanjutnya. Gambar 3.2 adalah desain topologi pengujian pada Icarus.



Gambar 3.2 Topologi pengujian

3.2.2 Strategi *In-network Caching* yang Diujikan

Strategi *in-network caching* yang diujikan pada penelitian ini terdiri dari 3 strategi *caching* yang bersifat *on-path* dan 3 strategi *caching* bersifat *off-path*. Strategi *on-path caching* tersebut antara lain Leave Copy Everywhere, ProbCache, dan Random. Strategi *off-path caching* yang diuji antara lain Hash-routing Symmetry, Hash-routing Asymmetry, dan Hash-routing Multicast. Sedangkan strategi *content replacement* yang diterapkan adalah *Least-recently Used* (LRU).

3.2.3 Perangkat Lunak yang Digunakan

Perangkat lunak utama yang digunakan dalam penelitian ini adalah simulator Icarus versi 0.6.0. Simulator Icarus membutuhkan bahasa pemrograman Python versi 2.7 dan beberapa pustaka pendukung seperti FNSS, NetworkX, Numpy, Scipy, dan Matplotlib agar dapat berjalan dengan baik. Seluruh perangkat lunak tersebut berjalan di atas sistem operasi Ubuntu 16.04.02 LTS 64 bit.

3.2.4 Perangkat Keras yang Digunakan

Agar perangkat lunak yang digunakan untuk pengujian dapat berjalan dengan baik, pengujian dilakukan pada sebuah laptop yang memiliki spesifikasi CPU AMD A8 Quadcore 2,4 GHz, RAM 4 GB, dan *hard drive* 500 GB.

3.3 Persiapan Simulasi

Sebelum dilakukan proses simulasi, proses konfigurasi simulator Icarus diperlukan untuk membuat berkas konfigurasi Icarus yang di dalamnya memuat parameter-parameter simulasi. Parameter simulasi yang dikonfigurasi antara lain

topologi yang digunakan, jumlah konten, jumlah permintaan, nilai *α* untuk popularitas konten, kapasitas *cache*, serta strategi *in-network caching* yang diuji. Selain itu, perlu dikonfigurasi pula *data collectors* yang akan mengumpulkan hasil simulasi.

Agar lingkungan simulasi mirip dengan kondisi riil Internet, jumlah konten yang digunakan di simulasi adalah 252255 konten, diambil dari jumlah video YouTube kategori *Science* hasil studi Cha, et al. (2007). Untuk jumlah permintaan konten adalah 2 kali jumlah konten, yaitu 504510 permintaan, dengan asumsi bahwa tiap konten akan diminta sebanyak 2 kali. Selain itu, simulator Icarus juga membutuhkan jumlah *warmup request* agar *cache* berisi konten sebelum dilakukan simulasi sebenarnya. Jumlah *warmup request* adalah sama dengan jumlah konten, dengan asumsi tiap konten akan diminta sebanyak paling tidak 1 kali. Selain itu, simulator Icarus sudah menentukan besarnya paket permintaan dan paket konten, yaitu 150 Bita untuk paket permintaan dan 1500 Bita untuk paket konten. Konten yang disimulasikan adalah bilangan bulat sejumlah banyaknya konten, yang dalam penelitian ini dimulai dari 1 sampai 252255.

Popularitas konten didasarkan pada distribusi *Zipf-like*. Menurut Rossi dan Rossini (2011), hingga saat ini belum ada konsensus dalam memodelkan popularitas konten, terutama mengenai nilai α yang tepat untuk memodelkannya. Dalam penelitian ini, nilai α yang digunakan adalah 0,7, 0,8, dan 0,9, dengan 0,8 sebagai basis berdasarkan pada hasil studi Breslau, et al. (1999). Kapasitas *cache* pada simulator Icarus adalah persentase konten yang dapat disimpan dalam *cache* dari jumlah keseluruhan konten. Kapasitas *cache* yang diujikan adalah 1%, 5%, dan 10% dari jumlah keseluruhan konten. Kapasitas tersebut dibagi rata di semua *cache* dalam topologi. *Request rate* disesuaikan dengan jumlah *receiver* dalam topologi, yaitu 15 permintaan per detik. Tabel 3.1 adalah tabel rangkuman parameter pengujian beserta nilainya.

Tabel 3.1 Parameter Pengujian

Parameter	Nilai
Topologi	Biznet
Strategi yang diuji	LCE, Hash-routing Symmetry, Hash-routing Asymmetry, Hash-routing Multicast, Random, ProbCache
Content replacement	LRU
Nilai α	0,7, 0,8, 0,9
Kapasitas *Cache*	1%, 5%, dan 10% dari jumlah keseluruhan konten
Kolektor Data	*Cache Hit Ratio*, Latensi, *Link Load*
Jumlah Konten	252.255 Konten
Request Rate	15 permintaan per detik

Tabel 3.1 Parameter Pengujian (lanjutan)

Parameter	Nilai
Jumlah *Warm-up Request*	252.255 Permintaan
Jumlah *Measured Request*	504.510 Permintaan

3.4 Implementasi dan Pengujian

Pada tahapan ini dilakukan simulasi kinerja *in-network caching* berdasarkan persiapan yang telah dibuat dengan mengeksekusi Icarus melalui aplikasi terminal. Pada tahap *scenario generation*, Icarus akan membaca rancangan topologi Biznet beserta penempatan *node* yang berfungsi sebagai *content source*, *receiver*, *cache router*, dan *router* biasa yang telah dibuat lalu mengatur *random event generator*. Selanjutnya, pada tahap *experiment orchestration*, Icarus akan membaca parameter-parameter dari berkas konfigurasi yang telah dibuat. Konten akan dibagi secara merata sejumlah banyaknya *content source* dan diletakkan secara acak pada seluruh *content source*. Seluruh *node* yang bertugas sebagai *cache router* akan diberi ruang penyimpanan berdasarkan kapasitas *cache* pada berkas konfigurasi. Sebagai contoh, dengan kapasitas *cache* sebesar 1% dan jumlah konten sebanyak 252.255, maka jumlah konten yang dapat disimpan di *cache* seluruh jaringan adalah 2.522,55. Jumlah tersebut dibagi secara merata ke seluruh *cache router* (yang berjumlah 10), sehingga tiap *cache router* dapat menyimpan 252 konten.

Pada tahap *experiment execution*, *receiver* akan meminta konten sesuai dengan distribusi *Zipf-like* berdasarkan parameter nilai α. Permintaan konten tersebut akan dikirimkan ke *content source* langsung atau ke *cache router*, tergantung strategi yang sedang disimulasikan. Simulasi akan berjalan sebanyak jumlah kombinasi dari strategi yang diuji, *content placement*, nilai α, dan kapasitas *cache*. Karena jumlah strategi yang diuji adalah 6 strategi, parameter *content placment* berjumlah 1, nilai α berjumlah 3, dan kapasitas *cache* berjjumlah 3, maka total simulasi yang dilakukan adalah 54 simulasi dalam satu kali eksekusi Icarus. Sebagai contoh, strategi LCE akan disimulasikan sebanyak 9 kali. Kombinasi simulasi tersebut antara lain:

1. LCE dengan α = 0,7 dan kapasitas *cache* 1%
2. LCE dengan α = 0,7 dan kapasitas *cache* 5%
3. LCE dengan α = 0,7 dan kapasitas *cache* 10%
4. LCE dengan α = 0,8 dan kapasitas *cache* 1%
5. LCE dengan α = 0,8 dan kapasitas *cache* 5%
6. LCE dengan α = 0,8 dan kapasitas *cache* 10%
7. LCE dengan α = 0,9 dan kapasitas *cache* 1%
8. LCE dengan α = 0,9 dan kapasitas *cache* 5%

9. LCE dengan α = 0,9 dan kapasitas *cache* 10%.

Pada tahap terakhir, *results collection*, hasil simulasi akan dikumpulkan dalam sebuah berkas berformat Pickle. Hasil simulasi tersebut berisi informasi kinerja dari tiap strategi yang diujikan.

3.5 Analisis Hasil Pengujian

Dari simulasi yang sudah dilakukan, dilakukan analisis terhadap kinerja *in-network caching*. Analisis dilakukan dengan membandingkan kinerja strategi *in-network caching* berdasarkan aspek *cache hit ratio*, latensi, dan *link load*. Proses analisis dilakukan secara bertahap berdasarkan nilai α dan kapasitas *cache*, sehingga akan diketahui pula pengaruh dari nilai α dan kapasitas *cache* terhadap kinerja *in-network caching*.

3.6 Penarikan Kesimpulan

Penarikan kesimpulan dilakukan setelah hasil pengujian dianalisis. Kesimpulan disusun berdasarkan hasil pembahasan mengenai kinerja *In-network caching*. Kesimpulan yang dihasilkan diharapkan mampu menjawab rumusan masalah penelitian ini dan dapat menjadi acuan bagi penelitian selanjutnya yang terkait.

BAB 4 IMPLEMENTASI DAN PENGUJIAN

Pada bab ini akan dijelaskan proses pembuatan topologi, konfigurasi simulasi, pengujian pada simulator Icarus, dan pemrosesan hasil simulasi. Sebelum proses simulasi dimulai, perangkat lunak yang digunakan dalam penelitian harus dipastikan sudah terinstal pada komputer atau laptop. Perangkat lunak tersebut antara lain bahasa pemrograman Python versi 2.7 atau yang lebih baru, Simulator Icarus, dan pustaka pendukung simulator Icarus (FNSS, NetworkX, Numpy, Scipy, dan Matplotlib).

Proses instalasi simulator Icarus cukup dengan mengunduh berkas simulator Icarus versi 0.6.0 dari situs https://icarus-sim.github.io. Berkas yang sudah terunduh lalu diekstrak dan diletakkan pada direktori yang diinginkan. Penambahan *path* direktori Icarus pada variabel *environment* PYTHONPATH juga diperlukan agar memudahkan proses simulasi. Gambar 4.1 adalah perintah untuk menambahkan direktori Icarus ke dalam variabel *environment* PYTHONPATH.

```
$ export PYTHONPATH=<DIREKTORI ICARUS>:$PYTHONPATH
```

Gambar 4.1 Perintah penambahan variabel pada PYTHONPATH

4.1 Pembuatan Topologi Simulasi

4.1.1 Pembuatan Berkas GML

Proses selanjutnya adalah pembuatan berkas GML dari topologi pengujian. Berkas GML tersebut berisi seluruh *node* beserta *link*/*edge* yang menghubungkan tiap *node*. Dalam berkas GML, sebuah *node* paling tidak harus memiliki 2 informasi, yaitu nomor identitas dan label *node*. Sedangkan *link* membutuhkan 3 informasi, yaitu *node* awal, *node* target, dan nomor/kode identitas dari *link* tersebut. Tiap *node* dari topologi Biznet diberi nomor identitas mulai dari 0 sampai 75 yang dimulai dari *node* Serang sampai *node* Jimbaran, seperti pada Gambar 3.2. *Link* yang menghubungkan tiap node tersebut diberi label mulai dari "e0" sampai "e77". Tabel 4.1 adalah potongan kode sumber berkas GML dari topologi yang akan disimulasikan sesuai dengan desain topologi pada Gambar 3.2. Berkas GML topologi yang telah dibuat lalu disimpan pada direktori ~/Icarus-0.6.0/resources/topologies dengan nama berkas Biznet.gml.

