Washington University in St. Louis

Washington University Open Scholarship

McKelvey School of Engineering Theses & Dissertations

McKelvey School of Engineering

Summer 8-15-2019

Decoupling Information and Connectivity via Information-Centric Transport

Hila Ben Abraham
Washington University in St. Louis

Follow this and additional works at: https://openscholarship.wustl.edu/eng_etds

Part of the Computer Engineering Commons, and the Computer Sciences Commons

Recommended Citation

Ben Abraham, Hila, "Decoupling Information and Connectivity via Information-Centric Transport" (2019). *McKelvey School of Engineering Theses & Dissertations*. 468.
https://openscholarship.wustl.edu/eng_etds/468

This Dissertation is brought to you for free and open access by the McKelvey School of Engineering at Washington University Open Scholarship. It has been accepted for inclusion in McKelvey School of Engineering Theses & Dissertations by an authorized administrator of Washington University Open Scholarship. For more information, please contact digital@wumail.wustl.edu.

WASHINGTON UNIVERSITY IN ST.LOUIS

School of Engineering & Applied Science
Department of Computer Science and Engineering

Dissertation Examination Committee:
Patrick Crowley, Chair
Roger Chamberlain
Roch Guerin
Raj Jain
Brian Kocoloski
Christos Papadopoulos

Decoupling Information and Connectivity via Information-Centric Transport

by

Hila Ben Abraham

A dissertation presented to
The Graduate School
of Washington University in
partial fulfillment of the
requirements for the degree
of Doctor of Philosophy

August 2019
St. Louis, Missouri

© 2019, Hila Ben Abraham

Table of Contents

List of Figures v

List of Tables viii

Acknowledgments ix

Abstract xii

Chapter 1: Introduction 1

1.1 Motivation 1

1.2 Why is This a Hard Problem? 3

1.2.1 The Conflicted Role of Forwarding Strategies 5

1.3 Our Approach to the Solution 7

1.4 Contributions 8

1.5 Organization 10

Chapter 2: Forwarding in ICNs 12

2.1 ICN Background 12

2.1.1 NDN Forwarding Plane 14

2.2 Forwarding Strategies 15

2.2.1 Decomposing Forwarding Strategy Mechanisms 17

2.2.2 The Underspecified Role of Forwarding Strategies 19

2.3 On the Role of Forwarding Strategies 21

2.3.1 The Information and Connectivity Planes 21

2.3.2 From HTTP to ICN 22

2.3.3 The Forwarding Strategy's Role 23

2.4 Forwarding Strategies in Related Work 24

Chapter 3: Information-Centric Transport 28

3.1 Decoupling Information from Connectivity 28
3.2 Information-Centric Transport 30
3.3 Challenges with In-Network Transport 33
3.3.1 Scalability 33
3.3.2 Marking and Classification 34
3.3.3 Security and Trust Considerations 37
3.3.4 ICT Summary 38
Chapter 4: Data Synchronization 41
4.1 Sync in ICNs 42
4.1.1 Sync as Information-Centric Transport 43
4.2 Related Works 44
4.3 ICT-Sync 52
4.3.1 ICT-Sync Synchronization Model and API 53
4.3.2 ICT-Sync Protocol 54
4.3.3 The Intermediate ICT-Sync Component 56
4.3.4 Mapping Data Names to UIDs 58
4.3.5 ICT-Sync Evaluation 59
4.3.6 ICT-Sync Summary 66
4.4 iSync: Synchronizing Namespaces With Invertible Bloom Filters 68
4.4.1 IBF Background 68
4.4.2 iSync Synchronization Model 70
4.4.3 iSync's Protocol and Data Structure 71
4.4.4 iSync Evaluation 80
4.5 Conclusions 91
Chapter 5: Push Notifications 93
5.0.1 ICN Push in Related Works 96
5.1 ICT for Push Notifications 96
5.1.1 Abstractions for Push Notifications 97
5.1.2 Push Notifications Vs. Data Synchronization 99
5.2 ICT-Notify 100

5.2.1 ICT-Notify API ... 102
5.2.2 ICT-Notify Protocol and Namespace Design ... 104
5.2.3 Evaluation ... 107
5.3 Conclusions ... 114
Chapter 6: Data Partitioning ... 116
6.1 The Problem of Partitioned Data ... 118
6.1.1 Where is the Failure? ... 119
6.2 ICT for Fetching Distributed Partitioned Data ... 123
6.2.1 Considerations for ICT Mechanisms ... 125
6.3 Application Abstraction: Controlling Strategy Retransmissions ... 129
6.3.1 Retransmission Decoupling ... 130
6.3.2 Retransmission Differentiation ... 131
6.3.3 Empirical Results ... 134
6.4 Conclusions ... 143
Chapter 7: Conclusions ... 145
7.1 Future Research Directions ... 147
7.2 Publications of Dissertation Work ... 149
References ... 150

List of Figures

Figure 2.1: NDN Forwarding Information Base. 14

Figure 2.2: NDN Building Blocks as Described in [83] 16

Figure 3.1: ICT as a two-component transport mechanism in ICN: A library API resides at the end-point, and an intermediate process resides in the network 39

Figure 4.1: ChronoSync Sync Tree 47

Figure 4.2: ChronoSync Digest Log 47

Figure 4.3: CCNx Sync Tree 50

Figure 4.4: CCNx Sync Timeline. 51

Figure 4.5: ICT-Sync Synchronization Model 53

Figure 4.6: ICT-Sync API 55

Figure 4.7: ICT-Sync Simultaneous Updates 57

Figure 4.8: Tested NDN Topology 60

Figure 4.9: Mapping Topology onto Physical ONL Hardware 61

Figure 4.10: Average Sync Times over Low Loss Rates 62

Figure 4.11: Average Sync Times over High Loss Rates 63

Figure 4.12: Percentage of Fetched File of Different Communications with Alternating Links 65

Figure 4.13: Scaled Up ONL Topology 66

Figure 4.14: Traffic Overhead of ICT-Sync 67

Figure 4.15: iSync Data Synchronization Model 71

Figure 4.16: Hierarchical Synchronization Data Structure. 72

Figure 4.17: Alice's Hierarchical Data Structure. 73

Figure 4.18: iSync Timeline Example. 75

Figure 4.19: Periodical Synchronization. 78

Figure 4.20: Local and Global IBFs for a Sync Collection. 79

Figure 4.21: Impact of Number of Components in File Name on Synchronization Time. 83

Figure 4.22: File Insertion and Recovery Time for iSync Protocol. 84

Figure 4.23: Impact of File Size on Time Cost (left) and Ratios of CCNx Sync vs. iSync (right). 85

Figure 4.24: Traffic Overhead for Various File Sizes. 86

Figure 4.25: Number of bytes vs. File Size. 87

Figure 4.26: Average Synchronization Time of iSync and CCNx Sync in Networks of Various Topology Types (with max recorded results on top of each bar). 88

Figure 4.27: Recovery Time vs. Number of Items after 10% of items have been deleted randomly 91

Figure 5.1: ICT-Notify API for the Two Abstractions 103

Figure 5.2: ICT-Notify Example 108

Figure 5.3: Tested NDN Topology 109

Figure 5.4: Mapping Topology onto Physical ONL Hardware 109

Figure 5.5: Average Push Latency over Different Loss Rates (ms) 110

Figure 5.6: Push Time with no SEEP (sec) 112

Figure 5.7: Name Size (Bytes) 113

Figure 5.8: Push Time (ms) of Simultaneous Notifications 114

Figure 6.1: Distributed Producers with Partitioned Data 117

Figure 6.2: Distributed Database Use Case 119

Figure 6.3: FIB Tables of R2, R3 and R4 120

Figure 6.4: NACK problem 132

Figure 6.5: Emulated NDN testbed 135

Figure 6.6: Unsatisfied Interest Rates for Different Link Loss Rate: (a) best-route. (b) best-route-r, ncc-r, and ncc. Note the very different Y axis ranges 139

Figure 6.7: End Hosts Traffic over Time with best-route 140

Figure 6.8: End Hosts Traffic over Time with best-route-r 141

Figure 6.9: WU Traffic over Time with best-route 142

Figure 6.10: WU Traffic over Time with best-route-r 143

List of Tables

Table 4.1: Large Scale Comparison of ICT-Sync and ChronoSync 66

Table 4.2: Sync Properties as an ICT 92

Table 5.1: Push Properties as an ICT 115

Table 6.1: Percentage of Satisfied Requests over Different NDN Strategies 121

Table 6.2: Multiple Producers Results Summary 137

Table 6.3: Fetching Partitioned Data Properties as an ICT 144

Acknowledgments

I want to express my sincere gratitude to my doctoral advisor, Professor Patrick Crowley, for his patient guidance and valuable and constructive suggestions during my studies at Washington University. Professor Crowley has always been supportive and continuously provided insightful feedback and useful critiques to my research work. I am grateful for the opportunity to work with him on the NDN project, and to learn how very complex problems can be approached by asking simple questions.

I would also like to thank my other committee members, Prof. Roch Guerin, Raj Jain, Roger Chamberlain, Christos Papadopoulos, and Brian Kocoloski, for their valuable feedback on my work.

I am particularly grateful for the assistance given by the staff members of the Applied Research Lab (ARL) at Washington University, John DeHart and Jyoti Parwatikar. They have been very supportive, academically and personally, and offered continuous support by brainstorming problems, setting up testing environments, and running experiments. It was a pleasure working with them.

I want to thank James Ballard for the help he provided editing my papers, and for always providing valuable suggestions on writing.

I am grateful to have the opportunity to intern at Cisco, and I want to thank my Cisco mentors David Oran, and Ralph Droms. During my internship at Cisco, I was introduced to related research works conducted in industry, learned more about the different ICN architectures, and gained practical experience working on internal ICN code.

I want to thank my fellow students at ARL, Haowei Yuan, Jason Barnes, and Adam Drescher. Their support along the way was significant and is greatly appreciated.

Finally, I am grateful to my parents, my spouse, my incredible boys and the rest of my family members, for their endless encouragement and support.

In addition to a research gift from Cisco. The following National Science Foundation grants support this work: CNS-1040643, CNS-1345282, CNS-1719366, and CNS-1629807.

Hila Ben Abraham

Washington University in Saint Louis

August 2019

Dedicated to my parents, Mordechay Taibi and Elana Sayag Taibi.

ABSTRACT OF THE DISSERTATION

Decoupling Information and Connectivity via Information-Centric Transport

by

Hila Ben Abraham

Doctor of Philosophy in Computer Engineering

Washington University in St. Louis, 2019

Professor Patrick Crowley, Chair

The power of Information-Centric Networking architectures (ICNs) lies in their abstraction for communication — the request for named data. This abstraction was popularized by the HyperText Transfer Protocol (HTTP) as an application-layer abstraction, and was extended by ICNs to also serve as their network-layer abstraction. In recent years, network mechanisms for ICNs, such as scalable name-based forwarding, named-data routing and in-network caching, have been widely explored and researched. However, to the best of our knowledge, the impact of this network abstraction on ICN applications has not been explored or well understood. The motivation of this dissertation is to address this research gap.

