Analysis of LCD(Leave Copy Down) &LCE(Leave Copy Everywhere) caching scheme for tree topology.

By

S.M Rozibul Islam

ID: 2011-2-58-032

Al Rafi Moon

ID: 2011-3-55-022

Supervised By

Dr. Mohammad Arifuzzaman

Assistant Professor

Electronics and Communications Engineering Department

East West University

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of Bachelors of Science in Electronics and Communications Engineering

to the

Department of Electronics and Communications Engineering

East West University

Dhaka, Bangladesh

December, 2016

Abstract

In computing, a cache is a hardware or software component that stores data so future requests for that data can be served faster, the data stored in a cache might be the result of an earlier computation, or the duplicate of data stored elsewhere. In this thesis we compare two caching mechanism LCE (Leave Copy Everywhere) & LCD(Leave Copy Down) for tree topology of internet architecture.

In our comparison, we found that LCD(Leave Copy Down) performs better compared to LCE(Leave Copy Everywhere).

However, in case of very popular content & extremely less popular content the performance of both scheme are almost same. Here we use only tree topology network & get a successful output. So it is possible to work with this in another topology also.

Declaration

I hereby declare that, this Thesis has been done under ETE498 and has not been submitted elsewhere for requirement of any degree or diploma or for any purpose except for publication.

S.M Rozibul Islam
ID: 2011-2-58-032
Department of ECE
East West University

Al Rafi Moon
ID: 2011-3-55-022
Department of ECE
East West University

Letter of Acceptance

We hereby declare that this Thesis is from the student's own work and best effort of mine, and all other source of information used have been acknowledge. This Thesis has been submitted with our approval.

____

Dr. Mohammad Arifuzzaman

Supervisor

Assistant Professor

Department of ECE

East West University

____

Dr. Md. Mofazzol Hossain

Chairperson

Professor and Chairperson

Department of ECE

East West University

Acknowledgement

First of all, I would like to thank almighty Allah for giving me the strength & proper knowledge to complete my thesis work.

I would like to express my deep sense of gratitude and sincere thanks to my honorable supervisor Dr. Mohammad Arifuzzaman, Assistant Professor, Department of ECE, East West University, Aftabnagar, Dhaka, Bangladesh, for his at most direction, encouragement, kind guidance, sharing knowledge and constant inspiration throughout this project work.

My sincere gratefulness for the faculty of ECE whose friendly attitude and enthusiastic support that has given me for four years.

I am very grateful for the motivation and stimulation from my friends and seniors.

Finally my most heartfelt gratitude goes to my beloved parents and brother for their endless support, continuous inspiration, great contribution and perfect guidance from the beginning to the end.

Table of Contents

Chapter 1

Introduction

In computing, a cache is a hardware or software component that stores data so future requests for that data can be served faster, the data stored in a cache might be the result of an earlier computation, or the duplicate of data stored elsewhere. A cache hit occurs when the requested data can be found in a cache, while a cache miss occurs when it cannot. Cache hits are served by reading data from the cache, which is faster than re-computing a result or reading from a slower data store; thus, the more requests can be served from the cache, the faster the system performs.

To be cost-effective and to enable efficient use of data, caches are relatively small. Nevertheless, caches have proven themselves in many areas of computing because access patterns in typical computer applications exhibit the locality of reference. Moreover, access patterns exhibit temporal locality if data is requested again that has been recently requested already, while spatial locality refers to requests for data physically stored close to data that has been already requested.

1.1 Objective of the thesis

Our project "Analysis of LCD (Leave Copy Down) & LCE (Leave Copy Everywhere) caching scheme for tree topology" is focused to the following objectives:

- ✓ To discuss different caching methods.
- ✓ Analyze merits & demerits of each method.
- ✓ Reduce load on Web Services/ Database.
- ✓ Increase Performance.
- ✓ Reliability (Assuming db backed cache. Server goes down and db is backed by cache there is no time wasted to repopulate an in memory cache).

1.2 Methodology of the thesis

Here, in this thesis we simulated with OMNeT++. OMNeT++ is an extensible, modular, component-based C++ simulation library and framework, primarily for building network simulators.Although OMNeT++ is not a network simulator itself, it has gained widespread popularity as a network simulation platform in the scientific community as well as in industrial settings, and building up a large user community. OMNeT++ provides a component architecture for models.

Chapter 2

Description Of Thesis Work

In this thesis we compare two caching mechanism LCE (Leave Copy Everywhere) & LCD (Leave Copy Down) for tree topology of internet architecture.

In our comparison, we found that LCD performs better compared to LCE.

However, in case of very popular content & extremely less popular content the performance of both scheme are almost same.

2.1 Cache

In computing, a cache is a hardware or software component that stores data so future requests for that data can be served faster; the data stored in a cache might be the result of an earlier computation, or the duplicate of data stored elsewhere. A cache hit occurs when the requested data can be found in a cache, while a cache miss occurs when it cannot. Cache hits are served by reading data from the cache, which is faster than recomputing a result or reading from a slower data store; thus, the more requests can be served from the cache, the faster the system performs.

To be cost-effective and to enable efficient use of data, caches are relatively small. Nevertheless, caches have proven themselves in many areas of computing because access patterns in typical computer applications exhibit the locality of reference. Moreover, access patterns exhibit temporal locality if data is requested again that has been recently requested already, while spatial locality refers to requests for data physically stored close to data that has been already requested.

2.2 Cache Operation

Hardware implements cache as a block of memory for temporary storage of data likely to be used again. Central processing units (CPUs) and hard disk drives (HDDs) frequently use a cache, as do web browsers and web servers.

A cache is made up of a pool of entries. Each entry has associated data, which is a copy of the same data in some backing store. Each entry also has a tag, which specifies the identity of the data in the backing store of which the entry is a copy.

When the cache client needs to access data presumed to exist in the backing store, it first checks the cache. If an entry can be found with a tag matching that of the desired data, the data in the entry is used instead. This situation is known as a cache hit. So, for example, a web browser program might check its local cache on disk to see if it has a local copy of the contents of a web page at a particular URL. In this example, the URL is the tag, and the contents of the web page is the data. The percentage of accesses that result in cache hits is known as the hit rate or hit ratio of the cache.

The alternative situation, when the cache is consulted and found not to contain data with the desired tag, has become known as a cache miss. The previously uncached data fetched from the backing store during miss handling is usually copied into the cache, ready for the next access.