Tabel 4.1 Potongan kode GML topologi Biznet

Biznet.gml

```
1   graph [
2     Network "Biznet Jawa - Bali"
3     label "Biznet"
4     node [
5       id 0
6       label "Serang"
7     ]
8     node [
9       id 1
10      label "Tangerang"
11    ]
```

Tabel 4.1 Potongan kode GML topologi Biznet (lanjutan)

Biznet.gml

```
13   node [
14       id 2
15       label "Jakarta"
16     ]

...    ...

313    edge [
314      source 0
315      target 1
316      id "e0"
317    ]
318    edge [
319      source 1
320      target 2
321      id "e1"
322    ]

...  ...
```

Setelah proses pembuatan berkas GML, langkah selanjutnya adalah memasukkan topologi tersebut ke dalam simulator Icarus dengan mengedit berkas topology.py yang ada di direktori ~/icarus-0.6.0/icarus/scenarios. Pengeditan ini bertujuan agar topologi Biznet dapat dipakai pada saat simulasi.

Langkah awal pengeditan berkas topology.py adalah dengan memasukkan sebuah *value* ke variabel *namespace* _ _all_ _. Variabel *namespace* _ _all_ _ pada berkas topology.py berisi nama-nama fungsi yang digunakan untuk mendesain topologi. *Value* yang akan dimasukkan tersebut bersifat arbitrer. Pada penelitian ini, *value* yang dimasukkan pada variabel _ _all_ _ adalah topology_biznet. Langkah selanjutnya adalah membuat sebuah fungsi baru pada berkas topology.py untuk memasukkan berkas GML topologi Biznet dan mendesain lebih lanjut topologi tersebut. Nama fungsi tersebut harus sesuai dengan nama fungsi yang telah dimasukkan pada variabel namespace _ _all_ _. Fungsi tersebut memilih node mana saja yang berfungsi sebagai *content source*, *cache router*, *receiver*, dan *router* biasa seperti yang telah didesain. Selain itu, fungsi tersebut juga menentukan tipe *link* (internal atau eksternal), besarnya *internal link delay* dan *external link delay*, dan bobot *link*. Besarnya *delay* untuk *internal link* dan *external link* tersebut sudah ditentukan di dalam Icarus seperti yang disarankan oleh Zhang, et al. (2006) dan Rajahalme, et al. (2010). Tabel 4.3 adalah potongan kode sumber dari topology.py yang telah disisipi topologi yang akan disimulasikan sesuai dengan desain topologi pada Gambar 3.2.

Tabel 4.2 Potongan kode sumber topology.py

topology.py

```
...   ...

24    __all__ = [
25        'IcnTopology',
26        'topology_tree',
27        'topology_path',
28        'topology_ring',
```

Tabel 4.3 Potongan kode sumber topology.py (lanjutan)

topology.py

```
29      'topology_mesh',
30      'topology_geant',
31      'topology_tiscali',
32      'topology_wide',
33      'topology_garr',
34      'topology_rocketfuel_latency',
35      'topology_biznet'
36  ]

...     ...

990 @register_topology_factory('BIZNET')
991 def topology_biznet(**kwargs):
992     topology =
    fnss.parse_topology_zoo(path.join(TOPOLOGY_RESOURCES_DIR,
    'Biznet.gml')).to_undirected()
993     sources = ["Serang", "Bandung", "Purwokerto", "Yogyakarta",
    "Banyuwangi", "Jimbaran"]
994     icr_candidates = ["Jakarta", "Padalarang", "Purwakarta",
    "Tegal", "Kartasura", "Kertosono", "Pasuruan", "Muncar",
    "Tabanan", "Denpasar"]
995     receivers = [“Receiver-A”, “Receiver-B”, “Receiver-C”,
    “Receiver-D”, “Receiver-E”, “Receiver-F”, “Receiver-G”,
    “Receiver-H”, “Receiver-I”, “Receiver-J”, “Receiver-K”,
996 “Receiver-L”, “Receiver-M”, “Receiver-N”, “Receiver-O”]
    routers = [n for n in topology.nodes() if n not in
997 sources+receivers]
998     fnss.set_weights_constant(topology, 1.0)
    fnss.set_delays_constant(topology, INTERNAL_LINK_DELAY,
999 'ms')
1000    topology.graph['icr_candidates'] = set(icr_candidates)
1001    for v in sources:
1002        fnss.add_stack(topology, v, 'source')
1003    for v in receivers:
1004        fnss.add_stack(topology, v, 'receiver')
1005    for v in routers:
1006        fnss.add_stack(topology, v, 'router')
1007    for u, v in topology.edges():
1008        if u in sources or v in sources:
1009            topology.edge[u][v]['type'] = 'external'
                fnss.set_weights_constant(topology, 1000.0, [(u,
1010 v)])
                fnss.set_delays_constant(topology,
1011 EXTERNAL_LINK_DELAY, 'ms', [(u, v)])
1012        else:
1013            topology.edge[u][v]['type'] = 'internal'
    return IcnTopology(topology)
```

4.1.2 Konfigurasi Simulasi

Konfigurasi simulasi pada Icarus dilakukan dengan mengedit berkas config.py yang berada di dalam direktori ~/icarus-0.6.0. Bagian yang perlu diedit pada berkas tersebut agar sesuai dengan tujuan penelitian ini meliputi STRATEGIES, CACHE_POLICY, TOPOLOGIES, ALPHA, DATA_COLLECTORS, dan NETWORK_CACHE. Selain itu, konfigurasi jumlah konten dan jumlah permintaannya dilakukan pada bagian NETWORK_REQUEST_RATE, N_MEASURED_REQUEST, N_WARMUP_REQUESTS, dan N_CONTENTS. N_WARMUP_REQUESTS merupakan jumlah permintaan konten pada proses

warmup untuk mengisi *cache router* sebelum dilakukan simulasi yang sebenarnya, sedangkan N_MEASURED_REQUEST merupakan jumlah permintaan yang sebenarnya dari simulasi yang dilakukan. Jumlah N_WARMUP_REQUESTS adalah sama dengan jumlah N_CONTENTS dengan asumsi bahwa tiap konten mendapat paling sedikit satu permintaan pada proses *warmup*. Tabel 4.3 adalah potongan kode sumber berkas config.py yang telah diedit sesuai dengan parameter pengujian pada Tabel 3.1.

Tabel 4.3 Potongan kode sumber config.py

config.py

```
...  ...

11   DATA_COLLECTORS = ['CACHE_HIT_RATIO', 'LATENCY', 'LINK_LOAD']

12   ALPHA = [0.7, 0.8, 0.9]

13   NETWORK_CACHE = [0.01, 0.05, 0.1]

14   N_CONTENTS = 252255

15   NETWORK_REQUEST_RATE = 15

16   N_WARMUP_REQUESTS = 252255

17   N_MEASURED_REQUESTS = 504510

18   TOPOLOGIES =  ['BIZNET']

19   STRATEGIES = [
          'LCE',
          'HR_SYMM',
          'HR_ASYMM',
          'HR_MULTICAST',
          'RAND_CHOICE',
          'PROB_CACHE',
           ]

20   CACHE_POLICY = 'LRU'

...  ...
```

4.2 Pengujian

Metode pengujian dilakukan dengan mengeksekusi berkas konfigurasi config.py yang telah diedit. Simulator Icarus akan memerika berkas konfigurasi tersebut lalu menjalankan simulasi sesuai dengan parameter-parameter yang ada di dalamnya. Simulasi lalu berjalan berdasarkan kombinasi parameter-parameter konfigurasi. Setelah selesai, hasil simulasi akan dikumpulkan menjadi satu dalam sebuah berkas berformat Pickle. Gambar 4.2 adalah perintah untuk menjalankan simulasi pada Icarus.

```
$ python <DIREKTORI ICARUS>/icarus.py --results <NAMA BERKAS HASIL
SIMULASI> config.py
```

Gambar 4.2 Perintah untuk mengeksekusi Icarus

Hasil simulasi yang berformat Pickle masih susah untuk dipahami, sehingga perlu dikonversi ke format TXT yang lebih mudah dipahami. Konversi ke format TXT dilakukan dengan cara mengeksekusi berkas printresults.py yang ada di dalam direktori Icarus dan memberikan nama berkas beformat Pickle beserta nama berkas baru beformat TXT sebagai argumen perintah. Gambar 4.3 adalah Perintah untuk mengkonversi hasil simulasi yang berformat Pickle ke format TXT.

```
$ python <DIREKTORI ICARUS>/scripts/printresults.py <NAMA BERKAS HASIL
SIMULASI> > <NAMA BERKAS TXT>
```

Gambar 4.3 Perintah untuk konversi hasil simulasi ke format TXT

Sebagai tambahan, simulator Icarus juga dapat melakukan *plotting* hasil simulasi ke dalam grafik. Grafik hasil *plotting* tersebut cukup membantu proses analisis hasil simulasi. Gambar 4.4 adalah perintah untuk melakukan *plotting* hasil simulasi.

```
$ python <DIREKTORI ICARUS>/plotresults.py --results <NAMA BERKAS HASIL
SIMULASI> --output <DIREKTORI PLOT> config.py
```

Gambar 4.4 Perintah untuk melakukan *plotting* hasil simulasi

Pada perintah-perintah di atas, <NAMA BERKAS HASIL SIMULASI> disesuaikan dengan nama berkas untuk menyimpan hasil simulasi yang dapat berformat Pickle. Sedangkan <DIREKTORI PLOT> dapat disesuaikan dengan nama direktori tempat grafik hasil *plotting* disimpan.

BAB 5 ANALISIS DAN PEMBAHASAN

Pada bab ini dijelaskan analisis dan pembahasan hasil dari pengujian kinerja *in-network caching* yang telah dilakukan pada Bab 4. Analisis dilakukan untuk mengetahui perbandingan kinerja *in-network caching*.

5.1 Hasil Pengujian

5.1.1 *Cache Hit Ratio* (CHR)

Tabel 5.1 adalah tabel hasil pengujian CHR strategi *on-path* caching, sedangkan Tabel 5.2 tabel hasil pengujian CHR strategi *off-path* caching. Hasil pengujian pada kedua tabel dikelompokkan berdasarkan parameter nilai α dan kapasitas *cache*. Nilai CHR pada tabel tersebut merupakan rerata perbandingan antara jumlah permintaan yang berhasil dilayani oleh *cache* dengan total permintaan yang dikirimkan. Semakin tinggi nilai CHR, maka jumlah permintaan yang berhasil dilayani oleh *cache* juga semakin tinggi.

