UNIVERSITY OF MINHO

MASTER THESIS

# Distributed Databases Synchronization in Named Data Delay Tolerant Networks


*Author:*

Chong Liu

*Supervisors:*

Professor Joaquim Macedo

Professor António Duarte Costa


*A thesis submitted in fulfillment of the requirements for the degree of Master of Computer Networks and Telematic Service*

*in the*

Department of Informatics

October 26, 2016

# Declaration of Authorship

I, Chong Liu, declare that this thesis titled, "Distributed Databases Synchronization in Named Data Delay Tolerant Networks" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

*"Thanks to my solid academic training, today I can write hundreds of words on virtually any topic without possessing a shred of information, which is how I got a good job in journalism."*

Dave Barry

UNIVERSITY OF MINHO

# *Abstract*

Department of Informatics

Master of Computer Networks and Telematic Service

**Distributed Databases Synchronization in Named Data Delay Tolerant Networks**

by Chong Liu

Delay Tolerant Network (DTN) is a small regional network designed to provide better communications when the end-to-end connection is not always possible. DTN is well known for intermittent connections and long delays. Nodes store data packets in the buffers and forward later when the connection is restored. Recently, Named Data Networking (NDN) has been drawing wide attention as a Future Internet architecture. This architecture shifts the emphasis from host to content and pays little attention to where is the content. Routing in NDN is based on the name of the content. Named Data-Delay Tolerant (ND-DT) network is an integration of DTN and NDN. It takes the advantages of both architectures by applying named data approach in DTN scenarios. In ND-DT network, distributed databases are maintained by a group of fixed or moving nodes. Data inconsistency always exists because of the intermittent connections and long delays. However, data synchronization solutions can minimize this inconsistency, helping to reduce the data access delay.

ChronoSync is a well-known NDN state synchronization protocol. Data synchronization in ND-DT networks are challenging because of the intermittent connections and the nodes' mobility. Moreover, the connection between nodes is not assured, which may make synchronization to fail. In this work, it is assumed that there is at least one path between each pair of database nodes.

The aim of this work is to improve the recovery process of ChronoSync in order to enhance its adaptability to ND-DT network scenarios. For this goal, ChronoSync and our improved solution were implemented and tested on an ND-DT network simulator.

The results show that our improved ChronoSync is more adaptable to ND-DT networks. The improved ChronoSync consumes less time to finish synchronization tasks in all the scenarios. What's more, in three database scenarios, IChronoSync decreasing about 83% of the synchronization time while Chronosync decreases 62% when changed from sparse network to dense network. What's more, improved ChronoSync generates 27% fewer data packets, which can increase the probability of other network nodes getting connected.

# *Acknowledgements*

This dissertation would not have been completed without the great support I have received from so many people during the two years of my Master study. I wish to offer my most heartfelt thanks to the following people.

First, I would like to express my deepest appreciation to my supervisors, Professor Joaquim Macedo and Professor António Duarte Costa. Thank you for the continuous advice and encouragement that allowed me to pursue research on this interesting topic. Thank you for spending a lot of time to discuss the research issues with me and give me suggestions on research directions, approaches and writing skills. Things would not go so smoothly without you.

Then I would like to thank Professor Alexandre Santos who gives me a lot of help during my Master study in University of Minho, Portugal. I would like to acknowledge the Department of Informatics where I learned computer networks for two years. I benefit greatly from the courses I took. I would like to thank my friends and my colleagues in Lab Room 307 who helped and colored my life in Portugal. We spent a lot of happy times together.

Finally, I would like to thank my parents, my sister for their unwavering support and understanding. Thank you for your constant encouragement and for tolerating all the days I spent in Portugal instead of spending time with you. I could never have finished a Master without your support and encouragement.

# Contents

# List of Figures

# List of Tables

# List of Pseudocode

# List of Abbreviations

| | |
|---|---|
| **CDNs** | **C**ontent **D**ata **N**etwork**s** |
| **DP** | **D**elivery **P**redictability |
| **DTN** | **D**elay **T**olerant **N**etwork |
| **FIB** | **F**orwarding **I**nformation **B**ase |
| **ICN** | **I**nformation **C**entric **N**etwork |
| **ICONE** | **I**nformation **C**entric **ONE** |
| **NCLs** | **N**etwork **C**entral **L**ocation**s** |
| **ND-DT** | **N**ame **D**ata-**D**elay **T**olerant |
| **NDN** | **N**ame **D**ata **N**etwork |
| **ONE** | **O**pportunistic **N**etwork **E**nvironment |
| **PIFP** | **P**robabilistic **I**nteres **F**orwarding **P**rotocol |
| **PIT** | **P**ending **I**nterest **T**able |
| **RW** | **R**andom **W**alk |
| **RWP** | **R**andom **W**ay**P**oint |
| **SNI** | **S**omething **N**ew **I**nterest |
| **TTL** | **T**ime **T**o **L**ive |

# Chapter 1

# Introduction

Delay Tolerant Network (DTN) is consensually considered a good architecture to support intermittent connections, significant delays and disruptions for end-to-end communications. The network infrastructure is usually not available in DTNs. The DTN devices, such as cell phones or PDAs, are equipped with wireless network interfaces and move irregularly. The mobility of devices makes the end-to-end path and network topology to change frequently. DTNs can transport data in two different ways: 1) over a wireless (or wired) network interface, 2) by the user move from location to location with the data stored in the device [1]. DTN supports a store-carry-and-forward strategy. Data units can be stored temporarily during the disconnection time and delivered later to the "next hop" when the network is reconnected.

Named Data Network (NDN) is one of most promising Information Centric Networking (ICN) architecture, shifting the emphasis from end-host to content. NDN model is totally different from TCP/IP based network. It concerns more about content and pays little attention to where is the content. It defines a special naming scheme and uses a prefix-matching method to search the content in the network. NDN consumers focus on the content which they are interested in, and broadcast their interest information to the entire network. NDN offers content storage functionality by two different components, named Content Store (cache content) and Repository (persistent content). NDN gives good support for mobility because there is no relationship between the content name and the content location.

Named Data Delay Tolerant (ND-DT) networks have emerged to meet the requirement of future networks: 1) data transportation depends on the content name rather than on the IP addresses of the hosts; 2) future networks must support delay tolerance and network disruption. ND-DT network is an integration of DTN and NDN architecture and takes the advantages offered by both architectures. It combines connectivity resilience (e.g. through store-carry-and-forward) and information resilience (e.g. through caching and replication) [2] from both architecture. ND-DT network uses Named Data on top of Delay Tolerant Network and applies named data approach in DTN scenarios.

Distributed Database is a logically connected database, consisting a plurality of physically dispersed computers connected by networks. Contents are stored on those computers and managed by a distributed database management system. As a result, the database will work as if all contents were stored on the same computer. The management system synchronizes all the data periodically to ensure that multiple users access the same data.

The update and/or delete operations must be reflected automatically in everywhere.

In an ND-DT network, the distributed database could be maintained by moving devices. In most cases, these devices have limited storage space and movement unpredictability. Most of the traditional distributed data replication and synchronization algorithms are focused on fixed topology and they are not suitable for this mobile situation. Nowadays, many popular applications, such as Dropbox and Google Drive, synchronize data with a centralized solution. Every change on the database should first upload to a central server and then the server distributes the change to the other nodes of the distributed databases. This centralized solution supports mobility but wastes bandwidth. Therefore, traditional data replication and synchronization algorithms mentioned above are not suitable for this new kind of network. Effective algorithms and solutions, which consider storage space and node mobility, are desperately demanded. This work is dedicated to finding new algorithms and solutions in order to benefit the ND-DT networks.

## 1.1 Objectives

In several scenarios, it's not possible to keep permanent connections between network users. These users (nodes) may move irregularly, which makes network topology unstable and complex. The irregular mobility of nodes introduces opportunistic connections and long delays, which makes the data transmission and information sharing more difficult. Several applications in this kinds of network are based on distributed files or databases that are maintained by various network nodes.

The lack of network infrastructure, decentralized data collection, limited resources (for example, bandwidth, energy, and storage capacity), geographic dispersion and high mobility of nodes make the centralized solution not suitable in such scenarios.

To overcome the mentioned problems, data sharing schemes of DTN are used. DTNs enable the transmission of data between nodes with an opportunistic contact. This achievement is supported by persistent storage of received packets, which can be delivered later to the destination node (or to a node closer to the destination).

However, the network applications paradigm is changing nowadays. Applications are more concerned with the data itself and do not care about the data location. So, if there is in-network stored named data, it can be accessed quicker without the need of original server(s) or even the use of Content Data Networks (CDNs). One of most promising Information Centric Networking (ICN) architecture is the NDN.

So, using Named Data on top of Delay Tolerant Networks seems to be a fruitful research direction. Nodes exchange information using intermittent connections and a set of mobile nodes (moving vehicles or walking pedestrian) maintains the database. Still, the existing solutions and algorithms are not suitable for this type of network. Therefore, algorithms and techniques for data replication, synchronization, and consistency checking in Named Data/Delay Tolerant Networks are needed.

This work aims to find solutions and algorithms that work well in ND-DT networks, as well as to evaluate solutions and algorithms for replication, synchronization, and consistency of these distributed databases. To accomplish this goal, it is required to implement and validate the solutions and algorithms on an ND-DT network simulator.

## 1.2 Main Contribution

The dissertation's main contribution is to implement an NDN state synchronization protocol, ChronoSync, for ND-DT networkN scenarios, and improve it to be more adapted to such scenarios.

ND-DT network nodes move uncertainty, which makes the network topology change frequently. Therefore, there is no sure of a path between a pair of database nodes. Data synchronization will fail when there is no path between the database nodes. To implement ChronoSync in ND-DT network, we assumed that there is at least one path between each pair of database nodes.

Finally, we implement ChronoSync and our improved version in an ND-DT network simulator (ICONE [3]). The results show that our improved ChronoSync is more suitable for ND-DT networks. It consumes less time to finish synchronization tasks, especially in sparse networks. Improved ChronoSync generates fewer packets, which increases the probability to have more network nodes synchronized.

## 1.3 Structure of This Dissertation

The organization of this dissertation is as follows. Chapter 2 provides basic background on DTN and NDN architectures. Chapter 3 explains set reconciliation and the file synchronization methods, respectively. Chapter 4 introduces several new techniques related to strategies of data reconciliation within DTN and NDN networks, which makes a further understanding of the related research concerning about our work. Chapter 5 provides a description of ChronoSync and improved ChronoSync in ND-DT network. Chapter 6 shows and discusses the simulation results. Finally, Chapter 7 concludes this dissertation and presents recommendations for future work.

In more detail, the organization of the thesis is as follows:

In Chapter 2, we present the background knowledge of DTN including the concept of DTN, the architecture of DTN, the characteristics of DTN, the special custody-based transfer manner, the movement models and the routing protocols. Then, NDN is explained, namely the NDN architecture and its forwarding strategy.

In Chapter 3, we begin by explaining the problem of set reconciliation. Then we categorize the approaches into three groups. The first group is the Log-based Reconciliation with prior context. The second group is related to exact methods that always returns the correct set difference and the third group includes methods with high success probability, which means that the results have a small chance of incorrection. Then, popular methods of each group are described. In Approximate Reconciliation Tree method, we provide an in-depth explanation of Merkly Tree, which is used in ChronoSync along with Log-based method. We also consider the problem

of file synchronization. We discuss current well-known file synchronization strategies, such as Rsync.

In Chapter 4, we explain a group of synchronization solutions in detail. We explain ChronoSync protocol and ChronoShare in NDN, Prioritized Data Synchronization solution in DTN and a cooperative caching approach, which give a new solution to synchronization.

In Chapter 5, we begin with an explanation of the ND-DT network simulator. Then, we explain the implement of ChronoSync in ND-DT networks. Afterwards, we describe how to improve ChronoSync in order to make it more suitable for ND-DT networks.

In Chapter 6, we first present the rationale of experiments to do and the parameters used in simulations. Then we present the simulator parameters used in all simulation scenarios. After that, different simulation scenarios for sparse and dense networks and also for few and large database nodes are provided. Finally, simulation results using synchronization time and synchronization related packets are presented, analyzed and discussed.

In Chapter 7, we conclude the dissertation and discuss a set of directions for future work.

# Chapter 2

# Delay Tolerant and Named Data Networks

This chapter introduces the fundamental knowledge and the basic support for this work. In particular, it presents the architecture of Delay Tolerant Networks and Named Data Networks.

## 2.1 Delay Tolerant Network

Nowadays, the networks we usually explore are based on the following assumptions [4]: 1) There is an end to end paths between the source and destination when nodes communicate with each other; 2) The maximum round-trip time between nodes is not too long (sending and receiving data in milliseconds, not in hours or days); 3) There is relatively little data loss or corruption on each link. However, there is a class of network that does not meet the above assumptions, such as the civilian networks on Earth, the wireless military battlefield, and other space networks. These networks usually experienced very long delays, long periods of link disconnection, high error rates, and large data-rate asymmetries. Repair the existing link problems in these kinds of networks are not easy, while constructing network-specific proxies (which provide access from this kind of network to the Internet) is also not a good idea [5]. Therefore, the existing network architecture and protocols are not suitable for this kind of networks. Under this circumstance, researchers have proposed a new type of network named Delay Tolerant Network (DTN).

### 2.1.1 Concept of DTN

Delay Tolerant Network (DTN) is a small regional network which tolerates intermittent connections and long delays. It was proposed by Kevin Fall [4] in 2002 and it was initially developed for interplanetary use. Needless to say, data transmissions between planets experience long delays and the connections between them are intermittent. However, other networks, such as wireless sensor networks, meet the same constraints as the interplanetary networks. Nowadays, DTN is used in diverse fields, such as military, public service and safety, environmental monitoring, personal use, engineering, and scientific research. There is currently many applications for DTNs.

A DTN node is an entity with a "Bundle" layer over the low-layer communication protocols. The structure is shown in Figure 2.1. A node may be a host, router, or gateway acting as a source, destination, or forwarder node. A source or destination node sends or receives bundles to

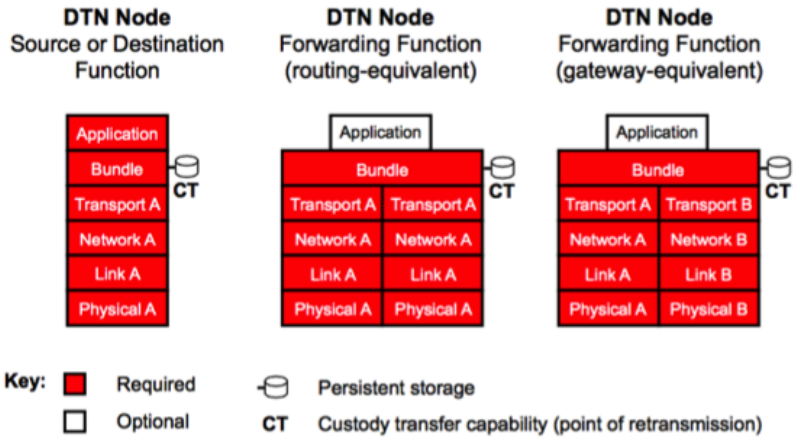



FIGURE 2.1: Structure of DTN Nodes [4]

or from another node, but it does not forward bundles received from other nodes. A forwarding node can forward bundles between two or more other nodes. There are two kinds of forwarding nodes: routing equivalent nodes and gateway equivalent nodes. The routing equivalent nodes forwards bundles between two or more other nodes, each of which implements the same lower-layer protocols. The gateway equivalent nodes forwards bundles between two or more other nodes, each of which implements different lower-layer protocols. A DTN node could be a pedestrian, a bus, a taxi or a train, each of which could have a different mobility pattern due to its own characteristic. Every DTN node has persistent storage to deal with the long delays. If the node operates over long-delay links, it stores the content in its persistent storage until links are available. To enhance the reliability of the transfer, the node may optionally support custody transfers, which is typically advisable in DTNs. Custody transfer is discussed in section 2.1.4.