Presumably, shifting from the IP's channel abstraction, in which two endpoints must establish a channel to communicate, to the request for named data abstraction in ICNs, should simplify application mechanisms. This is not only because those mechanisms are no longer required to translate named-based requests to addresses of endpoints, but mainly because application mechanisms are no longer coupled with the connectivity characteristics of the channel. Hence, applications do not need to worry if there is a synchronous end-to-end path between two endpoints, or if a device along the path switches between concurrent interfaces for communication. Therefore, ICN architectures present a new and powerful promise to applications — the freedom to stay in the information plane decoupled from connectivity.

This dissertation shows that despite this powerful promise, the information and connectivity planes are presently coupled in today's incarnations of leading ICNs by a core architectural component, the forwarding strategy. Therefore, this dissertation defines the role of forwarding strategies, and it introduces Information-Centric Transport (ICT) as a new architectural component that application developers can rely on if they want their application to be decoupled from connectivity. When discussing the role of ICT, we explain the importance of in-network transport mechanisms in ICNs, and we explore how those mechanisms can be scalable when generalized to provide broadly-applicable application needs.

To illustrate our contribution concretely, we present three group communication abstractions that can evolve into ICTs: 1) Data synchronization of named data. This abstraction supports applications that want to maintain data consistency over time of a group's shared dataset. 2) Push-like notifications for the latest named data. This abstraction supports applications that want to quickly notify and be notified about the latest content that was produced by a member(s) in the group. And 3) distributed named data fetching when the content is partitioned. This abstraction supports applications that their named data is partitioned and distributed in the group, and the names of content items in a partition cannot be generalized and hierarchically represented using one partition name.

For each ICT, we provide examples of known applications that can use it, we discuss different mechanisms for implementation, and we evaluate selected implementations. We show how by relying on an ICT instead of a forwarding strategy, the tested applications can maintain sustainable communication in connectivities where IP tools fail or do not work well.

Chapter 1

Introduction

The purpose of this dissertation is to introduce *Information-Centric Transport (ICT)* as a new architectural component that can effectively decouple the information and connectivity planes in Information-Centric Networking architectures (ICNs). In this Chapter, we explain how these planes are coupled in today's incarnations of ICNs, such as Named Data Networking (NDN) [83] and CICN [55], and we discuss why this coupling does not comply with the ICN abstraction — the request for named data [39].

1.1 Motivation

Although the success of the largest IP network – the Internet – is not underestimated, advocates for ICNs, argue that the underlying telephony-inspired IP abstraction, in which pairs of addressed endpoints must establish a connection to communicate (i.e., a telephone call), does not sustainably comply with the requirements of today's Internet. Those requirements include better information dissemination support when multiple consumers request the same piece of content (e.g., a chunk of a popular video), and better security and trust models

that shift from securing channels to securing data. Therefore, to adequately address IP's challenges, ICNs use another abstraction — the request for named data [39].

While this abstraction seems to achieve the requirements of today's Internet, its impact on applications has not been explored or well understood. The first objective of this dissertation is to address this research gap, and to explore if, and how, the change in the network abstraction impacts applications running on top of it.

In ICNs, applications simply request named data, and the network finds and retrieves this data. As a result, the application does not need to worry where the data is located, or what are the network characteristics between the consumer and the data provider (an endpoint producer, a repository or an intermediate cache). Thus, the request for named data abstraction promises to simplify applications, who no longer need to establish a channel with another endpoint, and to keep them in the *Information Plane*, decoupled from connectivity concerns. In other words, the application can be concerned only with data namespaces and trust identities of data producers and consumers, without worrying where the data is located or how it would be retrieved.

However, our system-oriented research approach, in which we explored NDN applications and tried to run them on the NDN testbed [53] and on the Open Network Lab (ONL)[76], revealed that ICNs do not live up to their promise, and that despite the power of the request for named data abstraction, ICNs applications are coupled with connectivity concerns. Specifically, in our exploration we found that a core architectural component, the forwarding strategy, couples applications to the details of the network connectivity in an unsustainable way [14, 15, 16].

In short [1], as presently designed, a forwarding strategy implements mechanisms concerning local connectivity characteristics, and a forwarding strategy can be paired with an application namespace to address the application needs. If an application name is paired with a specific forwarding strategy, and this strategy implements specific connectivity mechanisms, then the application is coupled with the characteristics of this connectivity. Hence, the forwarding strategy component couples the information and connectivity planes.

As a result, ICN applications tend to be complex, and their implementation requires a deep understanding of network mechanisms [14, 41]. This coupling not only makes ICN applications hard to develop, but also does not comply with the request for named data abstraction that promises to decouple applications from the details of connectivity.

This dissertation suggests to address the problem by providing ICN applications with a different architectural component to rely on in order to satisfy their application-level needs. We name this new architectural component Information-Centric Transport (ICT), and we define it as both an abstraction and a broadly applicable communication mechanism. Our goal is to show that by relying on ICTs, applications do not have to rely on forwarding strategies, and therefore they are no longer have to be coupled with connectivity mechanisms. Hence, applications can operate solely in the information plane, dealing only with namespaces and the trust relationship.

1.2 Why is This a Hard Problem?

The request for named data abstraction was introduced by the World Wide Web (WWW), and it was popularized by the HyperText Transfer Protocol (HTTP). The use and popularity of HTTP has given names to the world's data in the form of URLs, and has created several

[1] More details are provided in Chapter 2

generations of web-based applications whose function is organized around requesting data by name.

But there is a fundamental difference between HTTP and ICNs. While ICNs aim to use the request for named data as their a general-purpose network layer protocol, HTTP is an application-layer protocol, linked explicitly to the underlying channel-based TCP/IP protocol. To comply with the channel abstraction of the IP paradigm, an HTTP-based application relies on transport layer protocols that establish a host-to-host channel and translate name-oriented requests to IP addresses.

But ICN architectures rely on a different network abstraction and eliminate the notion of host-to-host transport. ICNs use the same abstraction in both application and network layer, and therefore applications can run directly on top of the network layer. In theory, this presents great benefits that include application simplification, because no translation is needed between the application and network layers, and because applications can simply request named data to get the content. In practice, implementing and running applications on top of ICNs is not that simple.

To understand why, consider that ICNs, just like IP, recognize that different applications may have different requirements for their communication. Therefore ICNs support different application needs by allowing applications to pair their content names with different forwarding mechanisms through the forwarding strategy component. However, there is presently no specification of what application requirements may be, and there is no understanding of what communication mechanisms should be provided to ICN applications. Moreover, the role of forwarding strategies has not been explicitly specified.

In Chapter 2, we provide detailed background information about ICN forwarding and the forwarding strategy component. In short, a forwarding strategy is a network-layer component in

ICNs that implements mechanisms for hop-by-hop packet forwarding. ICNs provide multiple forwarding strategies, each implements different forwarding mechanisms to provide different forwarding behaviors. For instance, the best-route strategy in NDN forwards requests for named data on one best path determined by the routing protocol; The multicast strategy in NDN simply multicasts requests for named data to all available upstreams; The ncc strategy in NDN is an adaptive forwarding strategy that tries different paths and adaptively switches between them according to real-time application and network performance; And the ASF strategy in NDN implements a combination of the ncc and best-route strategies;

In theory, pairing an application namespace with a specific forwarding strategy, a capability known as the named-based strategy selection, allow ICNs to implement general-purpose information-oriented mechanisms to support different application needs. However, we show in the next subsection and in Chapter 2 that in practice, every forwarding strategy must also implement specific connectivity-related mechanisms. Therefore, the name-based strategy capability couples applications to the details of connectivity.

As a result, the great promise presented by the ICN abstraction is not effectively satisfied, and ICN applications tend to be complex because they must implement mechanisms that are directly concerned with the characteristics of the network connectivity.

1.2.1 The Conflicted Role of Forwarding Strategies

To understand the complexity of the problem, which is also the key to the solution, we must discuss what brought the forwarding strategy component to play such a conflicted role.

First, consider that an ICN application simply sends a request for named data using an *Interest packet*, and that the network finds and retrieves the named data to the application

in a *Data packet* [2]. Moreover, consider that forwarding in ICNs is done hop-by-hop, and Data packets follow the same but reversed path of their Interest packets. When forwarding an Interest, every router must determine answers to questions such as 1) If routing rules permit multiple equivalent next-hops, which one should be chosen? 2) If a packet times out, should it be retransmitted? If so, should it be retransmitted on the same or another next hop(s)?

Clearly, the answers to such questions rely on connectivity characteristics, such as where in the network the node is located (e.g., core vs. access), the number and type (e.g., wired vs. wireless) of next hop links available, and the dynamics of the network (e.g., static vs. mobile). Therefore, a router must consider the local connectivity characteristics when forwarding Interests.

Second, our work has shown [14] that another set of forwarding questions are intrinsic to information flow, and therefore, are meaningful for applications. For instance, 1)Should an Interest be broadcast? 2)For how long should an Interest be saved before being dropped or retransmitted? 3) What if a new interest for the same name prefix comes along? Should it replace the previous one or be buffered as well? 4)Can multiple Data packets for the same Interest be aggregated into one?

Presently, the forwarding strategy component is the only architectural component that can address such information-oriented questions, and therefore it is the only component that can consider application-level preferences when forwarding Interests and Data packets. Pairing an application namespace with a specific forwarding strategy was designed to support different forwarding behaviors that can address a variety of application needs.

[2] Detailed background information provided in Chapter 2.

1.3 Our Approach to the Solution

It may appear that information and connectivity can be decoupled simply by voiding ICNs' capability to pair a forwarding strategy with an application name. However, this would also eliminate ICN's capability to support different application-level preferences in the network. While this may seem a minor tradeoff, and while we agree that the decoupling of applications from forwarding strategies is the first step towards a solution, we also argue that some degree of in-network application-oriented mechanisms is needed to fully decouple information and connectivity.

To see why, consider that ICNs' properties, such as the stateful forwarding plane and the symmetric form of Interest-data exchange, natively enable in-network mechanisms for resilient data communications. When an Interest packet is not satisfied with a Data packet, a router can immediately respond to guarantee continuous information flow.

But how exactly should a router respond to a network failure? Do all applications benefit from the same in-network mechanism for resilient data communication? We argue that the answer is no, and that the correct in-network mechanism for resilient data communication depends on the application needs. For example, in the case of intermittent links, a file-sharing application could benefit from in-network retransmissions of all the Interests for the file's chunks. However, a real-time video streaming application may prefer an intermediate router to buffer and retransmit only a window of its Interests and prefetch the following segments of the video for resilient user experience.

The frequently cited "Named Data Networking" paper [83] describes the forwarding strategy component as "the key to NDN's resiliency and efficiency", because it is the only architectural component that can implement the required in-network mechanisms for robust data

communication. Therefore, disabling the name-based strategy selection capability eliminates ICNs' capability to support various information-oriented mechanisms for resilient data communication, and transfers those mechanisms to the applications at the endpoints. In this case, an ICN application, similar to HTTP-based applications, would be forced to respond to connectivity events, and would not be decoupled from the details of connectivity.

One can argue that ICNs can resolve this problem by providing application support in the form of ICN libraries [51], similarly to the support provided to IP applications by socket APIs, and by implementing mechanisms that respond to connectivity events in the context of those libraries. This way, applications could be decoupled from both forwarding strategies and connectivity mechanisms. While our approach to the solution highly relies on endpoint libraries, we show in Chapters 4-6 that in some scenarios, in which the network links are highly intermittent and lossy, or when there is never a Synchronous End to End Path (SEEP) between a consumer and the producer, endpoint libraries by themselves are not sufficient to solve the problem.