Tabel 5.1 Hasil pengujian CHR *on-path caching*

Strategi	***Cache Hit Ratio***								
	α = 0,7			α = 0,8			α = 0,9		
	Cache 1%	*Cache* 5%	*Cache* 10%	*Cache* 1%	*Cache* 5%	*Cache* 10%	*Cache* 1%	*Cache* 5%	*Cache* 10%
LCE	0,061	0,145	0,206	0,134	0,252	0,323	0,262	0,404	0,475
Random	0,096	0,207	0,284	0,185	0,323	0,401	0,322	0,470	0,542
ProbCache	0,115	0,221	0,281	0,215	0,341	0,398	0,355	0,488	0,537
Rerata	**0,091**	**0,191**	**0,257**	**0,178**	**0,305**	**0,374**	**0,313**	**0,454**	**0,518**

Tabel 5.2 Hasil pengujian CHR *off-path caching*

Strategi	***Cache Hit Ratio***								
	α = 0,7			α = 0,8			α = 0,9		
	Cache 1%	*Cache* 5%	*Cache* 10%	*Cache* 1%	*Cache* 5%	*Cache* 10%	*Cache* 1%	*Cache* 5%	*Cache* 10%
HR Symmetry	0,115	0,254	0,352	0,217	0,38	0,478	0,365	0,534	0,620
HR Asymmetry	0,104	0,228	0,306	0,185	0,333	0,400	0,324	0,457	0,517
HR Multicast	0,114	0,253	0,351	0,218	0,380	0,479	0,367	0,532	0,619
Rerata	**0,111**	**0,245**	**0,336**	**0,207**	**0,364**	**0,452**	**0,352**	**0,508**	**0,585**

5.1.2 *Link Load*

Link load menunjukan seberapa tinggi beban yang ada di tiap *link* dalam suatu jaringan. Semakin tinggi *link load* berarti semakin tinggi pula sumber daya yang dipakai oleh jaringan, sedangkan semakin rendah *link load* berarti semakin rendah sumber daya yang dipakai oleh jaringan. Tabel 5.3 berisi hasil pengujian *link load* pada *on-path caching*, sedangkan Tabel 5.4 berisi hasil pengujian *link load* pada *off-path caching*. Hasil pengujian pada kedua tabel tersebut dikelompokkan berdasarkan parameter nilai α dan kapasitas *cache* yang telah diujikan. Nilai yang ada pada tabel hasil pengujian *link load* merupakan rerata *link load* yang terdapat pada keseluruhan topologi dalam satuan megabita per detik.

Tabel 5.3 Hasil pengujian *link load on-path caching*

Strategi	***Link load* (MBps)**								
	α = 0,7			α = 0,8			α = 0,9		
	Cache 1%	*Cache* 5%	*Cache* 10%	*Cache* 1%	*Cache* 5%	*Cache* 10%	*Cache* 1%	*Cache* 5%	*Cache* 10%
LCE	3046	2925	2837	2908	2712	2586	2681	2375	2230
Random	3028	2842	2728	2840	2594	2450	2548	2239	2086
ProbCache	2996	2826	2721	2835	2580	2454	2525	2219	2097
Rerata	**3023,3**	**2864,3**	**2762,0**	**2861,0**	**2628,7**	**2496,7**	**2584,7**	**2277,7**	**2137,7**

Tabel 5.4 Hasil pengujian *link load off-path caching*

Strategi	***Link load* (MBps)**								
	α = 0,7			α = 0,8			α = 0,9		
	Cache 1%	*Cache* 5%	*Cache* 10%	*Cache* 1%	*Cache* 5%	*Cache* 10%	*Cache* 1%	*Cache* 5%	*Cache* 10%
HR Symmetry	5721	5277	4971	5404	4879	4578	4923	4399	4124
HR Asymmetry	3338	3249	3195	3282	3172	3116	3180	3074	3034
HR Multicast	4565	4308	4149	4391	4081	3890	4098	3793	3633
Rerata	**4541,3**	**4278,0**	**4105,0**	**4359,0**	**4044,0**	**3861,3**	**4067,0**	**3755,3**	**3597,0**

5.1.3 Latensi

Latensi menunjukan waktu yang dibutuhkan *receiver* untuk menerima konten yang diminta, sejak dikirimnya paket permintaan ke *cache* atau *content source*. Latensi yang rendah berarti *receiver* menerima konten dengan segera. Sebaliknya, semakin tinggi latensi berarti semakin lambat proses pengiriman konten. Tabel 5.5 merupakan tabel hasil pengujian latensi pada *on-path caching*, sedangkan Tabel 5.6 merupakan tabel hasil pengujian latensi pada *off-path caching*. Hasil pengujian pada kedua tabel tersebut dikelompokkan berdasarkan parameter nilai α dan kapasitas *cache* yang telah diujikan. Nilai yang ada pada kedua tabel hasil pengujian latensi merupakan rerata latensi dari tiap permintaan konten dalam satuan milisekon.

Tabel 5.5 Hasil pengujian latensi *on-path caching*

Strategi	**Latensi (ms)**								
	α = 0,7			α = 0,8			α = 0,9		
	Cache 1%	*Cache* 5%	*Cache* 10%	*Cache* 1%	*Cache* 5%	*Cache* 10%	*Cache* 1%	*Cache* 5%	*Cache* 10%
LCE	147,7	138,5	131,9	138,8	125,5	117,1	123,6	105,7	96,9
Random	144,3	132,0	123,6	133,3	117,3	108,2	116,1	97,5	88,3
ProbCache	142,5	130,6	123,6	131,2	115,6	108,2	113,1	95,7	89,0
Rerata	**144,8**	**133,7**	**126,4**	**134,4**	**119,5**	**111,2**	**117,6**	**99,6**	**91,4**

Tabel 5.6 Hasil pengujian latensi *off-path caching*

Strategi	**Latensi (ms)**								
	α = 0,7			α = 0,8			α = 0,9		
	Cache 1%	*Cache* 5%	*Cache* 10%	*Cache* 1%	*Cache* 5%	*Cache* 10%	*Cache* 1%	*Cache* 5%	*Cache* 10%
HR Symmetry	217,6	196,0	180,7	201,6	176,2	161,2	178,4	152,4	139,0
HR Asymmetry	183,4	170,0	161,4	173,9	157,4	149,3	157,9	142,2	136,2
HR Multicast	181,4	165,6	154,7	169,8	151,1	139,7	152,3	133,5	123,6
Rerata	**194,1**	**177,2**	**165,6**	**181,8**	**161,6**	**150,1**	**162,9**	**142,7**	**132,9**

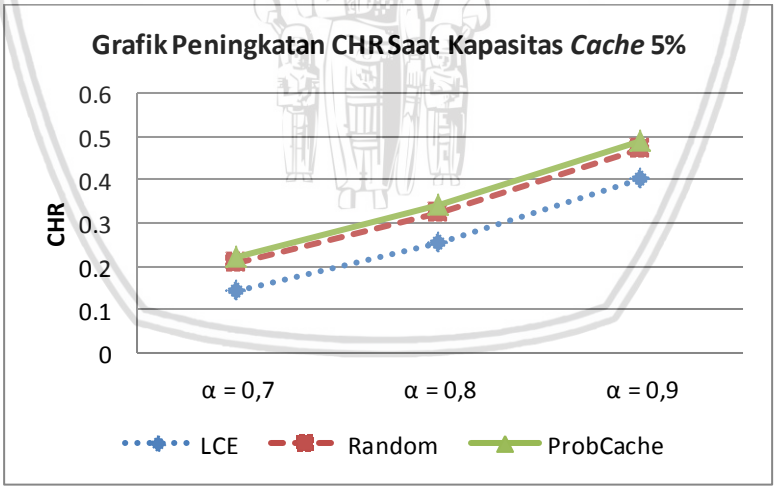
5.2 Analisis Hasil

Pada bagian ini, proses analisis kinerja *on-path caching* dan *off-path caching* dilakukan berdasarkan hasil pengujian pada Subbab 5.1. Analisis dilakukan terhadap metrik kinerja *in-network caching* tiap strategi, yaitu *cache hit ratio*, *link load*, dan latensi.

5.2.1 Nilai α

Analisis nilai α bertujuan untuk mencari tahu hubungan antara nilai α dengan kinerja *in-network caching* pada penelitian ini dengan mencari nilai α yang memiliki pengaruh paling baik terhadap kinerja *in-network caching*. Nilai α terbaik tersebut akan digunakan sebagai acuan untuk membandingkan kinerja *on-path caching* dan *off-path caching*.

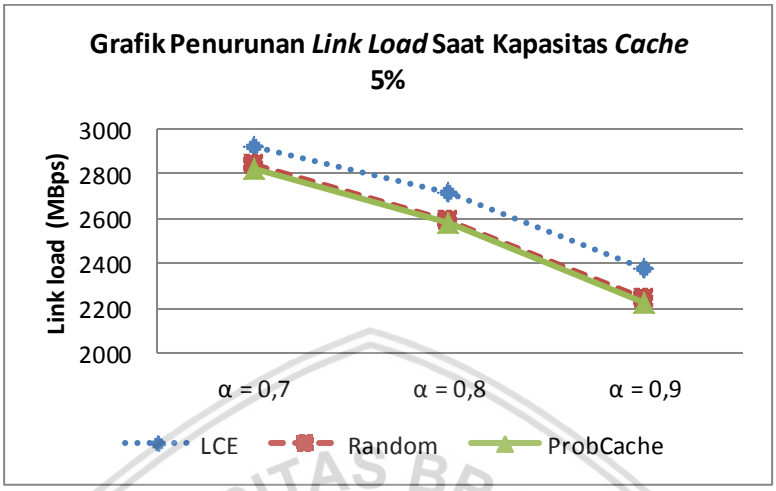
Berdasarkan hasil pengujian pada Tabel 5.1 dan Tabel 5.2, CHR seluruh strategi *on-path caching* dan *off-path caching* mengalami peningkatan seiring dengan meningkatnya nilai α. Gambar 5.1 menggambarkan pengingkatan CHR yang terjadi pada strategi berbasis *on-path caching* saat kapasitas *cache* 5%. CHR pada saat nilai α = 0,7 lebih rendah dibandingkan dengan CHR saat nilai α = 0,8, dan CHR saat nilai α = 0,8 lebih rendah dibandingkan dengan CHR saat nilai α = 0,9. Hal yang sama juga terjadi pada *off-path caching,* di mana CHR saat nilai α = 0,9 berada pada titik tertinggi sedangkan saat nilai α = 0,7 berada pada titik terendah. Dengan demikian, dapat disimpulkan bahwa nilai α = 0,9 memiliki pengaruh paling baik terhadap CHR.