During a cache miss, the CPU usually ejects some other entry in order to make room for the previously uncached data. The heuristic used to select the entry to eject is known as the replacement policy. One popular replacement policy, "least recently used" (LRU), replaces the least recently used entry. More efficient caches compute use frequency against the size of the stored contents, as well as the latencies and throughputs for both the cache and the backing store. This works well for larger amounts of data, longer latencies and slower throughputs, such as experienced with a hard drive and the Internet, but is not efficient for use with a CPU cache.

2.3 Buffer Vs. Cache

The semantics of a "buffer" and a "cache" are not necessarily mutually exclusive; even so, there are fundamental differences in intent between the process of caching and the process of buffering.

Fundamentally, caching realizes a performance increase for transfers of data that is being repeatedly transferred. While a caching system may realize a performance increase upon the initial transfer of a data item, this performance increase is due to buffering occurring within the caching system.

With read caches, a data item must have been fetched from its residing location at least once in order for subsequent reads of the data item to realize a performance increase by virtue of being able to be fetched from the cache's (faster) intermediate storage rather than the data's residing location. With write caches, a performance increase of writing a data item may be realized upon the first write of the data item by virtue of the data item immediately being stored in the cache's intermediate storage, deferring the transfer of the data item to its residing storage at a later stage or else occurring as a background process. Contrary to strict buffering, a caching process must adhere to a (potentially distributed) cache coherency protocol in order to maintain consistency between the cache's intermediate storage and the location where the data resides. Buffering, on the other hand,

reduces the number of transfers for otherwise novel data amongst communicating processes, which amortizes overhead involved for several small transfers over fewer, larger transfers,

provides an intermediary for communicating processes which are incapable of direct transfers amongst each other, or

ensures a minimum data size or representation required by at least one of the communicating processes involved in a transfer.

With typical caching implementations, a data item that is read or written for the first time is effectively being buffered; and in the case of a write, mostly realizing a performance increase for the application from where the write originated. Additionally, the portion of a caching protocol where individual writes are deferred to a batch of writes is a form of buffering. The portion of a caching protocol where individual reads are deferred to a batch of reads is also a form of buffering, although this form may negatively impact the performance of at least the initial reads (even though it may positively impact the performance of the sum of the individual reads). In practice, caching almost always involves some form of buffering, while strict buffering does not involve caching.

A buffer is a temporary memory location that is traditionally used because
CPU instructions cannot directly address data stored in peripheral devices. Thus, addressable memory is used as an intermediate stage. Additionally, such a buffer may be feasible when a large block of data is assembled or disassembled (as required by a storage device), or when data may be delivered in a different order than that in which it is produced. Also, a whole buffer of data is usually transferred sequentially, so buffering itself sometimes increases transfer performance or reduces the variation or jitter of the transfer's latency as opposed to caching where the intent is to reduce the latency. These benefits are present even if the buffered data are written to the buffer once and read from the buffer once.

A cache also increases transfer performance. A part of the increase similarly comes from the possibility that multiple small transfers will combine into one large block. But the main performance-gain occurs because there is a good chance that the same data will be read from cache multiple times, or that written data will soon be read. A cache's sole purpose is to reduce accesses to the underlying slower storage. Cache is also usually an abstraction layer that is designed to be invisible from the perspective of neighboring layers.

2.4 Network Topology

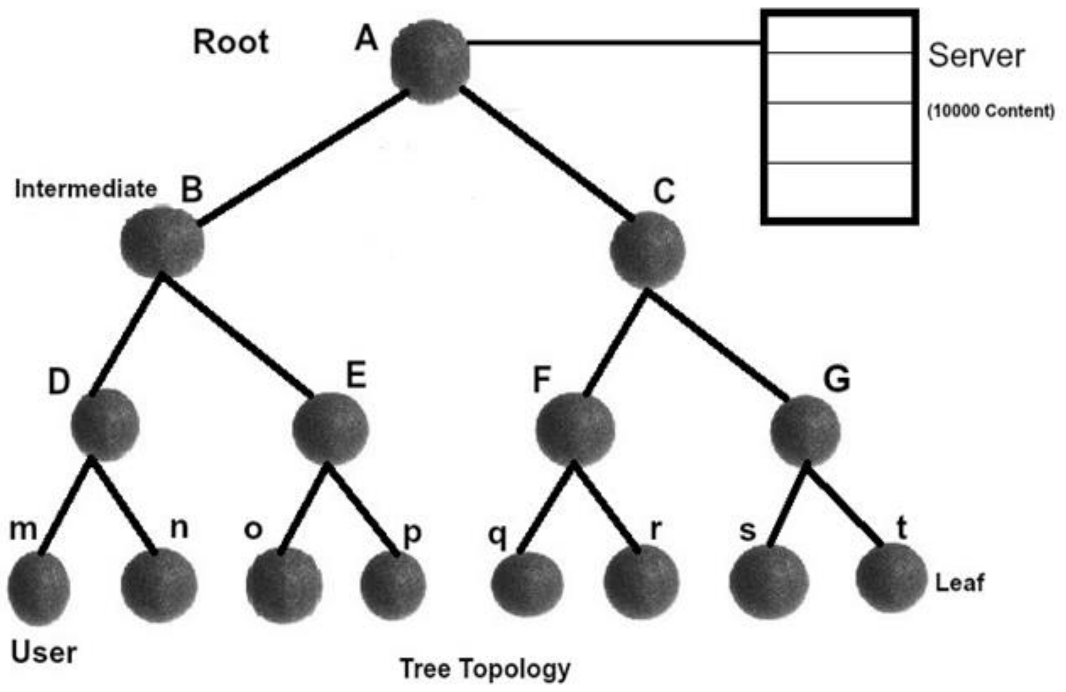

Figure- 2.4: Tree Topology.

Here, The line are considered as link, circle are considered as node. Node means routers.

- A (root) is connected to server.
- Server has 10000 objects.
- Intermediate node (router) B, C, D, E, F, G has cache capacity of 100 object.
- Users are connected with the leaf nodes (router) m, n, o, p, q, r, s, t.