To address these scenarios of intermittent connectivity, our approach to the solution consists of three steps: 1) decoupling applications from forwarding strategies, 2) supporting application needs by relying on endpoint libraries, and 3) allowing the implementation of in-network information-oriented mechanisms in ICNs outside of the forwarding strategy component.

1.4 Contributions

We reiterate that the motivation of this dissertation is to explore how the change in the network abstraction, from the channel abstraction in IP to the request for named data abstraction in ICNs, impacts applications. In our exploration, we found that ICN applications are coupled with connectivity mechanisms through the forwarding strategy component. We

argue that the problem lies in ICN having one architectural component that both reconciles application and network considerations and manages the interests of both applications and network operators.

Therefore, this dissertation proposes to decouple applications from forwarding strategies, and to add a new ICN component that manages the interests of applications and decouples them from connectivity mechanisms. We do this by specifying, for the first time, the role of forwarding strategies in ICNs, and by proposing a new abstraction for information-oriented mechanisms, named Information-Centric Transport (ICT) [3].

We define an ICT as both an abstraction and a communications mechanism designed to support a specific, but broadly applicable, set of application requirements. An ICT consists of an endpoint library for application developers, and an intermediate service for network operators. While forwarding strategies implement connectivity-oriented mechanisms, ICTs implement information-oriented mechanisms.

We show how the placement of ICT libraries at the endpoints, and the placement of intermediate ICT mechanisms in the network can simplify the tested applications, and can provide sustainable communications in the tested topologies. This, without deploying or relying on any application-specific code in the network, and while keeping the intermediate ICT mechanisms transparent to the application.

It is important to note that our work does not formally prove that ICT can solve the problem and decouple applications from connectivity in any given topology or connectivity. Additionally, our work does not argue that ICT is the only approach for the solution or the most comprehensive one. Instead, our system-oriented work shows how ICT abstractions can comply with the request for named data abstraction, and how ICT mechanisms can simplify ICN

[3] In Chapter 3, we explain how ICN transport is different from IP transport

applications by addressing connectivity challenges in a general-purpose way. Moreover, our work demonstrates how in-network information-oriented mechanisms can be used by ICNs outside of the forwarding strategy component, and decouple applications from connectivity in the tested intermittent environment.

To illustrate our contribution concretely, we explored three abstractions that can become ICTs: Data synchronization abstraction for applications with sync requirements; Push notification abstraction for applications with quick, latest data push requirements; And an abstraction for fetching distributed data for applications when the data is partitioned among distributed producers.

For each ICT abstraction, we 1) define the broadly applicable requirements it implements, 2) discuss potential applications that can use it, 3) explore mechanisms that can both implement the application abstraction and comply with the request for the named data abstraction, 4) demonstrate how they provide communications for their applications under a range of connectivity scenarios, where IP tools and native NDN applications fail or do not work well.

1.5 Organization

The remainder of this dissertation is organized as follows: In Chapter 2, we provide relevant background details of ICN forwarding, we decompose the mechanisms of existing forwarding strategies, and we explain how the underspecified role of the forwarding strategy component led it to coupling applications with the details of connectivity mechanisms. As a first step towards a solution, we define the architectural role of the forwarding strategy component in ICNs.

In Chapter 3, we explain why decoupling applications from the forwarding strategy component is not enough to decouple applications from the details of connectivity, and we propose Information-Centric Transport (ICT) as a new architectural component. We define ICT, we explain how it can sustainably decouple information and connectivity, and we explore the implications of in-network information-oriented mechanisms in ICNs.

In Chapters 4-6, we explore three ICN abstractions that can evolve into ICTs, discuss different mechanisms to implement each abstraction, and evaluate selected implementations.

In Chapter 7, we conclude our work, and discuss future work.

Chapter 2

Forwarding in ICNs

2.1 ICN Background

In the last few years, as interest in future information-centric network (ICN) architectures has increased, we have witnessed continuous growth in research efforts focusing on the design and development of ICN architecture prototypes. Two on-going research projects, Named Data Networking (NDN) [83] and Content-Centric Networking (CCN) [55, 56], provide two corresponding software prototypes: the NDN forwarder (NFD) [5, 54], developed by the NDN research group, and CCNx [52], initially developed by PARC and later acquired by Cisco and named CICN. To simplify the discussion, this subsection provides the background information on NDN and its forwarder.

NDN was introduced as one of the four projects funded by the NSF's Future Internet Architecture program [71]. In NDN, data is represented by a namespace, similar to the representation of URIs in HTTP. A namespace can represent any type of data, such as a file name, an application state, a chat message, or a video chunk. Unlike the host-centric IP

paradigm, NDN takes the content-centric approach, and a content item is cached, requested, and retrieved by specifying its namespace.

The NDN architecture introduces two types of packets: the *Interest* packet, used to request named data, and the *Data* packet, used to retrieved the named data. To request a content item, a consumer expresses an Interest packet that carries the name of the requested content. Upon receipt of an Interest packet, the NDN node looks in its Content Store (CS) to see if the Interest can be satisfied by a previously cached Data packet. If the CS does not hold the content, the node forwards the Interest to its next hop by matching the Interest's name with the entries in its Forwarding Information Base (FIB) table, and by selecting the FIB entry with the longest prefix match.

When forwarding the Interest, the node notes the name of the forwarded Interest and its incoming face in the local Pending Interest Table (PIT). Once the Interest packet arrives at a node that can satisfy the requested name, either from the CS or a local application, the node replies with a Data packet that carries the same name and contains the requested data. The Data packet is sent back to the consumer on the reverse path of the Interest packet by using the information stored in the PIT.

If more than one consumer asks for the same named content, the NDN node notes all the incoming faces in the PIT entry of the requested name. This way, NDN supports data dissemination and sends the same Data packet to all the consumers that requested it. Additionally, NDN nodes on the forwarding path(s) cache a copy of the Data packet in their Content Store (CS) as they forward it back to the consumer, and use this copy to satisfy future requests for the same name.

To detect loops, every Interest packet carries a nonce generated by the application. When an incoming Interest packet contains the same name and nonce as previously recorded in

the PIT, the Interest is detected as duplicated and is dropped by the router. In recent implementations of the NDN forwarder, the router responds with an upstream negative acknowledgment (NACK) when a duplicated Interest is detected.

Each NDN packet is encoded in a Type-Length-Value (TLV) format that provides a dynamic platform for adding new fields to either the Interest or the Data packet. In Chapter 6, we propose a solution for decoupling strategy retransmissions that uses this flexible encoding by adding a new TLV to the interest packet.

2.1.1 NDN Forwarding Plane

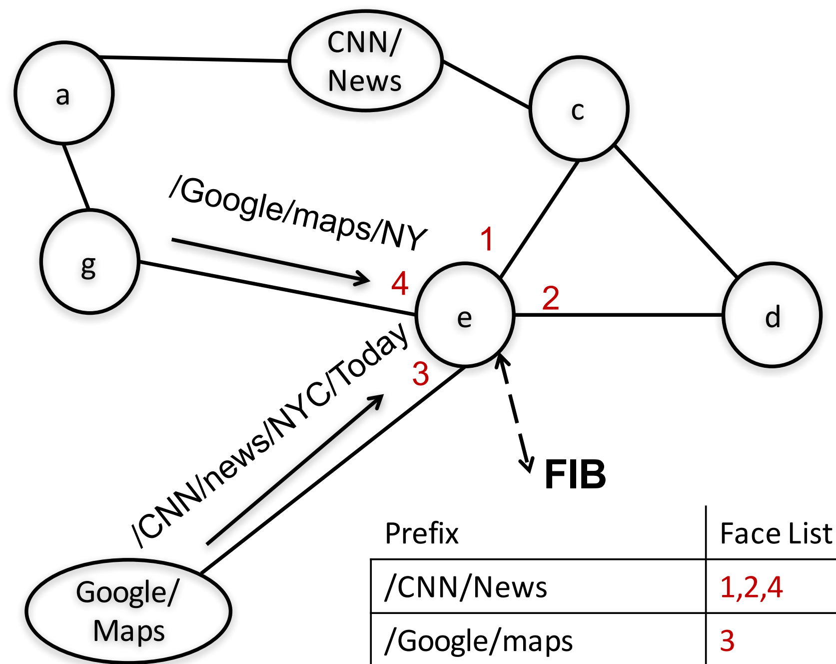


Figure 2.1: NDN Forwarding Information Base.

As in the IP architecture, the NDN router uses the information in its FIB table to determine the packet's next hop. However, while a FIB entry in the IP architecture consists of an IP

address and one port as the next-hop, each entry in the NDN FIB consists of a namespace and a list of possible faces. Each face represents an upstream interface to a possible next hop, which can be a remote NDN entity or a local application. When the list of faces consists of only one face, the Interest is forwarded on this face. However, when that list contains more than one face, the forwarding plane needs to decide on which face(s) to forward the Interest. Therefore, when forwarding an Interest, the NDN router performs two operations: 1) A FIB lookup to find the longest prefix match of the requested name. 2) The selection of one or more face(s) to be the Interest's next hop(s).

Figure 2.1 shows a network and the FIB table in node e. In this example, when e receives an Interest for /Google/maps/NY, it can forward it only on face number 3 towards the Google/Maps node. However, e can choose from a list of faces when receiving an Interest for /CNN/news/NYC/Today. In this case, after finding the correct FIB entry, e follows the forwarding strategy paired with /CNN/news namespace to decide on which face(s) the Interest should be forwarded.

2.2 Forwarding Strategies

ICNs present a unique architectural component, named the *forwarding strategy*, which is usually referred to as the forwarding layer. This architectural component is frequently described as the component that decides how to forward an Interest when a FIB entry contains multiple next hops. However, as we discuss in this chapter, the forwarding strategy component does much more.

The frequently cited "Named Data Networking" paper [83] describes this core architectural component as "the key to NDN's resiliency and efficiency". And in fact, in the past years, the forwarding strategy module has been demonstrated to be a key architectural component in

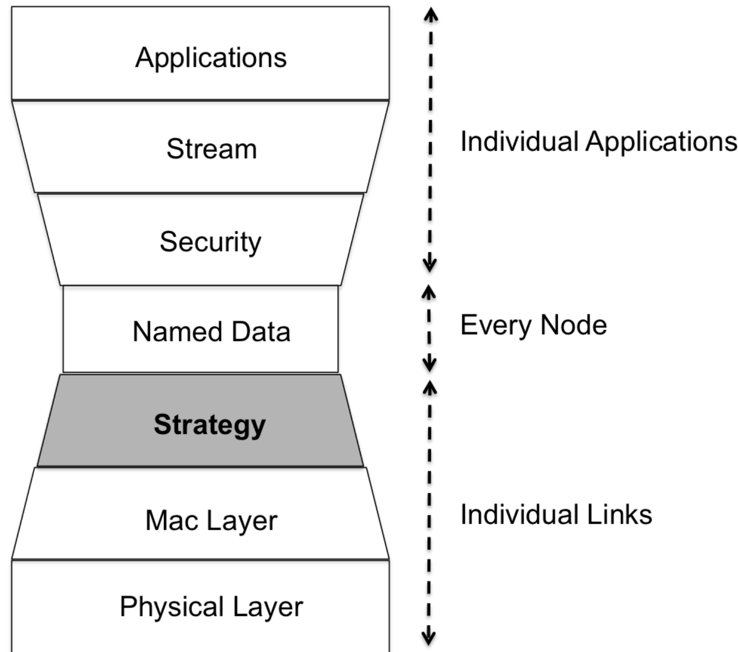


Figure 2.2: NDN Building Blocks as Described in [83]

the implementation of different applications and the design of different network connectivities [8, 26, 45, 59]. Figure 2.2 shows the building blocks of NDN, with the strategy layer residing between the MAC layer and the Named Data layer [83].

