IMPROVING EFFICIENCY OF INFORMATION RETREIVAL IN CONTENT CENTERED NETWORKS USING DATABASES

By

ASHWIN KUMAR THANDAPANI KUMARASAMY

Bachelor of Technology in Information Technology

Anna University

Chennai, Tamil Nadu, India

2010

Submitted to the Faculty of the
GraduateCollege of the
OklahomaStateUniversity
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
July, 2013

IMPROVING EFFICIENCY OF INFORMATION

RETRIVAL IN CONTENT CENTRIC NETWORKS

USING DATABASES

Thesis Approved:

Dr. Johnson Thomas

ThesisAdviser

Dr. Blayne Mayfield

Dr. Subhash Kak

Name: ASHWIN KUMAR THANDAPANI KUMARASAMY

Date of Degree: JULY, 2013

Title of Study: IMPROVING EFFICIENCY OF INFORMATION RETRIVAL IN CONTENT CENTRIC NETWORKS USING DATABASES

Major Field: COMPUTER SCIENCE

Abstract: Content centred Networking is a new networking architecture designed to work with existing network architecture and protocols. The emphasis is moved from the location of the data to its name. Content centred networks can work alongside TCP/IP or independently. They are capable of routing named pieces of content at the packet level. It supports application neutral caching that result in more efficient content delivery whenever needed. In spite of these advantages, content centered networks have some drawbacks too. All the data that is stored in cache and processed are in the form of flat files. When a user requires only specific data, the entire file has to be processed. This file processing is less efficient than a database management system in many ways. We have integrated an existing database management system to the cache. Specific parts of the cached data can be obtained by querying the database instead of processing the entire file, which improves efficiency in retrieving information of our interest more effectively. Instead of a relational database management system, graph based databases are used to interface with content centric networks because of their ability to quickly retrieve data that are inter-related, less restrictions on schema and the stored data can be of any type.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

1.1 Overview of Content Centered Networking

The basic principle involved in Content Centered Networking is allowing users to focus on data which they need rather referring to a specific location from which data can be retrieved. This arises from a content distribution network in which content is distributed to a large number of users from a single source. This amounts to around 80% of the current internet usage [1]. Content centered networking allows the caching of information at the intermediate routers which reduces congestion, increases delivery speed and enhances security at the data level [8]. An existing normal communication network is depicted in Figure 1.

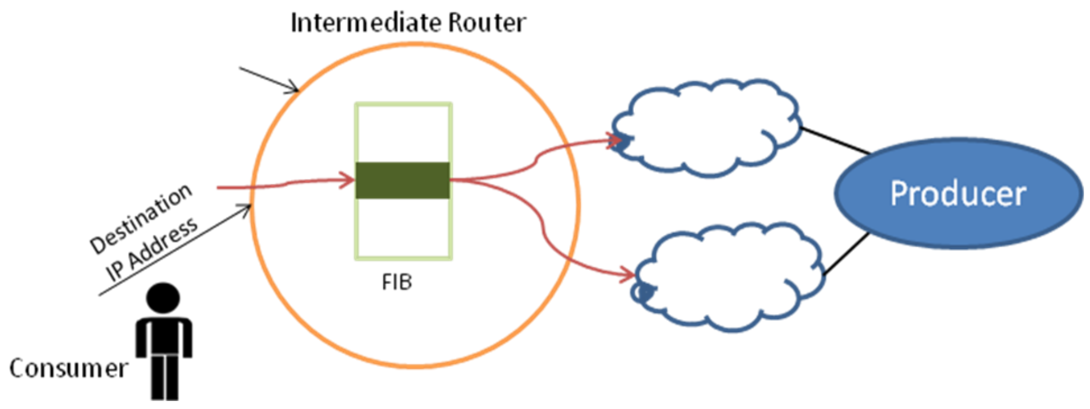


Figure 1: Communication Network

In a communication network, the request from a user (consumer) has the IP address from which the data can be fetched. This destination IP address is used by the intermediate routers to route the packet in the network [4]. The destination IP is matched with the Forwarding Information Base (FIB) at each router. The FIB acts like a dictionary, by storing information regarding where to route the packets for a corresponding IP address.

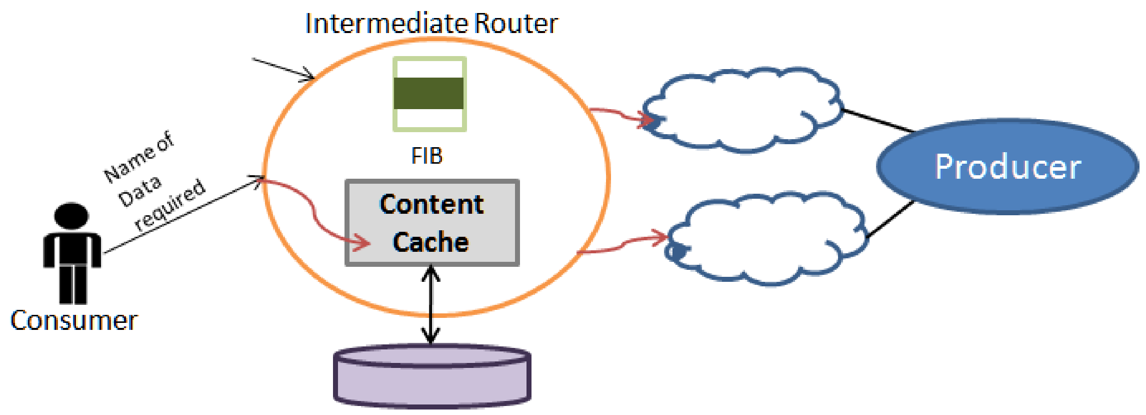


Figure 2: Content Distribution Network

In a content distribution network instead of passing the destination IP address, the content name is passed to the router (Figure 2). This router checks for the information in the content cache. If it is available then it sends the required information to the user. If not then it sends a broadcast message to the nearby routers with the same content name [1]. This continues until it hits the server or a router that has the corresponding data. Then the data is returned to the consumer. The data is cached in all the intermediate routers. Thus when a future requests arises, they can be handled more quickly.

Figure 3 shows the overall working of content centric networks. For example, consider a number of users wanting to read the “Daily O’Collegian” newspaper. Consider Client 1 initiating the request for the first time, it travels up the tree till it gets the desired content or hits the server. Once the required content is obtained then the information is passed down the path in the tree to the consumer. At every router the content passed is stored in the cache to handle future requests [5].

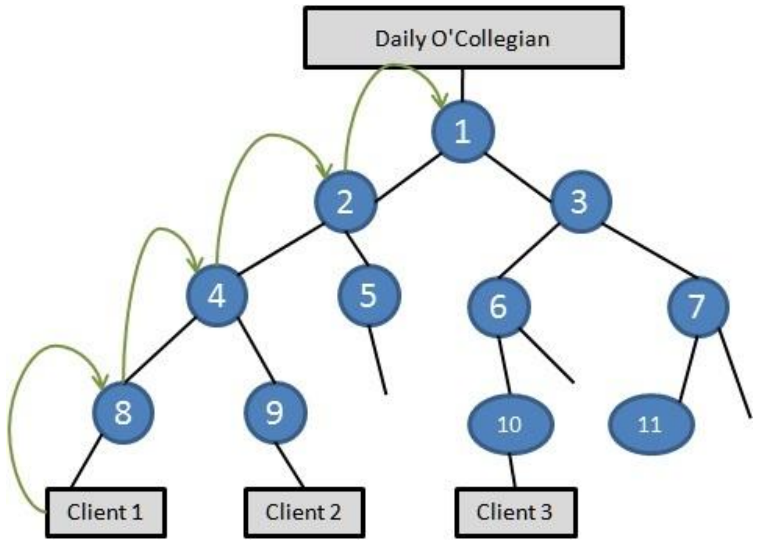


Figure 3: Working of Content centered Networks I

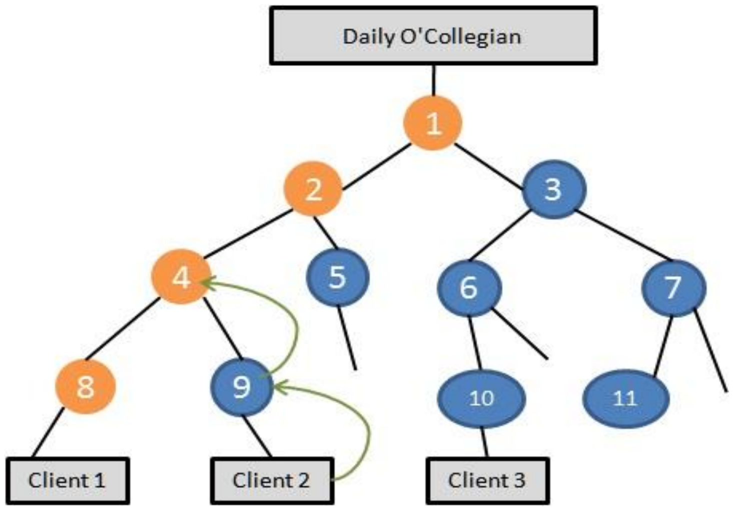


Figure 4: Working of Content centered Networks II

When client 2 requests for a copy of the “Daily O’Collegian”, the request is passed on to the routers until the content requested is found. Because of the previous request, the routers 1,2,4,8 have the content requested by client 2. Thus the data is obtained from router 4 instead of going all the way to the O’Collegian server again.

1.2 Problem Statement

Though content centered networks are efficient in information retrieval, there arises a problem when a user requests only for a part of the data. Even though the request is only for a part of the data, the entire data as a flat file is sent to the user. The flat file has to be processed at the user end to get the necessary information. This reduces the efficiency of information retrieval. To overcome this problem, there should be further enhancement in this area.

1.3 Research Objective

The main objective is to improve the efficiency of content centered networks, when the user requests for a part of data. To achieve this, an existing database management system is added to the cache that stores the content at the routers. Database queries are more efficient than processing a file [10]. Any part of available content can be retrieved very quickly and efficiently. Another research objective is to present data to the user from a query-based content.

1.3.1 Outline of Proposed Approach

In this thesis, Neo4J database is integrated with CCNx. CCNx based devices will be able to access NEO4J database through two commands namely ***ccnDBStore*** and ***ccnDBRetrieve.*** The data returned using ccnDBRetrieve is grouped based on a similarity measure and then presented to the user. This grouping involves in elimination of duplicates and clustering similar data.

1.4 Organization of Thesis

The thesis is organized into five chapters. Chapter 1 has the basic introduction to CCNx. In Chapter 2, the CCNx protocol is discussed in detail. This includes CCNx terminologies, CCNx repositories. Furthermore graph databases are also discussed in this chapter. Chapter 3 covers the idea of integrating a graph database to an existing CCNx system. It also includes the proposed system architecture and the implementation principle. Chapter 4 includes the software specifications and the proposed system specifications.

CHAPTER II

REVIEW OF RELATED LITERATURE

2.1 Distribution Networks over Communication Networks

There has been a tremendous increase in the use of internet and social networking, by which users create own content and share it with others. Various contents are hosted by several media servers. The only way to retrieve them is to establish an end-to-end communication with these corresponding servers. The usage pattern of internet has become content oriented, (i.e.) the users are just focused on the content rather than how and where to obtain it [1].

Internet was primarily designed to act as a communication network and not as a content distribution network. Content Centric networking is an approach to solve this limitation. In a Content centered network, the focus is shifted from the location to the data. This initiative was started by Xerox Palo Alto Research Center (PARC).