"Contact" is a way of message dissemination in DTN networks. DTN nodes communicating with each other during the contact time. There are two kinds of popular contacts in DTN: Opportunistic Contact and Scheduled Contact. During the opportunistic contact time, the sender and receiver communicate with each other at an unscheduled time. In real life, people often use this type of contact to communicate one another. This kind of contact is considered unexpected. A Scheduled Contact is an agreement to establish a contact at a certain time for a given duration [6]. For examples, a bus will pass a specific place at a certain time. In this way, we could predict and arrange the future communication according to this time schedule.

### 2.1.2 DTN Architecture

The traditional TCP/IP network architecture has five layers (including Physical layer, Data-linking layer, Network layer, Transport layer, and Application layer). DTN adds a bundle layer between the transport layer and the application layer. The architecture of DTN network is shown in Figure 2.2. The Bundle layer has two functions: 1) sends data

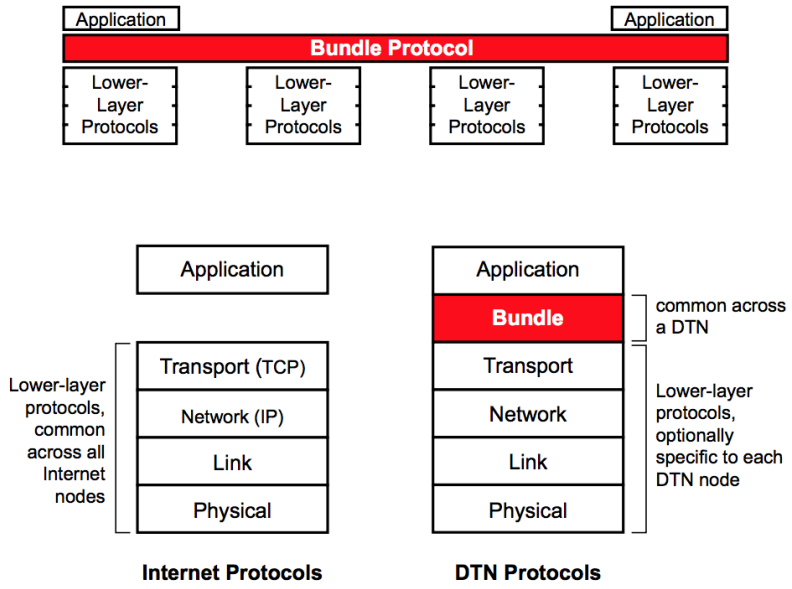



FIGURE 2.2: Architecture of Delay Tolerant Network [4]

to its application layer; 2) store and forward data between the various network nodes. With this mechanism, DTN can tolerate high latency and disconnection during data delivery.

The Bundle protocol is a new communication protocol on top of lower layer protocols, which ties the lower-layer protocols together. The bundle protocol agent stores and forwards bundle between nodes. In the same way, application programs can communicate with each other whether they use the same lower-layer protocols or not. The Bundle protocol works well on intermittent links, unlike other communication protocols, such as TCP.

### 2.1.3 Characteristics of DTN

DTN is a special network that has its own characteristics. DTN networks often suffer low transmission rates, asymmetric data rates, and very long delays. Connections may be intermittent suddenly during end-to-end communications. In some extreme cases, there may be no return channel at all.

In the traditional networks, queuing time often dominant the transmission and propagation delay. If the next-hop neighbors are not reachable, the routers discard the packet. So the queuing time rarely exceeds a second. But for DTN networks, the disconnection is common and the queuing time could be extremely large. In addition, due to the limited transfer opportunity, retransmissions can be even hard to achieve. Therefore, the message may be stored in the router for a long time. DTN networks use store-carry and forwards strategy to perform message exchanging.

DTN nodes usually work in special places (e.g. without power systems). In many cases, the lifetime of DTN node is limited, and the power is also a parameter. Nodes can stop working at any time, the duty cycle is usually lowered to increase network lifetime.

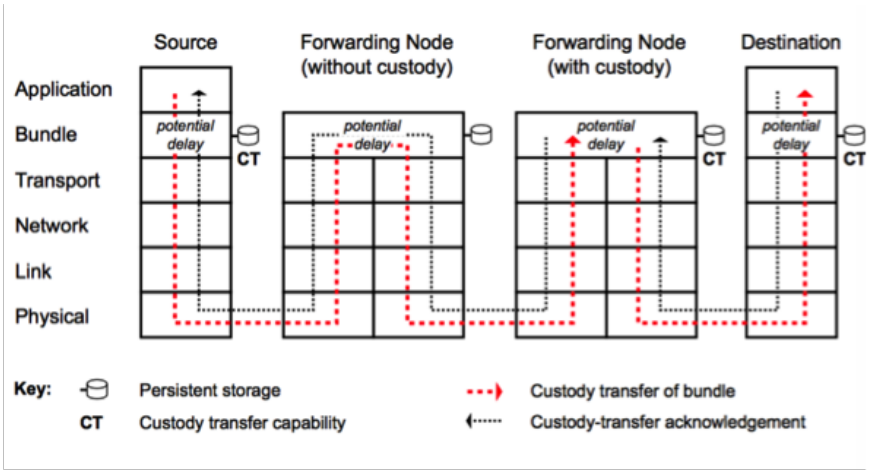



FIGURE 2.3: Custody Transfer in Delay Tolerant Network [4]

### 2.1.4 Custody Transfer

Custody transfer is a mechanism to enhance the reliability in DTN networks. It transfers the responsibility for reliable delivery towards the ultimate destination node [7]. Nodes holding a message with custody are called custodians. Usually, there is only a single custodian for a message. However, a message has more than one custodian may still exist in some circumstances. This type of custody is called joint custody.

The process of custody transfer is shown in Figure 2.3. When the current custodian sends a message to the next custodian, it requests a custody transfer and starts a time-to-acknowledge retransmission timer. If the node accepts custody, it returns an acknowledgment to the sender. If no acknowledgment is returned before the sender's time-to-acknowledge expires, the sender retransmits the message. A custodian must store the message until either another node accepts custody or its time-to-live expired.

### 2.1.5 Movement Models

A movement model should simulate the movements of real mobile nodes. Nodes change their speed and direction in reasonable time slots. In order to evaluate the performance of routing protocols in an ad-hoc network, it is better to use a movement model that accurately represents the mobile nodes. In the following, we will describe four movement models frequently used.

Random Walk is a simple mobility model described by Einstein in 1926. In this model, nodes wander away randomly with independent speed and direction. The direction and speed are selected randomly at the end of each movement [8]. Random Walk is an extremely unpredictable model and it is also one of the most popular and useful models for evaluating communication performance in DTN networks. Random Walk is a memoryless model knowing nothing about its past locations and speed values.

The Random Waypoint model is similar to the Random Walk model. Nodes choose a destination and a walking speed randomly, but pause times are added when the direction or speed is changed. After arriving the destination, the node pauses for a specific time and then it walks again.

The Map-Based Movement model is a derivative of the Random Walk model. Nodes move to random directions following the map roads. Shortest Path Map-based Movement is an improvement of Map-Based Movement model. It calculates shortest paths from the current location to a randomly selected destination by using Dijkstra's shortest path algorithm.

Working Day Movement model simulates the everyday life of ordinary people. People go to work in the morning, spend their daytime at the workplace, and go back home at evenings. There are three main activities in Working Day Movement model: being at home, working, and some evening activity with friends. Every node assigns a position on the map as its home location. After wake up, nodes leave their homes and use different transport methods to travel to work [9]. Then, they spent the day working at the desk or having a meeting in the office. After the work, they may go shopping, take part in different activities or walk on the street. Finally, they return home.

### 2.1.6 Routing Protocols in DTN

DTN is a network with long delay and intermittent connection. The storage capacity and energy of nodes are limited. Nodes have very little information about the network state and have little opportunities to transmit the information. Routing in such environments is a very difficult task. For this reason, the traditional routing algorithms are not suitable for DTNs.

In Direct Deliver protocol, the source nodes deliver message itself. The source node constantly keeps the message until the destination is in the proximity or directly reachable [10]. By using the Direct Deliver protocol, the nodes require seldom transmission knowledge. The message is only related to the destination node. There is only one hop to deliver the message.

First Contact routes data based on the first available contact. Nodes deliver the message through a set of intermediate nodes, then the intermediate node delivers the message to the destination nodes. A node forwards the message to nodes who never carried the message before.

Epidemic Routing algorithm is a flooding algorithm. Each node forwards a copy of packets to all encounter nodes that do not have these packets. Epidemic Routing algorithm guarantee the delivery ratio but waste a lot of resources. When two nodes meet, they first exchange the summary of their message repository. Then, each node transmits messages that the other node does not have [11].

Spray and Wait protocol makes a tradeoff between Epidemic protocol and Direct Deliver protocol. It performs significantly fewer transmissions than Epidemic protocol and achieves better deliver delay than Direct Deliver protocol. It consists of two phases: spray phase and wait phase. In spray phase, it spreads messages in a similar way as Epidemic routing. $L$ copies of each message are spread to $L$ distinct "relays". The spread can be done by the source node or by other nodes that receive this message copy.

If the destination node is found at spray phase, the delivery is finished. Otherwise, it starts the wait phase. The $L$ nodes carrying the message copy perform a direct transmission in the wait phase. [12].

MaxProp [13] is a scheduling message transport protocol. It maintains a queue ordered by the delivery probability of a future transitive path to the destination. The delivery probability is calculated by historical data and several complementary mechanisms, including acknowledgments, a head-start for new packets, and lists of previous intermediaries. MaxProp assigns a higher number to new packets and uses a list of previous intermediaries to prevent repeated data transfer.

The Prophet approach is a protocol based on delivery predictability metric. It uses a predicted value to describe the successful transfer probability between nodes. When two nodes meet each other, they exchange the value of the metrics for different known destinations and update their transmission predictive value. Nodes only forward the packet to the nodes that have higher values [14].

## 2.2 Named Data Network

Nowadays, the Internet has changed every aspect of people's lives. However, it also facing many challenges. For instance, IP address exhaustion, security problems, insufficient bandwidth and so on.

Moreover, the applications nowadays are more concerned about content and pay little attention to where are the contents. Under this circumstance, NDNarchitecture has been proposed. It changed the end-to-end Internet paradigm to a new content-centric model. Therefore, NDN requires new forwarding strategies and routing protocols.

### 2.2.1 NDN Architecture

NDN is an entirely new architecture based on our understanding of problems and limitations of current Internet. The network communication is driven by the consumer forwarding interests. It has new packet format, content-centric naming system, useful in-network storage, and forwarding strategies.

Packet carries data names rather than destination or source addresses. There are two types of packets in NDN: Interest packet and Data packet. Their structure is shown in Figure 2.4. Interest packet consists of three parts: Content Name, Selector, and Nonce. Every interest packet has a Name and a Nonce parts. Name is a string that responsible for routing and Nonce is a random number used to detect and discard duplicate packets. Data packet consists of four parts: Content Name, Signature, Signature Info and Data. Content Name is a unique identification decided by the content publisher. Usually, people use an easy understanding string for Content Name. Content Name does not need to be globally unique, though retrieving data requires some kind of global uniqueness. Names are used for local communication and may be heavily based on the local context. So, only local routing is required to find the corresponding data.

Consumers request content by broadcasting their interest over all the available connections. Any node who hears the Interest and has the

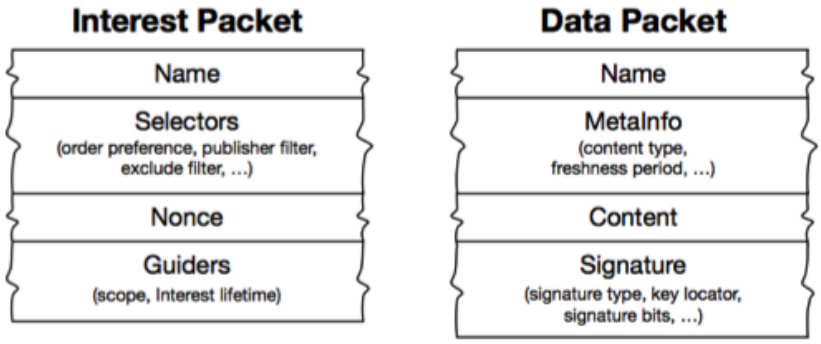



FIGURE 2.4: Packets in Named Data Architecture [15]

data, can response with a Data packet. Data packet is transmitted only in response to an Interest and consumers that Interest [16]. Note that neither Interest packets nor Data packets carry any host or interface address information.

NDN naming system[17] is the most important part of NDN architecture. NDN names are transparent to the network. Router does not know the specific meaning of the name, which makes names of content independent from the network. NDN use a hierarchical naming structure to express the relationship between various contents, which is similar to IP address. IP uses this prefix matching to resolve <network, subnet, host > hierarchy structure of IP address, which exhibits a high efficiency in practice. This structure has the ability to find the address quickly. The design of NDN structure retains the prefix-matching principle, using the longest prefix match when carrying out a Data packet matching. An interest request is satisfied when the name of Interest packet is a prefix of the Data packet.

The operation of an NDN node is very similar to an IP node. Both of them responsible for storing, forwarding and routing packets. NDN node consists of three parts: Content Store (CS), Pending Interest Table (PIT) and Forwarding Information Base (FIB).

The CS is used to store data, which similar to the buffer of an IP router but use a different data replacement policy. IP router forgets a packet after forwarding since each packet belongs to a single connection. In NDN, the packet is potentially useful to many consumers and also can be shared with other consumers. Therefore, contents are stored in node after the transfer, which improves the performance of upstream bandwidth and downstream latency. Data packets can stay in CS for a very long time if the node has a larger storage capacity.

PIT is used to record the Interest name and the arrival interface that have been forwarded but are still waiting for the matching Data. The structure of PIT is <Content Name, interface list>. An Interest packet is not forward if the router received the same interest requests from multiple different nodes. Only the corresponding interface is added to the list and the appropriate data is sent to all interfaces in the table when data is returned.