However, despite the central role played by the forwarding strategy, its architectural role has not been well understood, and it remains an underspecified piece in the ICN architecture. As a result, there is no clear determination of what mechanisms strategies should implement, or who chooses a forwarding strategy. We address these questions in Section 2.3.

Presently, ICNs allow different forwarding strategies to co-exist, and therefore, support a range of different forwarding algorithms. Each strategy relies on a different set of input considerations to implement a specific forwarding behavior. For instance, ad-hoc networks

can use a dynamic routing-less forwarding strategy [33], while a more traditional routing-based forwarding strategy can be used in core networks [45, 78]. Therefore, ICNs support a range of different forwarding algorithms for different connectivities.

In this dissertation, we argue that while the forwarding strategy component is crucial for the success of ICNs, its underspecified role couples applications to the details of connectivity, and breaks the promise of the request for named data abstraction.

2.2.1 Decomposing Forwarding Strategy Mechanisms

Before we discuss the conflicted role of forwarding strategies, we decompose the current operations of known forwarding strategies in ICNs into two core mechanisms: face selection and failure response.

Next Hop Selection

When the FIB entry consists of multiple faces, the forwarding strategy must decide on what face an Interest should be sent. When selecting the Interest next hop, the strategy may choose to send the Interest on a single face, a subset of faces, or all available faces. Moreover, the forwarding strategy may send Interests on different faces to probe the performance of upstreams, or to discover data providers.

An essential attribute of a forwarding strategy is its adaptation to changes. In NDN, a data packet is forwarded on the reverse path of the Interest packet. Therefore a strategy can record the performance of each face to learn if it works and how well it performs. Then, it can use this knowledge to update the face's *rank* and improve future next-hop decisions.

When choosing the face's next hop, a forwarding strategy can elect to exclusively rely on the face's *cost*, provided by external services such as a routing protocol, or the face's *rank*, provided by internal feedback monitored by the forwarding strategy, or on any combination of the two. For instance, the best-route simply relies on routing costs, the ncc strategy relies on the internal measurements of the strategy, and the access strategy relies on both.

When relying on internal strategy feedback, every strategy is free to choose the metric of the collected feedback according to its goals. For instance, the face rank can be determined according to the upstream round-trip-time (RTT), the number of hops to the producer, or the face successful delivery rate.

Failure Response

After selecting the Interest next hop(s), the forwarding strategy must decide how to react when the Interest is not satisfied within a specific amount of time. For each forwarded Interest, the NDN router initiates a timer for the period in which it expects to receive back a Data packet. When a Data packet is received within this period, the packet is sent back to the consumer by following the information kept in its PIT. At this time, the forwarding strategy can use the time passed to determine the face's rank.

There are three possible outcomes when the timer expires prior to the reception of a Data packet: 1) The strategy **drops** the Interest packet. 2) The strategy **retransmits** the packet on the same or a different face(s). 3) The strategy replies with a NACK packet to the previous hop [24]. Presently, a forwarding strategy can choose the algorithm it follows when implementing its timers. While some strategies, such as best-route, use a fixed-interval timer, other strategies, such as ncc and ASF, use exponential-interval algorithms when setting an Interest's timer.

The failure response mechanism can critically affect the application correctness. Thus, the application developer must be aware of the approach taken by the forwarding strategy to best decide how to design the application namespace and how to handle the application's retransmissions. We discuss this in depth in Chapter 6.

2.2.2 The Underspecified Role of Forwarding Strategies

To determine the strategy used, the router uses a per-packet name-based strategy selection, which means that the strategy is selected dynamically according to the incoming packet name. The current design of the NDN and CICN software prototypes gives the application developer the option to pair a forwarding strategy with its application namespace, and therefore to control the way the application packets are forwarded in the network. Thus, the application developer gets the freedom to choose an existing forwarding strategy, or, alternatively, to develop a new one to satisfy its application-specific needs. This is known as the name-based strategy selection capability. While this capability might present new opportunities to ICN application developers, it also poses new and significant challenges.

First, while an application developer can pair a forwarding strategy with its application namespace in the localhost, forwarding strategies are assigned within nodes interior to the network by the operators of those specific nodes. This difficulty can be mitigated in isolated environments where application developers also operate the entire network, such as with the global NDN testbed [53]. However, in general, it is not feasible for an application developer to choose an in-network mechanism.

One can argue that an application's strategy choice can be propagated everywhere in the network by the routing protocol or another network mechanism. However, we argue that

the true challenge does not lie in propagating an application's choice, but in letting the application choose a forwarding strategy.

Although it was never clearly specified by the ICN community, a strategy must implement its 'face selection' and 'failure response' mechanisms with respect to local connectivity. We provide a thorough explanation for that argument in the next subsection. However, even without relying on a clear known statement, we witness numerous forwarding strategies that were designed to address specific connectivity needs. For instance, the access strategy was designed to forward Interests at the edge, the best-route strategy was designed to forward Interests in the core network, and the strategies described in [11, 33, 63] were designed for Wireless networks.

If an application pairs its name with a specific forwarding strategy, and if this forwarding strategy implements specific connectivity mechanisms, then the application is coupled with the network connectivity. Hence, the problem is not the distribution of an application's strategy choices, but it is the coupling of information and connectivity by the forwarding strategy component. Therefore, neither application developers nor network operators can optimally select a forwarding strategy, because the right choice depends on knowledge that neither party alone possesses in its entirety.

To summarize this discussion, the lack of clear definition of the forwarding strategy role leads the two contradictory assumptions: 1) A forwarding strategy can be paired with an application namespace, and 2) A forwarding strategy can address desired connectivity characteristics. If both assumptions are correct, then an application that chooses a forwarding strategy is coupled with the connectivity the strategy implements. Hence, a forwarding strategy couples both applications and network mechanisms, and therefore introduces challenges for an application developer who 1) cannot guarantee that the same strategy is used

everywhere along the path(s) to the producer, 2) must modify its application whenever the strategy changes due to changes in connectivity, and 3) should potentially develop different versions of its application to address different network mechanisms. In other words, coupling the mechanisms of both the network and applications in the forwarding strategy module does not scale and makes it hard to develop ICN applications.

We argue that strategies cannot serve both applications and connectivity, because if they do, then they couple applications to the details of connectivity. Therefore, the name-based strategy selection capability does not comply with the request for named data abstraction. In order to solve this conflicted role of the forwarding strategy component, we first seek to provide a clear definition of its role.

2.3 On the Role of Forwarding Strategies

To clearly define the forwarding strategy's role, we look at the ICN architecture as a whole and identify the abstraction it aims to provide and how it provides it.

2.3.1 The Information and Connectivity Planes

We define that ICN applications operate in the *Information Plane*, and the network operates in the underlying *Connectivity Plane*. To see why, consider that ICN applications ask for data by name, and the network must find and retrieve that data. But how does the network do that? Unlike IP, ICN is channel-less and consists of different hop-by-hop mechanisms to find requested data. In practice, there is always an actual, real-world connectivity present — e.g., the collection of one or more connectivity options, including WiFi links, Ethernet links, TCP channels, BT, and UDP multicast. Because the properties of these different

connectivities differ so widely, the best choice of mechanism in any given circumstance may depend strongly on the specific connectivity available.

2.3.2 From HTTP to ICN

In the TCP/IP model, an HTTP name is translated to an IP address at the endpoint. However, in the information-centric approach, the name used by an ICN application is also used by the ICN network as the identifier of core network operations, including name-based Interest forwarding, name-based routing, and name-based caching. The great benefit of using the same identifier in both the information and connectivity planes is that the application is not coupled with the connectivity properties of the channel. By contrast, in the absence of sophisticated middleware which greatly increases application logic, HTTP-based applications can break when 1) devices change IP addresses, 2) devices have and try to use multiple concurrent interfaces, and 3) Internet connectivity is lost. It is true that IP-based applications can implement mechanisms to respond to such events, but they are still coupled with the events' occurrences.

Unlike HTTP, ICN uses the request for named data abstraction in the network layer rather than the application layer. Suppose that a consumer application asks: "What is the content for this name?" Here, the consumer does not specify where the content can be found, or how to get it. In theory, the consumer's question can be answered simply by broadcasting it until someone replies with the requested named data. However, broadcasting is an expensive network operation, and flooding the network is not a scalable solution. Therefore, this abstraction must somehow be translated by the network to a practical mechanism that can efficiently find and retrieve the requested content. In other words, ICN must somehow move from the information plane to the connectivity plane.

We illustrate the process of moving from the information plane to the connectivity plane by asking two more questions: 1) "Who might have the content for this name?" and 2) "What is the most efficient way to retrieve it"? These two questions should be answered differently, according to the characteristics of the network and the nature of the underlying links.

We argue that the strategy module answers these two questions in the context of its specific network environment, and therefore bridges the information and connectivity planes. Allowing a spectrum of strategies to co-exist under the umbrella of the ICN architecture provides flexible forwarding behavior that can be adapted to the characteristics of the local connectivity. Hence, an application asks "What is the content for this name?" in the information plane, and a strategy relies on a set of input considerations in the connectivity plane when answering the questions of "Who might have the content?" and "How to retrieve it?".

2.3.3 The Forwarding Strategy's Role

We specify the forwarding strategy as the architectural component that bridges the information and connectivity planes in ICN. Moreover, we argue that choosing the right mechanism when moving between the information and connectivity planes — the role of the forwarding strategy — is a critical element in the design of ICN, and what makes ICN operate in both Internet-like infrastructures and dynamic, non-stable topologies where current Internet methods do not work [1, 2, 3, 8, 48].

To be clear, the design and choice of specific mechanisms to bridge the two planes in any given circumstance is a fascinating future problem. However, this chapter focuses on resolving the tensions created because forwarding strategies, as presently defined, reconciles both application and network considerations and manages the interests of both application and

network operators. Therefore, the first step for decoupling information and connectivity is to clearly specify which of the two the strategy component should address.

From our definition of its architectural role, it is clear that every forwarding strategy mechanism must consider network characteristics. Therefore, in order to decouple information and connectivity, we argue that forwarding strategies should not implement information-oriented mechanisms, but should contain only connectivity-oriented mechanisms. When decoupled from application-level mechanisms, forwarding strategies can be safely chosen and deployed by network operators, according to the connectivity they manage.