	Communication Network	Distribution Network
Naming	End Points	Data
Memory	Invisible	Explicit

Table 1: Difference between Communication and Distribution Networks [1]

In a communication network only endpoints can be named i.e. the source and destination, whereas the content that is of interest cannot be named. It is difficult to obtain a particular content by knowing what it is. In a content distribution network instead of the end points, the content of interest is named and thus it makes the retrieval process simpler.

Memory in a communication system is invisible. The routers that are in the route do not remember anything, they just route data. But in content distribution the intermediate routers store the data that they pass on to the next level router. In this way any future requests can be handled by these routers themselves [4].

2.2 Differences between TCP/IP Communication & CCN Distribution Model

Being a distribution model, content centric networks differ from the TCP/IP networks in the following aspects [5]:

- Receiver-Centric Communication Model - The receiver may be able to pull information only after sending an interest message. An interest message indicates that a receiver wants to receive a particular data. At most one content message is delivered in response to an interest message.

- Hierarchical content naming – In Content Centric Networks, instead of addressing specific hosts, the content is itself addressed. They are similar to Uniform Resource Locator (URS)’s. Interest packets are forwarded by longest prefix matching at forwarding decision phase.

- Cache and forwarding architecture – Every CCN device can cache data and use it to serve future requests.

Due to the above characteristics CCN is able to resolve the problems in today’s internet such as mobility, security, multi-path support and so on.

2.3 Overview of CCNx Protocol

Content sharing in a CCN takes place with the help of the CCNx protocol. This acts as a transport protocol for CCN. The main aim of the CCNx protocol is to deliver named data instead of making connections between hosts [2]. Every packet of data is cached at any CCNx router. These routers support broadcast or multicast, leading to efficient usage of the network when many people are in need of the same information.

The CCNx protocol provides location independent delivery services for named data. This includes multi-hop forwarding for end-to-end delivery, flow control, transparent and automatic multicast delivery using buffer storage available in the network, loop-free multipath forwarding, and verification of content integrity and carriage of arbitrary application data [3].

Applications that run on CCNx protocol over some low-layer communications service capable of transmitting packets. There are no restrictions on the nature of the low-layer service. This can be a physical transport or another network or transport protocol. For example, the applications that run on the top of CCNx protocol over UDP take advantage of existing IP connectivity [3]. As content is named independent of location in CCNx protocol, it may be preserved indefinitely in the network, providing effective form of distributed file system service.

The CCNx protocol supports a wide range of network applications. Apart from distribution of files and videos, CCNx provides support in real-time communication and discovery protocols, which is general enough to carry conversations between hosts such as TCP connections. The CCNx protocol leaves the naming convention application specific. CCNx generally specifies common functions that are independent of the content names and semantics of exchanges. In addition to CCNx, there must be some specifications for naming, data formats and message semantics. Such specifications of application protocols on the top of CCNx are called profiles [3].

The CCNx protocol is generally designed for end-to-end communication and hence it should be integrated into application processing rather than being implemented as a separate layer.

2.3.1 Terminologies in CCNx

The following are some of the important terminologies that are used in CCNx

Terminology	Description
Node	A CCNx network entity implementing forwarding and buffering
Party	Any entity in the network using CCNx protocol to communicate
Message	A CCNx packet. A lower-layer packet (e.g. UDP) may have more than one CCNx message.

Table 2: Terminologies in CCNx [3]

2.3.2 Message formats and Encodings

The CCNx protocol does not have fixed length fields. CCNx data formats are defined by XML schemas and encoded with explicit field boundaries. This permits field values of arbitrary length, optional fields and nested structures. The use of XML does not mean that the values in XML are string and are in human readable form. Most fields have arbitrary binary values. This includes even the fields that identify the content.

The wire format of CCNx messages is an efficient binary encoding of XML structures called ccnb [3], defining things such as byte order. There is also text XML encoding that is used for debugging, testing and presentation, etc; but not used on the wire.

2.3.3 Content Identification

In the CCNx protocol, content transfer is done by the name of content, irrespective of its location. CCNx content names are hierarchically-structured consisting of a number of components. These hierarchic structures correspond to an IP address, but they are of arbitrary lengths, and are explicitly identified. The protocol entirely relies upon these names, which may contain encrypted binary data. CCNx content names are not interpreted during the process, but matched. All the assignments of meaning to names come from application, institution, and/or global conventions reflected in prefix forwarding rules.

A CCNx name sometimes identifies a specific chunk of data, but in other cases identifies a collection of data by naming a point in the name tree under which there may be multiple pieces of data [3]. A name identifying a collection of data is analogous to the address of a network in host addressing scheme of IP, where network address can be seen to identify the collection of hosts attached to that network. In naming a collection of data, the CCNx name is a prefix of the name of every piece of content in the collection. Hence CCNx name is referred to name prefix or prefix.

CCNx name can be represented using URI representation or using XML. A name element represents a hierarchical name for CCNx content. It has a sequence of component elements. Each component may contain zero to many bytes individually.

2.3.4 URI Representation

It is preferred and more convenient to use Universal Resource Identifier (URI) to represent a CCNx name. The scheme identifier for CCNx scheme is “ccnx”, and ccnx URI need not have the authority component.

For example a HTTP URI: http://www.ccnx.org/releases/latest/doc/technical/Name.html can be represented in ccnx URI as ccnx://releases/latest/doc/technical/Name.html

2.3.5 CCNx Message Types

The CCNx protocol is based entirely on two packet types namely Interest and Data packet [5].

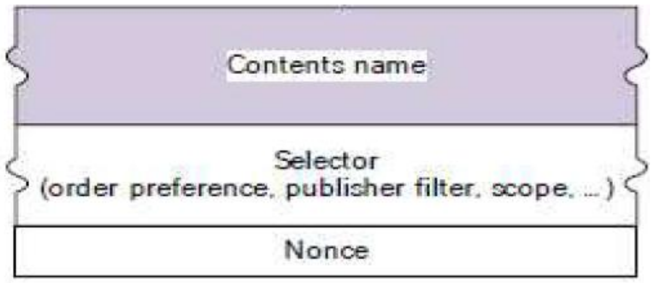


Figure 5: Interest Packet [5]

The interest message [5] is used to request data by its name. The most important field from these packets is content name. An example for interest packet is shown in figure 5. An interest message is used to identify a chunk of content to retrieve very specifically. Apart from that an interest message can provide a name prefix and other qualifications to restrict what data is acceptable from the collection named by the prefix.

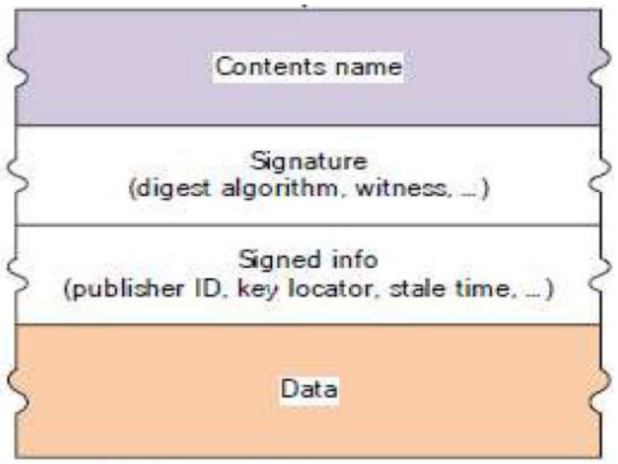

Contents name
Signature
(digest algorithm, witness, ...)
Signed info
(publisher ID, key locator, stale time, ...)
Data

Figure 6: Data Packet [5]

The data packet [5] is as shown in figure 6. It is also referred as content object. The main aim of this packet is to supply the required data. It not only contains the data payload but also the content name, a cryptographic signature and identification of the signer (publisher). Formally, a data packet is an immutable binding of name, a publisher and a chunk of data. Every packet has a valid signature. In this manner all the data in CCNx is attested.

Any CCNx party can verify the signature on any content object message that it receives. Applications that use this data must perform the verification process first. This process may require public keys which are not included in the data packet. A CCNx party discards packets that fail the verification process.

The CCNx protocol does not have any separate scheme for key distribution, as they can be distributed like any other data using general features of protocol. Profiles for key distribution are separate specifications, not a part of the CCNx protocol itself [5]. Signature verification can confirm whether a content object has not been corrupted in transit, since it was originally signed and that it was signed by the identified publisher. However verifications are not the same as determining that the publisher in question is a trustworthy source of data for a particular application purpose. CCNx parties use trust management practices to decide whether to trust data from a particular party. The CCNx protocol is designed to ensure the attestation of data, without constraining decisions about key distribution and trust management.

2.3.6 CCNx Node Model

A full CCNx node contains the following to provide buffering/caching and loop-free forwarding [5].

2.3.6.1 Content Store (CS)

A buffer memory is organized for retrieval by prefix match lookup on names. Since CCNx Content object messages are self-identifying and self-authenticating, each one is potentially useful to many customers. The content store needs to implement a replacement policy that maximizes the possibility of reuse such as Least Recently Used (LRU) or Least Frequently Used (LFU). The content store may retain the content objects indefinitely; CS is a cache and not a persistent storage.

2.3.6.2 Face

A face is a generalization of the concept of interface. A face may be a connection to a network or directly to an application party. A face may be configured to send and receive broadcast or multicast packets on a particular network interface, or to send and receive packets using point-to-point addressing in underlying transport, or using a tunnel. A face may also be the connection to a single application process running on the same machine. All messages arrive through the face are sent out through a face.

2.3.6.3 Forwarding Information Base (FIB)

The FIB is a table of outbound faces for interests, organized for retrieval by longest prefix lookup on names. Each prefix entry in the FIB may point to a list of faces rather than one.

2.3.6.4 Pending Interest Table (PIT)

A table of sources for unsatisfied interests, organized for retrieval by longest prefix match lookup on names. Each entry in the PIT may point to the list of sources. Entries in the PIT must timeout rather than being held indefinitely.

2.3.7 Message processing in CCN

The interest and data packets are processed in a different manner and are illustrated below.

2.3.7.1 Processing Interest Messages

An interest message is processed in the following sequence [3]

1. Lookup is performed on CS. If a matching content object is found, it is transmitted out at the arrival face as a response to the interest message. To match a content object must satisfy all the given specifications in given interest message. There may be multiple content objects matching the given interest message in this case the specification in the interest message will be used to determine which object to return. When a match is found CS processing stops and interest message is discarded as it is satisfied.

2. Lookup is performed on the PIT. If a matching interest message is found in the PIT it means the equivalent interest message has already been forwarded or is pending. The arrival face of new interest message is added to the list of sources of unsatisfied interests in a PIT entry and the interest message is discarded.

3. Lookup is performed in the FIB. If a matching prefix is found in the FIB, an entry is created in PIT identifying the arrival face of the interest message and the message is transmitted according to the strategy rules to one or more of the outbound faces registered for the prefix in FIB. One of the outbound faces may actually be connected to a local agent that uses the semantics of the name to dynamically configure new faces.

4. If there is no match in the previous steps then the node has no way to satisfy the interest message. It may be held for a short time before being discarded, since the creation of a new entry in the FIB may provide a way to satisfy it.

2.3.7.2 Processing Content Messages

A content object is processed as follows [3]

1. Lookup is performed on CS. If a matching content object is found, then the new content object is a duplicate and thus it can be discarded safely.