2.5 LCE (Leave Copy Everywhere)

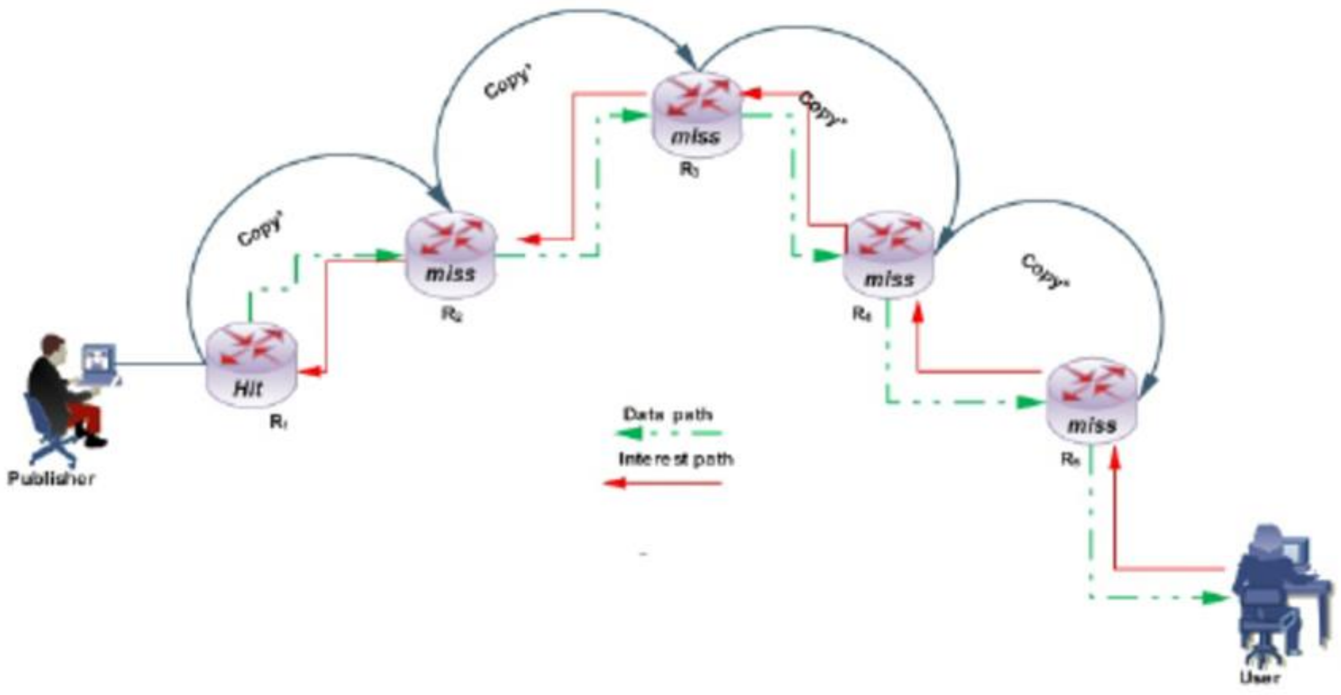

Figure- 2.5: LCE(Leave Copy Everywhere).(Adopted from Ref.1)

Cache management in LCE is defined by its operation of caching data in every node crossed. Part of the practice of the ICN is the ability to make information readily and easily accessible as described in ICN initial proposal. As a user sends out a request using LCE, the nature of the network serves the interest using hierarchical search and ordering of nodes to acquire a cache-hit. Given a network with a hierarchy ordered nodes n1, n2, ..., nm, once a request for a data is sent in the order 1,..., m (from the subscriber to Publisher R5-R1) and the data gets a hit (cache-hit) at a node ni, the copy of the data is cached at all immediate nodes on the route of content delivery in the order: {ni-1, n(i+1)-1, ..., n(1-m)}.This as a result of having to save a copy of the content in all single leveled node causing the high content redundancy. Its replacement algorithm could take the First In First Out (FIFO),Least Recently Used (LRU), Most Frequently Used (MFU), Random etc. Figure 2.5 presents a LCE caching operation.

2.6 LCD (Leave Copy Down)

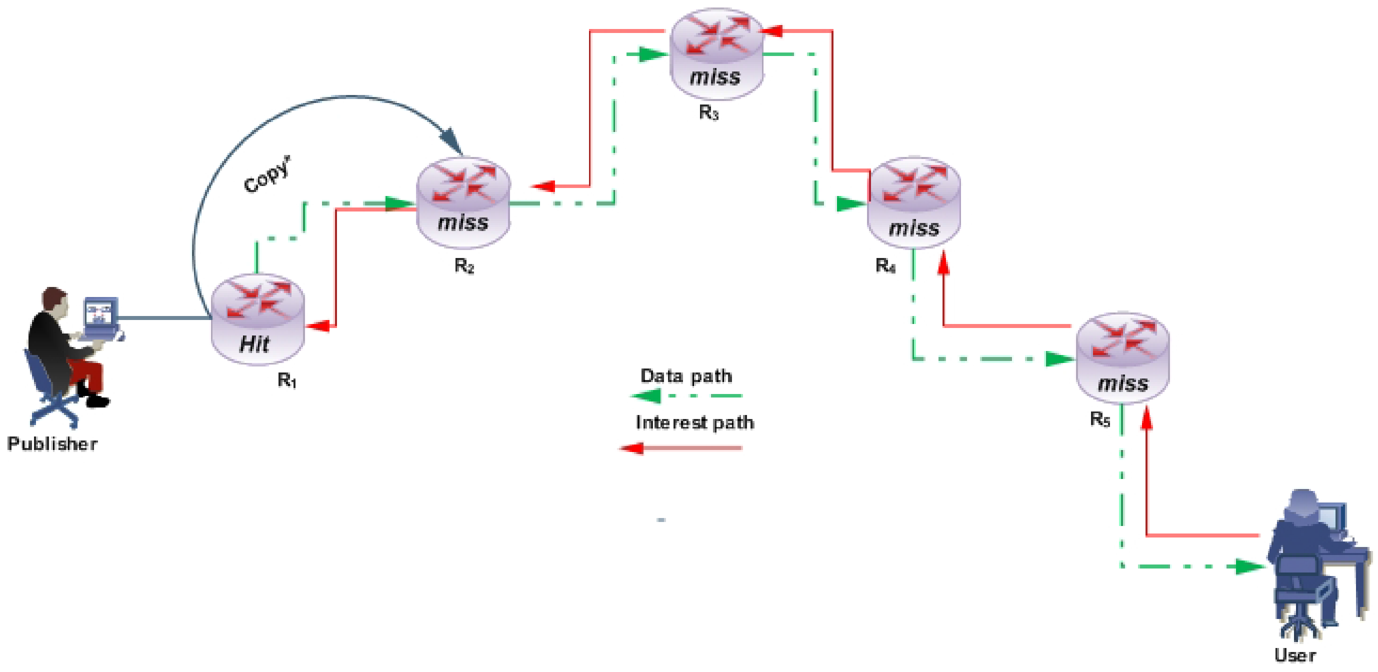

Figure- 2.6:LCD (Leave Copy Down). (Adopted from Ref.2)

The Leave Copy Down (LCD) is a cache management strategy that defines the form and manner of content caching on nodes. Its operation works in a similar fashion as the popular "drop at the first neighbor" process. In LCD once request for an interest is posed, a path link is created in the form of {n1, n2, ..., nm)} to the Publisher of the data or the content holding node. As the content records a hit on R1,it only stores the copy in its direct neighbor node using the traversing path to the subscriber R2. For LCD to gain the high LCE states of availing contents in almost all nodes, it needs a content-hit at almost all levels of {n1, n2,...,nm} (see Figure 2.6).