Forwarding Information Base (FIB) is used to forward Interest packets to potential Data sources. Its mechanisms are almost the same as the FIB of

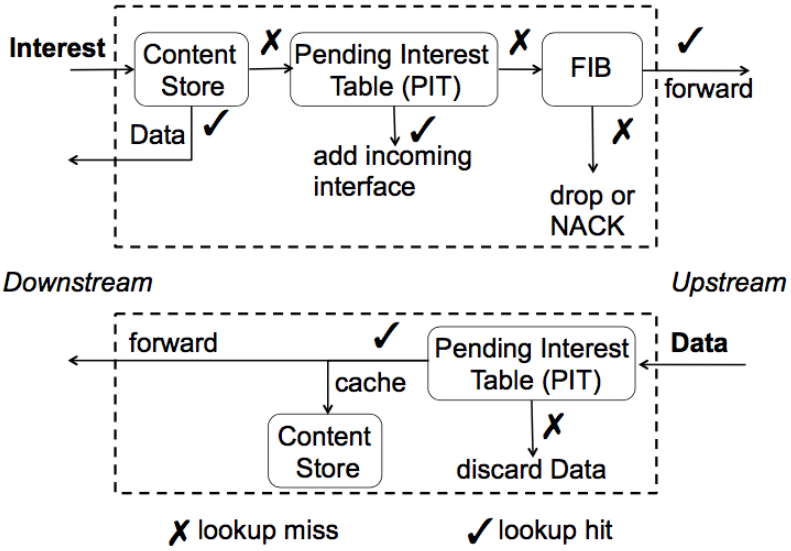



FIGURE 2.5: Interest and Data Packets Forwarding Processing in Named Data Networks [18]

IP router. But NDN FIB allows forwarding Interest packets through a list of outgoing interfaces rather than one and supports parallel queries.

### 2.2.2 Forwarding Strategy

The information is stored in CS, PIT and FIB tables. These tables are all indexed by the content name. The interface used to forward an Interest is determined by a strategic module, which makes the forwarding decisions adaptive to the network conditions. When a node receives a packet from an interface, it will query according to the largest prefix match based on the content name, and then operates as shown in Figure 2.5, and explained next:

1. Query CS table. When it receives an Interest packet from an interface, the router first checks whether there is a matching in its CS table. If a match is found, it forwards a copy of packet through the coming interface, and drop the Interest packet. Otherwise, continue with step 2.

2. Query PIT. If the name is in PIT, the node adds the coming interface to PIT interface list and also drops the Interest packet. When a Data packet is returned, the router looks up in the PIT and sends the Data packet to all interfaces from which this Interest packet was received. Afterwards, it stores this Data packet in the CS table. Otherwise, it continues with step 3.

3. Query FIB table: If there is a match in FIB, the router forwards the Interest packet according to the information in FIB, as well as the router's adaptive forwarding strategy. Otherwise, continue with step 4.

4. If there is no match in the three tables, the router discards the Interest packet, since the router cannot handle this Interest packet.[18]

# Chapter 3

# File Synchronization

In many distributed systems, common data is always shared among users and each user updated its local data independently. These isolated updates always bring database inconsistency. File synchronization method could efficiently solve this problem. File synchronization is a process that keeps two or more locations' files update. It uses in many situations such as file system backup and replications, cooperative work, website mirroring and chatting room. The significant problem in this update process is finding the outdated version of file effectively, which could solve the theoretical set reconciliation problem. Therefore, in this chapter, we will explore the set reconciliation and file synchronization more in-depth, and this will provide a solid foundation for our proposed synchronization method in Chapter 5.

Section 3.1 covers a group of popular set reconciliation methods including exact method and high probability method. Section 3.2 covers the file synchronization problems.

## 3.1 Set Reconciliation

Set reconciliation is a significant problem in file synchronization. In this section, we will dig down a little deeper in the set reconciliation problem involving various methods disposing of this problem. We separated the methods into three groups based on prior context and on the precision of the result.

For the method with prior context, we introduce a common method-Log-based Reconciliation. The second group covers two exact methods (Naïve Approach, Characteristic Polynomials), and the third group describes three methods with high probability (Bloom Filter, Approximate Reconciliation Trees, Invertible Bloom Lookup Tables ).

Set reconciliation is a process to obtain the union of distributed data sets. The primary procedure in the set reconciliation process is to get the different elements through data transmission. These data sets could be two or a plurality of data sets. To simplify the description, we starting the problem by a simple set reconciliation model in the case of two data sets. Imagine that we have two distributed data sets $S_A$ and $S_B$. The first step of set reconciliation is to get the datasets difference respectively (elements exclusively in $S_A$ and elements exclusively in $S_B$). The second step is to transmit $(S_A - S_B)$ to set $S_B$ and $(S_B - S_A)$ to $S_A$.

### 3.1.1 Log-based Reconciliation

Log-based reconciliation is an efficient technique in data set reconciliation. Each user on this distributed system recorded its local updates in log containing the isolated operations [19]. The local machine computes a new consistent state among these logs from different users. A simple example below introduces this Log-based reconciliation in detail.

Suppose there are two users $U_A$ and $U_B$. Each of them has a large collection of content on local machine. The synchronization between them can simply achieve by exchanging the lists of all their content (which is called Naïve Approach that we will describe in section 3.1.2). Since $U_A$ and $U_B$ could have a lot of same content, the size of data be transferred would be extremely large than the difference. To improve this situation, an often-used alternative is to maintain an update time-stamped log together with a record of the last communicated time [20]. When two users conduct the synchronization, they just need to exchange all of the updates since last communication. Obviously, Log-based data reconciliation requires prior context.

The operations of logs are the input of the reconciliation process. Some of the operations are non-conflicting operations. This means that the order of the operation is independent and we do not need to consider the sequence of these operations. When conducting this kind of reconciliation, we just combine the isolated operation. A set of isolated updates may contain conflicting operations as well. So, the updates to the same object must be executed in the same sequence. This brings much complexity to the set reconciliation.

Logs have more specific disadvantages as well. The log will be updated when there is a change in user data. This will add much overhead to the system, which is not worth if the reconciliation progress is carried out frequently. Furthermore, some items would appear multiple times in logs due to the frequent write of the "hot" content. This problem can be avoided by using Hash table. However, the system would increase more overheads on reconciliation. Moreover, users may receive the same update information from different users, which will lead to redundant communication. Finally, logs require stable storage and synchronized time, which are often unavailable on networking devices.

### 3.1.2 Exact Method

The exact method needs no prior context and always returns the correct set difference. Here we mentioned two methods: Naïve Approach and Characteristic Polynomial Set Reconciliation.

- **Naïve Approach**

  Naïve approach may be the simplest method appear in our mind when determining the set difference. Users exchange a list of all contents stored in is local machine. The list contains identifiers for all elements in their sets. Set reconciliation carried out by scanning the lists and removing the common elements. This requirements of message exchange size is O($|S_A| + |S_B|$) and computation time is O($|S_A| \cdot |S_B|$). The computation time could be decreased to O($|S_A| + |S_B|$) if each party sent its list in sorted order. This could achieve by

inserting all of the set elements into a hash table. That means to insert one list into a hash table, then querying the table with elements of the second list.

- **Characteristic Polynomials**

  Characteristic polynomials set reconciliation [21] is discovered by Minsky, Trachtenberg, and Zippel. They use characteristic polynomials to represent data sets.

  Assume that the data sets are integer numbers from set $S = \{x_1, x_2, \ldots, x_n\}$ and the characteristic polynomials is $P_s(Z) = \prod_{i=1}^{n}(Z - x_i)$. Those elements that make the $P_s(Z)$ equals *0* are considered to be elements of *S*. Finally, we observe the difference between the two data sets by a rational function: $\frac{P_(s_A)(Z)}{P_(s_B)(Z)} = \frac{\triangle P_(s_A)(Z)}{\triangle P_(s_B)(Z)}$. This function cancels out the same elements and leaves the differences elements in synchronizing sets.

### 3.1.3 Set Reconciliation with High Probability

The methods describe in section 3.1.2 find all elements of set difference providing perfect accuracy. However, there are some error tolerant applications need not find all the different elements. In this section, we will introduce some approximate reconciliation methods, which trade-off accuracy with transmission size and computation time. They determine a large portion of difference with little communication overhead.

- Bloom Filter

  In many daily applications, we often needed to determine whether a data is a member of the data set. For example, when we enter a word to a dictionary application, the spelling mistake is checked automatically. This checking process is to find whether the entered word is in the defined word collection. This kind of problems could be easily solved by hash table when the data set is small. However, shortcomings of hash tables are exposed with the increase of data size. Query time increasing as the table grows. Large space was used to store data. Under these circumstances, Bloom filter can be used.

  Bloom filter [22] is a space-efficient probabilistic data structure proposed by Burton Howard Bloom in 1970. It is a good solution to determine whether a data is an element of a large given data set. Bloom filter receives widespread attention since it was proposed. It needs very little spatial complexity than hash table when solving the same problem. Bloom filter is widely used in dictionaries, databases and web cache sharing [23]. Elements in Bloom filter can be inserted and lookup, but not allow to delete the existing elements.

  Bloom filter consists of an $m$ bits array and $k$ random independent hash functions. The functions are used to determine $k$ positions in the array. In order to express a set $S = \{x_1, x_2, \ldots x_n\}$ with $n$ items, Bloom filter uses an $m$ bit array and $k$ independent random hash functions $h_1, h_2, \ldots h_k$. Each of the hash function maps to a position in $\{0, 1, \ldots m-1\}$. The array is initialized $0$. When inserting an arbitrary element $x_1$ to the Bloom filter, we apply the $k$ hash functions

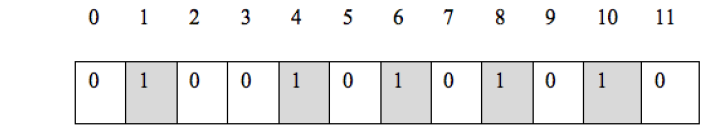



FIGURE 3.1: An Example of Bloom Filter

respectively and set those $k$ positions to 1. Every position could be set only once. So, if many elements point the same position to 1, only the first setting works. Looking in bloom filter carried out a similar process as the insert. $k$ functions were calculated and the corresponding positions were checked when querying an element. If there is at least one position is 0, we consider that the element is not in data set. This looking up may result in small false positives. The probability of a false positive depends on the number of bits used per item $n/m$ and the number of hash functions $k$. The false positive following equation: $f = (1 - e^{-kn/m})^k$. We could not delete elements from Bloom filter. This problem can solve by a transformed Bloom filter named countable Bloom filter [24]. An array of $n$ counters was used to track the number of elements currently hashed to these locations. The deletion could be safely done by decrementing the relevant counters.

Now we give a simple example to explain the false positive of Bloom filter clearly. Assuming that $k = 3$, element $A$ set bit $1, 4, 8$, element $B$ set bit $4, 6, 10$, and element $C$ set bit $2, 6, 8$. Set $S = A, B$. After inserting $A, B$ the Bloom filter shows in Figure 3.1. A true will get, if $C$ is queried in the Bloom filter. We will consider that $C$ is in set $S$. Obviously, this is a false in the query. The hash table will fail in queries whether we delete element $A$ or element $B$.

Bloom filter reconciliation is calculated as flows: insert all $S_A$ elements into a Bloom filter $F_A$, and then send $F_A$ to $S_B$. $S_B$ looks each of its elements in $F_A$. Set $S_B$ considers $S_A$ already has this key when the element is found. Even though sometimes the key does not really exist in $S_A$. After the calculation, $S_B$ sends these elements which exist in $S_B$ but not found in $F_A$ to $S_A$. Generally, set up the Bloom filter require $O(|S_A|)$ and find the difference need $O(|S_B|)$.

- Approximate Reconciliation Trees

  Approximate Reconciliation Trees [25] [26] is an extension of Bloom filter approach. It uses Bloom filter on top of a tree structure, which has a similar spirit of the Merkle tree. The structure of Merkle tree and a prefix tree structure will introduce first in this section. After that, we will involve the construction process of Approximate Reconciliation Trees.

  Merkle Trees [27] is a data structure widely used in comparison and validation process since proposed by Ralph Merkle in 1979. It is a tree with values stored in leaf nodes. The value of non-leaf nodes is obtained by applying the same hash function to the corresponding values storing in their children. There is a small chance of a false

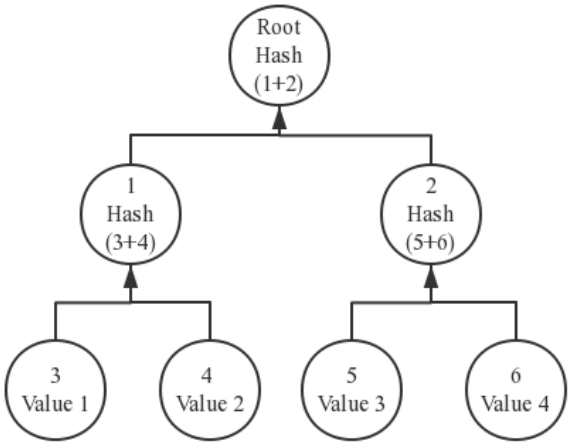



FIGURE 3.2: An Example of Merkly Tree

positive due to the hashing. Figure 3.2 shows an example of Merkle tree.

Actually, we could do set reconciliation only with Merkly tree. The process is similar to other tree reconciliation method. Set $A$ constructs its Merkly tree $T_A$ and send it to set $B$. Set $B$ creates a similar tree $T_B$ but with its own data set. The two data sets were considered identical if the root of $T_A$ matches of that $T_B$. Otherwise, set $B$ recursively traverses the children nodes of both trees.

Approximate Reconciliation Trees also learn from another prefix tree structure. Set $A$ constructs a binary tree of depth $\log u$, where $u$ is the number of elements in the universe $U$. The root corresponds to whole set $S_A$ and the children are subsets of $S_A$ in each half of $U$. That means the left child is $S_A \cap [0, u/2-1]$ and the right child is $S_A \cap [u/2, u)$ The remaining nodes use the same construction methods. Similarly, set $B$ creates a tree with the keys in set $S_B$.

Set $A$ sends its tree to set $B$. Set $B$ compares the two trees. There is no difference between the two sets if the root sets are the same. Otherwise, set $B$ recursively considers the children of the root of both trees. Finally, set $B$ will find all keys in $S_B - S_A$ at its leaf nodes. This kind of tree structure is unsuitable when the universe is large. So, we can hash each element initially before inserting it into the tree. Each internal node in the tree represents a subset of the elements.

Approximate Reconciliation Trees combining the sprite of Merkly tree with the prefix tree. The same hash function (such as exclusive-or) was used to get the internal nodes. The value of these nodes is the hash of its children nodes. The checking of two nodes could be done using associated values. This checking process will be finished in constant time. In order to save the transmission time, set $A$ inserted the leaf nodes and internal nodes into two Bloom filter and sent them

to set *B*. This may cause a small mistake when computing elements in $S_B - S_A$ as describe in Bloom filter.