Gambar 5.1 Grafik peningkatan CHR

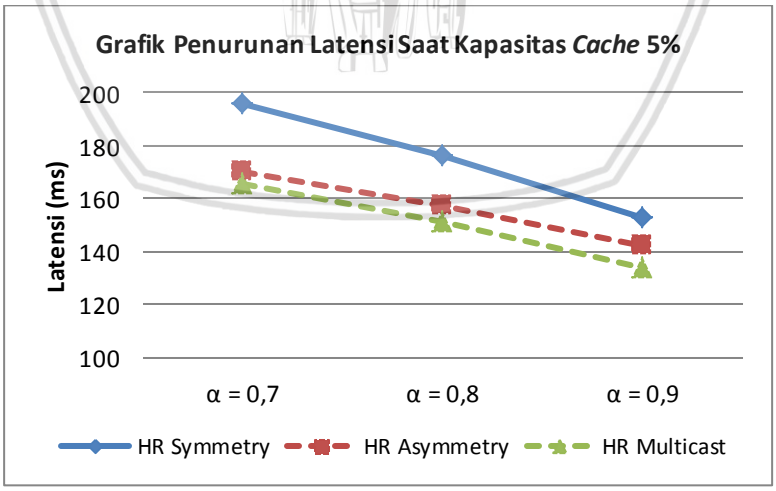
Pada hasil pengujian kinerja *link load*, meningkatnya nilai α diikuti oleh menurunnya *link load* pada seluruh strategi *on-path caching* dan *off-path caching* yang diujikan. *Link load* pada *on-path caching* dan *off-path caching* saat nilai α = 0,7 lebih tinggi dibandingkan dengan *link load* saat nilai α = 0,8, dan *link load* saat nilai α = 0,8 lebih tinggi dibandingkan dengan *link load* saat nilai α = 0,9.

Gambar 5.2 menunjukkan penurunan *link load* yang terjadi pada strategi berbasis *on-path caching* saat kapasitas *cache* 5%. Dari analisis tersebut, dapat disimpulkan bahwa nilai α = 0,9 memiliki pengaruh paling baik terhadap *link load*.



Gambar 5.2 Grafik penurunan *link load*

Sementara pada hasil pengujian latensi, meningkatnya nilai α diikuti oleh menurunnya latensi pada seluruh strategi yang diujikan. Gambar 5.3 menunjukkan penurunan latensi yang terjadi pada strategi berbasis *off-path caching* saat kapasitas *cache* 5%. Latensi pada *on-path caching* dan *off-path caching* saat nilai α = 0,7 lebih tinggi dibandingkan dengan latensi saat nilai α = 0,8, dan latensi saat nilai α = 0,8 lebih tinggi dibandingkan dengan saat nilai α = 0,9. Hal tersebut menunjukkan bahwa nilai α = 0,9 memiliki pengaruh yang paling baik terhadap latensi.



Gambar 5.3 Grafik penurunan latensi

Berdasarkan analisis pengaruh nilai α terhadap kinerja *in-network caching* di atas, dapat disimpulkan bahwa nilai α = 0,9 memiliki pengaruh paling baik terhadap CHR, *link load*, dan latensi dibandingkan dengan nilai α = 0,7 dan 0,8. Nilai α = 0,9 berarti sebuah permintaan konten memiliki peluang 0,5 yang

ditujukkan pada hanya 1,2% konten peringkat teratas berdasarkan perhitungan menggunakan persamaan 2.2. Kondisi tersebut cukup sesuai dengan kondisi riil Internet, di mana konten popular yang hanya berjumlah sedikit lebih sering diakses dibandingkan dengan mayoritas konten yang kurang popular. Nilai α = 0,9 digunakan sebagai acuan untuk menganalisis dan membandingkan *on-path caching* dan *off-path caching* di subbab-subbab selanjutnya, dengan asumsi bahwa pada nilai α tersebut kinerja *in-network caching* berada pada kondisi terbaik. Tabel 5.7 adalah tabel rerata kinerja keseluruhan berdasarkan nilai α = 0,9 dari Tabel 5.1 sampai Tabel 5.6.

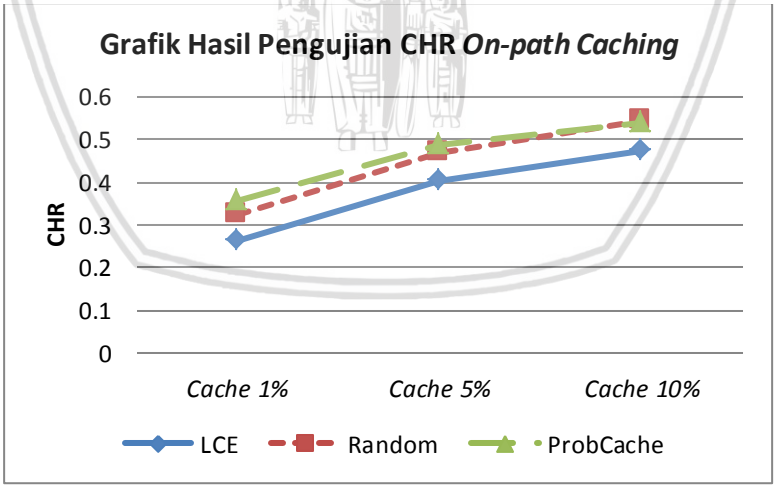
Tabel 5.7 Rerata kinerja keseluruhan

Kapasitas *Cache*	*On-path Caching*			*Off-path Caching*		
	CHR	*Link Load* (MBps)	Latensi (ms)	CHR	*Link Load* (MBps)	Latensi (ms)
1%	0,313	2584,7	117,6	0,352	4067,0	162,9
5%	0,454	2277,7	99,6	0,508	3755,3	142,7
10%	0,518	2137,7	91,4	0,585	3597,0	132,9
Rerata Keseluruhan	**0,428**	**2333,4**	**102,9**	**0,482**	**3806,4**	**146,2**

5.2.2 *Cache Hit Ratio*

Pada analisis *cache hit ratio*, hasil pengujian yang dianalisis hanya pada saat nilai α = 0,9 sesuai hasil analisis di Subbab 5.2.1.

5.2.2.1 *On-path Caching*

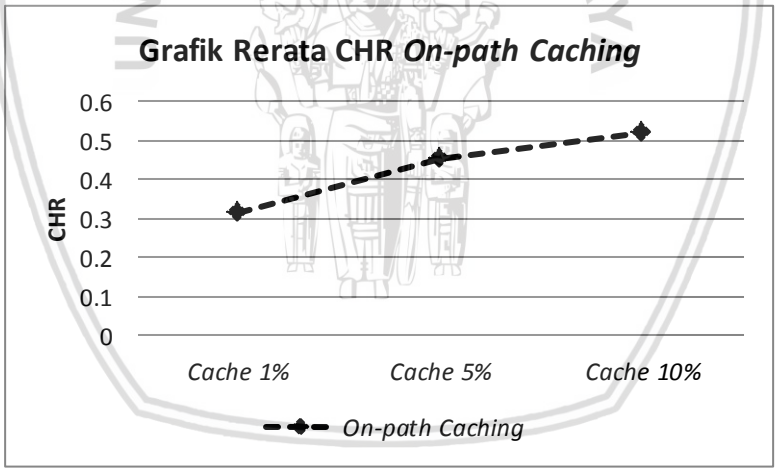


Gambar 5.4 Grafik hasil pengujian CHR *on-path caching*

Untuk hasil pengujian CHR *on-path caching*, CHR seluruh strategi menunjukkan peningkatan seiring dengan meningkatnya kapasitas *cache*, seperti yang ditunjukan pada Gambar 5.4. Saat kapasitas *cache* 1%, CHR strategi ProbCache adalah 0,355, CHR strategi Random adalah 0,322, dan CHR strategi LCE adalah 0,262. Pada kapasitas *cache* 5%, CHR strategi ProbCache adalah 0,488 (terjadi kenaikan 37,31%), CHR strategi Random adalah 0,470 (naik 45,78%), dan

CHR strategi LCE adalah 0,404 (terjadi kenaikan 54,06%). Dan saat kapasitas *cache* 10%, CHR strategi ProbCache naik 10,04% menjadi 0,537, CHR strategi Random naik 15,32% menjadi 0,542, dan CHR strategi LCE naik 17,57% menjadi 0,475. Berdasarkan analisis CHR *on-path caching* tersebut, strategi ProbCache menunjukkan kinerja yang paling baik dengan rerata CHR 0,460, disusul oleh strategi Random dengan rerata CHR 0,445 dan LCE dengan rerata CHR 0,380. Meskipun strategi LCE menyimpan konten di setiap *cache* yang dilewati, namun rerata CHR LCE lebih rendah dibandingkan dengan CHR strategi Random yang menyimpan konten di *cache* secara acak.

Rerata CHR keseluruhan strategi *on-path caching* juga menunjukkan peningkatan CHR seiring dengan meningkatnya kapasitas *cache*, seperti yang ditunjukan pada Gambar 5.5. Saat kapasitas *cache* 1%, rerata CHR strategi *on-path caching* adalah 0,313, sementara kapasitas cache 5%, CHR naik 45,04% menjadi 0,454. Sedangkan saat kapasitas cache 10%, rerata CHR *on-path caching* naik 14,09% menjadi 0,518. Meningkatnya kapasitas *cache* berarti semakin banyak konten yang dapat ditampung, sehingga peluang terjadinya *cache hit* semakin besar. CHR 0,518 pada saat kapasitas *cache* 10% menunjukkan bahwa *cache* pada strategi *on-path caching* mampu melayani lebih dari separuh permintaan konten. Sedangkan sisanya dilayani oleh *content source* karena terjadi *cache miss*.