2. Lookup is performed on PIT. If there is a match in the PIT, the content Object is transmitted on all the source faces for the interest represented at the PIT. A node may perform verification of the Content Object before forwarding it, and may apply various policy restrictions.

2.4 CCNx Repositories

A repository in CCNx preserves the content and responds to the interests, which request the content it holds [9]. A repository uses a local file system for persistent storage of CCN content objects. A disk-resident index facilitates rapid start-up and limits the memory usage.

Interactions with the repository are through the ***ccnd*** command. The primary method of instructing a repository to obtain and preserve content is through the ***CCNx repository protocol***. A client (writer) wishing to store information in the repository expresses a specially constructed Interest, in the form of a "request for Interests" that triggers the repository to start retrieving content with a given name prefix that the writer provides. Retrieval of content is performed with standard CCNx Protocol features.

The CCNx Name Enumeration Protocol provides means for a CCN component to obtain a list of names under a specified prefix, like a directory listing. The CCNx Synchronization Protocol allows CCN components to define collections of content that reside in Repositories in multiple nodes and are to be kept in sync. The Repository utilizes embedded Synchronization facilities to synchronize copies of the collections automatically.

A Repository has a Policy that specifies the namespaces (name prefixes) for which it accepts and holds content. An initial Policy may be specified by configuration information; after that, Policies may be updated using the CCNx Repository Protocol like any other content. The Repository may be initiated with the ***ccnr*** command.

2.5 CCNx Repository protocol

The repository protocol provides a means to store the content objects by an application. A content storage request is represented as a CCNx interest with a command marker as a component of the name indicating the desired action. A response is represented as a content response data. There are three commands in CCNx repository protocol namely [9]

2.5.1 Start Write

This command requests the repository to retrieve and store content. It is constructed as an interest with a name, which is expressed as a part of an URI.

ccnx: (/ <component>) * %C1.R.sw/<nonce>

- Component - The components are required and represent the name prefix of the content
- <Nonce> - Nonce is required

2.5.1.1 Start Write Response

The implementation of start write is as shown in figure 7. The repository responds to a start write command issued by a client application or library with a content object of type INFO. The repository then starts fetching content from the writer using normal CCNx protocol messages. Retrieval is asynchronous. The elements that are included in the RepositoryInfo object are type (which is INFO), Version, Global Prefix Name (name under which repository stores its objects), local name (Repository).

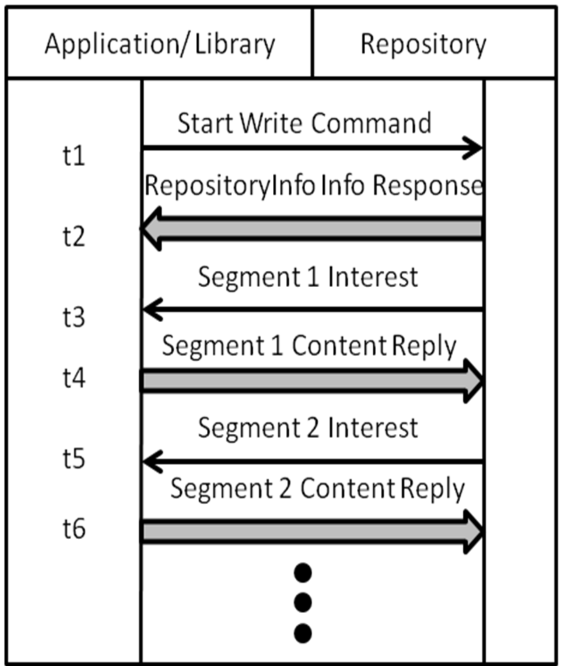


Figure 7: Start Write Implementation [9]

The single arrows in the figure indicate, interest messages and dashed arrows indicate content response.

2.5.2 Checked Start Write

Checked Start Write is similar to Start Write command, but the repository first checks whether it holds the content and does not retrieve and store if already present. The repository responds in either case, when the content is already present or not present. This command is expressed is as shown below when expressed as an URI.

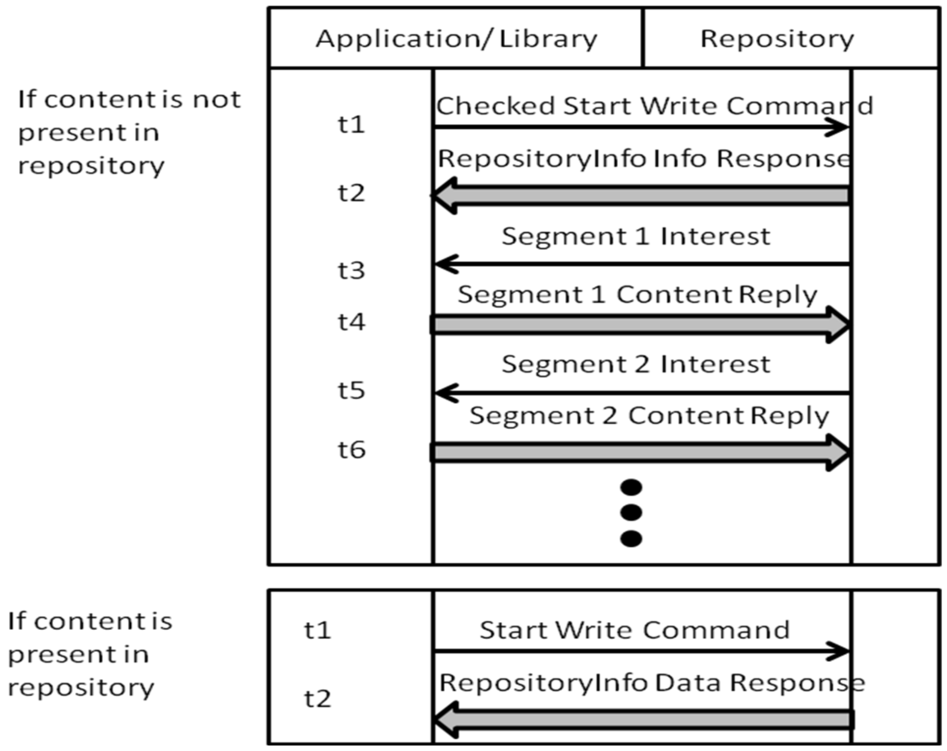
ccnx: (/<component>) */%C1.R.sw-C/<nonce>/<segment>/<digest>

- Components – Required, represents the name prefix of content to be stored.
- Nonce – Required
- Segment – Required, it is sequence component of first segment to be stored.

- Digest – required, it is SHA - 256 digest of entire ccnb – encoded content object.

2.5.2.1 Checked Start Write Response

Implementation of checked start write response is as shown in figure 8.



Application/ Library
Repository
If content is not present in repository
t1
Checked Start Write Command
RepositoryInfo Info Response
t2
Segment 1 Interest
t3
Segment 1 Content Reply
t4
Segment 2 Interest
t5
Segment 2 Content Reply
t6
If content is present in repository
t1
Start Write Command
RepositoryInfo Data Response
t2

Figure 8: Checked Start Write Implementation [9]

If the object is not present in the repository, the repository responds with a content object containing RepositoryInfo of the type "INFO", which is similar to start write. Te repository then begins fetching content from the user (writer) using normal CCNx protocol messages. Retrieval is asynchronous. Elements of RepositoryInfo object are type (which is INFO), version, Global Prefix Name (name under which the repository stores its own content), local name.

If the object is present in the repository, then the repository returns a content object of type "DATA". In this case the data returned is the complete name of the target of checked start wire and the repository does not fetch any data from the writer. If the object is present, the elements contained in RepositoryInfo object are type (which is “DATA”), repository version, global prefix name, local name, name (complete name of the target of checked start wire).

2.5.3 Bulk Import

Bulk import requests that the repository merge content objects from a local file into repository store. This is provided for backward compatibility. Bulk import is constructed as an interest with a single component name. When expressed in URI it is as shown below

ccnx:/ %C1.R.af~<file name>

File name – the ASCII name of a file that must exist within the import directory of the repository to which the content is being imported. It is a simple name. If the command is accepted then the repository attempts to open the file and then to parse it. If there are no errors, content objects in the file that are not already in the repository are imported, content objects that are in the repository already will be ignored and the file is deleted.

2.5.3.1 Bulk Import Response

Implementation of Bulk import is as shown in figure 9. The repository responds to the bulk import with a content object containing RepositoryInfo object of type “INFO”. Various elements of the returned content object are type (which is INFO), version, Global Prefix Name, Local Name, InfoString (reflects success or failure of the request).

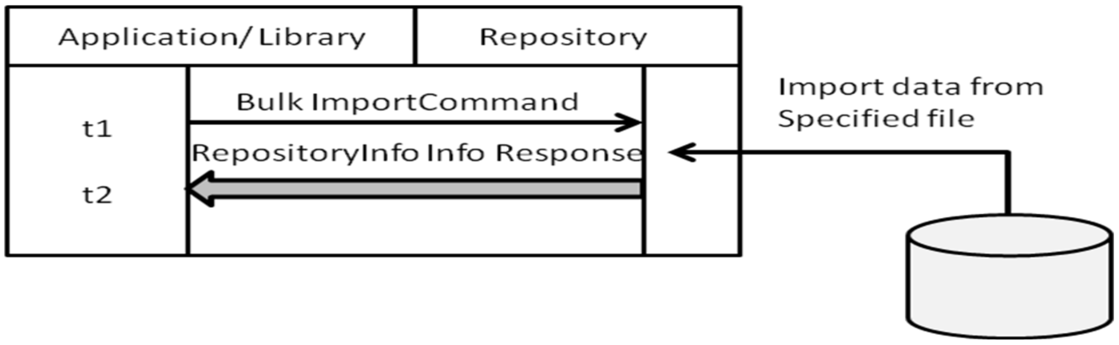


Figure 9: Bulk Import Implementation [9]

2.6 Fetching Repository Content

Standard interest is used to fetch content from the repository. The interest specifies a prefix, if it has a matching content; the repository returns it from its storage. The prefix is the only required component of interest, although other components may help in narrowing the selection.

Figure 10 explains how content from repository is fetched generally. If a matching object is present, the repository responds with the requested content object. If a matching object is not present, the repository does not respond at all. This protocol is illustrated below.

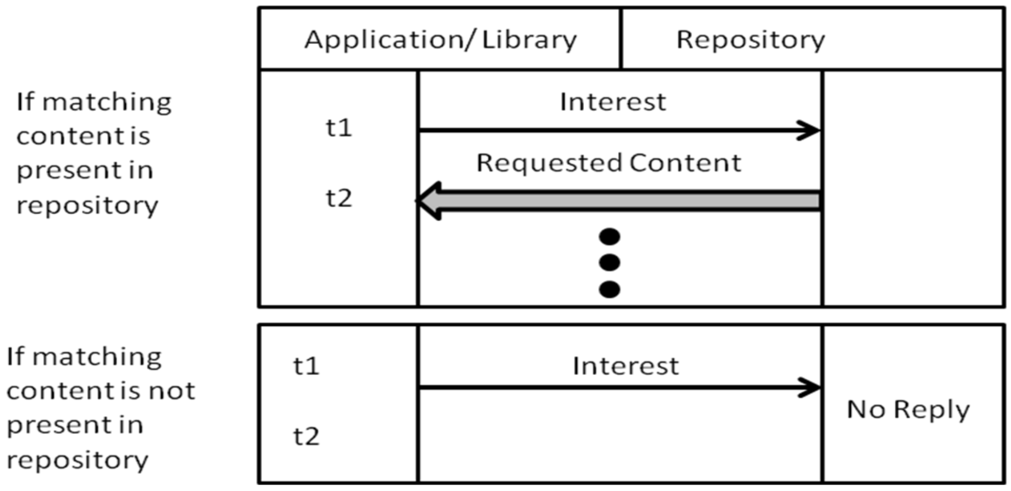

Application/ Library
Repository
If matching content is present in repository
t1
t2
Interest
Requested Content
If matching content is not present in repository
t1
t2
Interest
No Reply

Figure 10: Fetching Repository Content [9]

2.7 Database over flat file processing

There are several drawbacks in file processing when compared to a database management system [10]. Some of them are listed below.