This Approximate Reconciliation Trees method allows a faster search when the difference is small. It has the advantage of Merkly tree and Bloom Filter, but still has the common weakness of tree-based search strategies. An incorrect pruning from false positive could lead to large difference. The result will be totally different if there is a false positive when checking the root of Approximate Reconciliation Trees.

- Invertible Bloom Lookup Table

  Invertible Bloom Lookup Table [28] is a randomized data structure that stores a set of key-value pairs. Unlike Bloom filters, IBLTs support both a lookup operation (given a key, return a value) and a listing operation that lists out all the stored key-value pairs.

  IBLT consists of a table with *m* cells and *k* random hash functions $h_1, h_2, \ldots, h_k$. The hash functions are used to determine the position of key-value among *m* cells. That is to say, each of the function maps the key-value to a cell. Each cell has a *keySum* field ( which is the sum of all keys mapped to this cell), a *valueSum* field ( that is the sum of all values mapped to this cell), and a *Count* field ( that counts the number of keys mapped to this cell). All these fields are initially *0* and can be updated by operations supported by IBLT.

  There are three main operations in IBLTs: insert, delete, and listing. To insert a *x* to the cell, we first compute $h_1(x), h_2(x), \ldots, h_k(x)$ to determine which cells to insert. After that, we add the key to keySum and value to valueSum (add operation could be executed in many ways, we use exclusive-or as an example) and increment the count. The delete operation performs a reverse operation to insert. The listing operation is a little complex. We first go through the cells whose count field equals to *1* and add these cells to a queue. Then, a cell of the queue is popped to get the value stored in keysum and valuesum. Since the count field is *1*, the values in two fields are the original values we inserted. We add the value to the retrieved set and perform a delete operation to the IBLT. Repeat this process for the IBLT until every cell has a count value *0*. However, there is a small probability that no cell has a count equals *1*. In this situation, the lookup operation will respond with "not found".

  Before set reconciliation, set $A$ creates an IBLT $T_A$ containing all the elements from $S_A$ and send to set *B*. Set *B* creates a similar IBLT $T_B$ containing all the elements from $S_B$. As for conciliation, set *B* subtracts $T_A$ from $T_B$ to get a final IBLT $T_c$. The subtract operation is done by going through each of the cells in turn, compute the value from both IBLTs (For example, exclusive-or the value in the corresponding cell). After the computation, the value in the count field is subtracted. The value in $T_c$ will correspond exactly to the set difference, and we can just use a listing operation to retrieve them. Then, a listing operation on $T_c$ will be performed to find the difference of both sets. If a element *k* is in both $S_A$ and $S_B$, then *k*'s contribution to the relevant cell of $T_c$ (will be exclusive-ored twice, once for $T_A$, and

once for $T_B$), and *k*'s contribution to count will be *0* (*+1* for it being in $T_A$, and *-1* for $T_B$). Therefor, $T_c$ will not have the element *k*.

The element in $S_A - S_B$ will be compute once (exclusive-ored once) for each corresponding cell in $T_c$ (added to $T_A$ but not to $T_B$), and will contribute *+1* to count. Any key in $S_B - S_A$ will be execute a similar operation with the contribute *-1* in count (added to $T_B$ but not to $T_A$).

## 3.2 File Synchronization

File synchronization plays a very important role in many network situations. As we all know, bandwidth is a bottleneck of network affecting the performance of the network seriously. Good synchronization solution can decrease data transmitted effectively, and bring very large benefit to the network bandwidth. In this section, we will describe a famous single-round remote file synchronization protocol-rsync [29].

Rsync is an effective protocol used to update file by sending only the differences in data set. The main idea of rsync is to split a file into blocks and compute these blocks using a hash function. At the beginning of the synchronization, set $A$ and $B$ agree on the block size and split their file according to the negotiation. After the splits, set $A$ and $B$ compute the hash for each block. Synchronization begin by set $A$ sends its hashes to set $B$. After receiving those hashes, set $B$ compare its own hash with the corresponding hashes from set $A$. If a match is found, it returns $A$ with an acknowledgment of that match. Otherwise, send the corresponding block. To ensure the file is correct, set $B$ send a longer hash corresponding to the file block (for example 128-bit MD5) to set $A$. Set $A$ compute the new block and compare with the longer hash. If the computed and the longer hash does not match, we considered an error has happened during the transmission. Set $A$ send a request to set $B$ asking a retransmission of the block. RSync is a capable synchronization solution without sending the whole file across the network.

# Chapter 4

# Synchronization on NDN and DTN

To have a deeper and better understanding of the key technologies in data replication and synchronization for the used network scenarios, we analyze the state-of-the-art for synchronization in Delay Tolerant and Named Data networks, focused in the past five years. We will discuss and describe in detail a group of typical algorithms and solution including ChronoSync, ChronoShare, Prioritized Data Synchronization for DTN, and CPI synchronization algorithm.

## 4.1 ChronoSync Protocol and ChronoShare

ChronoSync protocol is an effective and robust protocol to synchronize the state of distributed database in Named Data Network [30]. It encodes the state of a dataset into a crypto digest form, named *state digest*. Each database node broadcasts *Interest* packets with the *state digest* calculated by its own knowledge.

The *Interest* packet recipients compare the difference of incoming *Interest*'s *state digest* and the local maintained *state digest*: 1) If they are the same, it indicates that dataset is identical; 2) If the *state digest* is the same as one of the previous locally *state digest*, they respond with the *Data* packet containing the changes; 3) if the state is unknown, it indicates that new data is generated or there are some problems in the network (e.g. network partitions). Then the receivers use a state reconciliation method to determine the differences.

When the recipient receives an unknown *state digest*, it sends out a recovery Interest with the unknown *state digest*. Those who produce or recognize the unknown *state digest* reply with recent dataset status. After receiving the recovery reply, the recipient compares with its local state digest and updates the state if it is more recent.

Many NDN applications implement ChronoSync protocol to maintain dataset synchronization. Core ChronoSync-based application components are shown in Figure 4.1, which provides a detailed explanation of ChronoSync. There are two main interdependent components: ChronoSync module and Application Logic module. The ChronoSync module is responsible for the dataset state synchronization and the Application Logic module is used to fetch the change content.

Current messages are maintained in the form of a digest tree. The *state digest* is kept at the root of digest tree, while the history of dataset state changes is stored in the form of a digest log.

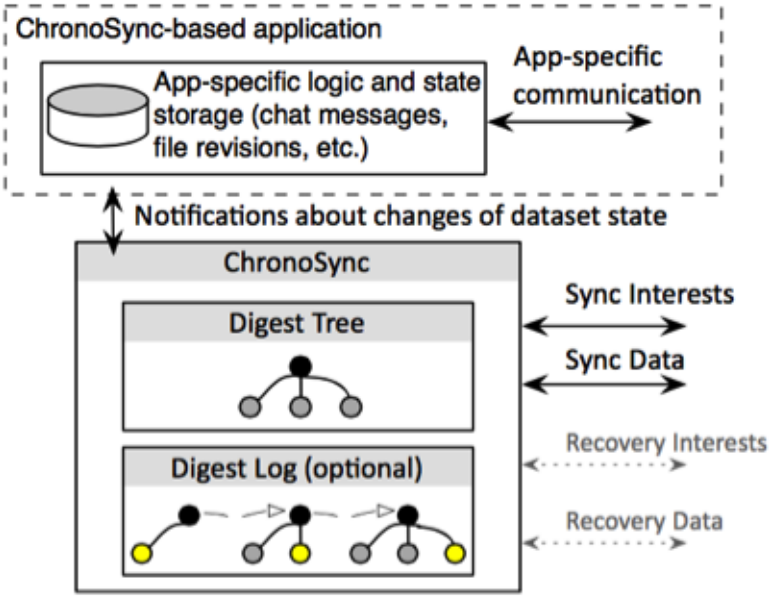



FIGURE 4.1: An Overview of ChronoSync [30]

ChronoSync module sends out a synchronize *Interest* with its current *state digest* to discover dataset changes. Once received a synchronize *Interest*, the ChronoSync module checks the dataset change with the help of the digest tree and digest log. If the received *state digest* is the same as its own *state digest*, it considers the two datasets are the same. If the receiver finds the same digest in its digest log, then it replies a synchronize *Data* packet, meaning that it contains the changes. If there is no match in both digest tree and digest log, it will use a recovery *Interests* to discover the differences. Finally, it updates the state once the recovery *Data* packets are received.

After discovering the difference between the datasets, ChronoSync module notifies the application logic module to deal with the dataset change. The application itself decides whether to fetch the chang messages. Or it may fetch a part of the important message firstly.

ChronoShare is an important NDN distributed file sharing application using ChronoSync protocol. Like any other NDN application, ChronoShare delivers packets according to the names, and the routing information is contained in Interest and Data packets. However, ChronoShare also has some special aspects. It chooses an action-based approach to synchronize file changes and treats the user operations as streams of *actions*. Each *action* records what changes have been made and where the modifications happened.

The typical ChronoShare scenario is the use of shared folder, as shown in Figure 4.2. Each shared folder consists of a set of files. These files were created locally or fetched from others user device through *actions*. Each user device has a local file manager to detect file changes and notifies to ChronoShare. After receiving information from a manager, ChronoShare updates the knowledge of *actions* at local areas and then synchronizes the *actions* to other user devices through ChronoSync. Other users fetch all

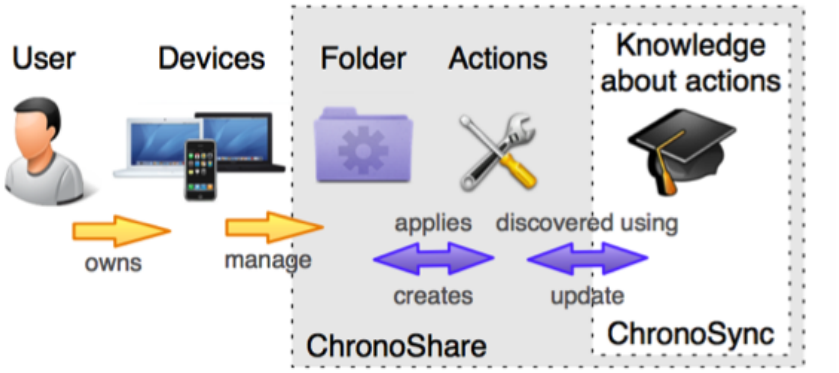



FIGURE 4.2: ChronoShare Entities [31]

the *actions* and have a consistent up-to-date view of the shared folder. Finally, the users fetch the missing information by applying these *actions* to their own folder. Users can decide whether or when to fetch the changed information. In this respect, ChronoShare is agnostic to the network infrastructure support and it is mobile-friendly[31].

## 4.2 Prioritized Data Synchronization for DTN

Due to the intermittent connections in DTNs, data transmission often suffers long delays. Replicate and store copies on several nodes is a good way to speed up the data delivery and ensure the reliability. In spite of reduction of data transmission delay, data replication uses a lot of storage resources. This Prioritized Data Synchronization method enables two nodes to exchange a subset of its differing packets, which offers a tradeoff between them. Packets with the highest priority would be transmitted first. It is based on the previous CPISyn approach for the reconciliation of two remote prioritized data sets in DTNs.

### 4.2.1 CPI Synchronization Algorithm

The CPISync algorithm is a mathematic solution to solve the reconciling problem of two remote sets. It focuses on the differences of datasets rather than the entire data. The difference between two data sets is calculated using the Characteristic Polynomials set reconciliation method.

The process of CPISync is described as follows:

- Host $A$ and host $B$ evaluate their characteristic polynomials on $\overline{m}$ samples respectively.
- Host $A$ sends its evaluations to host $B$.
- Host $B$ evaluate the difference by $\frac{\triangle P_(s_A)(Z)}{\triangle P_(s_B)(Z)}$ (which has been described in section 3,1.2) getting $\triangle S_A$ and $\triangle S_B$.
- Host $B$ sends $\triangle S_B$ back to Host $A$.

### 4.2.2 Protocol: Priority CPI (P-CPI)

In the case of large datasets, the difference of two datasets maybe larger than $\overline{m}$. Priority-based CPI (P-CPI) is used to support efficient prioritized data synchronization in DTNs. It partitions recursively the space of all possible numbers until the partition succeed with a prescribed bound $\overline{m}$. Data split first is assigned to high priority. Then, synchronization is run according to this priority. This method guarantees that the limited network bandwidth is used first for data entries with high priority[32].

## 4.3 A Cooperative Caching Approach in DTN

Caching among multiple nodes is a good technique for file share and could reduce the data delay in DTN. This cooperative caching strategy is an effective caching approach in DTN. It enables sharing and coordination of cached data among multiples nodes. It caches data at a set of Network Central Locations (NCLs) intentionally. NCLs correspond to a group of nodes that is easily accessed by other nodes in the network. The central nodes of NCLs are prioritized for caching data. Nodes that are close to the central node will be used first when central node's buffer is full. The NCLs selection is based on the probabilistic of data transport delay and caching overhead. NCLs coordinate multiple caching nodes to optimize the tradeoff between data accessibility and caching overhead [33].

### 4.3.1 NCL Selection Method

It is assumed that the contact time of two nodes in DTN form a Poisson process and the inter-contact time is exponentially distributed. A r-hop opportunistic path $P_{AB} = (V_P, E_P)$ between node $A$ and node $B$ consists of a set of nodes set $V_P = \{A, N_1, N_2..., N_{r-1}, B\} \in V$ and a set of edge set $E_P = \{e_1, e_2, \ldots, e_r\} \in E_P$ with edge weights $\{\lambda_1, \lambda_2, ..., \lambda_r\}$. Path weight $p_{AB}(T)$ is the probability of data transmitted from $A$ to $B$ within time $T$ opportunistically. inter-contact time between node *k* to node *k+1* is $X_k$. So, the total time needed to transmit data from $A$ to $B$ through path $P_{AB}$ is $Y = \sum_{k=1}^{r} X_k$. The probability density function (PDF) of node *k* to node *k+1* is $pX_k(x) = \lambda_k e^{-\lambda_k x}$. So, the PDF of total transmit time through path $P_{AB}$ is $pY(x) = \sum_{k=1}^{r} C_k^r pX_k(x)$ [33], where the coefficients is $C_k^r = \prod_{s=1, s\neq k}^{r} \frac{\lambda_s}{\lambda_s - \lambda_k}$. The path weight is $p_{AB}(T) = \int_0^T pY(x)dx = \sum_{k=1}^{r} C_k^r(1-e^{-\lambda_k T})$. The data transmission delay between nodes $A$ and $B$ is measured by the path weight. Each pair of nodes only maintains the shorter opportunistic path information. As for the average probability in Time $T$, this cooperative caching approach uses a metric $C_i = \frac{1}{N-|N_c|} \sum_{j \epsilon V \setminus N_c} p_{ij}(T)$ (where *i* indicates node *i*, $N$ is the total number of nodes, $N_C$ is the central nodes set).

In each NCL node, there is a central node that has the highest metric. After the selection of central node, the network administrator notifies each node about the central node's information. This central nodes selected approach is based on the nodes popularity in the network, rather than their computation or storage capabilities.