To summarize this chapter, we argued that the underspecified role of the forwarding strategy component brought it to couple information and connectivity, and as a result to break the ICN promise. To address the problem, we first illustrated the questions ICN must answer when translating its abstraction into a set of practical network protocols. Second, we specified the forwarding strategy component as the one that answers those questions with respect to local connectivity characteristics and therefore bridges the information and connectivity planes in ICN. Third, we argued that forwarding strategies should implement connectivity-oriented mechanisms, and be decoupled from information-oriented mechanisms. Therefore, forwarding strategies should be selected by network operators according to the connectivity they manage, and should not be paired to namespaces by application developers.

2.4 Forwarding Strategies in Related Work

In this subsection, we briefly discuss the mechanisms of forwarding strategies in NFD, CICN, and related research.

The first forwarding strategy implemented in ICN was the *default CCNx* strategy, which is also known as the *ncc* strategy in NFD. In this strategy, the NDN router forwards a received Interest packet on one face and waits for a Data packet to be returned. If the packet arrives within a specific predicted time set by the strategy, usually referred to as the *prediction* timer, then the face is remembered as the "best" face, and it is used to forward future interests of the same name. If the prediction timer expires before the arrival of a Data packet, the strategy retransmits the Interest again to another available face. The CCNx default strategy is distinctive in the way it adjusts its prediction timer. Every time a Data packet is returned on the selected best face, the predicted wait time is adjusted down so that the prediction timer will expire faster the next time. When the Interest is not satisfied within the predicted wait time, the prediction timer is adjusted up. Thus, the strategy tries another available face whenever the prediction timer is too short to allow a successful response from the previously working face. When that happens, the predicted time is adjusted up again to allow the new face to respond with data. Thanks to this mechanism, the strategy timer approaches the actual round trip times after an initial exploration phase. In addition, this mechanism guarantees that other faces will be eventually given a chance to satisfy a namespace.

The *best-route* strategy, also used in NFD 0.4, is the default strategy for new applications and the gateway routers in the NDN testbed [53]. In *best-route*, every Interest packet is forwarded to the cheapest face, which is determined according to the cost assigned by the routing protocol. The named-data link state routing protocol (NLSR) [37] is currently the routing protocol configured to work with the best-route strategy on the testbed. When the face fails to respond on time, the strategy drops the Interest, and the application can choose whether to retransmit the Interest again. The strategy decides whether to suppress or to forward the application retransmission on a different face. This decision is made by a suppression timer set by the strategy. The suppression timer algorithm has changed several

times in recent NFD versions. The *best-route* strategy keeps sending future Interests on the same face as long as that face has the cheapest cost, regardless of its success in returning the requested data. If the face is unresponsive, the routing protocol might delete the face from the FIB table [79].

The *loadsharing* strategy as implemented in CCNx 0.82, follows the same logic and forwards the Interest to the best available face selected according to feedback received in previous transmissions.

The *multicast* strategy, as implemented in NFD 0.4, forwards the Interest packet to all the available faces simultaneously. If there is no available face to forward the packet on, the strategy replies with a NACK packet. This strategy is similar to the *parallel* strategy implemented in CCNx 0.82.

The *access* strategy trades off between the best-route and multicast strategies. It first learns which next hop can satisfy an Interest by multicasting it to all possible next hops. The first upstream face to respond with the content is then remembered and used for future Interests. If the preferred upstream face later fails to satisfy an Interest with a similar name, the Interest's retransmission triggered by the consumer causes the strategy to start a new discovery phase by multicasting the Interest again.

The principles of an adaptive forwarding strategy are discussed in [80], and the details of such a strategy, the *GreenYellowRed* strategy, are described in [78]. A dynamic forwarding mechanism designed to discover temporary copies of content items is presented in [22]. The work in [31] proposes a revised forwarding strategy that can better prevent or detect loops in NDN.

Related works have also explored potential strategies in Wireless networks, such as a strategy for vehicular ad hoc networks in [33], and a set of adaptive forwarding strategies that can use multiple access networks simultaneously in [63].

The work in [57] presents a probability-based adaptive forwarding strategy, including a statistical model to compute strategy retransmission intervals.

While related works explore different mechanisms and approaches for forwarding strategies in ICN, our work is mainly focused on exploring the dynamics between applications and forwarding strategies.

Chapter 3

Information-Centric Transport

In Chapter 2, we identified that the underspecified role of the forwarding strategy component couples applications with the details of connectivity mechanisms, and we explained how this coupling complicates applications and prevents ICNs from fulfilling their promise. As a first step towards a potential solution, we specified the role of the forwarding strategy component. In this Chapter, we take a step forward and discuss how applications could be decoupled from connectivity using a new architectural component — Information-Centric Transport (ICT).

3.1 Decoupling Information from Connectivity

Specifying the role of forwarding strategies in Chapter 2 led to the conclusion that forwarding strategies should not be exposed to applications. However, decoupling applications from forwarding strategies is not enough to decouple information from connectivity, because disabling the name-based strategy selection capability eliminates all in-network information-oriented mechanisms from ICNs. It may not seem to be a problem because, as stated by the

end-to-end principle, applications should always implement their own mechanisms and never trust any network component to do that for them. However, we argue that, to some degree, in-network information-oriented mechanisms are required to truly decouple information and connectivity.

To see why, consider a network failure such as intermittent or lossy links, in which one or more Interest and Data packets are lost. What should ICNs do? There are two options: ICNs can either wait for the application to decide how to deal with the failure, or ICNs can use their stateful forwarding plane and address the failure at the point it was discovered.

If ICNs simply wait for the application to decide what to do, then ICNs' applications are not different from HTTP-based applications. For both types, the application is coupled with network failures, and therefore with connectivity concerns. It is true that an ICN application can use an endpoint library to implement connectivity-related mechanisms, but it would still be coupled with the occurrences of network failures. Moreover, a consumer's mechanisms by themselves are not sufficient to guarantee data resiliency. For instance, consider an intermittent-path scenario, where there is never a Synchronous End-to-End Path (SEEP) between a consumer and the source of the requested content (either a producer or a network cache). In this case, the consumer may never get the requested named data. Therefore, implementing information-oriented mechanisms only at the endpoint does not fully decouple the information and connectivity planes.

The second option, responding to network events at the point of failure, is also challenging. It is true that the stateful ICNs' forwarding plane can be utilized to react to network failures [57, 62], but do all applications benefit from the same in-network mechanisms? Consider the following questions: Is it always right to retransmit an Interest? Is there a point where the data becomes irrelevant to the application and an Interest for a different name would better

serve the application needs? If so, when is this time? What about Data packets? Should ICNs store all previous versions of Data packets for the same name? We argue that different types of applications can provide different answers to such questions. For instance, a video streaming application may want to retransmit only Interests for a specific frame, while a dropbox style application would not have any constraints on the timing, and would always benefit from Interest retransmissions. Moreover, a location-based application may want to receive the latest Data packet, while a chat application would require all Data packets to display all chat messages.

To conclude, we argue that the requirements for information-oriented mechanisms vary between different types of applications. We also argue that in some scenarios, such as lossy links and lack of SEEP, in-network mechanisms are required to fully decouple the information and connectivity planes. Furthermore, as explained in Chapter 2, we also argue that forwarding strategies should not be exposed to applications, and therefore, they cannot be the architectural component that implements information-oriented mechanisms in the network.

This conclusion leads us to discuss Information-Centric Transport (ICT).

3.2 Information-Centric Transport

Our approach to the solution is to add a new architectural component to ICNs, named *Information-Centric Transport (ICT)*, as a component that can address application-level needs, and can implement in-network mechanisms where needed to decouple information and connectivity.

We define ICT to be a communication mechanism that implements information-oriented communication mechanisms, and presents an abstraction for applications. We propose that

ICNs implement a set of ICTs to support different abstractions for different application requirements. Every ICT consists of two components: an API for applications at the endpoint, and an intermediate service that can run on selected devices in the network.

The primary goal of this dissertation is to demonstrate how by relying on ICT abstractions instead of forwarding strategies, applications can achieve different application-level requirements, and control in-network mechanisms for data resiliency while staying in the information plane. To do that, our system-oriented approach implements different application abstractions, and evaluates them on lossy connectivities including lack of SEEP in the network.

To illustrate the concept of ICT, consider how it relates to traditional notions of transport. Existing transport concepts can readily be seen in the IP protocols, which can be viewed as *Connection-Centric Transport*.

- Connection-Centric Transport (CCT): concerned with endpoints and channel characteristics, such as reliability and in-order delivery.
- Information-Centric Transport (ICT): concerned with application abstractions, data names, and the trust relationships between named identities.

The properties of a CCT are channel-based, and CCTs such as TCP and UDP enable applications to meet different reliability requirements. An IP-based application can also implement its own transport mechanisms by following the Application-Level-Framing (ALF) concept [23]. Related work has shown that ICN can provide similar transport mechanisms for applications with CCT requirements [30, 36, 51]. Therefore, it is important to note that ICT does not preclude CCT transport mechanisms for ICN applications. Instead, ICT extends

the concept of transport to include new abstractions for applications that are concerned with namespaces, and that want to stay in the information plane, decoupled from connectivity.

To be clear, the concept of ICT does not guarantee that applications will always operate perfectly under any given connectivity. Instead, it promises that ICNs will provide a set of abstractions for their applications, and that the network will make its best effort to support those abstractions in different connectivities.

Moreover, as we show in Chapters 4-6, in contrast to the simplicity of the request for named data abstraction, implementing ICN applications is not always simple or intuitive. We argue that it is challenging to implement relatively simple applications when the application requirements are not aligned with the consumer-producer pull-based paradigm. The examples discussed in this dissertation include applications that want to push content instead of pulling it, and applications that want to partition their data and distribute the subsets among multiple producers. Therefore, we argue that the goal of ICTs is twofold — one, provide essential abstractions for ICN applications, and two, decouple applications from connectivity.

The intermediate ICT mechanism is deployed by the network operator where connectivity characteristics require it (such as in intermittent links or in dynamic and mobile networks). When deployed in the network, an ICT must address requirements raised by the Information plane, and a forwarding strategy addresses concerns raised by the connectivity plane. Therefore, ICT is information-oriented, while the forwarding strategy is connectivity-oriented. For instance, an ICT can express relevant Interest packets and store data packets if needed for resilient data delivery, and the forwarding strategy can add, remove, or probe potential next-hops in response to connectivity changes.

3.3 Challenges with In-Network Transport

To scope the discussion in this subsection, we reiterate that in some network scenarios, such as the lack of SEEP or when network links are intermittent, in-network information-oriented mechanisms are needed to decouple information and connectivity. Therefore, we defined the ICT component as an architectural component that can implement in-network information-oriented mechanisms. However, this has three main challenges:

1. Scalability: How can an in-network information-oriented mechanism be scalable?
2. Marking and Classification: How can an intermediate ICT mechanism classify the application-level needs?
3. Security and Trust: Can an intermediate transport be trusted?

In this subsection, we discuss these challenges and set the high-level principles for in-network information-oriented mechanisms.

3.3.1 Scalability

The end-to-end principle [61] determines that, for scalability, application-specific features should remain at the endpoints and never reside in the network. Although ICN already maintains an in-network state in its PIT and CS, we argue that to ensure scalability, an ICT should never implement any application-specific mechanisms. Therefore, an intermediate ICT mechanism must be implemented to capture abstractly a specific set of application-level needs, and for scalability reasons, those needs must be shared among different types of applications. In other words, an ICT must implement a primitive mechanism.