1. Data redundancy and inconsistency: For file processing, the application programs are heavily dependent on the file format. Any changes in file format should be compensated with corresponding changes in the application program. Data can be duplicated in many places and leading to inconsistencies during updating or deleting of information.

2. Difficulty in accessing data: The application programs are very specific, slight changes in requirements would need a new application or change in existing ones.

3. Data isolation: As data can be present in many files, extracting them would be very difficult.

4. Integrity Issues: Some data are sensitive and have some constraints on them. If new constraints are added to the existing data, it is difficult to change the programs to enforce them.

5. Atomicity problems: Any transaction should be performed completely or not performed at all. A program might crash all of a sudden. This creates problems for a file processing system. Thus it is difficult to ensure atomicity in a conventional file processing system.

6. Concurrent Access Anomalies: Multiple users might try to run the application at the same time; hence the file can be accessed by many users. The data of one user can be over-written with another user. This makes data inconsistent.

7. Security issues: Not every user can access every data. Securing a part of data from the user is very tedious and complex in a file-processing system.

2.8 Graph Database

Graph databases comes under the category called NOSQL (Not Only SQL). Some examples are Google's BigTable, Facebook's Cassandra, and Neo4J.

2.8.1 Features of a Graph Database

A Graph database uses features of a graph such as nodes, edges to store and represent data [11]. In a graph database there are no indices. Each element in the graph has a pointer to its adjacent elements. This eliminates the overhead in index lookups.

Relational databases are generally efficient unless the data contains many relationships requiring joins of large tables [12]. Graph databases do not use join operations and hence they can associate large sets of data more effectively. The data can be mapped directly to object-oriented applications. Schema in a graph database is less rigid and can be modified more easily compared to a RDBMS.

2.8.2 Need for a Graph Database

After the creation of Relational Databases, it became the primary storage mechanism. RDBMS range from small-scale to large-scale. Some of them are Oracle, MySQL, Microsoft SQL Server, etc. With the increase in use of the internet as a tool for public data, data has grown enormously in terms of volume and interconnectedness [14]. Usually graphs are used to represent this interconnectedness. RDBMS can also handle the graph data, but they are inefficient in this process. Thus a database handling graphs more efficiently became more important.

2.8.3 NoSQL

NoSQL stands for Not Only SQL, these databases are high-volume data stores that actively reject the relational and object relational models. Atomicity, consistency, isolation and durability are the set of governing principles (ACID) for a relational model. The ACID properties ensure that the relational database is consistent. NoSQL doesn't follow these ACID properties. They are schema-free, easy to replicate, have simple API and are eventually consistent [13].

2.8.4 Neo4J

In this thesis, Neo4J a graph database will be investigated on how to be integrated into the CCNx devices. NEO4J is used for integration with CCNx instead of a traditional relational database management system because, real-time data is highly interconnected. Data are interdependent on each other forming a complex graph. RDBMS can handle these graphs but not as efficiently as NEO4J. NEO4J and other graph based databases are specifically designed to handle these graph based structures. The retrieval is much more efficient by the graph based databases in comparison with the relational databases.

CHAPTER III

IMPROVING EFFICIENCY IN INFORMATION RETRIEVAL

3.1 Integrating a DBMS with CCNx repository

To increase the efficiency of information retrieval an open source database management system (Neo4J) is integrated with the CCNx repository. When the user requires only part of data this approach helps a lot in eliminating the unnecessary file transmission and processing.

This approach aims in adding a DBMS to the existing CCNx implementation. Two separate commands are developed to facilitate the user to store the contents into a database and to retrieve as well. The contents of the database are not persistent like the content cache. The user is expected to know which data to store in database.

3.1.1 Architecture

The architecture of the proposed system which is implemented by integrating CCNx and a DBMS is shown below.

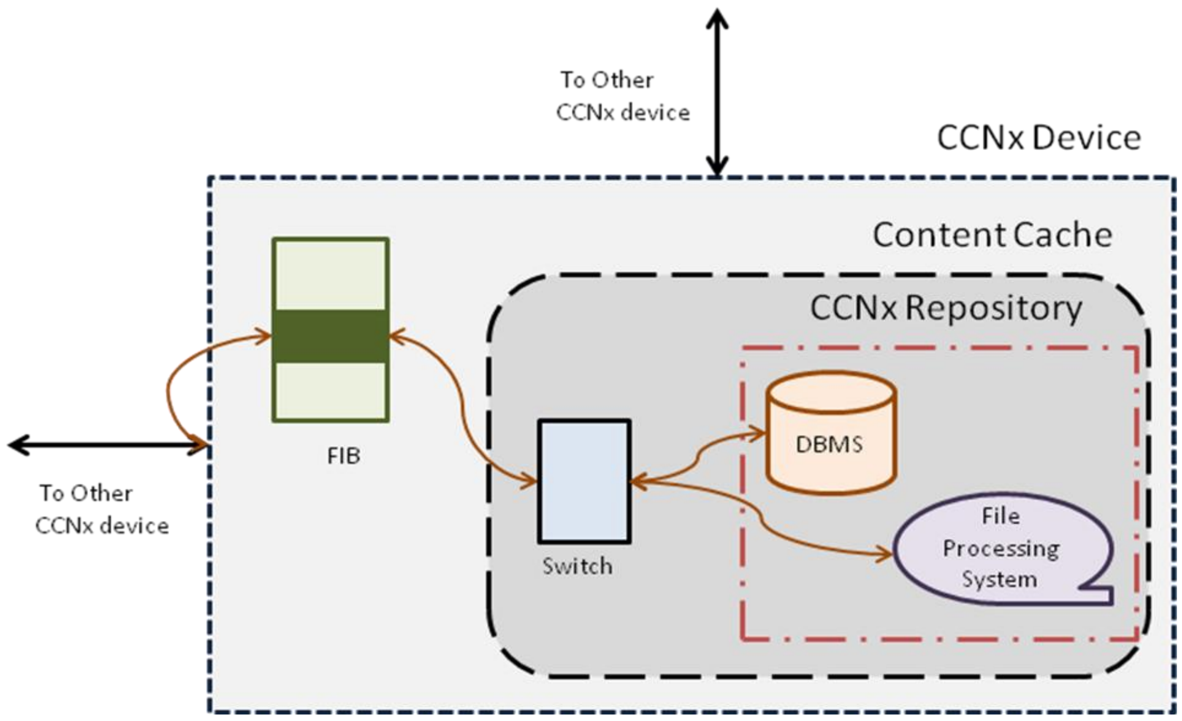


Figure 11: Proposed System Architecture

The architecture is similar to a normal CCNx device. The only difference is the addition of a DBMS to CCNx repository and a switch. The switch allows the user to decide which storage to select. Whenever the CCNx device receives an interest message, it checks with its FIB. If there is an entry in FIB, then it looks for the data in the cache. Based on the interest message, the content is searched either in the Database or File System.

Data is being searched either on database or on the file system. The CCNx devices differentiate this by the header content of interest message. If these interest messages are from the command ***ccnDBRetrieve,*** the contents of database are searched. Otherwise the file system is searched.

3.1.2 Implementation

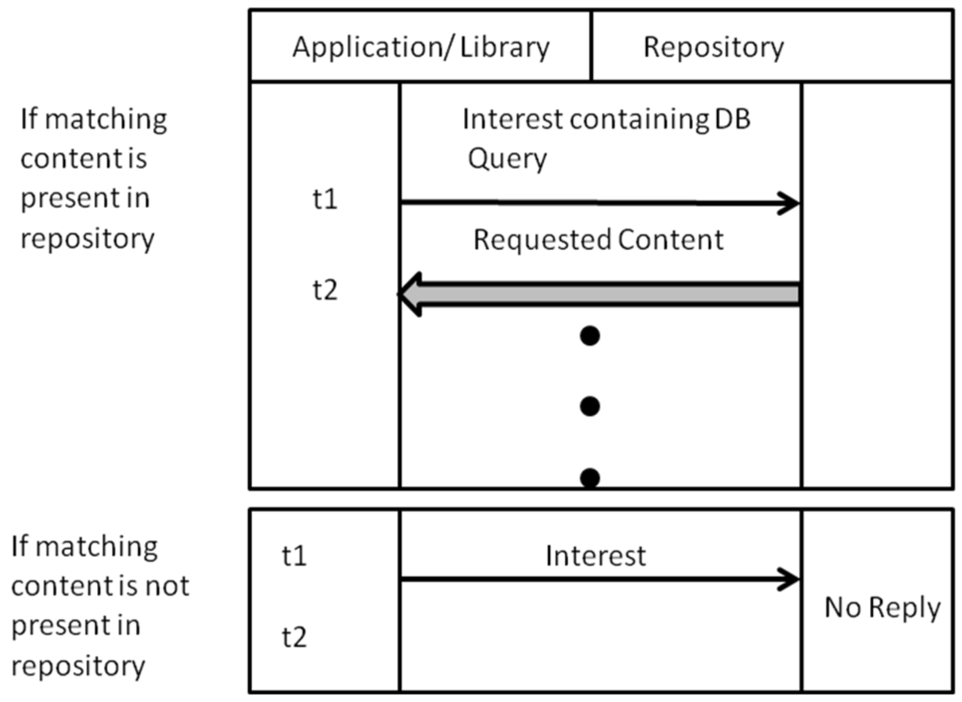

Application/ Library
Repository
If matching content is present in repository
Interest containing DB Query
t1
Requested Content
t2
If matching content is not present in repository
t1
Interest
t2
No Reply

Figure 12: Proposed System Working Principle

The repository receives the request as an interest message, containing a query (optional). The interest message requests for a content object. If a query is contained within the interest message and if it is available in the repository, then the query is processed by the DBMS and the corresponding content is returned to the user posting a query. If the interest message has a query and if the repository does not have the content object, then there would be no reply from the repository, the interest message will be discarded. If the interest message doesn't have a query, then it will be processed as described in section 2.6.

3.1.3 Presenting data to the user

The user can request the entire data; say today's news paper. Parts of it will be stored in different CCNx devices. The destination might receive duplicates of several parts. The destination CCNx device will have to eliminate these duplicates and to present the user with the original data.

When the search term given by the user is very general, many parts of different data will reach the destination CCNx device. In this case the destination must use an algorithm that group data based on a similarity measure. In this way different parts of data can be grouped and the user will be presented with the choice of data corresponding to his search term. The user will be able to select one of the choices from the returned list. When the search term is very specific, this whole process is not needed. In this case the various parts of the data need to be assembled together.

The search term from the user is in the form of a query or just data names. These queries will be in ***Cypher Query Language*** **(*CQL*)**. The Task of relating the queries to content-based context will be investigated, as future work.