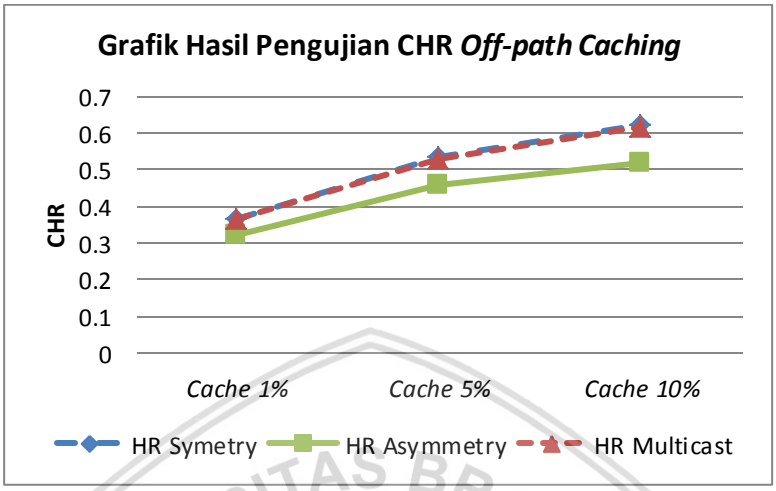


Gambar 5.5 Grafik Rerata CHR *on-path caching*

5.2.2.2 *Off-path Caching*

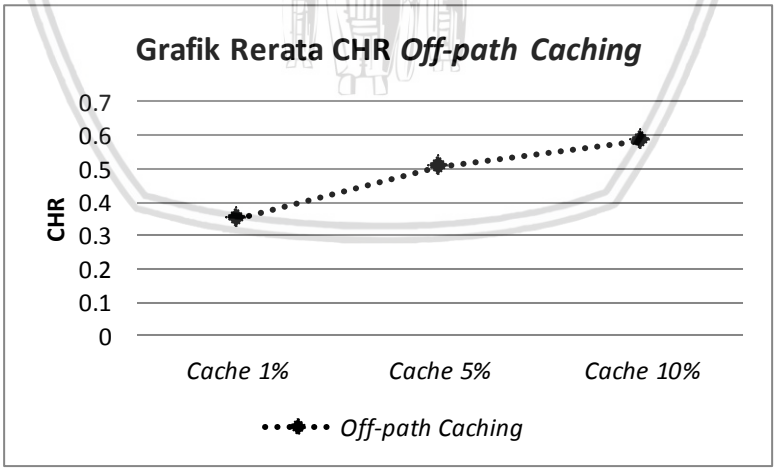
Berdasarkan hasil pengujian CHR *off-path caching*, CHR seluruh strategi juga mengalami peningkatan seiring dengan meningkatnya kapasitas *cache*, seperti pada strategi-strategi *on-path caching*. Saat kapasitas *cache* 1%, CHR strategi Hash-routing Symmetry adalah 0,365, CHR strategi Hash-routing Asymmetry adalah 0,324, dan CHR strategi Hash-routing Multicast adalah 0,367. Sementara saat kapasitas *cache* 5%, CHR strategi Hash-routing Symmetry naik 46,12% menjadi 0,534, CHR strategi Hash-routing Asymmetry naik 41,15% menjadi 0,457, dan CHR strategi Hash-routing Multicast naik 44,88% menjadi 0,532. Dan saat kapasitas *cache* 10%, CHR strategi Hash-routing Symmetry adalah 0,620

(terjadi kenaikan 16,19% dari CHR sebelumnya), CHR strategi Hash-routing Asymmetry adalah 0,517 (naik 13,18% dari CHR sebelumnya), dan CHR strategi Hash-routing Multicast adalah 0,619 (terjadi kenaikan 16,33% dari CHR sebelumnya).



Gambar 5.6 Grafik hasil pengujian CHR *off-path caching*

Grafik hasil pengujian CHR *off-path caching* dapat dilihat pada Gambar 5.6. Pada gambar tersebut, kesamaan karakter Hash-routing Symmetry dan Hash-routing Multicast yang dapat menyimpan konten di *cache* menghasilkan CHR yang hampir sama di tiap kenaikan kapasitas *cache*. Sementara pengiriman konten melalui jalur terpendek saat *cache miss* pada strategi Hash-routing Asymmetry menghasilkan CHR yang lebih rendah dibandingkan dengan strategi Hash-routing Symmetry dan Hash-routing Multicast.



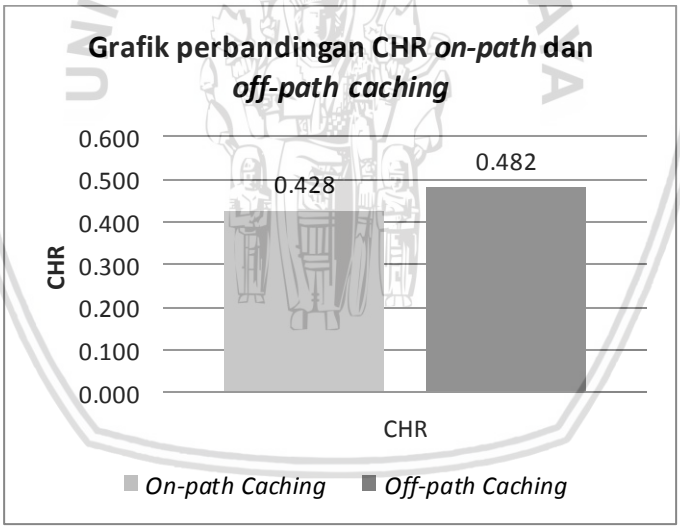
Gambar 5.7 Grafik rerata CHR *off-path caching*

Sementara itu, rerata CHR keseluruhan strategi *off-path caching* juga menunjukkan peningkatan CHR seiring dengan meningkatnya kapasitas *cache*, seperti pada Gambar 5.7. Meningkatnya kapasitas *cache* berarti semakin banyak konten yang dapat ditampung, sehingga peluang terjadinya *cache hit* semakin besar. Saat kapasitas *cache* 1%, rerata CHR strategi *off-path caching* adalah

0,352, sementara saat kapasitas cache naik menjadi 5%, CHR *off-path caching* naik 44,22% menjadi 0,508. Dan saat kapasitas *cache* 10%, rerata CHR *off-path caching* naik 15,29% menjadi 0,585. CHR 0,585 pada saat kapasitas *cache* 10% menunjukkan bahwa *cache* pada strategi *off-path caching* mampu melayani 58,5% permintaan konten. Sedangkan sisanya dilayani oleh *content source* karena terjadi *cache miss*. Strategi *off-path caching* yang memiliki CHR tertinggi adalah Hash-routing Symmetry dan Hash-routing Multicast, dengan rerata CHR 0,506. Sedangkan Hash-routing Asymmetry memiliki rerata CHR 0,433. Kesamaan karakter Hash-routing Symmetry dan Hash-routing Multicast yang dapat menyimpan konten di *cache* menghasilkan CHR yang hampir sama di seluruh pengujian. Kemampuan menyimpan konten tersebut meningkatkan probabilitas terjadinya *cache hit*, sehingga bila dibandingkan Hash-routing Asymmetry yang mengirimkan konten secara melalui jalur terpendek tanpa menyimpannya, Hash-routing Symmetry dan Hash-routing Multicast memiliki CHR yang lebih tinggi.

5.2.2.3 Perbandingan CHR *On-path Caching* dan *Off-path Caching*

Perbandingan CHR *on-path caching* dan *off-path caching* dilakukan terhadap rerata CHR secara keseluruhan. Hal tersebut bertujuan untuk mengetahui mekanisme *in-network caching* mana yang memiliki CHR lebih baik. Rerata CHR keseluruhan dapat dilihat pada Tabel 5.7.



Gambar 5.8 Grafik perbandingan CHR *on-path* dan *off-path caching*

Gambar 5.8 adalah grafik perbandingan rerata CHR keseluruhan *on-path caching* dan *off-path caching*. Berdasarkan gambar tersebut, rerata CHR keseluruhan *off-path caching* 12,45% lebih tinggi dibandingkan dengan rerata CHR keseluruhan *on-path caching*. Hal tersebut cukup beralasan mengingat mekanisme tersebut menyimpan konten di *cache* yang sudah ditentukan saat terjadi *cache miss*, sehingga memperbesar peluang terjadinya *cache hit* pada permintaan-permintaan selanjutnya. Sedangkan *on-path caching* menyimpan konten di *cache* yang dilalui oleh paket konten. Rerata CHR 0,428 untuk *on-path caching* dan 0,482 untuk *off-path caching* menunjukkan bahwa rata-rata 42,8% permintaan konten berhasil dilayani oleh *cache* pada *on-path caching* dan 48,2%

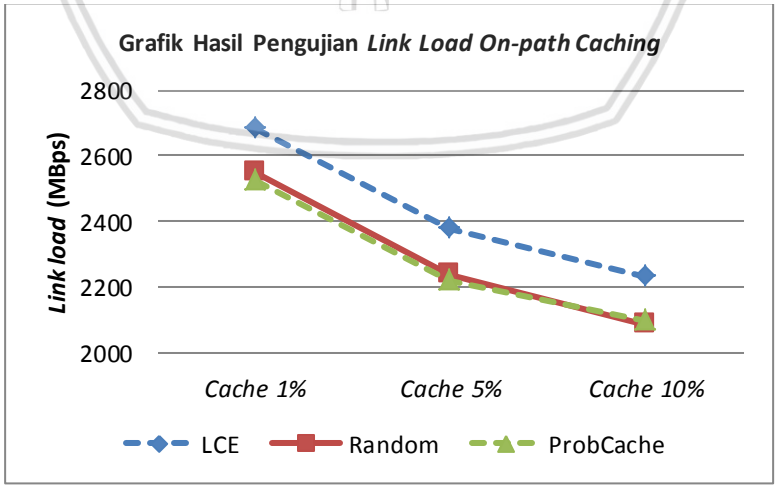
permintaan konten berhasil dilayani oleh *cache* pada *off-path caching*. Selain itu, strategi *off-path caching* dengan rerata CHR tertinggi, Hash-routing Multicast dan Hash-routing Symmetry, memiliki rerata CHR 10% lebih tinggi dibandingkan dengan CHR strategi ProbCache, yang merupakan strategi *on-path caching* dengan rerata CHR tertinggi.

5.2.3 *Link Load*

Pada analisis *link load*, hasil pengujian yang dianalisis juga hanya pada saat nilai α = 0,9 sesuai hasil analisis di Subbab 5.2.1.

5.2.3.1 *On-path Caching*

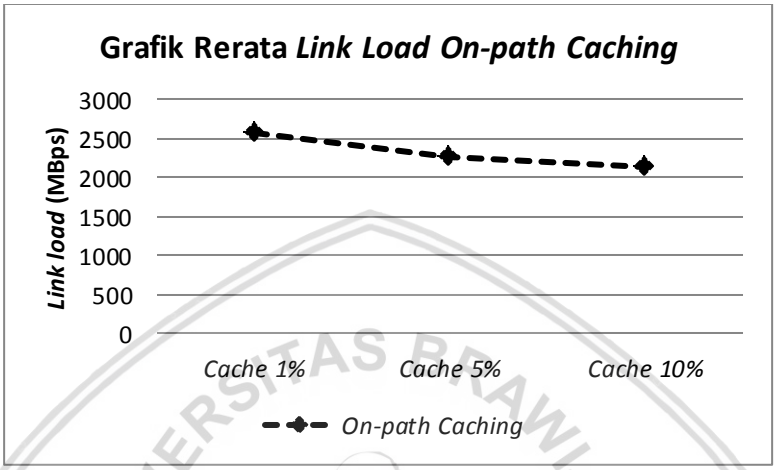
Pada pengujian *link load on-path caching*, hasil pada seluruh strategi menunjukkan penurunan *link load* seiring dengan meningkatnya kapasitas *cache*. Saat kapasitas *cache* 1%, *link load* strategi ProbCache adalah 2525 MBps, strategi Random adalah 2548 MBps, dan strategi LCE adalah 2681 MBps. Sementara saat kapasitas *cache* 5%, *link load* strategi ProbCache turun 12,11% menjadi 2219 MBps, strategi Random turun 12,12% menjadi 2239 MBps, dan strategi LCE turun 11,40% menjadi 2375 MBps. Dan saat kapasitas *cache* 10%, link load strategi ProbCache adalah 2097 MBps (terjadi penurunan sebesar 5,52%), strategi Random adalah 2086 MBps (turun 6,82%), dan strategi LCE adalah 2230 MBps (terjadi penurunan 6,11%). Hasil pengujian tersebut menunjukkan bahwa strategi ProbCache dan Random memiliki *link load* yang hampir sama, seperti yang dapat dilihat pada Gambar 5.9. Strategi *on-path caching* yang memiliki *link load* terendah adalah ProbCache, dengan rerata *link load* 2280,3 MBps. Sementara strategi Random memiliki rerata *link load* yang sedikit lebih tinggi dibandingkan link load ProbCache, yaitu 2291,0 MBps. Sedangkan strategi *on-path caching* dengan *link load* tertinggi adalah LCE, dengan rerata *link load* 2428,7 MBps.