2.7 Probabilistic Cache

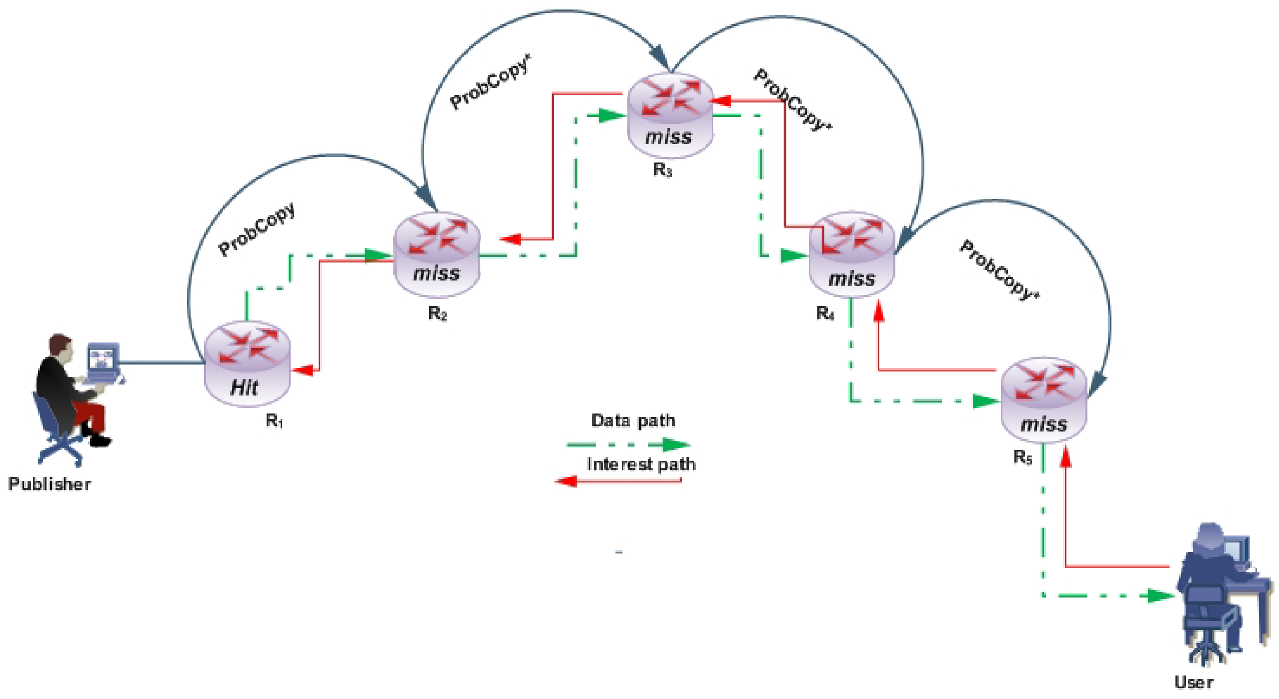

Figure- 2.7: Probabilistic Cache. (Adopted from Ref.3)

The Probcache is assumed as a probabilistic LCE in a randomized manner. Its main distinction to the LCE is the ability to keep its cache copy in an immediate neighbor node (n) with probability p. Probcache therefore does not keep a cached copy in a node with q (1-p) probability. Once a cache is hit, a probability computation is acted on depending on the chosen distribution from the subscriber to Publisher R5- R1(see Figure 2.7).

2.8 Cache Replacement Strategies

The size and catalog nature of caches makes it almost impossible to store data and information for long. The major difference between caches and secondary, primary storage is the amount of time dished out for a content to reside in a memory. Several replacement policies have been studied and proposed to be inculcated into ICN. A great number of researchers have however argued that the LRU still provides the best results against other types. In this study, we shall analyze the LRU against Random and Semantic replacement strategies.

A.Semantic Replacement

Semantic replacement can be said to be the replacement that adhere to a define syntax and rules. In Semantic replacement strategy, a cache administrator defines the tags, fields and forms of condition. This condition can be called the grammar nature of the replacement policies. In semantics, the envisioned disadvantage is the strong building in the set of rules defined. The rules are usually not flexible and thus may require a complete redesign if the needed content are unfairly replaced.

B. Random Replacement

In Random replacement policies, once data object or name data objects (NDO) are requested, a copy of the data is found and served known as cache-hit. The resulting data is then saved temporarily in a neighbor node based on the strategy adopted as discussed in Section III. However, the replacement in Rand uses a uniform distribution. According to the distribution, contents are thus replaced based on the uniform selection after exhausting the available memory and catalog sizes in the nodes. Typically, researchers have thus challenged this replacement policy as sometimes the most important data are replaced in lieu of non-popular contents.

C. Least Recently Used

Least Recently Used (LRU) replacement strategy has been agreed as the most efficient cache replacement policy in ICN (Information-centric network). LRU replaces contents in caches according to recency of usage. A popular content is usually demanded more than a least popular content in the network. The recency of a data or content is directly proportional to its usage which thus serves the main objective of adequate information sharing in and out of the network.

2.9 Cache Management Comparison with Replacement Policies

This section presents our empirical analysis and comparison of cache managements against replacement policies to verify the submission by several researchers about the sufficiency of LRU. ProbCache, LCD and LCE are presented as thus: This work adopted the nature of traffic with the following parameters. The simulation was ran for forty (40) occurrences to attain an acceptable rates of instances in ICN traffic manageability using SocialccnSim.

Simulation Parameters

Parameters	**Value**
Number of users	**3980**
Number of files	**10,000**
Mapping Algorithm	**Random**
Time limit	**86,400**
Social graph	**Facebook**
Number of communities	**5**
Cache replacement	**LRU, Semantic and Rand**

A. ProbCache Comparison

In ProbCache, the simulation was ran on an Abilene network topology with a randomly generated traffic using the social graph known as Facebook. The results shows that on several ran, the LRU was performing magnanimously to achieving better hit ratios. Figure 4 depicts the obtained results against the selected cache replacement policies in our analysis.