3.2 Commands to Integrate DBMS with CCNx repository

To provide an interface between NEO4J (graph-based) database and CCNx two commands were developed. These commands are ***ccnDBStore*** and ***ccnDBRetrieve***. They are used to store data into the database and fetch data from the database respectively. The brief description of these commands is provided in the following sections.

3.2.1 ccnDBStore

This command facilitates user to store the data in NEO4J database. ccnBDStore acts as an interface between the user and the NEO4J database. This command allows the user to enter data in one of two ways. The user will be able to enter a plain text or contents of a file into database. All the data in CCNx are identified by data-names. Hence the user should provide the data name along with the corresponding option. The plain text is followed by the option or the file path is followed by the option.

3.2.2 ccnDBRetrieve

This command facilitates the user to retrieve the data stored in the NEO4J database. The data is retrieved using the data name. The user provides the data name to be searched for and constraints on the data. Based on the constraints the data is looked up in the local database. If there are any records corresponding to the data name then they are displayed to the user. If not the CCNx broadcast is made use of. The data name and constraint is sent to all the neighboring CCNx devices. If the data is available in those machines, the data is sent to the machine requesting it. Else the request is ignored. The requesting device receives the data and stores it so that future requests can be serviced in an efficient manner.

3.2.2.1 Similarity algorithm

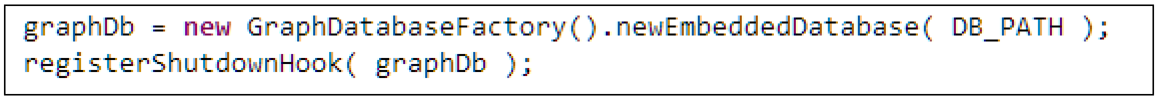
If the data name provided by the user is very general, it would result in data belonging to various categories. These data must be categorized and presented to the user for easy understandability. To serve this purpose a similarity algorithm is used. The similarity algorithm classifies the data names and categorizes them based on their relevance. The categories of the data names are then presented to the user. Then the user selects the category of interest. All the data names of the corresponding category are displayed. In case of more specific data names, classification is not necessary. Apart from this elimination of duplicate data is also necessary. The duplicate data names are eliminated. This prevents ambiguity and confusion.

3.3 Data storage format

The data is stored in a NEO4J database. The database is accessed by a JAVA program using an API. A separate database for CCNx is created at a specified path. The means of accessing this database by the JAVA program is discussed in the following sections.

3.3.1 Graph database service

To create a new database or to open a database, ***EmbeddedGraphDatabase*** [15]. ***EmbeddedGraphDatabase*** facilitates in creating an instance of graph database service. Only one instance of this service can be created for a database. This instance can be shared by multiple threads. Multiple instances of this service cannot be created on the same database. The graph database should be stopped at the end of the program or if it ends abruptly. To shut down the database service, the method ***shutdown ()*** is called

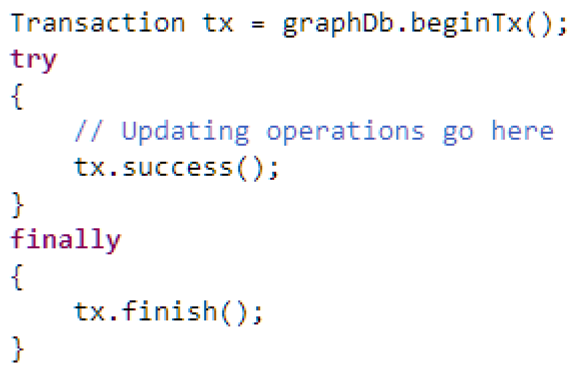


```
graphDb = new GraphDatabaseFactory().newEmbeddedDatabase( DB_PATH );
registerShutdownHook( graphDb );
```

Figure 13: Instantiating Graph Database Service

3.3.2 Graph database transaction

All the write operations that comprises of creation, deletion and updating operations must be performed in a transaction. This is because transaction demarcation is an important part of working with a real enterprise database. Transaction handling is simple in NEO4J as it involves the creation of a transaction object and bringing all the operations under it in a try block. The try and catch block is used to handle any exceptions in case the transaction fails.



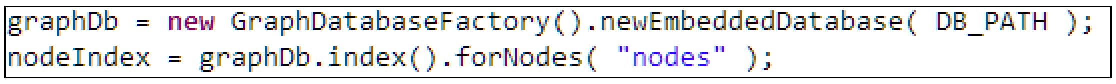
```
Transaction tx = graphDb.beginTx();
try
{
    // Updating operations go here
    tx.success();
}
finally
{
    tx.finish();
}
```

Figure 14: Transaction Structure

In figure 14, graphDb is an instantiation of a Graph database service. If the transaction code is successful then ***Transaction Success ()*** method is called. If there are any exceptions, the transaction is aborted. It is based on the all-or-nothing principle i.e. a transaction takes place completely or it doesn't take place at all.

3.3.3 Node Index

In a graph database all the records are stored in the form of nodes. This node structure makes the table structure more flexible and the properties of a node can have any type of values. Node Index is an instantiation of graph database service – index method.



```
graphDb = new GraphDatabaseFactory().newEmbeddedDatabase( DB_PATH );
nodeIndex = graphDb.index().forNodes( "nodes" );
```

Figure 15: Creating Node Index

Figure 15, depicts the creation of a Node Index from the graph database service. The index can be created for a nodes or relationships. Here the data nodes are indexed. Index in a graph database is similar to the index in a relational database. It enhances efficiency of the data retrieval process. Similar to the relational database, a property of a node can be indexed. In this case the data names are indexed. Hence the search process can be far more effective. An advantage of using indices in a graph database is that it supports regular expressions in finding required data names.

3.3.4 Properties of Nodes

In the process of integrating CCNx with NEO4J, all the data nodes are assumed to have certain properties. These properties are similar to attributes of an entity in a relational database. The nodes in NEO4J correspond to individual records. These nodes' attributes are referred to as properties. In this thesis the data nodes, are assumed to have two basic properties, namely data name and data. The property data name corresponds to the name of the corresponding data and the property data corresponds to the relevant data.

CHAPTER IV

SYSTEM IMPLEMENTATION

4.1 Tasks involved in building the proposed system

- Integrating Neo4J graph database into CCNx
- Accessing Neo4J through CCNx. This will be achieved by implementing two commands at the CCNx interface. These commands will be to enter and retrieve data from Neo4J.
- Grouping the returned data based on similarity measures.
- Translating CQL into content, that is, CQL query is translated into CCNx content for retrieval.

4.2 Software Specifications

For the further study, a stable version of CCNx release is used (CCNx 0.7.1). Java source code of this release is investigated to integrate with an open source database management system. The DBMS used here is Neo4J. Netbeans IDE is used to make modifications to the CCNx release.

4.3 ccnDBStore - Implementation

This command as described in the previous chapter (section 3.2.1) is used to enter data into NEO4J database. Implementation details and the command usage will be discussed in this section. ***ccnDBStore*** is implemented in JAVA. This command accepts data name, option, and data as command line arguments.

Validation of these arguments is done at the initial part of the program and any illegal usages are thrown as exceptions. The second part of the implementation creates a graph database service from ***EmbeddedGraphDatabase*** [15]. The database path specified during the graph database instantiation remains fixed for a CCNx device. Then the shutdown service is registered. A node index is created for the database to index the nodes based on the data names.

Based on the data from the user, a graph node is created and its properties are set. The properties of importance here are data name and data. Creation of the node and assigning values to the properties are done within a transaction, as all the writes to the database must be within a transaction. The new node is indexed based on the data name. Once the data is written into the database the shutdown method is called and the graph database service exits. Exceptions are thrown if any error occurs when the data is saved. They are handled accordingly by try-catch blocks.

4.4 ccnDBStore – Usage

Usage format for the command ***ccnDBStorage*** is as shown below:

ccnDBStore <options> <data-name> <data>/<file-path>

Any deviation from the above usage will result in an usage exception. The options parameter can take values “-F” or “-n”.

Options	Description
F	This option specifies that data is available in the specified file path
n	This option specifies the data is a plain string.

Table 3: ccnDBStore Options

If the option specified by the user is "-F", then the file path should follow the data name. If the file path is incorrect or the file does not exist then an exception is thrown. The contents of the file are stored as the value for the property data. The data name is specified by the user, when the command is called.

If the option specified by the user is "-n", it implies that the data is a plain string passed as a command line argument. This becomes the value for the property data. The data name is specified by the user.

Blank data names in both the cases will result in an exception, as it is mandatory and cannot be null.

4.5 ccnDBRetrieve – Implementation

This command as described in previous chapter (Section 3.2.2) is used to retrieve data from NEO4J database. Implementation details and the command usage will be discussed in this section. ***ccnDBRetrieve*** is implemented in JAVA. This command has no command line arguments.

In the first part of the implementation, ***ccnChatNet*** [16] is instantiated. This instantiation is crucially important for this command to function. After the instantiation, the user is expected to enter the data name and the constraints which he/she needs to be retrieved. Once the user enters the data name and constraints, the same is searched for in the local NEO4J database. If the data is found then it is presented to the user. Else, it is broadcasted to the nearby CCNx devices. When a CCNx device receives a request for a data, it checks its local NEO4J database and then it sends its local data to the device from which the request was handed over.

The second part of the implementation creates a graph database service from ***EmbeddedGraphDatabase*** [15]. The database path specified during the graph database instantiation remains fixed for a CCNx device. Then the shutdown service is registered. A node index is created for the database to index the nodes based on the data names.

If the data name is generic, then the similarity algorithm is invoked to group the data names into various categories. These categories are listed to the user and the user selects one among them. The selected category and its data are then displayed.

4.6 ccnDBRetrieve – Usage

Unlike ***ccnDBStore***, this command has no command line arguments. Once this command is invoked, it waits for the user to enter the data name and constraints.

4.7 NEO4J – Setup

NEO4J can be downloaded from http://www.neo4j.org/download, depending on the type of operating system. Then the service NEO4J must be started, using the command "***neo4j***

start". To be installed as a separate service, NEO4J must be installed using the install command and the service should be started.

Once the service is setup or neo4j is started, it is ready to use.

A separate NEO4J database is used in all CCNx devices. This database is used only by the commands used as an interface between CCNx and NEO4J. This path can vary from one device to another, depending on the access rights and permission.

4.8 Properties File

Properties files are used in the implementation of these commands. These files contain the NEO4J database path and the CCNx chat URI.

These properties can be changed according to need.

4.9 Similarity Algorithm Implementation

Similarity algorithm is used to classify the data names into categories. This is done when the search term is generic. This algorithm is used in the implementation of ***ccnDBRetrieve***.

Classification is done by analyzing the data names and finding out how many adjacent character pairs are contained in both the strings. The similarity measure is calculated by the following formula [17].

$$similarity(s1, s2) = \frac{2 \times |pairs(s1) \cap pairs(s2)|}{|pairs(s1)| + |pairs(s2)|}$$

Figure 16: Similarity Measure [17]

The value of this metric is between 0 and 1. If the similarity measure is greater than 0.5 or 50% then the data names are considered to belong to the same category. Thus the data is categorized and presented to the user.

CHAPTER V

SYSTEM EVALUATION

5.1 System Evaluation Tools

To evaluate the performance of the above system, a set of tools is used namely Cytoscape [18], Random Network Plug-in [19], Pesca plug-in [20].