We think that a future set of ICT abstractions can address a substantial range of abstract application needs, from purely semantic needs to performance and reliability. However, this dissertation does not define the ultimate set of ICTs in ICNs because, to the best of our knowledge, there is no comprehensive understanding of the needs and requirements of ICN applications. We argue that this set should be finite and kept small for practical deployment, but exploring the different application-level requirements that should be implemented by different ICTs is a rich area for future work. We anticipate that a number of widely useful ICTs will emerge over time.

Once an ICN abstraction has been introduced and an ICT has been implemented, an application can pair its code with the ICT's endpoint library and API. Using an ICT's library results in three outcomes: 1) The application declares that its application-level requirements are aligned with the information-level requirements implemented by the ICT. 2) The ICT library satisfies the application-level needs by implementing information-oriented mechanisms, including sending and receiving Interest and Data packets. 3) The ICT library expresses the broadly applicable application-level requirements, so that an intermediate ICT component, deployed in the network, can identify and classify its packets.

3.3.2 Marking and Classification

The next challenge of in-network transport is marking and classification. While this challenge can be discussed as a scalability concern, we discuss it separately due to the special characteristics of ICN packets.

In the past, packet classification was defined as the process that categorizes packets into flows by following a pre-defined set of rules [13, 28, 34, 35]. In IP, the area of packet classification has been widely researched and explored in the context of Quality of Service (QoS) [60, 77].

But there is a fundamental difference between algorithms for QoS and ICTs. The purpose of QoS is to control and manage network resources by setting priorities for specific types of data. However, the purpose of ICT is to provide different transport mechanisms to different types of packets. In other words, QoS is the ability to provide different priorities to different applications, while ICT is the ability to provide different processing mechanisms for the applications' packets.

Related ICN work suggests adding functions to Interest packets [67], and executing those functions in the network. However, this approach relies on fundamental changes to the CCN and NDN architectures. Adding a Transport layer to ICN packets might be an appropriate long-term solution, however in this work, we aim to use existing NDN code and tools to demonstrate the ICT concept without modifying the architecture or the packet format.

Another option for marking and classifying transport requirements in the network is to decode them in the application payload. This approach is known as the Application-Level-Framing (ALF) approach [23], and it was suggested in the context of ICNs in [51]. However, this approach cannot be taken by in-network transport because it breaks the ICN trust model. In ICNs, every packet is signed and validated at the endpoints, thus, extracting transport requirements from the payload would require in-network nodes to decode and validate signed packets. For this reason, adding transport information to the application presents efficiency challenges and violates the ICN trust model.

To comply with the request for named data abstraction, we propose that the transport requirements will be encoded in the Interest and Data names. Hence, an intermediate ICT can understand the application-level requirements from the Interest and Data names. But encoding transport requirements in the name can be a challenging task because it either requires that an intermediate ICT mechanism would recognize arbitrary application names,

or determines that an application name should be designed to meet the expectations of an intermediate ICT mechanisms.

Clearly, an intermediate ICN node cannot understand arbitrary application names, and therefore the application must encode its transport requirement in a general-purpose way that can be recognized by an intermediate ICT mechanism. However, if the application must design its data names to meet the expectations of an intermediate ICT mechanism, then the application must be modified every time the intermediate mechanism is changed or updated. This solution is not scalable and does not comply with the request for named data abstraction that serves as the motivation of this dissertation.

Therefore, we argue that it is the ICT library at the endpoint that should encode the application's transport requirements into the Interest's name, and that it should do it in a general-purpose way that is transparent to the application but known to the intermediate ICT component. Every change to the intermediate ICT should be handled either by an update to the ICT's endpoint library or by having an intermediate ICT mechanism support earlier versions of endpoint encoding.

The second challenge of encoding transport requirements in the name is to determine what exactly should be encoded. An intuitive way to mark ICT requirements in the name is to add an ICT name component as a name prefix. However, in Chapter 4-6, we show that simply marking the ICT name as a name component is not always sufficient to provide a context-based, but broadly applicable, transport mechanism. For instance, an ICT-Sync prefix can classify the application's packets as sync-based, but it can not provide the required context to determine which names have already been synced and which have not. Similarly, an ICT-Notify prefix can classify the application's packets as notification-based, but it does

not provide the information required to understand which of the latest Data packets should be pushed.

Therefore, we argue that there is no definitive answer to the question of what should be encoded as transport information, and that the answer lies in the type and the semantics of the abstraction the ICT aims to provide. For instance, as discussed in Chapters 4-6, the transport requirements for sync-based applications can be encoded as the state of the shared dataset, while the transport requirements for notification-based applications can be encoded as the timestamps of Data packets.

We conclude this subsection by saying that in order to mark and classify an application's transport requirement for in-network processing, the endpoint ICT library must translate the arbitrary application name into an ICT name that provides broadly-applicable context, and is not application-specific.

3.3.3 Security and Trust Considerations

A core principle in the design of an ICT is to address broadly-applicable application needs. And as discussed in subsection 3.3.2, those needs should be represented in the Interest's name. Thus, an intermediate ICT component is not required to look into the application payload. As a result, ICT can maintain the application's trust model and is not required to decrypt or validate Data packets.

However, in some scenarios, an intermediate ICT must express Interests and construct Data packets to support the application's transport requirements. For instance, ICT-Sync must express sync packets when it learns about a change in the state of the shared dataset, and ICT-Notify needs to aggregate multiple Data packets into a new one when it receives simultaneous Data packets for the same name.

When an ICT generates a packet, either an Interest or a Data, it must sign it so it can be validated by the endpoint ICT component. Using an intermediate service to sign or encrypt packets leads to key management questions, in which the endpoints must hold a copy of the intermediate's public key. Algorithms for key management in ICNs were proposed in [7, 38, 81]. However, future work should explore the trust implications of an intermediate node signing and encrypting ICN packets.

3.3.4 ICT Summary

To conclude this Chapter, we have proposed *Information-Centric Transport (ICT)* as both an abstraction and a communication mechanism that aims to decouple applications from connectivity and allows applications to operate in the information plane, free from connectivity concerns. To achieve this goal, an ICT provides both endpoint and in-network transport mechanisms by implementing a specific, but broadly applicable, set of well-defined application requirements. Figure 3.1 shows that an ICT consists of two components: an API for applications at the end hosts, and an intermediate service that runs on selected devices in the network.

Although we cannot predict the final set of ICTs in ICNs, the next chapters explore several challenges of existing ICN applications, identify three sets of broadly applicable application needs, and propose ICTs to address them. Chapters 4-6 not only show that ICT libraries address significant implementation challenges, but also show how the intermediate component of an ICT allows applications to stay in the Information plane, decoupled from connectivity and from in-network mechanisms.

Three ICT abstractions are discussed in the following chapters of this dissertation are:

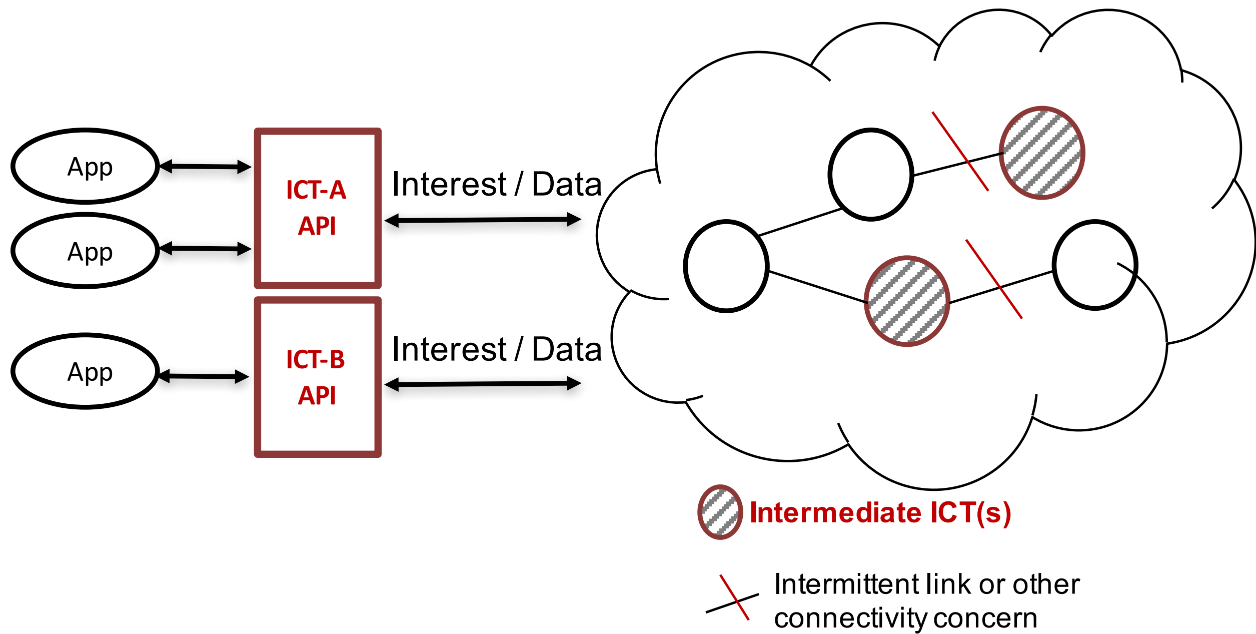


Figure 3.1: ICT as a two-component transport mechanism in ICN: A library API resides at the end-point, and an intermediate process resides in the network

- ICT for data synchronization: this chapter discusses sync abstractions for applications that want to maintain data consistency over time of all the content under a shared namespace. Chapter 4.5 explores different mechanisms to provide data synchronization in NDN and CCN, and discusses their implementation details and evaluation.

- ICT abstractions for distributed applications that want to reliably push their latest data. Chapter 5 explores the challenges in implementing push notifications in ICNs, and more specifically, when multiple parties push simultaneous notifications under the same prefix.

- ICT abstractions for fetching distributed partitioned named data. This abstraction is meant for applications that want to keep their dataset partitioned and distributed among multiple producers, while allowing consumers to reach the right producer without being coupled with routing and forwarding mechanisms.

For every proposed ICT we discuss the following:

- The broadly-applicable requirements the ICT aims to satisfy.
- Examples of applications that can use the ICT's abstraction.
- Known challenges in the area and related works.
- Either the implementation details of an ICT mechanism, or proposed mechanisms. Including the roles of the API and the intermediate ICT components.

Chapter 4

Data Synchronization

This chapter explores data synchronization, usually referred to as sync, as an application abstraction. The communication model of Sync is a group of endpoints, in which every party in the group wants to have a replica of the content produced by the other members of the group. Therefore, the process of data synchronization can be described as the process that provides data consistency of a shared dataset over time, and the dataset is the union of the distributed data items.

Although in this chapter we discuss data synchronization in the context of ICN, the premise of sync is widely used by many of today's popular applications, such as cloud storage, group communication, and media sharing. The files in a Dropbox [73] directory are an example of a synchronized dataset, in which a copy of each file is created and kept up-to-date among the user's devices and the Dropbox server. Another example of a synchronized dataset could be all the messages typed in a chat room. Here, the up-to-date dataset must consist of all messages typed by all the chat participants.