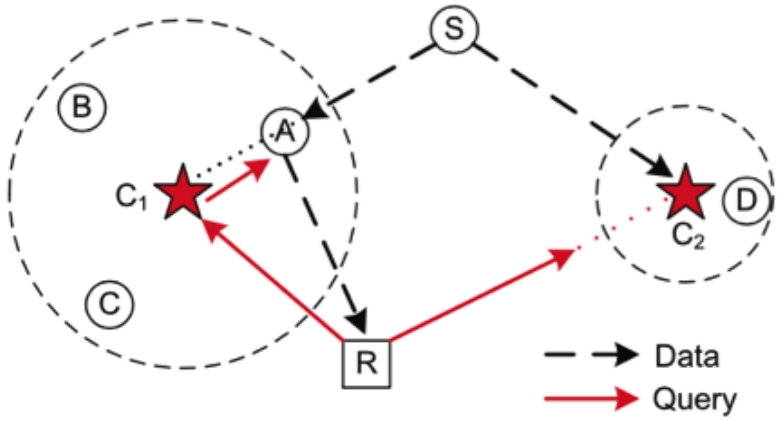



FIGURE 4.3: A Cooperative Caching Scheme in DTN [33]

### 4.3.2 Caching Scheme

The basic idea of the cooperative caching scheme is intentionally caching data at a set of NCLs. As shown in Figure 4.3, the stars represent the central node of NCLs. The source node sends a data copy to each NCL central node when it generates new data. Central node is prioritized to cache data in this scheme. Other nodes (e.g. node $A$) near central node start to catch data after central node's buffer is full. This data push phase is automatically executed according to the buffer conditions of nodes. A requester multicasts the query to all central nodes when it requests the data. Central nodes will reply data to the requester, if the requested content is in their buffers. Otherwise, they will forward the request to other caching nodes. After received the query, caching nodes will reply with the data to the requester. Obviously, this caching scheme will return multiple data copies to the requester. The number of copies is an important metric, since it controls the trade-off between data accessibility and transmission overhead.

### 4.3.3 Cache Replacement

Cache Replacement is used to adjust data cache locations dynamically. Unlike traditional cache replacement strategies (such as LRU), this cache replacement strategy dynamically calculates data utility and places popular data nearer to the central nodes of NCLs. The popularity of data is estimated based on the past requests.

# Chapter 5

# Improving ChronoSync in ND-DT Network

ChronoSync, described in Chapter 4 is an effective protocol that implements state synchronization in Named Data Network. Since our ND-DT network uses the same data structure as NDN, it is possible to solve the synchronization problem in Named Data Delay Tolerant Network.

In this Chapter, we will make a deep study of ChronoSync protocol, focusing on it's adaptability to ND-DT scenarios. This study will identify some aspects in which ChronoSync can be improved.

Section 5.1 gives a detailed introduction of ICONE simulation platform. Section 5.2 covers the implementation of ChronoSync in ND-DT network including the naming rules and the processing of packets. Section 5.3 introduces the Improved ChronoSync (IChronoSync) and explains its implementation on an ND-DT network simulator.

## 5.1 Simulation Platform

Information Centric ONE (ICONE) is a modified version of ONE simulator written in Java.

ONE is a well-known simulator specifically designed for evaluating DTN routing and application protocols [34]. It allows users to simulate various scenarios with different synthetic movement models and observes node's interaction process conveniently in the simulator's GUI. Users could easily get simulation data through the report modules and also simulate different scenarios by modifying the frequency, duration and other properties of the nodes in the setting files.

However, ONE only supports node-to-node communications. It has to know the IP addresses of source and destination node and does not support NDN simulations. To meet the NDN requirements, ICONE simulator was designed. ICONE retains the good features of ONE simulator and makes some important changes to satisfy the NDN communication requirements.

### 5.1.1 ICONE Components

ICONE changes the communication mode of ONE because source and destination addresses are no longer important for NDNs. The messages must carry the name of the data and forwarding strategies must change from address to content name matching. Nodes must include CS, PIT, and FIB components. The architecture of ICONE is shown in Figure 5.1. It is similar to ONE but has some unique components highlighted in green.

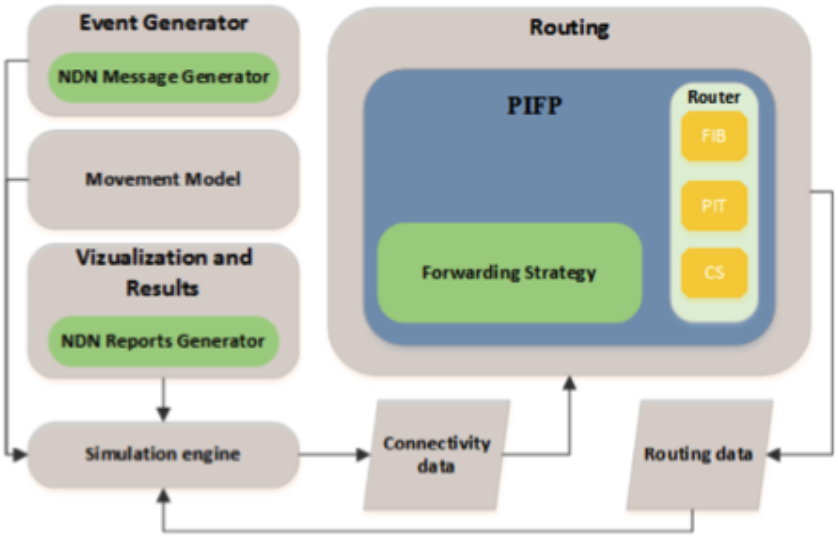



FIGURE 5.1: Information Centric ONE (ICONE) Framework [3]

### 5.1.2 Event Generator

*NDNMessage* was defined with a common set of fields including Content-Name, MessageType and Time To Live (TTL). Messages will be discarded when its lifetime exceeds the TTL. Two types of messages are defined in ICONE: Interest packet and Data packet. Interest packets play an important role in NDN communications. Consumers express their interest in a specific content by multicasting an Interest packet. The content name is on the ContentName field. Nodes who have the content in its CS and received the Interest message will reply a Data packet.

The event generator also called Message generator was also redesigned in ICONE. All events are generated according to a set of input parameters and stored in a trace file. A trace file is a simple text file that can be saved and loaded. Message generator modules are normal Java classes that can create requests number, request size and time intervals between sessions. The data size can also be configured with a minimum and a maximum value.

### 5.1.3 Movement Model

Movement Model specifies the node mobile capabilities and defines the rules of nodes movement. It is the same with the ONE simulator. Three types of movement models are implemented in ICONE: random movement, map-constrained random movement and human behavior based movement. They are detailed in Section 2.1.5.

### 5.1.4 NDN Reports

Report module is an important component in ONE simulator to get the results after the simulation run. It records connections, messages, and movement-related events by calling for the related object methods. It can

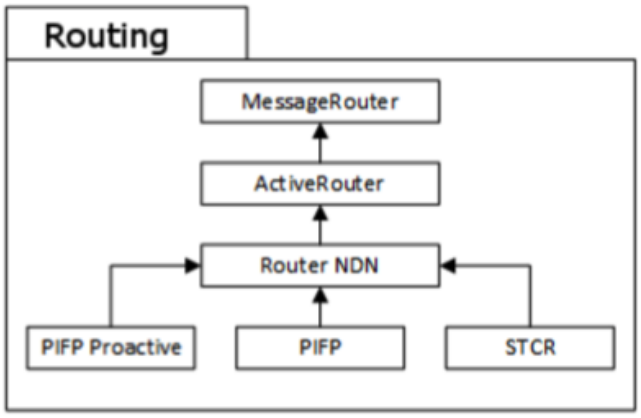



FIGURE 5.2: NDN Routing Classes in Information Centric ONE (ICONE) [3]

either write information about the event into a output report file or create a summary when the simulation is done.

ICONE implemented a number of new report models to register NDN communication metrics, such as the number of Interest packets generated, sent and satisfied by all nodes in the network, the number of Data packets sent, the number of satisfied Interest at the local node, the hops from provider to consumers, and so on.

### 5.1.5 NDN Router

NDN Router implements an abstraction of the basic components of NDN: PIT, FIB, and CS. PIT and CS components provide create, delete and check methods for PIT and CS entries. The *createPITEntry* method is invocated, when a new Interest packet arrives. The *deletePITEntry* method is invocated when the Interest is satisfied or lifetime exceeds the TTL. Both PIT and CS components provide updating functionality in PIT and CS table. FIB component has more updates when nodes exchange routing information [35].

Figure 5.2 shows the architecture of NDN routing classes in the *Routing* folder. From the architecture, we can view the dependencies between modules. *NDNRouter* is a new class extended from the *ActiveRouter*, which allows to deal with connection and transportation issues during running time. *NDNRouter* also provides message-forwarding strategy, and handles new NDN protocols.

### 5.1.6 PIFP Protocol

Probabilistic Interest Forwarding Protocol(PIFP) is a routing strategy for Interest packet forwarding. It is derived from Prophet DTN protocol [3]. PIFP provides a way to find the best node to send messages. This protocol designed a new metric Delivery Predictability (DP) to evaluate the possibility of encounters between nodes and content. DP is a combination of frequency and freshness. Frequency describes how many times one

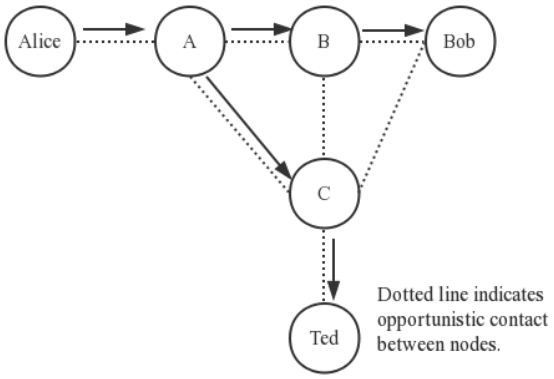



FIGURE 5.3: Synchronization Model in ND-DT Netwotk

node meet with a specific content. Freshness reflects how many time this information is updated. Nodes who carry Interest packets should forward them to nodes with higher DP. This protocol is implemented in a *PIFP* class. PIFP extends *NDNRouter* and uses all its data structures and functions.

### 5.1.7 Network Face

Node ID is used as the network faces in ICONE. When a node receives an Interest packet, it records the requesting node ID and uses the ID as the incoming face.

## 5.2 Impletementing ChronoSync in ND-DT Network

This section details the implementation of ChronoSync on three aspects: Naming, Packet Handling Process, and its implementation on ICONE simulator.

### 5.2.1 An Overview of ChronoSync in ND-DT Network

The main function of ChronoSync is to discover the database state change in NDN. A detailed description was provided in Chapter 4. Here we only focus on its specific implementation in ND-DT networks.

ND-DT network's biggest feature is the intermittent connection. Nodes carry data while moving in the network and transmit to other nodes when they have connections established. Unlike wired NDN, data transmission can only occur when there is a connection. An important issue is that synchronization process may not occur immediately after a database change. Database node should wait for a connection with other nodes. Therefore, there is a possibility that data exchange will never occur if the node does not contact other nodes at all. In such case, the synchronization will fail.

In order to implements ChronoSync protocol in ND-DT network, we make the following assumptions: database nodes are not isolated. There is at least one node that connects the database node and sends its messages to the network. This means that there is at least one path between two

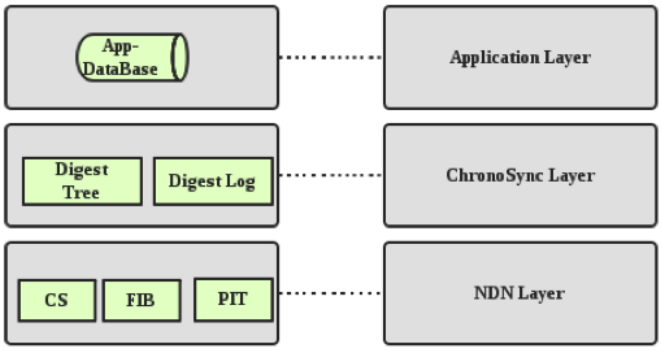



FIGURE 5.4: Structure of ChronoSync Based Application

database nodes. The data model is shown in Figure 5.3. Nodes $Alice, Bob$ and $Ted$ are database nodes with large storage space. Nodes $A, B, C$ are relay nodes (there could be a lot of relay nodes in the network, we only depicted three). $Alice$'s messages always arrive at $Bob$ and $Ted$. The same happens with $Ted$ and $Bob$.

Figure 5.4 shows the structure of our ChronoSync based application in ND-DT network. NDN layer is mainly used for data transmission in ND-DT network. Above the NDN layer, we designed a ChronoSync layer, which implements the ChronoSync protocol. It is independent of data forwarding and transmitting method used in NDN layer. After finding out the differences between the Database state, the application layer decides whether to fetch the missing message from other Database nodes.

There are two main components in the ChronoSync layer : Digest Tree and Digest Log. Digest Tree is used to compute the current database state, while Digest Log records the database change of each synchronizing process or data generate process.

ChronoSync uses a Merkly Digest Tree to record the current database state. Merkly Tree was introduced in Chapter 3. The database state is encrypted using SHA-256 and stored in the root of Digest Tree. A union of the subsets of data generated by all producers represents the state of the application database. To simplify the writing, ChronoSync assumes that the producer generates data in sequence. As a result, we could use its name prefix and the latest application data's sequence to represent the state of each producer. An example of database Merkly Tree structure is shown in Figure 5.5. Each branch of the tree represents a state of a producer. The name prefix and the latest sequence of the producer are stored in leafs of the tree. Then, it is recursively applied the SHA-256 hash function to all child nodes. The whole database state is at the root of the tree. The digest stored in the root is called state digest. ChronoSync uses the Log Tree to record the change database state digest and what kind of changes was made in the database.

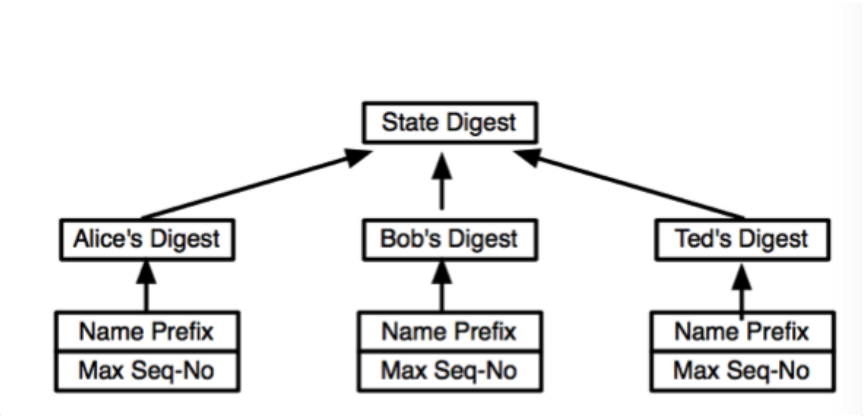



FIGURE 5.5: An Example of Digest Tree Structure [30]

### 5.2.2 Naming Rules

Naming rules take an import role in Information Centric Network (ICN) because both the forward decision and routing depend on names. By using the name of the data packet, a router could figure out where to forward the packet and which process to deal with this packet. To better implement ChronoSync in ICONE, we integrate the naming rule of ChronoSync with ICONE rule. We keep the philosophy of ChronoSync and only change a format. Table 5.1 shows our new naming rules used in the implementation. There is three type of naming rules used in ChronoSync: application data names, synchronization data names and recovery synchronization data names.