Gambar 5.9 Grafik hasil pengujian *link load on-path caching*

Rerata *link load* keseluruhan strategi *on-path caching* juga mengalami penurunan seiring dengan meningkatnya kapasitas *cache*. Pada saat kapasitas *cache* 1%, rerata *link load on-path caching* adalah 2584,7 MBps. Saat kapasitas

cache naik menjadi 5%, rerata *link load* turun 11,87% menjadi 2277,7 MBps. Dan saat kapasitas *cache* naik menjadi 10%, rerata *link load on-path caching* turun 6,14% menjadi 2137,7 MBps. Grafik penurunan rerata *link load* tersebut dapat dilihat pada Gambar 5.10. Kapasitas *cache* yang semakin besar memungkinkan terjadinya *cache hit* yang semakin sering, sehingga permintaan tidak perlu diteruskan ke *cache* lain atau *content source* yang dapat menyebabkan naiknya *link load*.

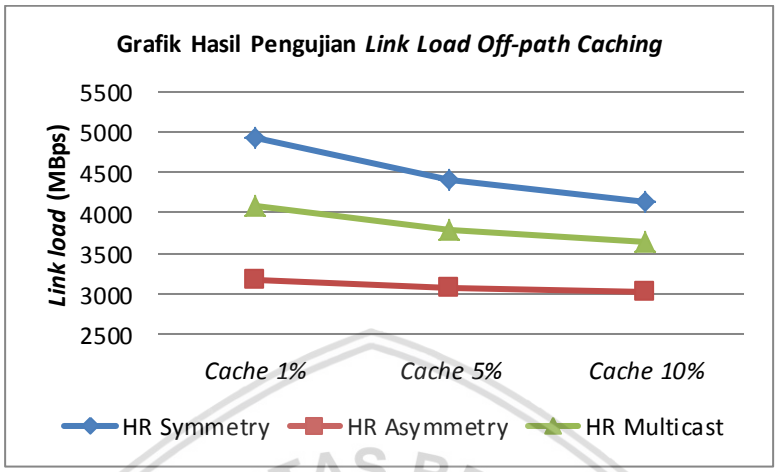


Gambar 5.10 Grafik rerata *link load on-path caching*

5.2.3.2 *Off-path Caching*

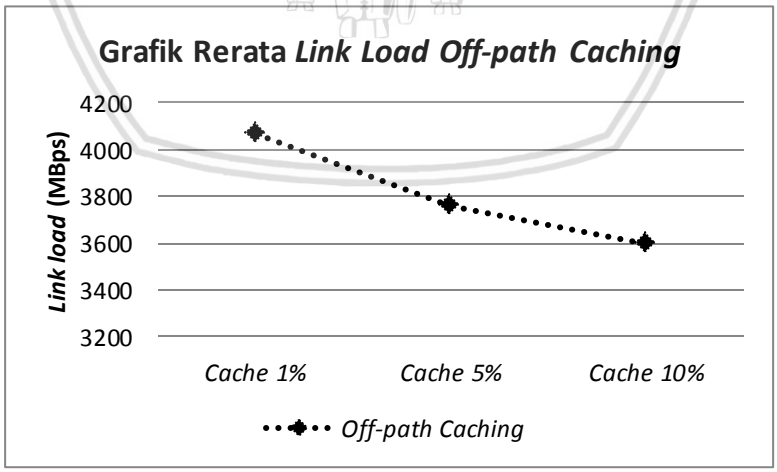
Berdasarkan hasil pengujian *link load off-path caching*, *link load* seluruh strategi mengalami penurunan seiring dengan meningkatnya kapasitas *cache* seperti pada hasil pengujian *link load on-path caching*. Saat kapasitas *cache* 1%, *link load* strategi Hash-routing Symmetry adalah 4923 MBps, sementara strategi Hash-routing Asymmetry adalah 3180 MBps, dan strategi Hash-routing Multicast adalah 4098 MBps. Pada saat kapasitas *cache* 5%, *link load* strategi Hash-routing Symmetry turun 10,64% menjadi 4399 MBps, strategi Hash-routing Asymmetry turun 3,33% menjadi 3074 MBps, dan strategi Hash-routing Multicast turun 7,45% menjadi 3793 MBps. Dan saat kapasitas *cache* 10%, *link load* untuk strategi Hash-routing Symmetry adalah 4124 MBps (terjadi penurunan 6,25%), strategi Hash-routing Asymmetry adalah 3034 MBps (turun 1,32%), dan strategi Hash-routing Multicast adalah 3633 MBps (turun 4,21%). Strategi *off-path caching* yang memiliki rerata *link load* terendah adalah Hash-routing Asymmetry, dengan rerata *link load* 3096,0 MBps. Sementara rerata *link load* Hash-routing Symmetry adalah 4482,0 MBps dan rerata *link load* Hash-routing Multicast adalah 3841,3 MBps. Hasil pengujian tersebut menunjukkan bahwa strategi Hash-routing Asymmetry dengan metode pengiriman konten melalui jalur terpendek memiliki *link load* yang paling rendah dibandingkan dengan strategi Hash-routing yang lain. Pengiriman melalui jalur terpendek dapat mengurangi jumlah *link* yang dipakai sebagai media pengiriman, sehingga *link load* pun lebih rendah. Strategi Hash-routing Multicast juga mengirimkan konten melalui jalur terpendek, tetapi ia mengirimkan konten ke *cache* untuk disimpan yang menyebabkan link load yang lebih tinggi dibandingkan dengan Hash-routing

Asymmetry. Sementara strategi Hash-routing Symmetry memiliki *link load* tertinggi yang disebabkan jalur pengiriman paket *request* dan *content* bersifat simetris. Grafik hasil pengujian *link load off-path caching* dapat dilihat pada Gambar 5.11.



Gambar 5.11 Grafik hasil pengujian *link load off-path caching*

Rerata *link load* keseluruhan strategi *off-path caching* juga mengalami penurunan seiring dengan meningkatnya kapasitas *cache*. Kapasitas *cache* yang semakin besar memungkinkan terjadinya *cache hit* yang semakin sering, sehingga permintaan tidak perlu diteruskan ke *content source* yang dapat menyebabkan naiknya *link load*. Pada saat kapasitas *cache* 1%, rerata *link load off-path caching* adalah 4067 MBps. Saat kapasitas *cache* naik menjadi 5%, rerata *link load* turun 7,66% menjadi 3755,3 MBps. Dan saat kapasitas *cache* naik menjadi 10%, rerata *link load off-path caching* menurun 4,21% menjadi 3597 MBps. Grafik penurunan *link load* tersebut dapat dilihat pada Gambar 5.12.

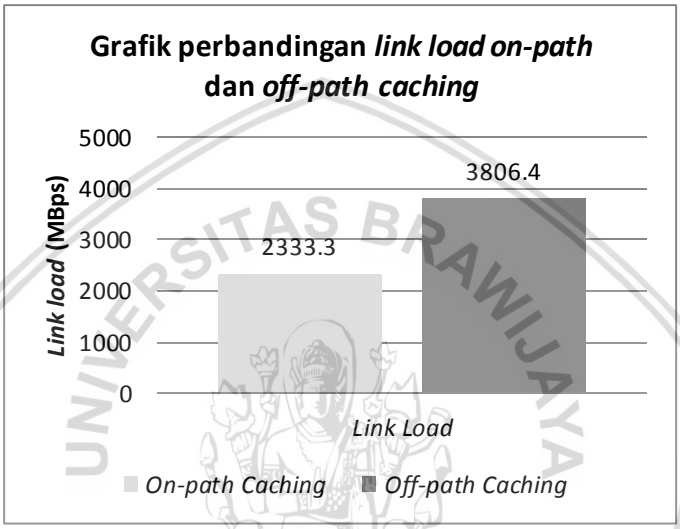


Gambar 5.12 Grafik rerata *link load off-path caching*

5.2.3.3 Perbandingan *link load On-path Caching* dan *off-path Caching*

Untuk perbandingan *link load,* berdasarkan Tabel 5.7, rerata *link load* keseluruhan mekanisme *on-path caching* 63,14% lebih rendah dibandingkan dengan rerata *link load* keseluruhan mekanisme *off-path caching*. Hal tersebut

cukup beralasan karena pada mekanisme *on-path caching* hanya terjadi satu kali pengiriman konten melewati jalur yang simetris bila terjadi *cache hit* atau *cache miss*, sedangkan mekanisme *off-path caching* harus mengirim dua kali saat terjadi *cache miss* (ke *node receiver* dan ke *cache* untuk menyimpan konten) yang menyebabkan bertambahnya lalu lintas data dalam jaringan. Gambar 5.13 adalah grafik perbandingan rerata keseluruhan *link load on-path caching* dan *off-path caching* yang didasarkan pada Tabel 5.7. Selain itu, strategi *on-path caching* dengan rerata *link load* terendah, ProbCache, memiliki rerata *link load* 35,77% lebih rendah dibandingkan dengan *link load* strategi Hash-routing Asymmetry, yang merupakan strategi *off-path caching* dengan *link load* terendah.



Gambar 5.13 Grafik perbandingan *link load on-path* dan *off-path caching*

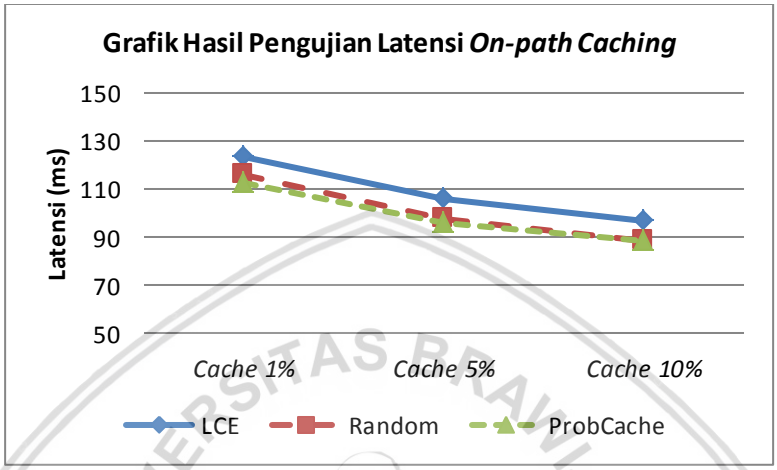
5.2.4 Latensi

Pada analisis latensi, hasil pengujian yang dianalisis juga hanya pada saat nilai α = 0,9 sesuai hasil analisis di Subbab 5.2.1.