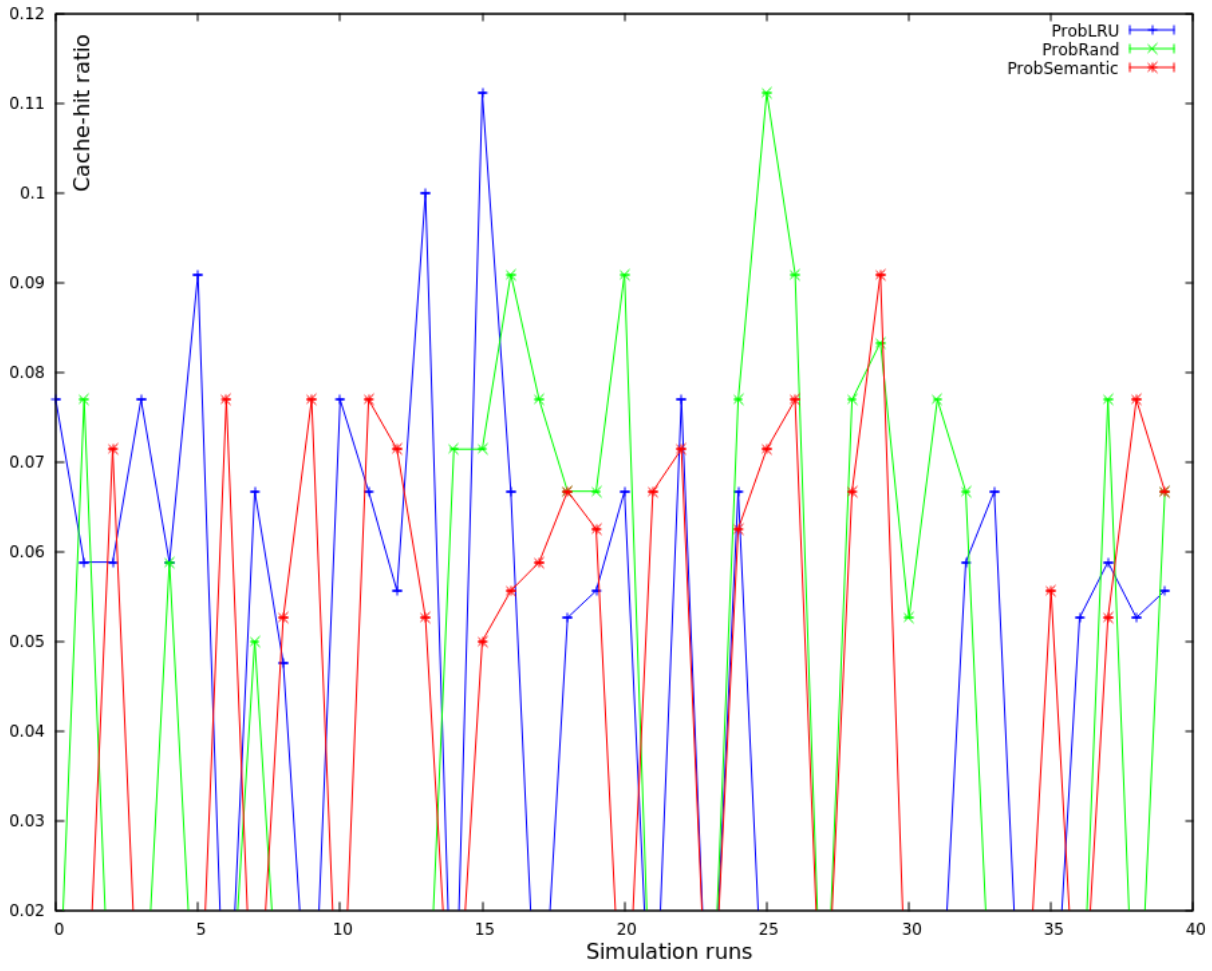

Figure- 2.9(A): ProbCache and cache replacement policies.(Adopted from Ref.4)

B. Leave Copy Down Comparison

Leave Copy Down in our simulation shows the relationship of having a content only cached at an immediate neighbor after a cache-hit. However, when the operation of replacement is needed, semantic, Rand and LRU was tested and the obtained results shown on Figure 5 presents the facts of LRU being a more acceptable replacement concept in ICN. It is fair to note that from simulation run 8-25 LRU out-perform the other policies which can be concluded as the best also when using the LCD.