5.1.1 Cytoscape [18]

Cytoscape is an open source software platform for visualizing complex networks and integrating these with any type of attribute data. This tool was originally developed to visualize molecular interaction networks and biological pathways. Even though it was basically developed for biological research, it is now a general platform for complex network analysis and visualization. There are three versions of Cytoscape namely Cytoscape 2.x, Cytoscape 3.x and Cytoscape.js. Cytoscape 2.x is a stable, production version of Cytoscape. Hence a release in Cytoscape 2.x is used to evaluate the model.

5.1.2 Random Network Plug-in [19]

This plug-in was developed for Cytoscape. It creates a random network for the given number of nodes. This plug-in is used to create a random network to evaluate the system. The graphs are created based on Erdos-Renyi Model [21]. The number of nodes and the probability of edge creation are specified by the user. A complex graph is created and every edge is independently created with the probability of edge creation.

5.1.3 Pesca 3.0 Plug-in [20]

This plug-in is used to compute possible shortest paths in a graph. This plug-in is used to find multi-shortest paths tree, multi-shortest paths, s-p cluster and connect isolated nodes. Multi-shortest paths tree is used to compute the tree of all the shortest paths from the selected node to all the other nodes in a network. Multi-shortest path is used to compute all shortest paths connecting two selected nodes. S-P cluster is used to compute all the shortest paths connecting a set of nodes (more than two). Connected isolated node is used when a node is to be connected to a sub-network. All the shortest paths are displayed in the results panel of Cytoscape tool. To evaluate the system developed in this thesis, Multi-shortest path is used. This provides a list of shortest path between two nodes and one among them is selected. This is used to compare the developed model with the existing CCNx and the communication network with the CCNx network integrated with DBMS.

5.2 Evaluation Metrics

There are two metrics that are considered to evaluate the system in this implementation namely hop count and network traffic. An efficient system must minimize network traffic.

5.2.1 Hop Count

Hop count refers to the number of intermediate nodes in its path from source to destination. A single hop is a portion of the path between sender and receiver. Each time a data packet passes to next device such as routers, gateways a hop occurs. Hop count is a measure of distance between the hosts.

5.2.2 Network Traffic

Network traffic refers to the amount of data flowing through a network. It is also referred as data traffic. In an efficient network the traffic should be optimum. High network traffic can cause congestion and packet drop. High traffic also deteriorates Quality of Service in a network. If the data traffic can be minimized in a network then the network can be utilized by every node in a more efficient manner.

5.3 Evaluation Assumptions

A random graph with 50 nodes and edge probability 0.5 is generated using random network plug-in. A node is picked from these 50 nodes randomly and is assumed to be the node that has the requested data. In this evaluation Node 7 is picked to be source node.

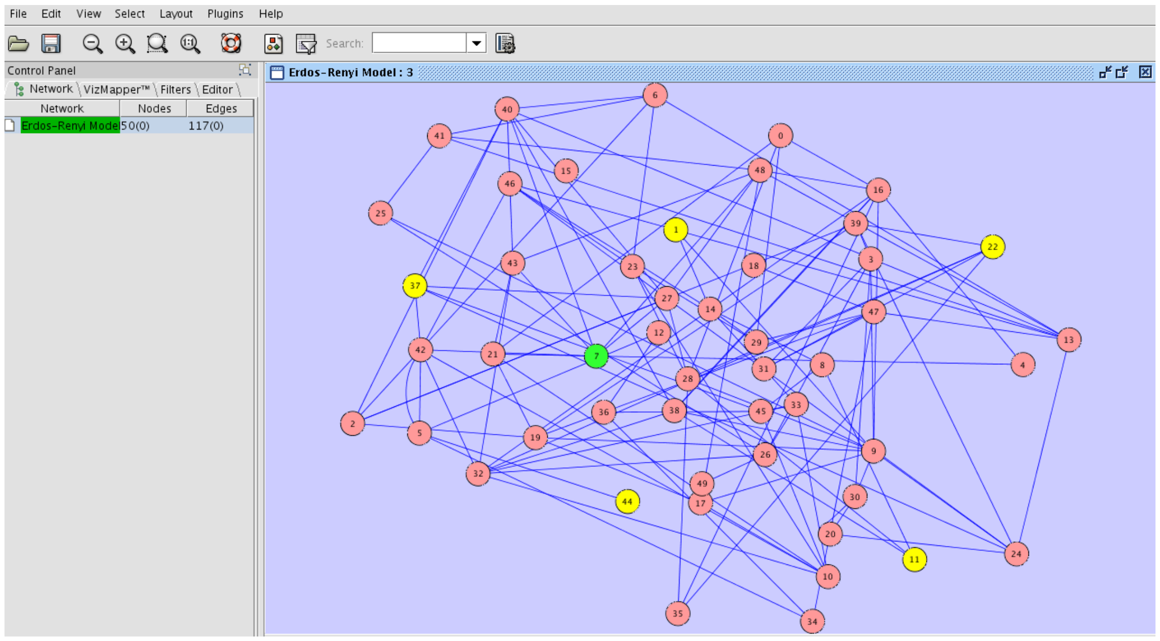

File Edit View Select Layout Plugins Help
Search:
Control Panel
Network VizMapper™ Filters Editor
Network Nodes Edges
Erdos-Renyi Model 50(0) 117(0)
Erdos-Renyi Model : 3

Figure 17: A Random Graph generated in Cytoscape

Figure 7 shows the random graph generated in Cytoscape tool. Node 7 is Green in color to indicate that it is the source node. There are two primary cases in the evaluation. The best case is when the requests are for the same resource and the worst case is that when the requests are for different resources or different parts of same resource.

It is assumed that there are requests from 5, 10 and 15 nodes in the network to demonstrate its efficiency. This evaluation is done in two cases namely best case and worst as described above.

5.4 System Evaluation – Best Case

In figure 17, Node 7 is the source that has the data. The nodes that are marked in yellow correspond to nodes that requests for the same data. Initially let's consider that 5 nodes request for data and later on it will be increased to 10 and 15.

Let the nodes requesting for data initially be Node 1, 11, 37, 22 and 44. The shortest path between them and the source is calculated by Pesca plug-in.

Node	Shortest Path
Node 1	1-9-38-40-7
Node 11	11-38-40-7
Node 37	37-40-7
Node 22	22-28-27-7
Node 44	44-5-42-7

Table 4: Shortest path for requests from 5 Nodes

In a normal communication network, the data packets are expected to take these shortest paths. Thus the number of hops in this case is 15 (4+3+2+3+3). Consider the resource transmitted to be a file of size 1MB. Then the total data transmitted in the network is 15MB i.e. network traffic is 15MB.

In a CCNx network, data is saved in the intermediate nodes unlike the communication network in which the data is just transmitted to its successor. This saved data in the intermediate nodes can be used to handle the same kind of request in the future. Consider Node 1 requests for data initially, then the request traverses to node 7 through nodes 9, 38 and 40. At the completion of the request all these nodes have the corresponding data so that they can handle any future requests for that data without contacting the source. Then node 11 requests for the same data and it traverse through node 38 to 40. Since node 40 already has the data, it sends the data. The same principle occurs for requests from nodes 37, 22 and 44. The final hop count for a CCNx network is 12 (4+1+1+3+3). Hence the network traffic is 12*1 MB = 12 MB.

In CCNx integrated with a DBMS, the hop counts are same as that of CCNx when the nodes request for the same data or same parts of data. Let us assume that the nodes are requesting for the same part of the data which is 1/5th of the file size. Thus the network traffic is 2.4 MB.

Now let 10 nodes request for the data. Let these nodes be 29, 8, 1, 11, 18, 37, 31, 22, 44 and 49. This is pictorially depicted as shown below.

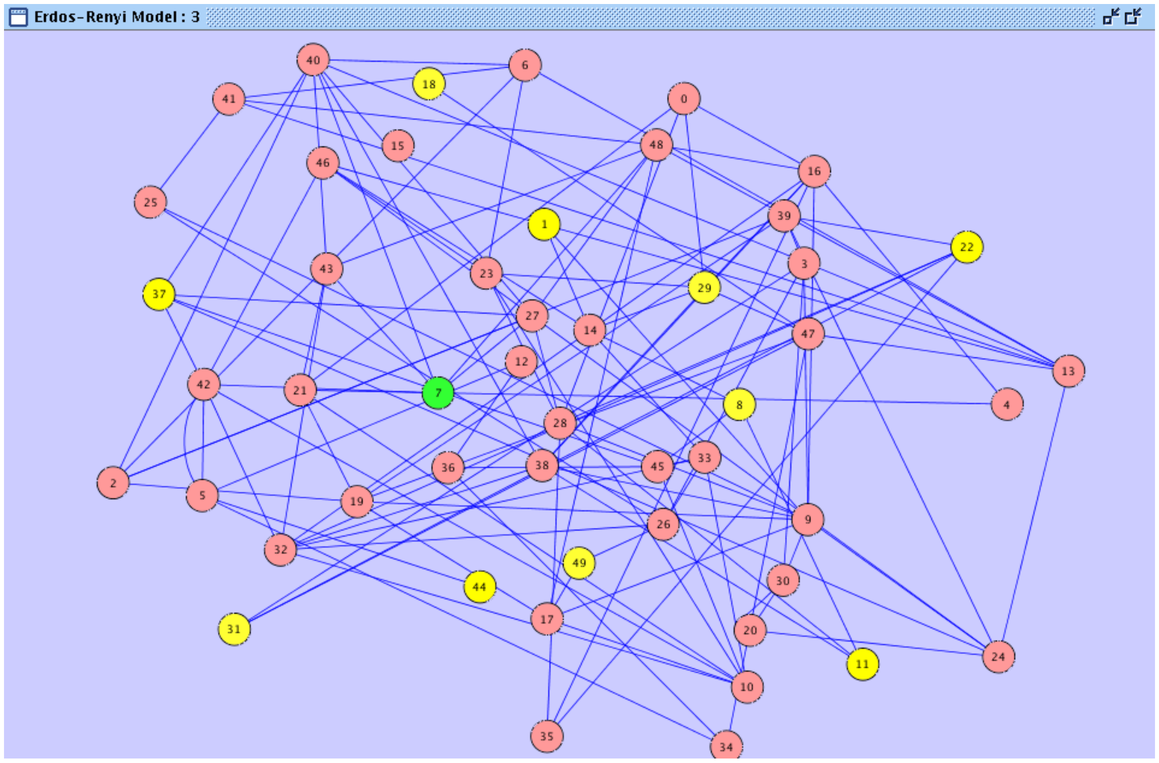


Figure 18: Request from 10 Nodes

Shortest path calculated for these 10 nodes by Pesca plug-in is shown below in Table 5. In a normal communication network, the data packets are expected to take these shortest paths. Thus the number of hops in this case is 34 (4+4+4+3+4+2+4+3+3+3). Consider the resource transmitted to be a file of size 1MB. Then the total data transmitted in the network is 34MB i.e. network traffic is 34MB.

Node	Shortest Path
Node 29	29-47-13-40-7
Node 8	8-11-38-40-7
Node 1	1-9-38-40-7
Node 11	11-38-40-7
Node 18	18-47-13-48-7
Node 37	37-40-7
Node 31	31-22-28-27-7
Node 22	22-28-27-7
Node 44	44-5-42-7
Node 49	49-17-48-7

Table 5: Shortest path for requests from 10 Nodes

In a CCNx network, data is saved in the intermediate nodes. Consider Node 29 requests for data initially, then the request traverses to node 7 through nodes 47, 13 and 40. At the completion of the request all these nodes have the corresponding data so that they can handle any future requests for that data without contacting the source. Then node 8 requests for the same data and it traverse through nodes 11, 38 and 40. Since node 40 already has the data, it sends the data. The same principle occurs for requests from nodes 1, 11, 18, 37, 31, 22, 44 and 49. The final hop count for a CCNx network is 21. Hence the network traffic is 21*1 MB = 21 MB.