- **ChronoSync Interest and ChronoSync Data**

  *Outstanding Interest* and ChronoSync *Data* are used to do the general synchronization process in ChronoSync. ChronoSync *Outstanding Interest* packet is used to broadcast the current database state and drives the synchronization process. These packets start with the broadcast prefix *"NDNBroadcast"*. *"Sync-ChronoSync"* indicates the application that generates this packet. The last part is the digest of these database nodes. As for the response of *Outstanding Interest*, ChronoSync *Data* uses a similar rule but with request Interest Digest at the last part of the name.

- **Recovery Interest and Recovery Data**

  Recovery *Interest* and Recovery *Data* is used to set the unknown Interest Digest. Similar to *Outstanding Interest* and ChronoSync *Data*, the name starts with *"NDNBroadcast"*. It is followed by *"Recovery-ChronoSync"*, which predicts the recovery process of ChronoSync. The last part is the received unknown digest .

- **Application Data**

  Application *Data* is the real content generated by producers. The first part identifies the synchronization group and the second part indicates which application generates this content. The third part is the name of the producer and the last part is the sequence number,

TABLE 5.1: Naming Rules used in ICONE

| Type of Packet | Examples |
|---|---|
| ChronoSync Interest | NDNBroadcast/Sync-ChronoSync/ a1234as.... |
| ChronoSync Data | NDNBroadcast/Sync-ChronoSync/ a1234as.... |
| Recovery Interest | NDNBroadcast/Recovery-ChronoSync/ a1234as... |
| Recovery Data | NDNBroadcast/Recovery-ChronoSync/ al234as... |
| Application Data | DBGroup1001/ProduceApplication/ Alice/01 |

which indicates how many pieces of content were generated by this producer.

### 5.2.3 Outstanding Interest Handing Process

Outstanding Interest Handing Process is an important part of the synchronization. The process is shown in Pseudocode 1. ND-DT network is a consumer-driven model network, the synchronization begins with the *Outstanding Interest* generated by every distributed database node. Each node multicast an *Outstanding Interest* carrying its current database digest. When a database node receives the *Outstanding Interest* packet, it compares with its own state digest. The following situations may occur:

- **The state digest in *Outstanding Interest* packet is the same with its current database digest**

  In this situation, we can conclude that the two database state are the same. There is no need to do synchronization. But the face from which the Interest is coming is added to the PIT. This record will help on sending ChronoSync Data packet when there is a local database change.

- **The state digest in *Outstanding Interest* packet is a prior digest**

  The data digest is not the same as the current digest but there is a match in the log. It means that this database node has a new database state. It should send a synchronization message to the *Outstand Interest* source node. This node will respond with a ChronoSync *Data* packet carrying the database changes.

- **The state digest in *Outstanding Interest* packet is an unknown digest**

  It means that the database node which the *Outstanding Interest* come from has a new state. So, the node that receives the Interest sends a recovery packet to request the database changes.

**Pseudocode 1. ChronoSync Interest Handing Process**

```
1: all database node which received an Outstanding Interest packet
2:     Compare with its own database state digest
3:     if the digests are the same then
4:         Add source Face to the corresponding PIT
5:     else if finds a match in its log then
6:         Responds with a ChronoSync Data packet
7:     else if unknown digest then
8:         Send a Recovery Interest with the received digest
```

### 5.2.4 Recovery Interest Packet Handing Process

When a database node receives a recovery *Interest* packet, it checks its CS (Content Store) or Log Tree. If there is a match, a recovery *Data* packet with the state of the current database is sent as the response to this Interest. If the node does not recognize the digest carried in recovery *Interest*, it just ignores.

### 5.2.5 Data Packet Handing Process

The Data Packet Handing Process is shown in Pseudocode 2. When a database node receives a distinct Data packet, it first checks whether the packet is requested by itself. If not, it stores the Data packet in its CS. Otherwise, it sends the Data packet to ChronoSync layer. The ChronoSync layer recomputes the database digest and puts the changes in its Log Tree. After that, ChronoSync notifies the application logic model about the database change and leaves the decision on what to do to the application layer. Finally, the node generates a new *Outstanding Interest*. If a recovery packet is received, it compares the database state with its own database. Updates the digest tree and adds an item to the digest log, if it is not an out-of-date packet. After the change, it also notifies the application logic model about the database change and sends a new *Outstanding Interest*.

### 5.2.6 Application Data Fetching

After the ChronoSync process, all database nodes have the same database state. ChronoSync leaves the decision of how to deal with the change to the application. The application could fetch the entire missing data message or fetch a part of the data. Of course, the application could also ignore the Database change. In the ND-DT network, fetching the entire database is not a good decision, since the opportunity for data exchange is precious. Nodes could exchange messages only when there is a connection between them. During the disconnection time, nodes store the content temporary. The nodes carry content during the move and forward it when they reach a node closer to the destination. So this network is not suitable to fetch the entire database content when the network is large.

**Pseudocode 2. Data Packet Handing Process in ChronoSync**

```
1:  all database nodes received Data Packet
2:    If data packet was requested by this database node then
3:      If it is ChronoSync Data Packet then
4:        Change Local Digest Tree; Change log
5:        Notify application layer about the change;
6:        Generate new Outstanding Interest;
7:      else if it is Recover Data Packet
8:        if not repeat content packet then
9:        Change Local Digest Tree; Change log;
10:       Notify application layer about the change;
11:       Generate new Outstanding Interest;
12:   else delete packet;
```

### 5.2.7 Implementation of ChronoSync in ICONE Simulator

This section introduces the implementation of ChronoSync protocol in ICONE simulator. The most important part of implementation is the message reception process, which shown in Figure 5.6.

When a node receives a message, it checks the type of packet. After that, it handles the message according to the packet processing.

When an Interest packet is received, the message reception process first checks in CS component whether there is a match. If so, the node answers with the corresponding Data packet. Otherwise, it checks whether there is a ChonoSync layer installed on the node. If so, the NDN layer sends the message to the ChronoSync layer.

When the node receives a Data packet, it checks its PIT to find whether this packet was requested by this node. If there is a ChronoSync application installed and Data packet was requested by this node, then it sends the Data packet to the application layer. Otherwise, the normal execution of Data packet is done.

### 5.2.8 Disadvantages of ChronoSync

ChronoSync uses a recovery process that brings back the missing information during network partitions periods.

The *Outstanding Interest* only brings the encrypted digest. A database node only knows that the database has changed when it receiving an unknown Interest packet. It has no idea of the change has been made. To get the specific change of database, the node must send a recovery packet and wait for recovery reply brings the change back. Node needs to wait a period if time to get the database change. Since ND-DT network is characterized by disconnections and long delays, network partitions and reconciliations often occur in a short period of time. Therefore, a database node needs to perform the recovery process frequently. Due to network delay, the node needs to wait more time to get the database change.

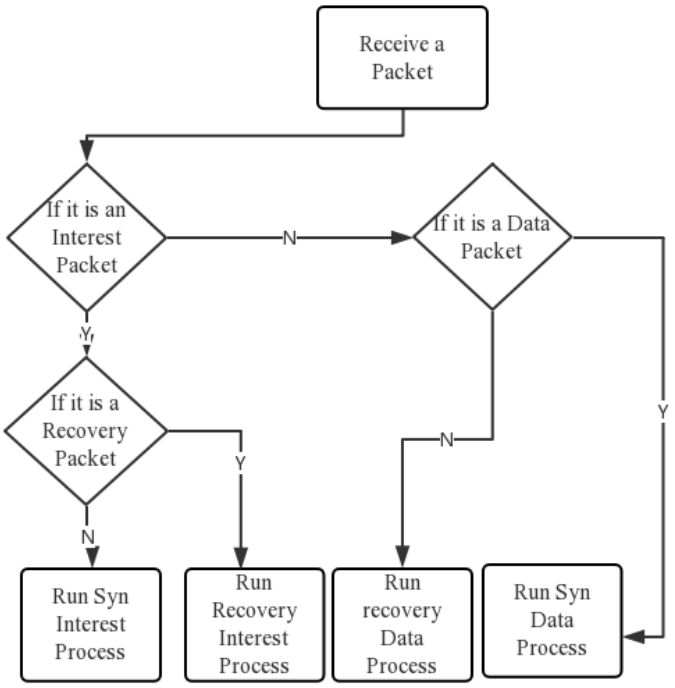



FIGURE 5.6: ChronoSync Flow Diagram in ND-DT Network

## 5.3 An Improvement of ChronoSync

### 5.3.1 An Overview of improved ChronoSync

To better suit the frequent network partition in ND-DT networks, we improved ChronoSync protocol. The recovery process is replaced with a more active interaction. A database node will inform others when there is database change caused by the local machine. Other database nodes just ignore the packet and wait for the notification when an unknown Interest packet is received.

Three components are used in the IChronoSync: Digest Tree, log of changes and log of newly generated data. The function of Digest Tree and log of changes is the same as ChronoSync. Digest Tree is used to store current database state and the log of changes is responsible for recording the change of database. The log of newly generated data is a new component used to record the content generated by local database node.

There are three type of Interest packets in IChronoSync : Outstanding Interest, Application Data Interest, and Something New Interest. Outstanding Interest and Application Data Interest were introduced in the previous chapter. Something New Interest is used to inform other synchronization members about the new data.

TABLE 5.2: An Example of Something New Interest (SNI) in Improved ChronoSync

| Type of Packet | Examples |
|---|---|
| Something New Interest | NDNBroadcast/somethingnew/ Alice |

**Pseudocode 3. Something New Interest Packet Handing Process in Improved ChronoSync**

```
1:  all database nodes received Something New Interest Packet
2:    if not out-of-date Something New Interest then
3:        Compare the state in Interest packet with its own state
4:          if state of Interest packet is the same or a previous one;
5:            Ignore this Interest
6:          else
7:            Change digest tree; Change log;
8:            Notifies application layer about the change
9:            Satisfy Outstanding Interest
10:           Sending synchronization Data Packet
11:           Sending new Outstanding Interest
12:     else
13:       delete packet
```

### 5.3.2 Naming Rule

The *Outstanding Interest* and application *Interest* use the same naming rule as ChronoSync. The *Something New Interest* packet was added in this improved solution. An example of its naming rule is shown in Table 5.2. *"NDNBroadcast"* is the multicast domain and*"Somethingnew"* indicates the type of this packet. *"Alice"* identifies the producer of this packet.

### 5.3.3 Outstanding Interest Handing Process

Outstanding Interest Handing Process is the same as ChronoSync when receiving the same digest and a prior digest. The IChronosync just ignores the Interest when an unknown *Outstanding Interest* is received. It leaves the recovery process to the Something New process.

### 5.3.4 Something New Interest Handing Process

The process of receiving *Something New Interest* is shown in Pseudocode 3. When the database node receives a *Something New Interest*, it first checks and deletes the out-of-date *Something New Interest*. This could assure that every node only keeps one *Something New Interest* for each producer. Then the node checks if it is a database node. If so, changes the database according to the state carrying in this packet. Otherwise, it only does the normal forwarding of Interests.

# Chapter 6

# Analysis and Discussion of the Simulation Results

This chapter compares the IChronoSync protocol with ChronoSync protocol. They are evaluated and tested through three simulation scenarios. To make the results simple and understandable, tables and figures along with detailed analysis will be used in this chapter.

Section 6.1 describes the settings used in all simulation scenarios. Section 6.2 gives a detailed description of the three different simulation scenarios. Section 6.3 covers the corresponding results and analysis, and Section 6.4 discusses the results.

## 6.1 Configuration

To understand the characteristics and tradeoffs of ChronoSync protocol and IChronoSync protocol, we conducted a number of simulation-based experiments using ICONE simulator, which was introduced in Chapter 5. The experiments are divided into three different scenarios. For each concrete scenario, the same configuration parameters were used for both protocols. All simulation scenarios correspond to the map of Helsinki, Finland with streets, pavements, and roads. The simulation area is 4500m x 3400m. The PIFP was used as the data transport protocol, which is a Probabilistic Interest Forwarding Protocol used in ND-DT network. A summary of the common settings is shown in Table 6.1.

The simulation scenario consists of groups of nodes that can be cars, pedestrians, and trams. Nodes move according to a Movement Model established in the configurations files. In our scenarios, pedestrians and cars move according to Shortest Path Map-Based Movement Model, while trams are forced to move according to predefined tracks. The speed of movement is set as follows: pedestrians move with a random speed selected from 0.5 to 1.5 m per second, cars and trams move with a random speed from 2.7 to 13.9 m per second and 10 to 30 m per second, respectively. All nodes use wireless interfaces, both Bluetooth interface and high speed interface were selected in these scenarios. There are two kinds of interfaces available in ICONE: simple interface and high speed interface. A simple interface has a transmission rate of 200 kbps within 10 meters while a high-speed interface has a rate of 10 Mbps in 100 meters. We set simple interfaces for pedestrians, cars and high-speed interfaces for trams. The storage space is an important parameter in our synchronization solutions. Usually, database nodes have a larger storage space than transmission nodes. Generally, the larger store space the better. Since our ND-DT

TABLE 6.1: Summary of ICONE Configuration used in all Scenarios

| Parameter | Value X |
|---|---|
| Simulation Time | 96400s (Near a day 26,6h) |
| Simulation Areas | 4500 m*3400 m |
| Movement Model | Pedestrians: Shortest Path Map-based Movement Model<br>Cars: Shortest Path Map-based Movement Model<br>Tram: Map-based Movement Model |
| Speed of Node | Pedestrians: 0.5 to 1.5 (m/s)<br>Cars: 2.7 to 13.9 (m/s)<br>Tram: 10 to 30 (m/s) |
| Messages TTL | 8 hour |
| Radio Interface | Bluetooth and highspeed Interface |
| Transmission Range | Pedestrians and cars: 10m<br>Tram: 100 m |
| Transmission Rate | Simple Interface: 200kbps<br>High Speed Interface: 10Mbps |
| Storage Space | Database node: 400MB<br>Transmission Nodes: 100MB |
| Database Nodes | Stationary Pedestrian |

network is a small region network, a small database storage space is enough. So, the storage space of database nodes is set to 400 MB and transmission nodes is set to 100 MB.