The objective of a distributed synchronization protocol is to send the smallest possible amount of information over the network, hence to avoid sending the entire shared dataset every time it changes, and to keep sync latency low and acceptable.

We decompose the problem of data synchronization into three main tasks: 1) Understanding whether an instance of a shared dataset is up-to-date or out-of-date, 2) Finding the set difference between two or more instances of the shared dataset , and 3) Retrieving the missing items found in the set difference. The second task lies at the heart of every synchronization mechanism.

4.1 Sync in ICNs

In ICNs, data is represented by a namespace, and a namespace can represent any type of content, such as a file name, an application state, a chat message, or a video chunk. Therefore, while a synchronization mechanism in IP must consider the type of the application's data, a synchronization mechanism in ICNs can be generalized to synchronize namespaces instead of content items. Hence, an ICN synchronization mechanism synchronizes a set of shared namespaces, and the application decides if and when to fetch the content associated with the namespace by expressing a regular Interest packet.

In recent years, keeping namespaces synchronized has emerged as a basic service required by many ICN applications, such as Dropbox-style file sharing [4, 46], mobile and ad-hoc vehicular communication [33], chat applications[86], routing protocols[37], and key management services deployed on the NDN testbed [19].

4.1.1 Sync as Information-Centric Transport

To become an ICT, ICNs must implement a sync mechanism, with a clear library API provided to application developers, and an in-network sync mechanism provided to the network operator. While the deployment of the in-network mechanism depends on the network's best-effort policy, an ICT is required to implement one. Applications with requirements to synchronize a shared set of namespaces should be able to use the ICT-Sync API to: 1) Register a dataset name to be synchronized, such as the directory name in a file-sharing application, and 2) Add a name to the shared set of namespaces, such as a new file name. The sync mechanism implemented by the ICT should monitor the participants who registered the same dataset name, and notify the application whenever a new name was added or deleted.

We show how the first two tasks of namespace synchronization, determining if the set is up-to-date and finding the set-difference, can be provided by the ICT, while the third task can be left to the application. This approach allows ICN applications to maintain the knowledge about a shared dataset of names, without having to fetch the entire set of content items. For instance, a chat application on Bob's laptop can find that another participant, Alice, added a chat message named Alice/message10, but can decide not to fetch the content of Alice's message if Bob previously blocked Alice. Moreover, we show in this chapter how the mechanisms that implement the first two tasks heavily rely on the data structure chosen to represent the synchronized set of namespaces, and therefore has a significant impact on the performance of the sync mechanism.

In the remainder of this chapter, we discuss related works, and we describe the details of two sync protocols we designed and implemented: ICT-Sync for the NDN architecture, and iSync for the CCN architecture. While both mechanisms share the same goal of synchronizing a

shared dataset of namespaces, they differ in their dataset representation, and therefore, in the way they identify and reconcile the set-difference. ICT-Sync uses a vector to represent the latest sequenced name of each party, while iSync uses a hierarchical invertible Bloom filter to represent arbitrary names in a shared dataset. By discussing the two different sync mechanisms we present the challenges and tradeoffs of different ICT mechanisms for data synchronization. However, it is important to note that standardizing different sync mechanisms into one global ICT remains a rich area for future work.

1. ICT-Sync: A synchronization protocol for NDN, designed and implemented as a two-component ICT. ICT-Sync uses a list of tuples to represent a set of sequenced namespaces. We evaluated ICT-Sync in a range of connectivities to demonstrate the concept of ICT, and compared its performance to another NDN synchronization protocol, named ChronoSync [85].

2. iSync: a synchronization protocol designed and implemented in earlier versions of CCNx. iSync uses a hierarchical invertible Bloom filter data structure to represent a set of arbitrary namespaces. Hence, the namespaces don't have to be sequenced. We evaluated iSync and compared it to the official synchronization protocol of the CCN architecture, named CCNx Sync [20].

4.2 Related Works

Data synchronization is widely used and plays an important role in traditional and emerging network premises. Different approaches to identify the set-difference are taken by different applications. The well-known rsync [66] was the first to suggest a solution that does not transfer the complete dataset over the network but send only the deltas instead. Rsync is a pairwise algorithm that synchronizes remote files and directories by sending only the

missing file chunks. Rsync discovers the set-difference by calculating the block checksums of a synchronized file on one host, and sending the list of the calculated checksums to the remote host. The remote host goes over its local copy of the file and compares its local checksums to the received list. Then rsync identifies the missing blocks in its local file and requests those blocks from the remote host.

Another common approach to identify the set-difference is a timestamped log. One host notes the changes to its local dataset in a local timestamped log, and therefore the set-difference can be identified by transferring the log notes added after the last synchronization cycle. While log-based synchronization solutions are practical and easy to implement, their performance dramatically decreases when the number of parties scales up, due to the complexity of transferring, parsing, and comparing multiple files in every sync cycle.

Additional research efforts have focused on synchronization protocols such as surveys [6] and on the synchronization of two nodes in scenarios of a small set of differences [75]. Most of the recent works in this field are well-known commercial projects such as BitTorrent Sync service [69], DropBox [73], and Google Drive [74].

In ICNs, the Custodian-Based Information Sharing (CBIS) system [40] was the first implementation of ICN-based sync service. In this early paper, the authors discussed the high-level principles of what later became the foundation of other sync protocols designed to support ICN applications.

The following sync services for ICN have been proposed in related work: ChronoSync [85], CCNx Sync [20], PartialSync[84], and vectorSync [64]. While these protocols can be differentiated by their implementation details, including their namespace design, mechanism, and

data structures, they all follow the same high-level sync goal of providing a continuous synchronization of namespaces. In the following subsections, we discuss the relevant background details of ChronoSync[85] and CCNx Sync [20].

chronoSync

In Section 4.3, we discuss the design and implementation of ICT-Sync, and evaluate it by comparing its performance to ChronoSync. Therefore, in this subsection, we provide the relevant background details of ChronoSync[85].

To represent a set of shared namespaces, ChronoSync uses a digest tree and a log of sequence numbers to keep track of the changes made by each participant. Every participant in the synchronization service is represented by a node in the sync tree, and this node holds the participant digest and its latest sequence number. In other words, the participant node represents the status of the participant dataset. The root of the sync tree holds the name of the dataset, and the digest that represents the state of the dataset. Therefore, two sync trees of the same name are considered up-to-date only if their root digests are equal.

Figure 4.1 shows an example of ChronoSync's digest tree, and figure 4.2 shows the digest log. In this example, Alice, Bob, and Ted participate in a chat room named /chatRoom/wustl, when Alice adds her 18th message. Here, Alice's digest changes to represent Alice's new message, and as a result, the root digest of the entire sync tree changes.

To participate in the synchronization process, a distributed party must first use ChronoSync's API to publish the sync prefix, for instance /chatRoom/wustl. Then, ChronoSync uses this sync prefix, together with the root digest, to notify all the registered parties about a change in their shared set of namespaces.

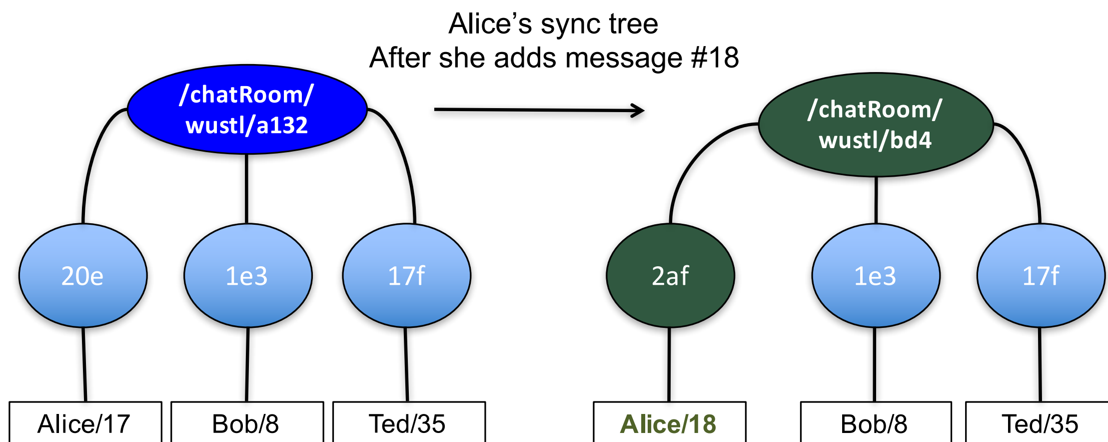


Figure 4.1: ChronoSync Sync Tree

State Digest	Changes
000	NULL
9w3	[Alice's prefix, 1]
...	...
a132	[Bob's prefix, 31], [Alice's prefix, 17]
bd4	**[Alice's prefix, 18]**

Figure 4.2: ChronoSync Digest Log

Periodically, ChronoSync triggers a sync Interest containing the root digest of the local sync tree to notify remote parties about the status of the dataset. Upon receiving a sync Interest, ChronoSync compares the received digest with the local root digest to determine whether its view of the dataset is up-to-date. If the incoming root digest is equal to the local root digest, ChronoSync determines that the set is up-to-date. If the digests are different, then ChronoSync looks for the incoming root digest in its digest log. If it finds it, then it shows that the recipient of the sync Interest has more recent knowledge about the shared dataset, than ChronoSync responds with a Data packet to reconcile the missing names. In our example, when Alice receives a sync Interest with digest 'a132', it would respond with a sync Data packet that consists of Alice/18.

If the incoming digest is not equal to the local root digest, and the incoming digest cannot be found in the digest log, then ChronoSync waits for a fixed amount of time to allow other parties to respond to its own sync Interest. If no sync Data packet arrives within the fixed time interval, ChronoSync enters *recovery mode*, and requests others for their entire set of names.

Upon receiving a sync Data packet, ChronoSync updates the local sync tree with the received data, and notifies the application about the new sequence number added by the participant. In our example, Alice's sequence number 18. The application then can fetch the content by exchanging application-level Interest and Data packets for Alice/18. Here, we simplified the participants' names to be Alice, Ted and Bob, but in practice, their names would consist of additional name components to indicate a routable name.

CCNx Synchronization Protocol

In section 4.4 of this dissertation, we compare the performance of iSync to the performance of the *CCNx Synchronization protocol* (CCNx Sync). Therefore, in this subsection, we provide the relevant background details of CCNx Sync.

The Content-Centric-Networking project (CCNx) project [55], implemented by PARC, was the first open source implementation of an ICN architecture. This project was later acquired by Cisco and renamed CICN. The initial implementation of CCNx consisted of two main components: the CCNx deamon (ccnd) and the CCNx repository (ccnr). The ccnd component implemented the forwarder as well as the FIB, PIT, and CS infrastructures. The ccnr component implemented the CCNx repository, which can be used by the network or by an application to preserve required data, such as a routing table, file contents, or an application state.

The CCNx Sync protocol defines a collection as a set of content items, all sharing the same name prefix. The protocol operates between two neighbor nodes that declared the same collection, and keeps the collection up-to-date by synchronizing the differences. For synchronization, CCNx Sync uses a tree-based structure called the *Sync Tree*. A single *Sync Tree* represents the prefixes of all the content items in a single collection.