5.2.4.1 *On-path Caching*

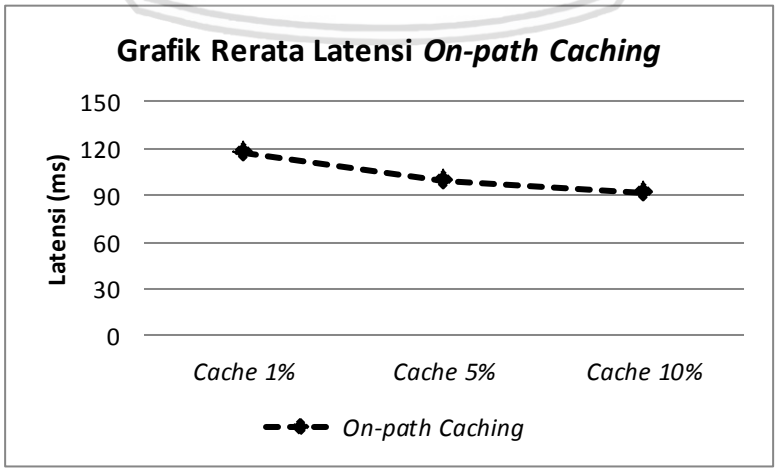
Pada pengujian latensi *on-path caching*, hasil pada seluruh strategi menunjukkan penurunan latensi seiring dengan meningkatnya kapasitas *cache*. Saat kapasitas *cache* 1%, latensi strategi ProbCache adalah 113,1 ms, strategi Random adalah 116,1 ms, dan strategi LCE adalah 123,6 ms. Sementara saat kapasitas *cache* 5%, latensi strategi ProbCache turun 15,39% menjadi 95,7 ms, strategi Random turun 15,99% menjadi 97,5 ms, dan strategi LCE turun 14,51% menjadi 105,7 ms. Dan saat kapasitas *cache* 10%, latensi strategi ProbCache adalah 89,0 ms (terjadi penurunan sebesar 6,93%), strategi Random adalah 88,3 ms (turun 9,39%), dan strategi LCE adalah 96,9 ms (terjadi penurunan 8,34%). Strategi *on-path caching* yang memiliki rerata latensi terendah adalah ProbCache, dengan rerata latensi 99,3 ms. Sementara strategi Random memiiki rerata latensi yang hampir sama dengan strategi ProbCache, yaitu 100,6 ms. Strategi LCE memiliki rerata latensi paling tinggi dibandingkan dengan rerata latensi ProbCache dan Random, yaitu 108,7 ms. Meskipun strategi LCE

menyimpan konten di setiap *cache* yang dilewati paket konten (yang memungkinkan terjadinya *cache hit* dengan cepat), tetapi bila dibandingkan dengan strategi Random yang menyimpan konten di sembarang *cache* latensi LCE masih lebih tinggi. Sedangkan strategi ProbCache yang menyimpan konten di *cache* terbaik berdasarkan persamaan 2.1 memiliki latensi yang hampir sama dengan latensi strategi Random yang menyimpan konten di *cache* secara acak. Grafik hasil pengujian latensi *on-path caching* dapat dilihat pada Gambar 5.14.



Gambar 5.14 Grafik hasil pengujian latensi *on-path caching*

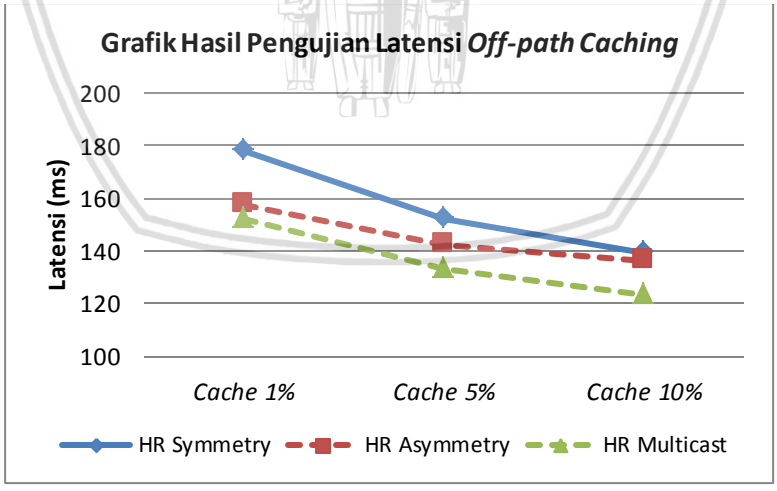
Pada pengujian latensi *on-path caching*, hasil pengujian pada Tabel 5.5 menunjukkan penurunan rerata latensi seiring dengan meningkatnya kapasitas *cache*. Saat kapasitas *cache* 1%, rerata latensi strategi *on-path caching* adalah 117,6 ms. Sementara saat kapasitas *cache* naik menjadi 5%, rerata latensi turun 15,27% menjadi 99,6 ms. Sedangkan saat kapasitas *cache* 10%, rerata latensi strategi *on-path caching* turun 8,26% menjadi 91,4 ms. Meningkatnya kapasitas *cache* memungkinkan terjadinya *cache hit* yang lebih cepat karena *cache* dapat menyimpan lebih banyak konten. *Cache hit* yang semakin cepat berarti latensi pengiriman semakin menurun. Gambar 5.15 adalah grafik rerata latensi keseluruhan strategi *on-path caching* di setiap kenaikan kapasitas *cache*.



Gambar 5.15 Grafik rerata latensi *on-path caching*

5.2.4.2 *Off-path Caching*

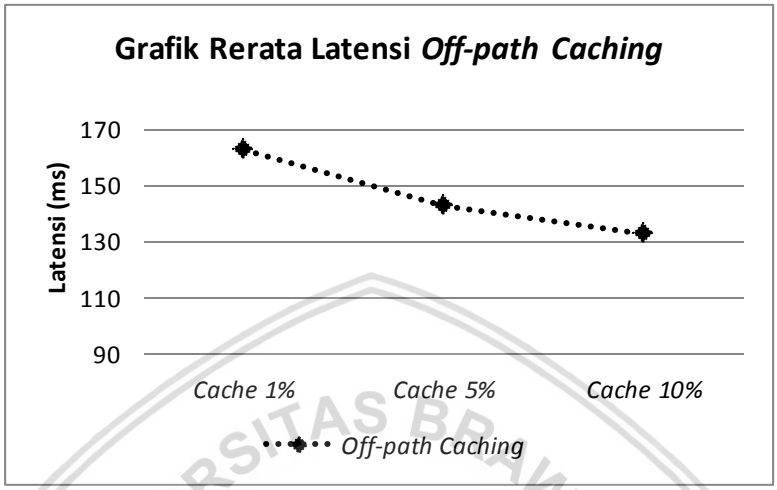
Berdasarkan hasil pengujian latensi *off-path caching*, latensi seluruh strategi menurun seiring dengan meningkatnya kapasitas *cache*, seperti pada Gambar 5.16. Saat kapasitas *cache* 1%, latensi strategi Hash-routing Symmetry adalah 178,4 ms, sementara strategi Hash-routing Asymmetry adalah 157,9 ms, dan strategi Hash-routing Multicast adalah 152,3 ms. Pada saat kapasitas *cache* 5%, latensi strategi Hash-routing Symmetry turun 14,55% menjadi 152,4 ms, strategi Hash-routing Asymmetry turun 9,94% menjadi 142,2 ms, dan strategi Hash-routing Multicast turun 12,34% menjadi 133,5 ms. Dan saat kapasitas *cache* 10%, latensi untuk strategi Hash-routing Symmetry adalah 139,0 ms (terjadi penurunan 8,8%), strategi Hash-routing Asymmetry adalah 136,2 ms (turun 4,19%), dan strategi Hash-routing Multicast adalah 123,6 ms (turun 7,39%). Strategi *off-path caching* yang memiliki rerata latensi terendah adalah Hash-routing Multicast, dengan rerata latensi 136,5 ms. Sedangkan rerata latensi strategi Hash-routing Symmetry adalah 156,6 ms dan strategi Hash-routing Asymmetry adalah 145,4 ms. Strategi Hash-routing Multicast memiliki latensi yang paling rendah karena mengirimkan konten melalui jalur terpendek. Sementara strategi Hash-routing Asymmetry, meskipun juga mengirimkan konten melalui jalur terpendek, memiliki latensi yang lebih tinggi dibandingkan dengan strategi Hash-routing Multicast. Hal tersebut disebabkan oleh tidak adanya mekanisme penyimpanan konten di *cache* pada saat terjadi *cache miss*, sehingga kemungkinan terjadinya *cache hit* kecil karena konten jarang masuk ke *cache*. Strategi Hash-routing Symmetry memiliki latensi tertinggi pada tiap hasil pengujian yang disebabkan oleh jalur simetris yang dilalui paket *request* dan *content*.



Gambar 5.16 Grafik hasil pengujian latensi *off-path caching*

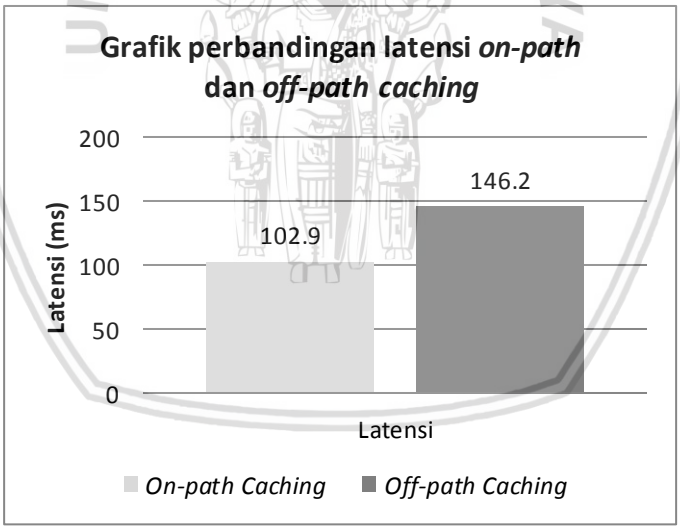
Berdasarkan hasil pengujian latensi pada Tabel 5.6, rerata latensi strategi *off-path caching* menunjukkan penurunan seiring dengan meningkatnya kapasitas *cache*. Saat kapasitas *cache* 1%, rerata latensi strategi *off-path caching* adalah 162,9 ms. Sementara saat kapasitas *cache* naik menjadi 5%, rerata latensi turun 12,38% menjadi 142,7 ms. Sedangkan saat kapasitas *cache* 10%, rerata latensi

strategi *off-path caching* turun 6,84% menjadi 132,9 ms. Meningkatnya kapasitas *cache* memungkinkan terjadinya *cache hit* yang lebih cepat karena *cache* dapat menyimpan lebih banyak konten. *Cache hit* yang semakin cepat berarti latensi pengiriman semakin menurun. Gambar 5.17 adalah grafik rerata latensi strategi *off-path caching* tiap kenaikan kapasitas *cache*.