## 6.2 Simulation Scenarios

In order to study the impact of network density on the two synchronization protocols, we implemented two simple simulation scenarios with three database nodes and a group of pedestrians, cars and trams moving in simulation area. Then, a scenario with 20 database nodes and a group of pedestrians, cars and trams were used to test the performance of synchronization when adding database nodes. In these scenarios, database nodes are fixed in a specific location (chosen manually). Moving nodes are only responsible for data transmission and do not need to synchronize. A summary of the three scenarios is presented in Table 6.2.

The dense scenario consists of two groups of pedestrians (each one has thirty-nine nodes), one group of cars (thirty-eight nodes) and two groups of trams (each has two nodes). The total number of nodes is 120.

The number of nodes used in sparse network scenario was reduced to 40. It consists of two groups of pedestrians (each has fifteen nodes), one group of cars (eight nodes) and two groups of trams (one for each).

Regarding the scenario of increasing database nodes, we tested with 20 database nodes in the dense network.

Before the simulation starting, there is no content in the database nodes, and the digest is null. Database nodes generate *Outstanding Interest* when the simulation begins. Database nodes generate content over a period of

TABLE 6.2: Summary of Simulation Scenarios

| Scenarios | Number of Nodes |
|---|---|
| Dense Scenario | Database Nodes: 3<br>Pedestrians: 39*2<br>Cars: 38*1<br>Trams: 2*2 |
| Sparse Scenario | Database Nodes: 3<br>Pedestrians: 15*2<br>Cars: 8*1<br>Trams: 1*2 |
| 20 database nodes Scenario | Database Nodes: 20<br>Pedestrians: 39*2<br>Cars: 38*1<br>Trams: 2*2 |

time. After 11500 seconds the database nodes stop the data generation. The network achieves its first steady state. At 24400 seconds, one of the database nodes starts generating content. After a while, the other two databases begin to generate content. They end content generation at 53380s, which brings the second steady state. After 67300 seconds, the database nodes continue the content generation.

For each of the scenario described above, we run 20 simulations with different seeds and used the arithmetic average as the final result.

The dense and sparse scenarios were selected to evaluate the performance of ChronoSync and IChronoSync in such conditions. The third scenario is used to test the performance of both protocols when increasing the number of database nodes.

## 6.3 Simulation Results

This section shows the results obtained in the three scenarios. After that, the impact of network density and database nodes number is analysed.

The evaluation metric is often critical when evaluating the performance of protocols. An evaluation metric may be naturally appropriated for one case, but not for the others. For example, the latency time is important for real-time networks but not for DTNs. An appropriate evaluation metrics will help to clarify the network performance. In the following, we will describe the metrics used in evaluating IChronoSync and ChronoSync.

- **Time to Synchronization**

  Time to synchronization is one of the most important criteria to measure the performance of synchronization protocols. It can directly affect the communication performance of the entire network.

- **Number of Generated Outstanding Interests**

  Number of Generated Outstanding Interest is the total number of Outstanding Interest packet generated by all database nodes. This

kind of packet is only generated when there is a state change in the network database. Therefore, we could consider that the distributed database achieves a stable state when this metric keeps unchanged for a long time period.

- **Number of Satisfied Outstanding Interests**

  Number of Satisfied Outstanding Interests is the total number of satisfied database synchronization Interests.

  There are two kinds of situations: 1) Interest satisfied by receiving ChronoSync Data packets (which means database changed by another database node); 2) Interest satisfied by local application data (which means there is new content generated by the local database node).

- **Number of Generated Recovery Interest**

  Number of Generated Recovery Interest is the total number of recovery Interest packet generated by all database nodes. This kind of packet is only generated when database node receives an unknown Interest packet.

- **Number of Satisfied Recovery Interests**

  Number of Satisfied Recovery Interests is the total number of recovery Interests satisfied. This metric represents how many times the database changes are caused by the recovery process.

- **Number of Generated Something New Interests**

  Number of Generated Something New Interest is the total number of Something New Interest packet generated by all database nodes. This kind of packet is only generated when the local database node generates new content.

- **Number of Satisfied Something New Interests**

  Number of Satisfied Something New Interests is the total number of satisfied Something New Interests. This metric indicates the number of times the database changed by receiving Something New Interest packets.

To evaluate the impact of network density on the data synchronization, two different scenarios have been created, which were detailed in section 6.2.

Figure 6.1 shows the synchronization time in both sparse and density scenarios. For each scenario, twenty simulations were run. The bar graph shows the average synchronization time and the vertical line is the 95% confidence interval.

Firstly, we figure out that, the sparse network needs more time to finish synchronization in both scenarios. This is easily understood because nodes have more connection opportunities in the dense network. Therefore, the data transmit chance is greater when compared with sparse networks.

Secondly, IChronoSync consume less time to finish synchronization in all the scenarios. What's more, IChronoSync decreasing about 83% of the synchronization time while Chronosync decrease 62% when the simulation changed from sprase network to dense network. Therefore, IChronoSync

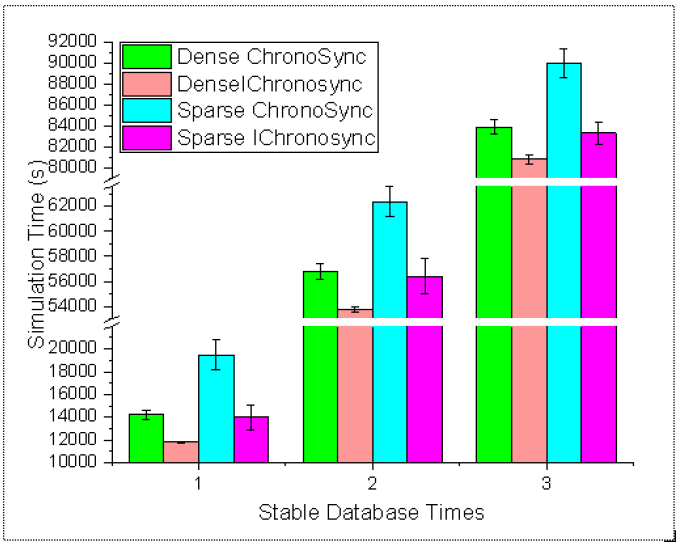


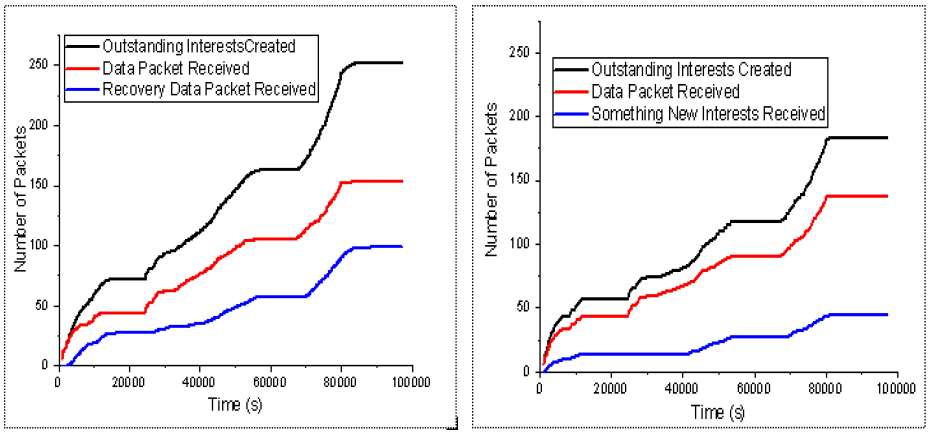

FIGURE 6.1: Time to Synchronize in Dense and Sparse Scenarios

has a better performance especially in dense network. This is mainly due to the fact that IChronoSync decreases the interaction time of the recovery process by using only one interaction with the database to finish the state synchronization. However, ChronoSync needs to request and wait for the reply from other database nodes when conducting recovery. Since data exchange opportunity is precious in ND-DT network, this change brings great improvement in the performance.

The total number of packets generated and satisfied in the dense scenario is shown in Figure 6.2. We can figure out that the number of Outstanding Interest Satisfy (Data packet received in figure ) plus the Recovery Interest Satisfy (recovery Data packet received or Something New Interest received in IChronoSync) is approximately equal to the Number of Outstanding Interest Created. Sometimes Number of Outstanding Interest Created may be a little bigger than the sum. This occurs when the recovery process (or Something New process) is working. One recovery Data packet (*Something New Interest* packets) can carry several changes in a single packet.

The results have shown that IChronoSync generates 27% fewer packets than ChronoSync. In Chronosync, any database node that receives an unknown database digest generates a recovery packet. In our improved solution, only the nodes who cause database change generate packet. Due to the intermittent connection and the long delay of ND-DT network, nodes request recovery synchronization frequently. Therefore, our IChronoSync has a better performance in the aspect of saving bandwidth.

Figure 6.3 shows the result of synchronization time with 20 database nodes. Apparently, both ChronoSync and IChronoSync need more time to finish the synchronization. In the first and second simulation tests, the additional time require to finish the database synchronization was lower with the improved solution than with ChronoSync. For instance in the first test, ChronoSync require additional average time of 20000 to 15000 seconds to finish synchronization, but the improved solution require only 3000




(a) ChronoSync (b) Improved ChronoSync

FIGURE 6.2: Interest and Satisfy Packets Created in Dense Scenario

seconds. In the third synchronization test, our improved solution required five hundred seconds more, but still finished the synchronization first than ChronoSync. However, the confidence interval significantly increased in both protocols. This means that there is a large dispersity in the results. It's not difficult to understand why this occured. ND-DT network nodes move randomly and packet transmission between database nodes become complex. Some database nodes may become out of manage during the synchronization time. Each result shown in figure 6.3 is the time when all database nodes finish the synchronization. Therefore, the results present some discrepancies.

## 6.4 Discussion of Results

In this section, the IChronoSync protocol is evaluated and compared with the ChronoSync from the simulation results.

IChronoSync needs less time to finish the synchronization in both dense and sparse scenarios. It has a better performance especially in dense scenario. This is mainly due to the characteristics of ND-DT networks. ND-DT network is well-known for intermittent connections, unexpected mobility and long transmission delays.

Nodes may be divided into different groups due to the intermittent connections and synchronize respectively. Nodes from different groups do not recognize each other's digest. Moreover, the long delays and out of order transmission can also cause other *New Outstanding Interest* arrive first than *Data* packet. This two situations cause the Recovery (Chronosync) Something New (IChronosync) process working.

IChronosync can finish data synchronization by using only one *Something New* interaction. It could be more useful when there is few communication opportunity. Therefore, The IChronoSync consumes less time to finish synchronization tasks in all scenarios. What's more, in three database scenarios, IChronoSync decreasing about 83% of the synchronization time while Chronosync decreases 62% when changed from sparse network to

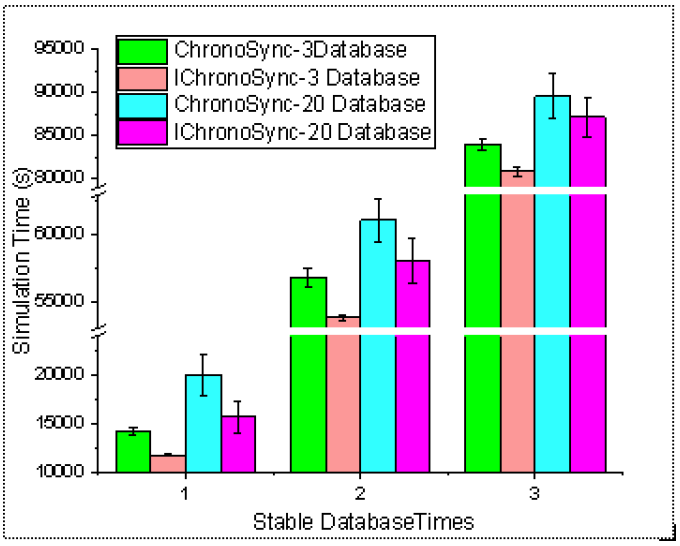



FIGURE 6.3: Time to Synchronize when Database Nodes Increases

dense network. In IChronoSync, only the database nodes that generate content creates a *Something New Interest* packet. Therefore, IChronoSync create 27% fewer packets than ChronoSync. This characteristic helps to save the bandwidth, making it avaiable to often communications packets.

Therefore, we conclude that IChronoSync is more suitable for distributed database synchronization in ND-DT networks than ChronoSync.

# Chapter 7

# Conclusion and Future Work

## 7.1 Conclusion

The distributed database synchronization problem is always a hot research topic in computer networks since it could minimize data inconsistency and help improve data accessibility.

This work contributed to improve the database synchronization in ND-DT networks.

ND-DT network is a new and promising future network architecture. It integrates the current popular Delay Tolerant Network (DTN) with the Named Data Network (NDN) by using Named Data on top of DTN, that is to say, it implements DTN scenarios in Named Data Network. Just like DTN, ND-DT network may suffer intermittent connections and long delays during communications.

Data inconsistency always exists in ND-DT networks. Therefore, the database synchronization becomes particularly important. The distributed databases in ND-DT network are maintained by a group of fixed and moving nodes. In our experiments, databases are supported in fixed nodes.

We reviewed the set reconciliation and file synchronization problems, which provided the theoretical concepts for distributed database synchronization. A few set reconciliation methods, including Log-based, Naïve Approach, Characteristic Polynomial, Bloom Filters, Approximate Reconciliation Trees, and Invertible Bloom Lookup Table methods were explained.

Then, we analyzed several existing file synchronization algorithms and solutions on the Internet and solutions for NDN and DTN. We selected a popular and interesting protocol (ChronoSync) in NDN and implemented this protocol in our ND-DT network. Finally, we made some improvements on ChronoSync to adapt better to ND-DT networks.

To evaluate ChronoSync and Improved ChronoSync protocols, we implemented them in ICONE, an ND-DTN simulator, and tested them in different scenarios. We first evaluated their performance in dense and sparse networks, and then we increased the database nodes. For each scenario, we run twenty times simulation with different seeds and calculated the average results and corresponding confidence intervals.

The result of three database nodes shows that the improved ChronoSync consumes less time to finish synchronization task in all scenarios. What's more, IChronoSync decreasing about 83% of the synchronization time while ChronoSync decrease 62% when simulation changed from sparse network to dense network. Therefore, IChronoSync performs better even in the dense network. What's more, the improved solution generates

27% fewer packets than ChronoSync. Although the confidence interval increased significantly in both protocols when increasing the Database nodes to twenty, the improved ChronoSync still require less time to finish synchronization. Therefore, improved ChronoSync performs better than ChronoSync.

## 7.2 Future Work

This work can only be viewed as the beginning of a research on distributed database synchronization in ND-DT network. Many works could be done in the future.

As referred in Chapter 5, the synchronize Interests packets are sent to all nodes that meet this message, while the mobility of ND-DT network nodes always has some regular rules. For example, cars must drive on roads, and tram can only move on the rails.

The storage capacity of nodes is limited and the data transmission opportunities are quite precious in ND-DT networks. Besides, redundant data transmission wastes the power and storage space of node. In the next, we can improve synchronization process by multicasting Interest packet to a group of nodes, and not to the entire network.

We can also migrate the implemented algorithms to devices such as mobile phones, laptop, desktop and servers and use them in a real prototype application.