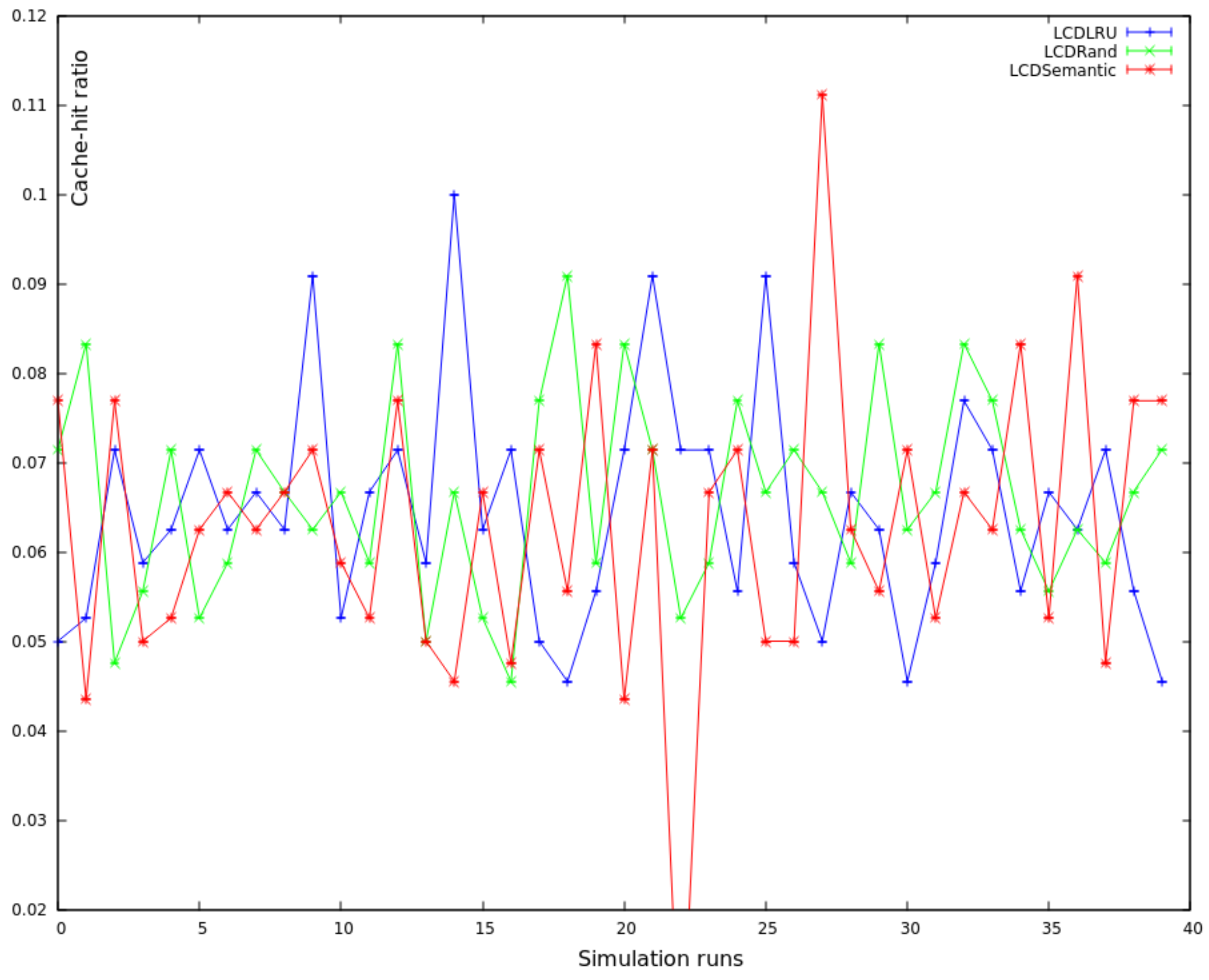

Figure- 2.9(B):LCD and cache replacement policies.(Adopted from Ref.5)

C. Leave Copy Everywhere Comparison

The LCE is the most popular ICN management strategy as it was the initial designed in the first ICN proposal. Since LCE deposits the copy of the demanded data on all nodes, this was challenged as a probable content and path redundancy causing practice. However, in a network that demands popular objects in nearly constant intervals, the LCE is still the best. But in terms of replacement, since it would barely be replacing all contents in its nodes, the simulation results shows that LRU is the optimal on LCE even though the Rand replacement was closely competitive given its uniform distribution nature. See Figure 2.9(C) for the obtained results.

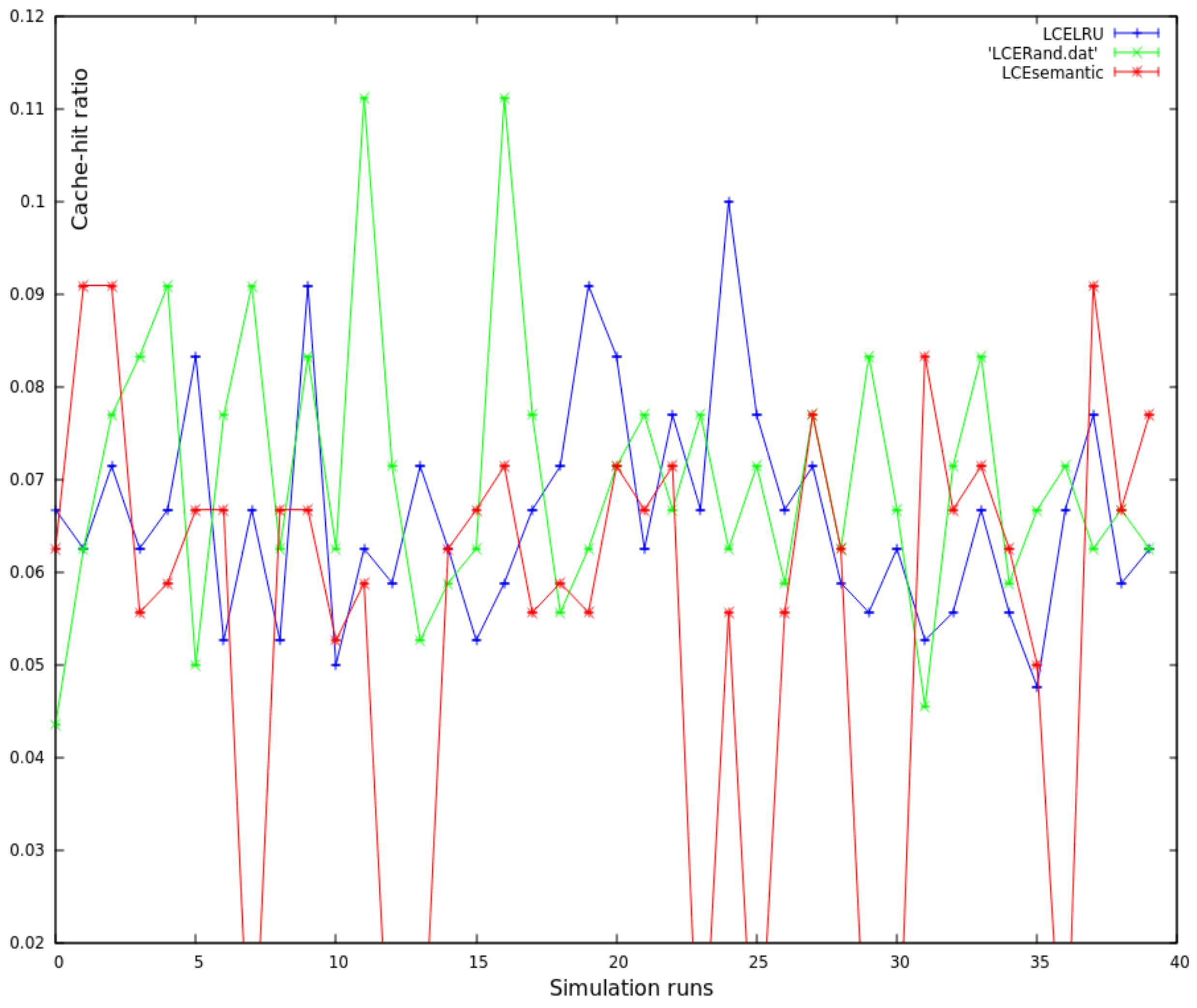

Figure- 2.9(C): LCE and cache replacement policies. (Adopted from Ref.6)

The study presented a highly simulated empirical comparison of different cache management strategies in ICN by analyzing the strength and weaknesses of each placement and replacement strategies. The study proved the conclusive submission by the leading ICN research communities in adopting the LRU as the finest replacement policy for an ICN design. Since ICN is still in its infancy in terms of standardization, the authors tried to present this study as an experimentally proven contribution to aid in future draft and design submission for the future Internet. The study therefore concludes its finding that LRU replacement policy has great significance to ICN performance in regards to minimal bandwidth consumption, better throughput by analogy since the cache-hit ratio are better with wide percentage against the others. Suggestion for the future will be to extend the simulation run to different network topology and other traffic generation platform.