Gambar 5.17 Grafik rerata latensi *off-path caching*

5.2.4.3 Perbandingan latensi *On-path Caching* dan *Off-path Caching*



Gambar 5.18 Grafik perbandingan latensi *on-path* dan *off-path caching*

Untuk perbandingan latensi, berdasarkan Tabel 5.7, mekanisme *on-path caching* memiliki rerata keseluruhan latensi 42,07% lebih rendah dibandingkan dengan rerata latensi keseluruhan mekanisme *off-path caching*. Hal ini karena mekanisme *off-path caching* menyimpan konten di *cache* yang seringkali tidak dekat (karena sudah ditentukan) dengan *receiver* sehingga membutuhkan waktu yang lebih lama dibandingkan dengan *on-path caching* yang menyimpan konten pada *cache* yang dilalui oleh konten. Gambar 5.12 adalah grafik perbandingan rerata keseluruhan latensi mekanisme *on-path caching* dan *off-path caching.* Selain itu, apabila terjadi *cache miss*, mekanisme *off-path caching* juga akan

meneruskan paket *request* ke *content source*, sehingga memperpanjang waktu yang dibutuhkan dalam satu kali sesi permintaan. Strategi *on-path caching* dengan rerata latensi terendah, ProbCache, memiliki rerata latensi 37,47% lebih rendah dibandingkan dengan latensi strategi Hash-routing Multicast.

BAB 6 PENUTUP

Pada bab ini akan dibahas kesimpulan dari analisis yang telah dilakukan serta saran-saran untuk penelitian lebih lanjut.

6.1 Kesimpulan

Kesimpulan yang dapat ditarik dari penelitian ini antara lain:

1. Rerata CHR keseluruhan mekanisme *on-path caching* adalah 0,428, sedangkan rerata CHR keseluruhan mekanisme *off-path caching* adalah 0,482. Dengan demikian, mekanisme *off-path caching* memiliki CHR 12,45% lebih tinggi dibandingkan mekanisme *on-path caching*. Strategi *in-network caching* yang memiliki CHR tertinggi adalah Hash-routing Symmetry dan Hash-routing Asymmetry dengan rerata CHR 0,506.
2. Rerata *link load* keseluruhan mekanisme *on-path caching* adalah 2333,4 MBps, sedangkan rerata *link load* keseluruhan mekanisme *off-path caching* adalah 3806,4 MBps. Dengan demikian, mekanisme *on-path caching* memiliki *link load* 63,14% lebih rendah dibandingkan dengan mekanisme *off-path caching*. Strategi *in-network caching* yang memiliki *link load* terendah adalah ProbCache dengan rerata *link load* 2280 MBps.
3. Rerata latensi keseluruhan mekanisme *on-path caching* adalah 102,9 ms, sedangkan rerata latensi keseluruhan mekanisme *off-path caching* adalah 146,2 ms. Dengan demikian, mekanisme *on-path caching* memiliki rerata latensi keseluruhan 42,07% lebih rendah dibandingkan dengan rerata latensi keseluruhan mekanisme *off-path caching*. Strategi *in-network caching* yang memiliki latensi terendah adalah ProbCache dengan rerata latensi 99,3 ms.
4. Berdasarkan kinerja yang sudah diuji, strategi *in-network caching* yang cocok pada topologi Biznet adalah ProbCache yang memiliki latensi dan *link load* paling rendah di antara seluruh strategi *in-network caching* yang diujikan, serta CHR cukup tinggi yang hampir sama dengan mekanisme *off-path caching* (hanya 10% lebih rendah dibandingkan dengan CHR tertinggi mekanisme *off-path caching*).

Selain kesimpulan-kesimpulan di atas, berdasarkan hasil analisis juga ditemukan bahwa semakin besar kapasitas *cache* dan nilai α berarti semakin tinggi CHR, semakin rendah *link load*, serta semakin rendah latensi. Selain itu, meskipun mekanisme *off-path caching* dapat menjamin CHR yang lebih tinggi, tetapi dalam hal pengiriman konten mekanisme *on-path caching* dapat mengirim lebih cepat.

6.2 Saran

Saran-saran untuk penelitian selanjutnya antara lain:

1. Proses pengujian dilakukan pada lebih banyak strategi *in-network caching* dan pada pada topologi riil lain.

2. Parameter pengujian ditambahkan selain dari CHR, *link load*, dan latensi.
3. Perlu ada penelitian lebih lanjut tentang jarak ideal antara *cache* dan *receiver*. Selain itu perlu dilakukan studi lebih lanjut tentang nilai α yang sesuai untuk memodelkan popularitas konten agar sesuai dengan kondisi riil Internet.

DAFTAR PUSTAKA

Auerbach, F. (1913). Das Gesetz der Bevoelkerungskonzentration. *Petermanns Geographische Mitteilungen*, 74-76.

Breslau, L., Cao, P., Fan, L., Phillips, G., & Shenker, S. (1999). Web Caching and Zipf-like Distribution: Evidence and Implications. *Proceedings of Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies, INFOCOM '99* (pp. 126-134). IEEE.

Cha, M., Kwak, H., Rodriguez, P., Ahn, Y.-Y., & Moon, S. (2007). I Tube, You Tube, Everybody Tubes: Analyzing The World's Largest User Generated Content Video System. *Proceedings of the 7th ACM SIGCOMM Conference of Internet Measurement, IMC'07* (pp. 1-14). New York, NY: ACM.

Cisco. (2016). *Cisco Visual Networking Index: Forecast and Methodology, 2015-2020.* Cisco.

Dannewitz, C., Kutscher, D., Ohlman, B., Farrel, S., Ahlgren, B., & Karl, H. (2013). Network of Information (NetInf) - An Information-centric Networking Architecture. *Computer Communications*, 721-735.

Danzig, P. B., Hall, R. S., & Schwartz, M. F. (1993). A Case for Caching File Objects Inside Internetworks. *ACM SIGCOMM 1993* (pp. 239-243). San Francisco, CA: ACM.

Ghodsi, A., Koponen, T., Raghavan, B., Shenker, S., Singla, A., & Wilcox, J. (2011). Information-Centric Networking: Seeing the Forest for the Trees. *Proceedings of the 10th ACM Workshop on Hot Topics in Networks, Hotnets'11* (pp. 1-6). New York, NY: ACM.

Gupta, A., Shailendra, S., & Girish, A. (2016). Analysis of In-Network Caching for ICN. *IEEE Annual India Conference (INDICON), 2016* (pp. 1-5). Bangalore, India: IEEE.

Himsolt, M. (2010). *GML: A Portable Graph File Format.* Passau: Universitaet Passau.

Jacobson, V., Smetters, D. K., Thornton, J. D., Plass, M. F., Briggs, N. H., & Braynard, R. L. (2009). Networking Named Content. *Proceedings of the 5th international conference on Emerging networking experiments and technologies, CoNEXT '09* (pp. 1-12). New York, NY: ACM.

Jeon, H., Lee, B., & Song, H. (2013). On-Path Caching in Information-Centric Networking. *15th Conference on Advanced Communication Technology (ICACT), 2013* (pp. 264-267). PyeongChang, South Korea: IEEE.

Kalla, A., & Sharma, S. (2016). Exploring Off Path Caching with Edge Caching in Information Centric Networking. *Proceedings of 2016 International Conference on Computational Techniques in Information and*

Communication Technologies (ICCTICT) (pp. 630-635). New Delhi, India: IEEE.

Koponen, T., Chawla, M., Chun, B.-G., Ermolinskiy, A., Kim, K. H., Shenker, S., et al. (2007). A Data-Oriented (and Beyond) Network Architecture. *Proceedings of the 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, SIGCOMM'07* (pp. 181-192). New York, NY: ACM.

Kuliesius, F., & Paulauskas, P. (2015). Simulation of Content Caching in Information Centric Networking. *Seventh International Conference on Ubiquitous and Future Networks (ICUFN), 2015* (pp. 391-394). Sapporo, Japan: IEEE.

Laoutaris, N., Syntia, S., & Stavrakakis, I. (2004). Meta Algorithms for Hierarchical Web Caches. *IEEE International Conference on Performance, Computing, and Communication 2004.* Phoenix, AZ: IEEE.

Psaras, I., Chai, W. K., & Pavlou, G. (2012). Probabilistic In-Network Caching for Information Centric Networks. *Proceedings of the second edition of the ICN workshop on Information-centric networking, ICN '12* (pp. 55–60). Helsinki: ACM.

Rajahalme, J., Sarela, M., Visala, K., & Riihijarvi, J. (2010). Routing On Name-Based Inter-Domain. *Computer Networks*, 975-986.

Rossi, D., & Rossini, G. (2012). Caching performance of content centric networks under multi-path routing (and more). *IEEE 17th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD), 2012* (pp. 105-109). Barcelona, Spain: IEEE.

Rossini, G., & Rossi, D. (2011). A Dive into the Caching Performance of Content Centric Networking.

Saino, L., Psaras, I., & Pavlou, G. (2013). Hash-routing Schemes for Information Centric Networking. *Proceedings of 3rd ACM SIGCOMM (ICN'13).* Hong Kong: ACM.

Saino, L., Psaras, I., & Pavlou, G. (2014). Icarus: a Caching Simulator for Information Centric Networking (ICN). *Proceedings of 7th International ICST Conference, SIMUTOOLS'14.* Lisbon: ACM.

Tarkoma, S., Ain, M., & Visala, K. (2009). The Publish/Subscribe Internet Routing Paradigm (PSIRP): Designing the Future Internet Architecture. In G. Tselentis, J. Domingue, A. Galis, A. Gavras, D. Hausheer, S. Krco, et al., *Towards the Future Internet: A European Research Perspective* (pp. 102-111). IOS Press.

Wang, L. (2015). *Content, Topology and Cooperation in In-network Caching.* Helsinki: University of Helsinki.

Yi, C., Afanasyev, A., Moiseenko, I., Wang, L., Zhang, B., & Zhang, L. (2013). A Case for Stateful Forwarding Plane. *Computer Communications*, 779-791.

Zhang, B., Ng, E., Nandi, A., Riedi, R., Druschel, P., & Wang, G. (2006). Measurement-Based Analysis, Modeling, and Synthesis of the Internet Delay Space. *Proceedings of the ACM/Usenix Internet Measurement Conference (IMC 2006).* Rio de Janerio: ACM.

Zipf, G. K. (1949). *Human Behaviour and the Principle of the Least Effort.* Cambridge, MA: Addison-Wesley.