# Appendix A

# Default Settings for Simulation

```
#
# Default settings for the simulation
#
Scenario.name = Scenario.name = default_scenario
Scenario.simulateConnections = true
Scenario.updateInterval = 0.1
# 21600s == 6h
# 43200s == 12h
# 86400s == 1d
# 172800s == 2d
# 259200s == 3d
# 345600s == 4d
Scenario.endTime = 97200
# Define 8 different node groups
Scenario.nrofHostGroups = 8
NDNRouter.secondsInTimeUnit = 30
NDNRouter.sizeHigh = 350
NDNRouter.sizeLow = 250
#
# Interface-specific settings:
# type : which interface class the interface belongs to
# For different types, the sub-parameters are interface-specific
# For SimpleBroadcastInterface, the parameters are:
# transmitSpeed : transmit speed of the interface (bytes per second)
# transmitRange : range of the interface (meters)
# "Bluetooth" interface for all nodes
btInterface.type = SimpleBroadcastInterface
# Transmit speed of 2 Mbps = 250kBps
btInterface.transmitSpeed = 250k
btInterface.transmitRange = 10
# High speed, long range, interface for group 4
highspeedInterface.type = SimpleBroadcastInterface
highspeedInterface.transmitSpeed = 10M
highspeedInterface.transmitRange = 100
# Group-specific settings:
# groupID : Group's identifier. Used as the prefix of host names
# nrofHosts: number of hosts in the group
# movementModel: movement model of the hosts (valid class name
from movement package)
```

```
# waitTime: minimum and maximum wait times (seconds) after
reaching destination
# speed: minimum and maximum speeds (m/s) when moving on a path
# bufferSize: size of the message buffer (bytes)
# router: router used to route messages (valid class name from routing
package)
# activeTimes: Time intervals when the nodes in the group are active
(start1, end1, start2, end2, ...)
# msgTtl : TTL (minutes) of the messages created by this host group,
default=infinite
## Group and movement model specific settings
# pois: Points Of Interest indexes and probabilities (poiIndex1, poiProb1,
poiIndex2, poiProb2, ... )
# for ShortestPathMapBasedMovement
# okMaps : which map nodes are OK for the group (map file indexes),
default=all
# for all MapBasedMovent models
# routeFile: route's file path - for MapRouteMovement
# routeType: route's type - for MapRouteMovement
# Common settings for all groups
Group.movementModel = ShortestPathMapBasedMovement
Group.router = PIFP
Group.bufferSize = 50M
Group.waitTime = 0, 120
# All nodes have the bluetooth interface
Group.nrofInterfaces = 1
Group.interface1 = btInterface
# Walking speeds
Group.speed = 0.5, 1.5
# Message TTL of 300 minutes (5 hours)
Group.msgTtl = 500
Group.initialEnergy = 10000
Group.scanEnergy = 0.1
Group.transmitEnergy = 0.2
Group.scanResponseEnergy = 0.1
Group.baseEnergy = 0.01
EnergyLevelReport.granularity = 1.0
Group.nrofHosts = 6
#
# group1 group2 group3 are database nodes
Group1.groupID = p
Group1.bufferSize = 400M
Group1.nrofHosts = 1
Group1.SynGroupID=DBGroup1001
Group1.nrofApplications = 1
Group1.application1 = ChronoSyncImprove
ChronoSyncImprove.type=ChronoSyncImprove
Group1.movementModel = StationaryMovement
#Dababase1's Location
Group1.nodeLocation = 1295,705
Group1.nrofInterfaces = 1
```

```
Group1.interface1 = btInterface
Group2.groupID = p
Group2.bufferSize = 400M
Group2.nrofHosts = 1
Group2.SynGroupID=DBGroup1001
Group2.nrofApplications = 1
Group2.application1 = ChronoSyncImprove
ChronoSyncImprove.type=ChronoSyncImprove
Group2.movementModel = StationaryMovement
#Dababase2's Location
Group2.nodeLocation = 3074,385
Group2.nrofInterfaces = 1
Group2.interface1 = btInterface
Group3.groupID = p
Group3.bufferSize = 400M
Group3.nrofHosts = 1
Group3.SynGroupID=DBGroup1001
Group3.nrofApplications = 1
Group3.application1 = ChronoSyncImprove
ChronoSyncImprove.type=ChronoSyncImprove
Group3.movementModel = StationaryMovement
#Dababase1's Location
Group3.nodeLocation = 2308,1862
Group3.nrofInterfaces = 1
Group3.interface1 = btInterface
# group4 specific settings
#group4 (pedestrians) specific settings
Group4.groupID = p
Group4.nrofHosts = 39
Group4.bufferSize = 400M
Group4.SynGroupID=DBGroup1002
Group4.okMaps = 1
Group4.movementModel=ShortestPathMapBasedMovement
#group5 (pedestrians) specific settings
Group5.groupID = p
Group5.nrofHosts = 39
Group5.bufferSize = 400M
Group5.SynGroupID=DBGroup1002
Group5.okMaps = 1
Group5.movementModel=ShortestPathMapBasedMovement
#group6 tram's specific settings
Group6.groupID = t
Group6.bufferSize = 100M
Group6.SynGroupID=DBGroup1002
Group6.movementModel = MapRouteMovement
Group6.routeFile = data/tram4.wkt
Group6.routeType = 2
Group6.waitTime = 10, 30
Group6.speed = 3, 5
Group6.nrofInterfaces = 2
Group6.interface1 = btInterface
```

```
Group6.interface2 = highspeedInterface
Group6.nrofHosts = 2
Group6.initialEnergy = 15000
#group7 tram's specific settings
Group7.groupID = t
Group7.bufferSize = 100M
Group7.SynGroupID=DBGroup1002
Group7.movementModel = MapRouteMovement
Group7.routeFile = data/tram10.wkt
Group7.routeType = 2
Group7.waitTime = 10, 30
Group7.speed = 3, 5
Group7.nrofHosts = 2
Group7.initialEnergy = 15000
#group8 car's specific settings
Group8.groupID = c
# cars can drive only on roads
Group6.okMaps = 1
# 10-50 km/h
Group8.speed = 2.7, 13.9
Group8.bufferSize = 50M
Group8.SynGroupID=DBGroup1002
Group8.nrofHosts = 38
#
## Message creation parameters
# How many event generators
Events.nrof = 1
Events1.class = ExternalEventsQueue
Events1.filePath = ChronoSyncmessage.txt
Events1.prefix = [A]
#
## Movement model settings
# seed for movement models' pseudo random number generator
(default = 0)
MovementModel.rngSeed = 1
# World's size for Movement Models without implicit size (width,
height;
meters)
MovementModel.worldSize = 4500, 3400
# How long time to move hosts in the world before real simulation
MovementModel.warmup = 1000
# MovementModel seeds used in batch Model
MovementModel.rngSeed = [1;2;3;4;5;6;7;8;9;10;11;12;13;14;15;16;17;18;19;20]
#
## Map based movement -movement model specific settings
MapBasedMovement.nrofMapFiles = 4
MapBasedMovement.mapFile1 = data/roads.wkt
MapBasedMovement.mapFile2 = data/main_roads.wkt
MapBasedMovement.mapFile3 = data/pedestrian_paths.wkt
MapBasedMovement.mapFile4 = data/shops.wkt
## Reports - all report names have to be valid report classes
```

```
#
# how many reports to load
Report.nrofReports = 1
# length of the warm up period (simulated seconds)
Report.warmup = 0
# default directory of reports (can be overridden per Report with output
setting)
Report.reportDir = reports/
# Report classes to load
Report.report1 = NamedDataReport
## Default settings for some routers settings
## Optimization settings – these affect the speed of the simulation
## see World class for details.
Optimization.cellSizeMult = 5
Optimization.randomizeUpdateOrder = true
## GUI settings
# GUI underlay image settings
GUI.UnderlayImage.fileName = data/helsinki_underlay.png
# Image offset in pixels (x, y)
GUI.UnderlayImage.offset = 64, 20
# Scaling factor for the image
GUI.UnderlayImage.scale = 4.75
# Image rotation (radians)
GUI.UnderlayImage.rotate = -0.015
# how many events to show in the log panel (default = 30)
GUI.EventLogPanel.nrofEvents = 100
```

# Bibliography


[1] Augustin Chaintreau et al. “Impact of human mobility on opportunistic forwarding algorithms”. In: *IEEE Transactions on Mobile Computing* 6.6 (2007), pp. 606–620.

[2] Gareth Tyson, John Bigham, and Eliane Bodanese. “Towards an information-centric delay-tolerant network”. In: *Computer Communications Workshops (INFOCOM WKSHPS), 2013 IEEE Conference on*. IEEE. 2013, pp. 387–392.

[3] Paulo Duarte et al. “A Probabilistic Interest Forwarding Protocol for Named Data Delay Tolerant Networks”. In: *International Conference on Ad Hoc Networks*. Springer. 2015, pp. 94–107.

[4] Forrest Warthman et al. *Delay-and disruption-tolerant networks (DTNs)*. A Tutorial. v. 0 Interplanetary Internet Special Interest Group. July 2012.

[5] Kevin Fall. “A delay-tolerant network architecture for challenged internets”. In: *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*. ACM. 2003, pp. 27–34.

[6] Stephan Olariu and Michele C Weigle. *Vehicular networks: from theory to practice*. Crc Press, 2009.

[7] Kevin Fall, Wei Hong, and Samuel Madden. *Custody transfer for reliable delivery in delay tolerant networks*. Tech. rep. Citeseer, July 2003.

[8] Tracy Camp, Jeff Boleng, and Vanessa Davies. “A survey of mobility models for ad hoc network research”. In: *Wireless communications and mobile computing* 2.5 (2002), pp. 483–502.

[9] Frans Ekman et al. “Working day movement model”. In: *Proceedings of the 1st ACM SIGMOBILE workshop on Mobility models*. ACM. 2008, pp. 33–40.

[10] Yue Cao and Zhili Sun. “Routing in delay/disruption tolerant networks: A taxonomy, survey and challenges”. In: *IEEE Communications surveys & tutorials* 15.2 (2013), pp. 654–677.

[11] Amin Vahdat, David Becker, et al. *Epidemic routing for partially connected ad hoc networks*. Tech. rep. CS-200006: Duke University, Apr. 2000.

[12] Thrasyvoulos Spyropoulos, Konstantinos Psounis, and Cauligi S Raghavendra. “Spray and wait: an efficient routing scheme for intermittently connected mobile networks”. In: *Proceedings of the 2005 ACM SIGCOMM workshop on Delay-tolerant networking*. ACM. 2005, pp. 252–259.

[13] John Burgess et al. “MaxProp: Routing for Vehicle-Based Disruption-Tolerant Networks.” In: *INFOCOM*. Vol. 6. 2006, pp. 1–11.

[14] Anders Lindgren, Avri Doria, and Olov Schelén. "Probabilistic routing in intermittently connected networks". In: *ACM SIGMOBILE mobile computing and communications review* 7.3 (2003), pp. 19–20.

[15] Alexander Afanasyev et al. "Named Data Networking". In: *ACM SIGCOMM Computer Communication Review* 44.3 (2014), pp. 66–73.

[16] Van Jacobson et al. "Networking named content". In: *Proceedings of the 5th international conference on Emerging networking experiments and technologies*. ACM. 2009, pp. 1–12.

[17] Lixia Zhang et al. "Named data networking (ndn) project". In: *Relatório Técnico NDN-0001, Xerox Palo Alto Research Center-PARC* (2010).

[18] Cheng Yi et al. "A case for stateful forwarding plane". In: *Computer Communications* 36.7 (2013), pp. 779–791.

[19] Yek Loong Chong and Youssef Hamadi. "Distributed Log-based Reconciliation". In: *Proceedings of the 2006 conference on ECAI 2006: 17th European Conference on Artificial Intelligence August 29–September 1, 2006, Riva del Garda, Italy*. IOS Press. 2006, pp. 108–112.

[20] David Eppstein et al. "What's the difference?: efficient set reconciliation without prior context". In: *ACM SIGCOMM Computer Communication Review*. Vol. 41. 4. ACM. 2011, pp. 218–229.

[21] Yaron Minsky, Ari Trachtenberg, and Richard Zippel. "Set reconciliation with nearly optimal communication complexity". In: *IEEE Transactions on Information Theory* 49.9 (2003), pp. 2213–2218.

[22] Burton H Bloom. "Space/time trade-offs in hash coding with allowable errors". In: *Communications of the ACM* 13.7 (1970), pp. 422–426.

[23] Andrei Broder and Michael Mitzenmacher. "Network applications of bloom filters: A survey". In: *Internet mathematics* 1.4 (2004), pp. 485–509.

[24] Li Fan et al. "Summary cache: a scalable wide-area web cache sharing protocol". In: *IEEE/ACM Transactions on Networking (TON)* 8.3 (2000), pp. 281–293.

[25] John Byers, Michael Mitzenmacher, and Jeffrey Considine. *Fast approximate reconciliation of set differences*. Tech. rep. Boston University Computer Science Department, 2002.

[26] John W Byers et al. "Informed content delivery across adaptive overlay networks". In: *IEEE/ACM Transactions on Networking (TON)* 12.5 (2004), pp. 767–780.

[27] Ralph C Merkle. "Protocols for Public Key Cryptosystems." In: *IEEE Symposium on Security and privacy*. Vol. 122. 1980.

[28] Michael T Goodrich and Michael Mitzenmacher. "Invertible bloom lookup tables". In: *Communication, Control, and Computing (Allerton), 2011 49th Annual Allerton Conference on*. IEEE. 2011, pp. 792–799.

[29] Andrew Tridgell. "Efficient algorithms for sorting and synchronization". PhD thesis. Australian National University, Apr. 2000.

[30] Zhenkai Zhu and Alexander Afanasyev. "Let's chronosync: Decentralized dataset state synchronization in named data networking". In: *2013 21st IEEE International Conference on Network Protocols (ICNP)*. IEEE. 2013, pp. 1–10.

[31] Alexander Afanasyev et al. "The story of chronoshare, or how NDN brought distributed secure file sharing back". In: *Mobile Ad Hoc and Sensor Systems (MASS), 2015 IEEE 12th International Conference on*. IEEE. 2015, pp. 525–530.

[32] Jiaxi Jin et al. "Prioritized data synchronization for disruption tolerant networks". In: *MILCOM 2012-2012 IEEE Military Communications Conference*. IEEE. 2012, pp. 1–8.

[33] Wei Gao et al. "Supporting cooperative caching in disruption tolerant networks". In: *Distributed Computing Systems (ICDCS), 2011 31st International Conference on*. IEEE. 2011, pp. 151–161.

[34] Ari Keränen, Jörg Ott, and Teemu Kärkkäinen. "The ONE simulator for DTN protocol evaluation". In: *Proceedings of the 2nd international conference on simulation tools and techniques*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering). 2009, p. 55.

[35] Paulo Alexandre Gomes Duarte. "Dados nomeados para redes tolerantes a atrasos". MA thesis. University of Minho, Oct. 2014.