In CCNx integrated with a DBMS, the hop counts are same as that of CCNx when the nodes request for the same data or same parts of data. Let us assume that the nodes are requesting for the same part of the data which is 1/5th of the file size. Thus the network traffic is 4.2 MB.

Now let 15 nodes request for the data. Let these nodes be 29, 8, 1, 11, 18, 37, 31, 22, 44, 49, 5, 14, 26, 34 and 41. This is pictorially depicted as shown below. Shortest path calculated for these 10 nodes by Pesca plug-in is shown below in Table 6.

In a normal communication network, the data packets are expected to take these shortest paths. Thus the number of hops in this case is 47. Consider the resource transmitted to be a file of size 1MB. Then the total data transmitted in the network is 47MB i.e. network traffic is 47MB. In a CCNx network, data is saved in the intermediate nodes. Consider Node 29 requests for data initially, then the request traverses to node 7 through nodes 47, 13 and 40. At the completion of the request all these nodes have the corresponding data so that they can handle any future requests for that data without contacting the source.

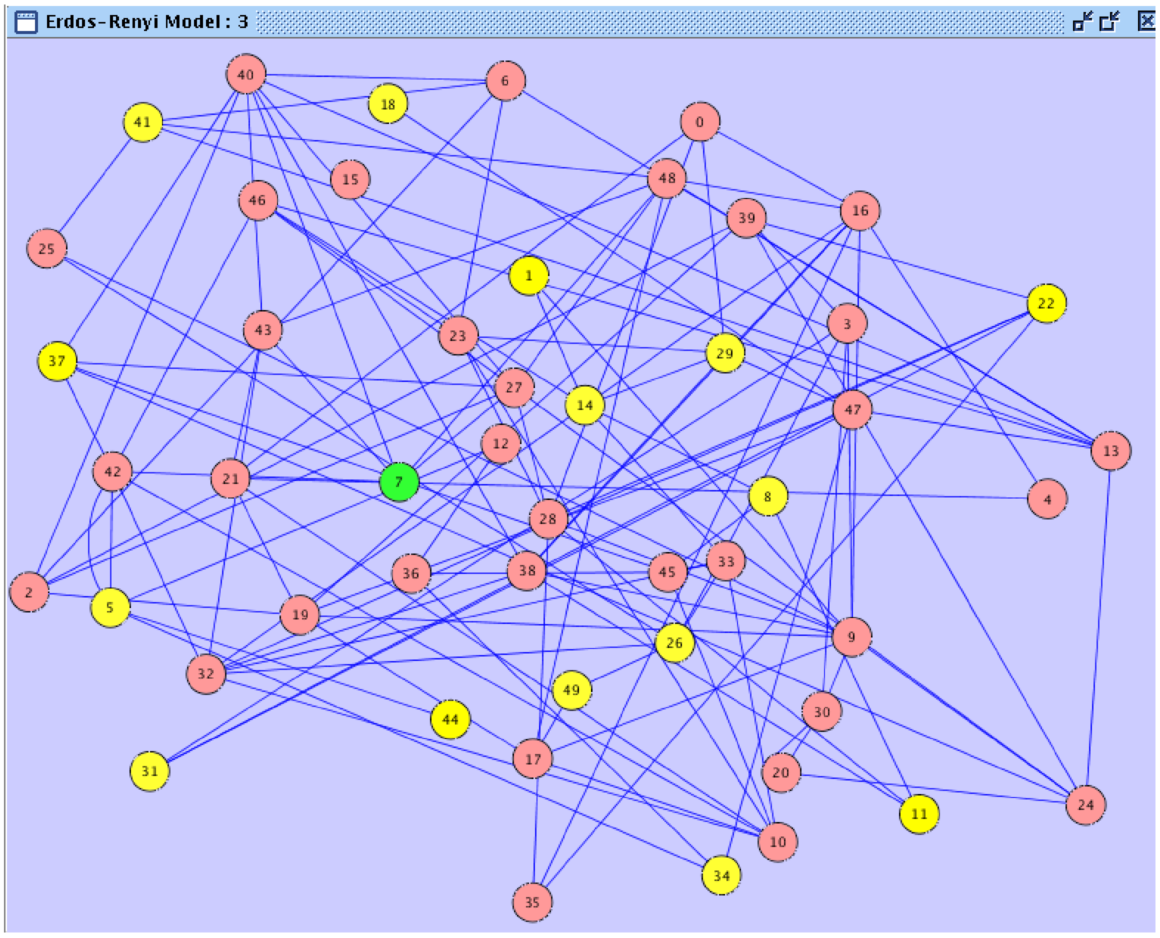


Figure 19: Request from 15 Nodes

Then node 8 requests for the same data and it traverse through nodes 11, 38 and 40. Since node 40 already has the data, it sends the data. The same principle occurs for requests from nodes 1, 11, 18, 37, 31, 22, 44 and 49. The final hop count for a CCNx network is 29. Hence the network traffic is 29*1 MB = 29 MB.

In CCNx integrated with a DBMS, the hop counts are same as that of CCNx when the nodes request for the same data or same parts of data.

Node	Shortest Path
Node 29	29-47-13-40-7
Node 8	8-11-38-40-7
Node 1	1-9-38-40-7
Node 11	11-38-40-7
Node 18	18-47-13-48-7
Node 37	37-40-7
Node 31	31-22-28-27-7
Node 22	22-28-27-7
Node 44	44-5-42-7
Node 49	49-17-48-7
Node 5	5-42-7
Node 14	14-46-42-7
Node 26	26-32-43-7
Node 34	34-5-42-7

Node 41	41-48-7

Table 6: Shortest path for requests from 15 Nodes

Let us assume that the nodes are requesting for the same part of the data which is 1/5th of the file size. Thus the network traffic is 5.8 MB.

Thus the total hop counts for requests from 5, 10 and 15 nodes in these three systems are as follows

# of Nodes Requesting data	Communication N/W	CCNx	CCNx integrated with DBMS
5	15	12	12
10	34	21	21
15	47	29	29

Table 7: Hop Count for Best Case

A graph plotted for the hop counts shown in Table 7. In this case the number of hop count in our proposed system is the same as that of the original CCNx. But the major difference is in the network traffic. In our proposed system, when the nodes request for part of data and not the whole information, the network traffic is reduced drastically in the system.

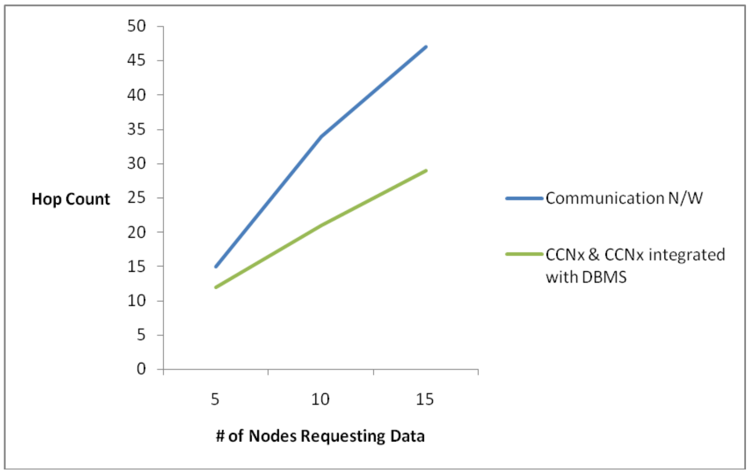


Figure 20: Hop Count plot for Best Case

Network traffic for three different networks is shown below in table 8.

# of Nodes Requesting data	Communication N/W (MB)	CCNx (MB)	CCNx integrated with DBMS (MB)
5	15	12	2.4
10	34	21	4.2
15	47	29	5.8

Table 8: Network Traffic for Best Case

A graph plotted for the network traffic shown in Table 8. Our system reduces network traffic drastically. Thus this system can be used when the user needs a part of data and not the whole file or data.

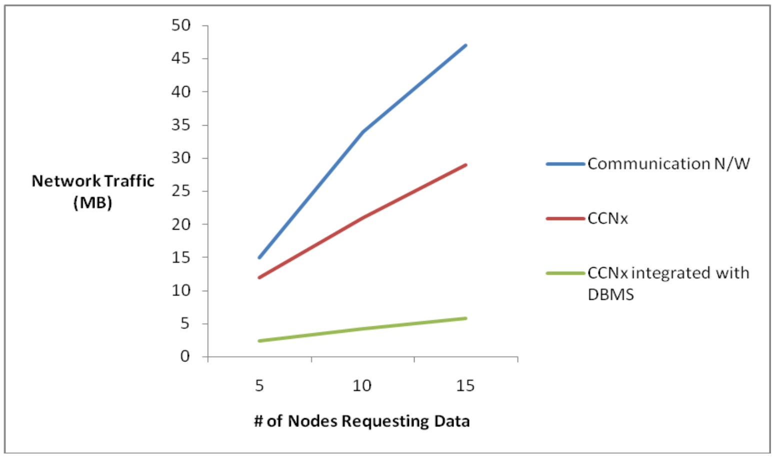


Figure 21: Network Traffic plot for Best Case

5.5 System Evaluation – Worst Case

In figure 17, Node 7 is the source that has the data. The nodes that are marked in yellow correspond to nodes that requests for the same data. Initially let's consider that 5 nodes request for data and later on it will be increased to 10 and 15.

Let the nodes requesting for data initially be Node 1, 11, 37, 22 and 44. The shortest path between them and the source is calculated by Pesca plug-in. These values are in Table 4.

In a normal communication network, the data packets are expected to take these shortest paths. Thus the number of hops in this case is 15 (4+3+2+3+3). Consider the resource transmitted to be a file of size 1MB. Then the total data transmitted in the network is 15MB i.e. network traffic is 15MB.

In a CCNx network, the hop count is same as the best case. Thus final hop count for a CCNx network is 12 (4+1+1+3+3). Hence the network traffic is 12*1 MB = 12 MB.

In CCNx integrated with a DBMS, the hop count varies from CCNx when every node request for the different parts of data. Let us assume that the nodes are requesting for the different part of the same data (1/5th of the file size). Let's assume Node 1 requests for 1st part of data, node 11 requests for 2nd part of data, node 37 requests for 3rd part of data, node 22 requests for 4th part of data and node 44 requests for 5th part of data. Requests are passed to the source if the intermediate node does not have the part of data requested. The hop count calculated is 15. Thus the network traffic is 15*200KB = 3MB.

Now let 10 nodes request for the data. Let these nodes be 29, 8, 1, 11, 18, 37, 31, 22, 44 and 49. This is pictorially depicted in figure 18. Shortest path calculated for these 10 nodes by Pesca plug-in is shown below in Table 5.

In a normal communication network, the data packets are expected to take these shortest paths. Thus the number of hops in this case is 34 (4+4+4+3+4+2+4+3+3+3). Consider the resource transmitted to be a file of size 1MB. Then the total data transmitted in the network is 34MB i.e. network traffic is 34MB.