Chapter 3

Operation Tools

3.1OMNeT++

OMNeT++ [7] is an extensible, modular, component-based C++ simulation library and framework, primarily for building network simulators. Although OMNeT++ is not a network simulator itself, it has gained widespread popularity as a network simulation platform in the scientific community as well as in industrial settings, and building up a large user community. OMNeT++ provides component architecture for models. Components are programmed in C++, then assembled into larger components and models using a high-level language. OMNeT++ has extensive GUI support, and due to its modular architecture, the simulation kernel (and models) can be embedded easily into applications. OMNeT++ runs on Windows, Linux, Mac OS X, and other Unix-like systems. The OMNeT++ IDE requires Windows, Linux, or Mac OS X. Here we use OMNeT++ for our simulation.

3.2 Algorithm

Algorithm For Upstream Using OMNeT++

```
i:Node   where interest is propagate
R: The storehouse that contains the requested data
J: The node attached/next to R
Initialize (Content router ←i)
Initialize (Visited←0)
Execute a loop (Content router !=j)
{
If (data is found in Data table of Content store)
Return Content router
ElseIf(Visited!= 1) and (data temporary location, T is found in Cache-Route table of Content
store)
Execute a loop (Content router !=T)
{
Visited←1
Content router← Next hop towards T
Forward interest to Content router
If ( data is found in data table of Content store)
Return Content router
}
Content router← Next hop towards j
Forward interest to Content router
}
```

Algorithm For Downstream Using OMNeT++

```
i:Node   where interest is propagate
S←Content router value retrieved from upstream Algorithm
Execute a loop(Content router !=i)
{
Content router ← Next hop towards i
Forward data to Content router
Ifhop_count(R)≥hop_count(C)
Update Cache-Route table of Content store
}
If(S= =R) Store data at Content store of i
If(S= =C) Update Cache-Route table of Content store
```

3.3 Simulation Results

We simulate with a scalable chunk-level simulator of CCN called ccnSim [8] under the OMNet++ framework. The network topology used in the simulation is depicted in Fig 2.1. There are 14 CCN nodes and each node has a cache size of 100 objects. There are 2 repositories connected with the node B and node C. Total number of objects is 10000 and they are stored in both the repository next to B and next to C. Clients are connected at each node. The aggregate request stream for an object i arrives to the each CCN node with a rate, $\lambda_i = p(i)\lambda$, request generation rate is a Poisson distribution and randomly tagged with one of the 10 CCN nodes provided that for each CCN node the aggregate request rate over all objects is $\lambda \leq 1$Hz. Cache object eviction policy and also cached content's rule eviction policy is LRU.

The Simulation is done for the following performance metrics:

Result of the simulation text file is plotted in Microsoft Office Excel Worksheet.

1). Average Hop-distance:

Minimization of average hop distance is one of the prime goals for most of the networks. From the user point of view, lowering the hop distance means low round trip delay. From the other point of view, lower hop distance denotes lower traffic flow.

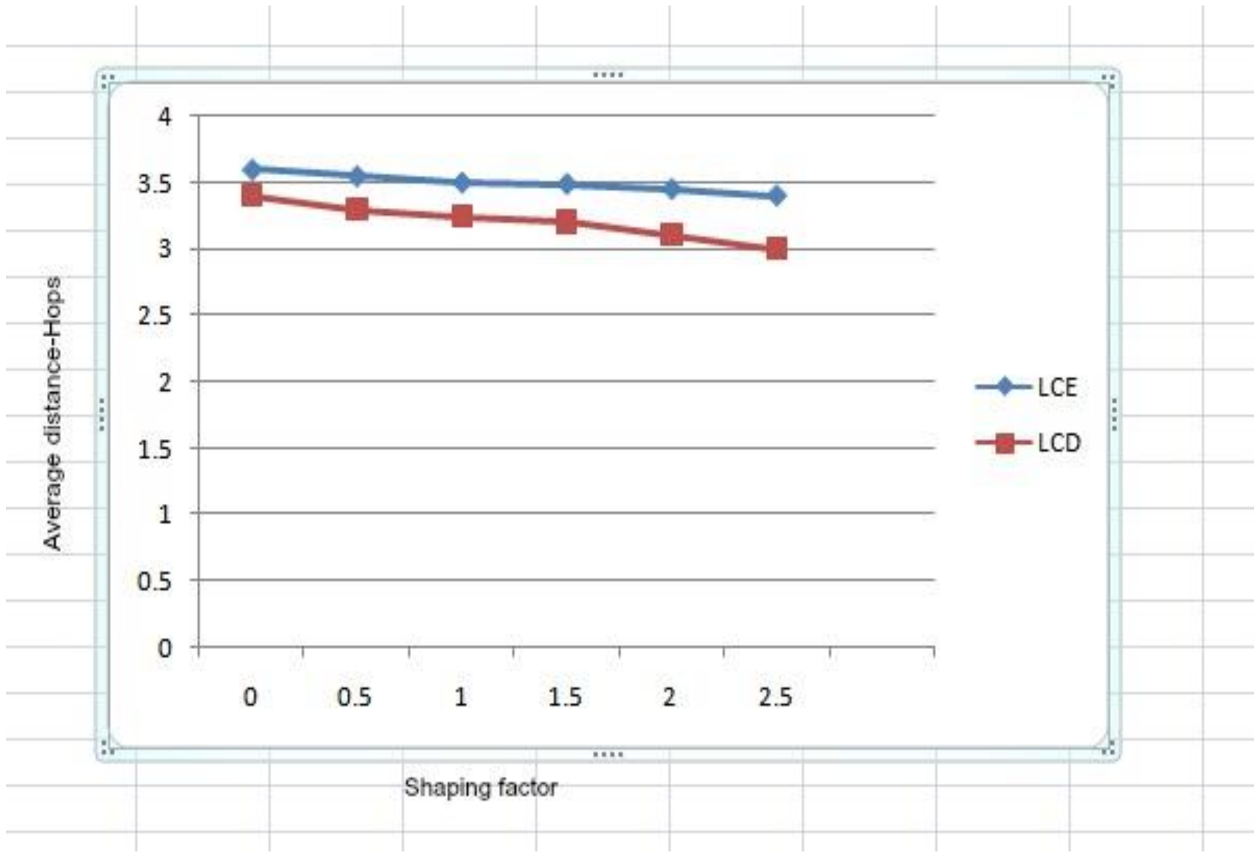

Figure- 3.3(1): Comparison of average hop-distance for LCD & LCE for policy.