In a CCNx network, data is saved in the intermediate nodes. Consider Node 29 requests for data initially, then the request traverses to node 7 through nodes 47, 13 and 40. At the completion of the request all these nodes have the corresponding data so that they can handle any future requests for that data without contacting the source. Then node 8 requests for the same data and it traverse through nodes 11, 38 and 40. Since node 40 already has the data, it sends the data. The same principle occurs for requests from nodes 1, 11, 18, 37, 31, 22, 44 and 49. The final hop count for a CCNx network is 21. Hence the network traffic is 21*1 MB = 21 MB.

In CCNx integrated with a DBMS, the hop count varies from CCNx when every node request for the different parts of data. Let us assume that the nodes are requesting for the different part of same data (1/5th of the file size). Let's assume Node 29 and 37 requests for 1st part of data, node 8 and 31 requests for 2nd part of data, node 1 and 22 requests for 3rd part of data, node 11 and 44 requests for 4th part of data and node 18 and 49 requests for 5th part of data. Requests are passed to the source if the intermediate node does not have the part of data requested. The hop count calculated is 32. Thus the network traffic is 32*200KB = 6.4MB.

Now let 15 nodes request for the data. Let these nodes be 29, 8, 1, 11, 18, 37, 31, 22, 44, 49, 5, 14, 26, 34 and 41. This is pictorially depicted as shown below. Shortest path calculated for these 10 nodes by Pesca plug-in is shown below in Table 6.

In a normal communication network, the data packets are expected to take these shortest paths. Thus the number of hops in this case is 47. Consider the resource transmitted to be a file of size 1MB. Then the total data transmitted in the network is 47MB i.e. network traffic is 47MB. In a CCNx network, data is saved in the intermediate nodes. Consider

Node 29 requests for data initially, then the request traverses to node 7 through nodes 47, 13 and 40. At the completion of the request all these nodes have the corresponding data so that they can handle any future requests for that data without contacting the source. Then node 8 requests for the same data and it traverse through nodes 11, 38 and 40. Since node 40 already has the data, it sends the data. The same principle occurs for requests from nodes 1, 11, 18, 37, 31, 22, 44 and 49. The final hop count for a CCNx network is 29. Hence the network traffic is 29*1 MB = 29 MB.

In CCNx integrated with a DBMS, the hop count varies from CCNx when every node request for different parts of the data. Let us assume that the nodes are requesting for the different part of same data (1/5th of the file size). Let's assume Node 29, 37 and 5 requests for 1st part of data, node 8, 31 and 14 requests for 2nd part of data, node 1, 22 and 26 requests for 3rd part of data, node 11, 44 and 34 requests for 4th part of data and node 18, 49 and 41 requests for 5th part of data. Requests are passed to the source if the intermediate node does not have the part of data requested. The hop count calculated is 43. Thus the network traffic is 43*200KB = 8.6MB.

Thus the total hop counts for requests from 5, 10 and 15 nodes in these three systems are as follows

# of Nodes Requesting data	Communication N/W	CCNx	CCNx integrated with DBMS
5	15	12	15
10	34	21	32
15	47	29	43

Table 9: Hop Count for Worst Case

A graph plotted for the hop counts shown in Table 9. In this case the number of hop count in the system built is greater than the CCNx. But the major difference is in the network traffic. In the system built, when the nodes request for part of data and not the whole information, the network traffic is reduced drastically in our proposed system.

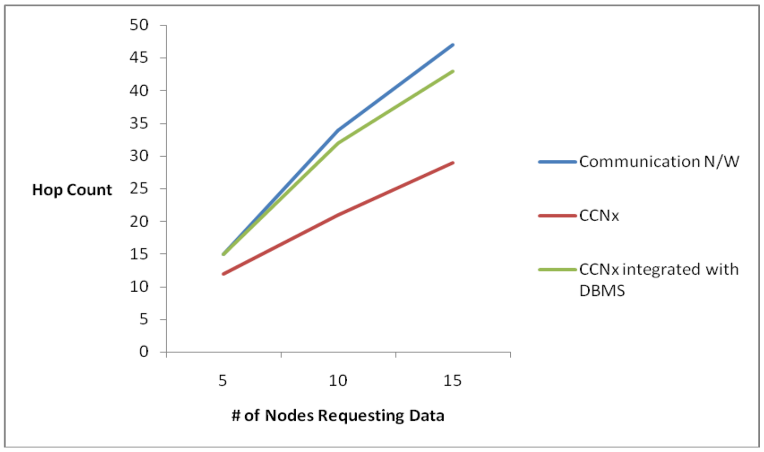


Figure 22: Hop Count plot for Worst Case

Network traffic for the three different networks is shown below in table 10.

# of Nodes Requesting data	Communication N/W (MB)	CCNx (MB)	CCNx integrated with DBMS (MB)
5	15	12	3.0
10	34	21	6.4
15	47	29	8.6

Table 10: Network Traffic for Worst Case

A graph plotted for network traffic shown in Table 10. Our system reduces network traffic drastically. Thus this system can be used when the user needs a part of data and not the whole file or data.

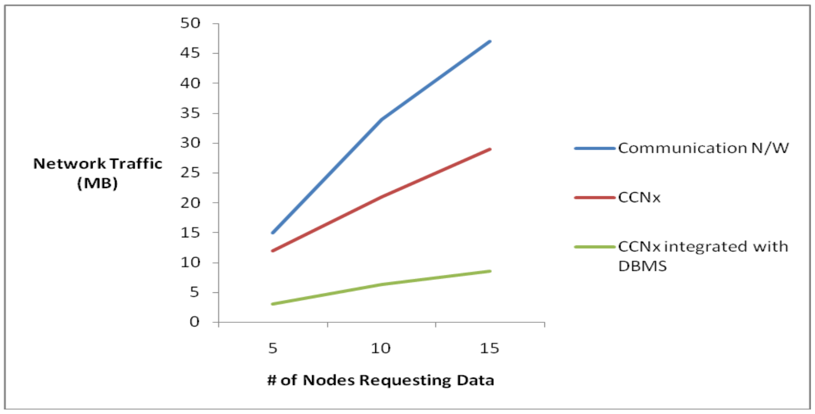


Figure 23: Network Traffic plot for Worst Case

5.6 System Performance

Our proposed system in both cases, that is, best case and worst case reduces the network traffic that prevails in a communication network and CCNx network. As a result of substantial reduction of traffic, congestion is avoided to a certain extent and thus packet drop is also avoided. This leads to more efficient utilization of the network and increases inefficiency of information retrieval of the CCNx networks.

CHAPTER VI

CONCLUSION

CCNx is an evolving technology. It has a lot of advantages such as

- content can be sent where there is an interest expressed to it
- content is sent in the shortest path
- content crosses any link at most once [1]
- average latency is reduced
- total bandwidth is minimized
- it can co-exist with the current network architecture

CCNx primarily is a form of distribution network, which is a lot more efficient than a communication network. A communication network is a subset of a distribution network. The reverse is not possible. In spite of the above advantages, the information retrieval efficiency is a question mark when the user requests only for a part of specified information. CCNx repositories generally use file processing systems, hence the efficiency of information retrieval is not very high. This thesis aims in optimizing the information retrieval efficiency by adding a database management system to existing CCNx repositories. By this the user can request any part of the data present in the repository. NEO4J is used instead of a relational database as the efficiency of retrieval is more for NEO4J for highly interdependent and interconnected data. Most of the real-time data are of this kind and they tend to form a complex graph. RDBMS will be able to process these graphs but not as efficient as graph databases like NEO4J. Apart from that the graph databases are schema less. Any type of data can be stored as properties of a node as there are no restrictions on schema.

6.1 Future Work

An expiration date can be added to data when it is sent from one system to another. With this expiration data, recent version of data can be identified from the old version. This prevents in presenting out-dated data to user. Support for Cypher Query Language (CQL) to fetch the data can be developed in the future to retrieve data from NEO4J database.

REFERENCES

[1] "A New Way to look at Networking", Van Jacobson's Google Tech Talk, http://www.youtube.com/watch?v=oCZMoY3q2uM. [03-03-2013]

[2] CCNx Introduction, http://www.ccnx.org . [03-03-2013]

[3] CCNx Protocol,http://www.ccnx.org/releases/latest/doc/technical/CCNxProtocol.html. [03-03-2013]

[4] Content-Centric Networking, PARC, http://www.parc.com/work/focus-area/content-centric-networking/. [03-10-2013].

[5] S.H. Shah,"Content Centric Networking and its application", ASM's international E-Journal of ongoing research in Management and IT, 2013

[6] Jim Kurose, "Technical perspective Content-Centric Networking", Communications of the ACM, Volume: 55, Issue: 1, Page: 116, 2012

[7] V Cerf and R Kahn,"A protocol for packet network interconnection" . IEEE Transactions on Communications Technology, Volume: 22, Issue: 5, Pages: 627–641, 1974 .

[8] Van Jacobson, "Content-Centric Networking", Computing Conversations" IEEE Volume:46,Issue: 1, Pages:11-13, 2013.

[9] CCNx Repositories, http://www.ccnx.org/releases/latest/doc/technical/RepoProtocol.html. [03-10-2013].

[10] Kamel et al, "The Federated Database Management System: An architecture of Distributed System for the 90's", Distributed Computing Systems, Pages: 346-352, 1990.

[11] Graph Database, http://en.wikipedia.org/wiki/Graph_database. [03-10-2013].

[12] Chad Vicknair et al, "A comparison of a Graph Database and a Relational Database – A Data Provenance Perspective", Department of Computer Science, University of Mississippi, http://www.cs.olemiss.edu/~ychen/publications/conference/vicknair_acmse10.pdf, 2010.

[13] NoSQL database, http://nosql-database.org/. [03-15-2013]

[14] R. Angles and C. Gutierrez, "Survey of graph database models". ACM, Volume 40, Issue: 1, Pages: 1–39, 2008.

[15] Embedded Graph Database API, http://components.neo4j.org/neo4j/2.0-SNAPSHOT/apidocs/org/neo4j/kernel/EmbeddedGraphDatabase.html. [07-01-2013].

[16] CCN Chat Net API, http://www.ccnx.org/releases/ccnx-0.4.0/apps/ccnChat/src/org/ccnx/ccn/apps/ccnchat/CCNChatNet.java. [07-01-2013].

[17] Similarity Algorithm by Matching adjacent character pairs, http://www.catalysoft.com/articles/StrikeAMatch.html. [07-01-2013].

[18] Cytoscape: An open source platform for complex network analysis and visualization, http://www.cytoscape.org/. [07-01-2013].

[19] Random Network Plug-in: https://sites.google.com/site/randomnetworkplugin/Home. [07-01-2013].

[20] Pesca 3.0 Beta Plug-in: http://profs.sci.univr.it/~scardoni/centiscape/pescadownload.php. [07-01-2013].

[21] Erdos-Reyni Model: http://www.renyi.hu/~p_erdos/1959-11.pdf. [07-01-2013].

VITA

ASHWIN KUMAR THANDAPANI KUMARASAMY

Candidate for the Degree of

Master of Science

Thesis: IMPROVING EFFICIENCY OF INFORMATION RETREIVAL IN CONTENT CENTERED NETWORKS USING DATABASES

Major Field: Computer Science

Biographical:

Education:

Completed the requirements for the Master of Science in Computer Science at Oklahoma State University, Stillwater, Oklahoma in July, 2013.

Completed the requirements for the Bachelor Technology in Information Technology at Anna University, Chennai, Tamil Nadu, India in 2010.