Fig: 3.3(1) shows the comparison of LCD (Leave Copy Down) scheme with the LCE (Leave Copy Everywhere) scheme. It is evident from the figure that LCD scheme outperforms the LCE scheme. However, at the lower value for α, the performance difference between the two schemes is low. And for the increasing value for α, LCD scheme achieves higher gain than LCE.

2). Server hit ratio:

Alleviating server hit ratio is an important objective of deploying wide-spread cache structure of ICN. In Fig: 3.2 the comparison of Server hit ratio for objects of different popularity between LCD & LCE policy is depicted. Again it is evident from the figure that LCD scheme outperforms the LCE scheme. It is because the LCD ensures to cache the data in the path only if it is found in the repository. If it is found in the cache, it fetches the data from the cache but do not make any further replica on the path. Rather it just updates the Cache-route table entry of the Content store. This ensures fewer replicas within the local network. In other words, wide range of popular contents is accumulated in the local network. As this concept assists to locate the temporary copy or cached data, the server hit ratio is minimized as a result.

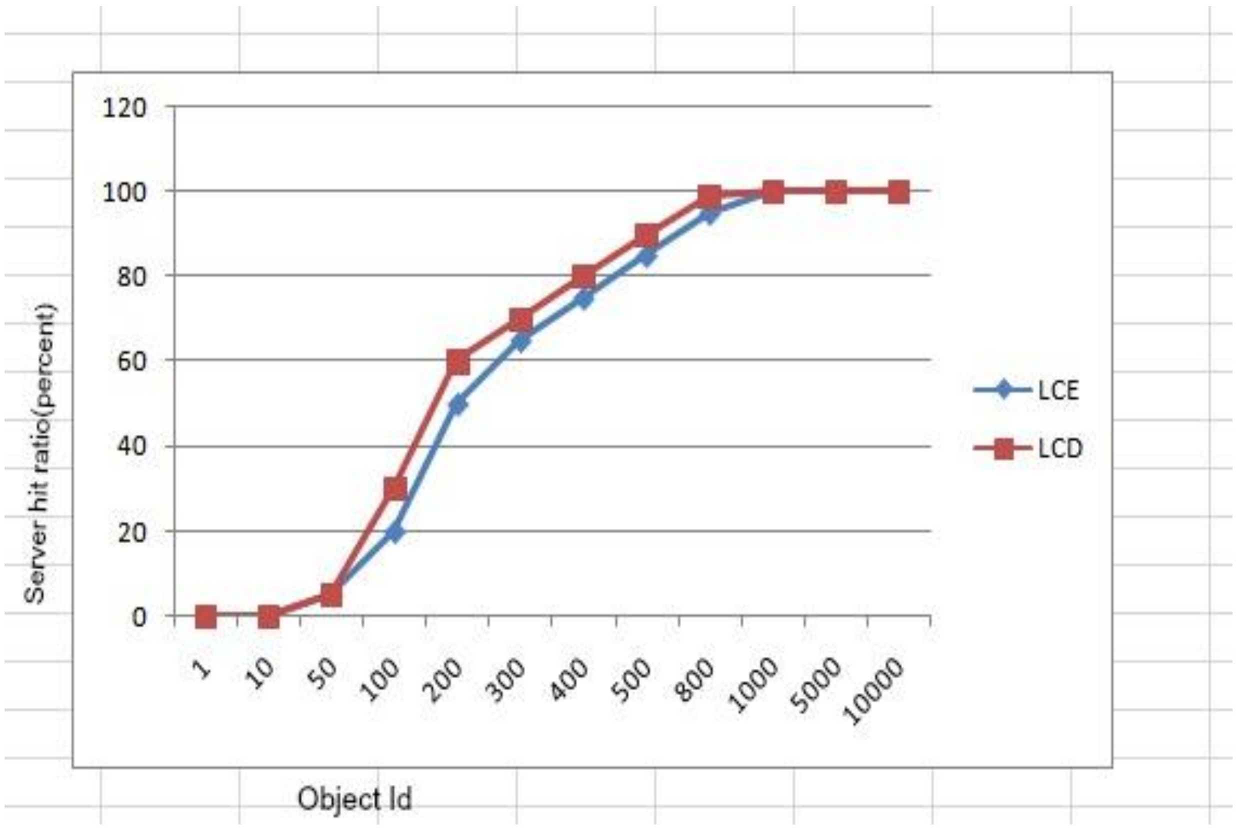

Figure- 3.3(2): Comparison hit ratio between LCD & LCE for the object of different popularity.

Chapter 4

Conclusion and Future Work

As this system saves both money and time. Here we use only tree topology network & get a successful output. So it is possible to work with this in another topology also. In this project, we take a holistic approach to revisit and address the on-path caching and forwarding strategy for ICN. We tried to introduce a novel concept of content’s cache route/path caching. We eliminate cache redundancy with help of the comparison between LCD & LCE schemes and at the same time we put deliberate effort to retrieve cache copy of content rather than accessing the remote server. As a scope of future work, we will work with other network topologies and get the same successful result.

References:

1. https://www.researchgate.net/figure/287723915_fig1_Figure-1-Leave-copy-everywhere

2. https://www.researchgate.net/figure/287723915_fig2_Figure-2-Leave-copy-down

3.https://www.researchgate.net/figure/287723915_fig3_Figure-3-Probabilistic-Cache

4. https://www.researchgate.net/figure/287723915_fig4_Figure-4-ProbCache-and-cache-replacement-policies

5. https://www.researchgate.net/figure/287723915_fig5_Figure-5-LCD-and-cache-replacement-policies

6. https://www.researchgate.net/figure/287723915_fig6_Figure-6-LCE-and-cache-replacement-policies

7. https://omnetpp.org/intro

8.M. Arifuzzaman