Upon the addition of a content item to the CCNx Repository, ccnr checks if the content name answers the definition of an existing collection. If it does, then the content's name is added to a *Sync Tree* leaf of the corresponding Sync collection. A hash value is computed for each of the inserted names. Thus, each leaf holds a list of content names and a combined hash representing the arithmetic sum of the local names' hash values. The other *Sync Tree* nodes hold the combined hashes of their children. Using this tree structure, the root node

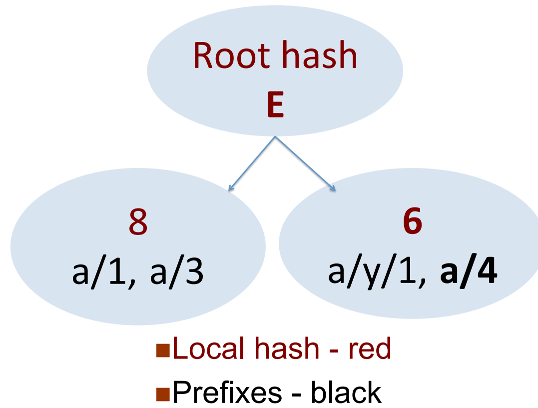


Figure 4.3: CCNx Sync Tree

of each *Sync Tree* holds the combined hash of all the names stored under the represented collection.

Figure 4.3 presents an example of a *CCNxSyncTree* containing four names: *a/1*, *a/3*, *a/y/1*, and *a/4*. In this example, the first two names compute a combined hash of 0x8, while the last two names compute a combined hash of 0x6. The root hash in our example is 0xE. Figure 4.4 shows the protocol timeline upon the insertion of *a/4* into Alice's collection. Thus, Alice's sync tree of a specific collection includes the inserted name *a/4*, while Bob's collection is represented by the same tree without *a/4*.

To keep the collection up-to-date, Alice sends a periodic *Root Advise* interest to Bob, including the collection name and its root hash. Upon reception of a *Root Advise* interest, Bob compares its local root hash with the remote node root hash. Equal root hashes imply the collection is up-to-date in both nodes, while different hashes imply a collection difference. In our example, Bob understands that its local collection is out-of-date. To reconcile the differences, Bob sends a *Node Fetch* interest that contains the collection name as well as the unrecognized hash value. Alice responds with the children's hash values list of the unrecognized hash. In our example the children of 0xe are 0x8 and 0x6. Bob recognizes 0x8, but does not recognize 0x6, and therefore sends an additional *Node Fetch* interest to requests

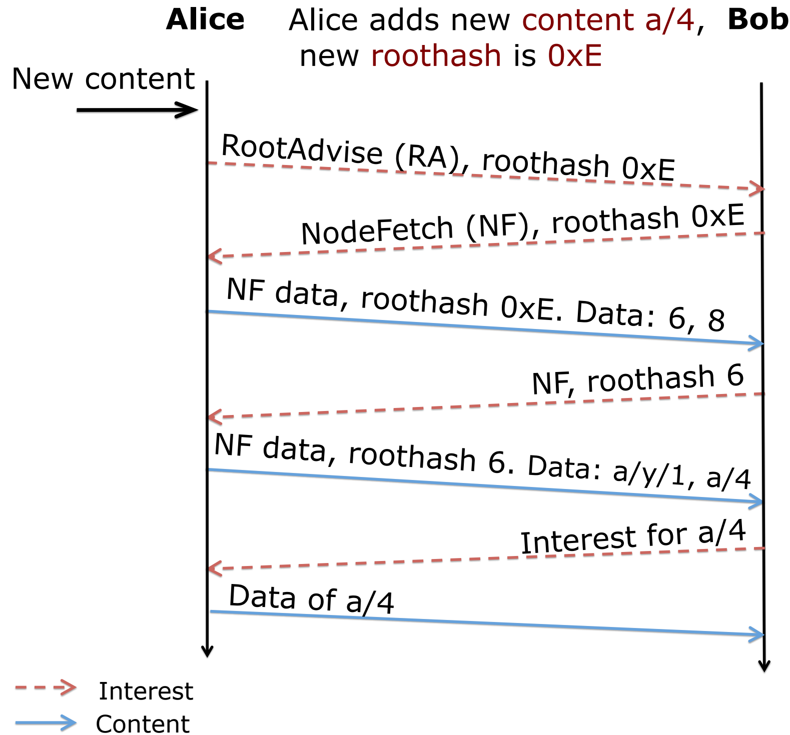


Figure 4.4: CCNx Sync Timeline.

for the content of 0x6. Now, the unrecognized hash represents a leaf in Alice's sync tree, and therefore the list of the names stored in that leaf is sent as the data response. Here, Alice responds with *a/y/1* and *a/4*. At this point, Bob understands that the set-difference of its collection is *a/4*. The remote and the local nodes can exchange *Node Fetch* interests and data packets until the data packet includes the list of all the missing contents names and, hence, the set difference. Once a node reconciles the names, it sends a regular interest packet to fetch the content of the reconciled names. In this example, seven packets are sent to reconcile the addition of a single name. An update to another collection will result in an additional set of packets as described in Figure 4.3.

It's important to note that the names described in the example were simplified to present names that share the same *a/* prefix. In reality, the prefix used by a CCNx Sync collection consists of additional components to indicate the forwarding routes.

4.3 ICT-Sync

As discussed at the beginning of this chapter, the core task of a distributed sync mechanism is to identify and reconcile the set-difference, using the smallest amount of data transferred in the network. As shown in subsection 4.2, ChronoSync, the leading sync protocol in NDN, uses a digest in the Sync interest name to represent the state of the shared dataset. By comparing two digests ChronoSync can tell whether the dataset is up-to-date or not. However, solely comparing two digests is not enough to identify the set-difference. To find the set-difference, ChronoSync maintains a local digest log to indicate the difference between two subsequent digests. However, in the case of simultaneous updates, the set-difference cannot be found in the digest log. When this happens, ChronoSync must use "recovery" packets that consist of the entire synchronized dataset, and result in (at least) one additional RTT.

ICT-Sync was designed to find the set-difference directly from the sync Interest name to avoid the penalty of recovery packets, and without the need to maintain a log. ICT-Sync follows the sequential data naming convention proposed by ChronoSync. However, unlike ChronoSync's digest tree, ICT-Sync represents a sync group — a shared set of sequenced namespaces — using two lists of tuples. The first list, named the *status list (sl)*, consists of two items: a UID and a sequence number. The UID represents an application party, and the sequence represents the number of names added to the shared dataset by the party. The second list of tuples, named the *mapping list (ml)*, maps every UID into a full party name.

For instance, a shared dataset of Alice, Bob and Ted contains all the sequenced data names of the three. If sl=([h1x1:10];[h2x2:13];[h4x2:11]), and ml=([h1x1:Alice/ProfilePicture/]; [h2x2:Bob/ProfilePicture/];[h4x2:Ted/ProfilePicture/];) shows that Alice, with party UID 'h1x1', published 10 names: Alice/ProfilePicture/1, Alice/ProfilePicture/2, ... Alice/ProfilePicture/10. Bob has a UID 'h2x2', and he published 13 sequenced names with

the prefix Bob/ProfilePicture, and Ted has a UID 'h4x2', and he published 11 sequenced names with prefix Ted/ProfilePicture.

While ICT-Sync approach to explicitly represent the synchronized dataset in the Interest name is similar to the high-level approach suggested in [64], ICT-Sync does not rely on vectors or membership protocol. Clearly, the two lists of tuples can be coded as a single list. However, we describe ICT-Sync's data structure as two lists to demonstrate how only sl is needed to find and reconcile the set-difference of every two parties, and therefore, is the information exchanged by sync packets.

Figure 4.5 shows how by exchanging their sl, Alice, Bob and Ted can not only identify if their view of the shared dataset is out-of-date, but can also find and reconcile the set-difference.

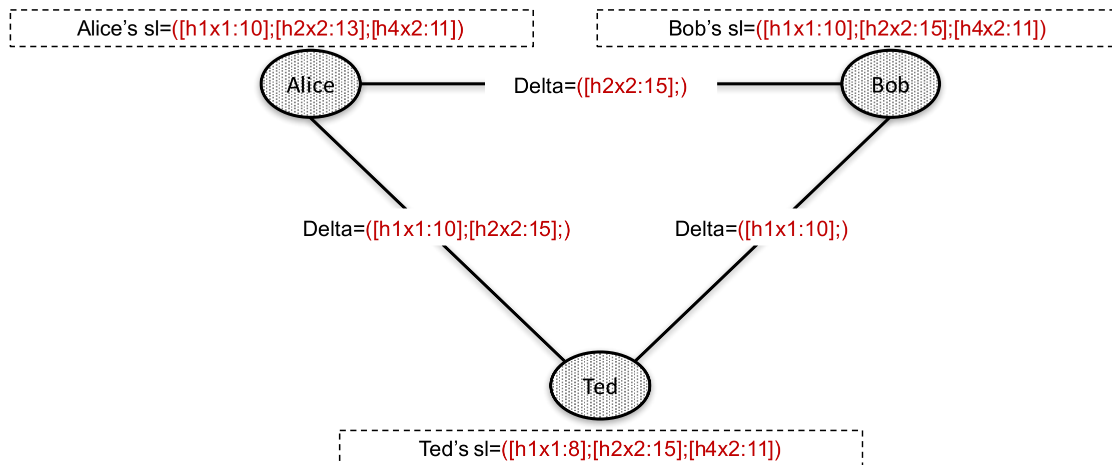


Figure 4.5: ICT-Sync Synchronization Model

4.3.1 ICT-Sync Synchronization Model and API

To participate in a sync group, an application party uses the ICT-API to register its UID and name, and to specify the name of the group it wants to join. The application does it by

using the *Register* API. All the parties that specify the same group name, are added to the group's sl and ml. To add a new sequenced name to the shared dataset, an application uses the *Add* API. When ICT-Sync discovers a remote update to the group's shared dataset, it updates the application using the SyncUpdate callback.

Similarly to ChronoSync, ICT-Sync's synchronization model uses *sync names* to synchronize a shared list of *data names*. ICT-sync name consists of an ICT-Sync prefix, the group name, and the group's sl. The group name can be represented by one or more name components, depending on the application in preference. A data name in the synchronized dataset consists of the party's routable prefix, the content name, and the content sequence number.

Figure 4.6 shows ICT-Sync synchronization model and its high-level API to applications. The red text in the picture is an example of a file sharing application, with a 'ProfilePicture' as its sync group name, and three known participants: Alice, Bob and Ted. The figure shows how the application communicates with ICT-Sync using the three API calls, Register, Add, and SyncUpdate, and can also communicate directly with the NDN forwarder to fetch named data.

4.3.2 ICT-Sync Protocol

To maintain an up-to-date view of the synchronized set of data names, ICT-Sync multicasts its sl to the other parties in the group. Upon the receipt of an sl, ICT-Sync compares its local sl with the remote sl to determine if the two sets are the same. If the lists are different, and the local sl has an up-to-date view of the set, hence a larger sequence for one or more UIDs, then it responds with its own sl so the remote party could update its view. If the lists are different and the remote sl has an up-to-date view of the set, then ICT-Sync updates its local view according to the information in the received sl.

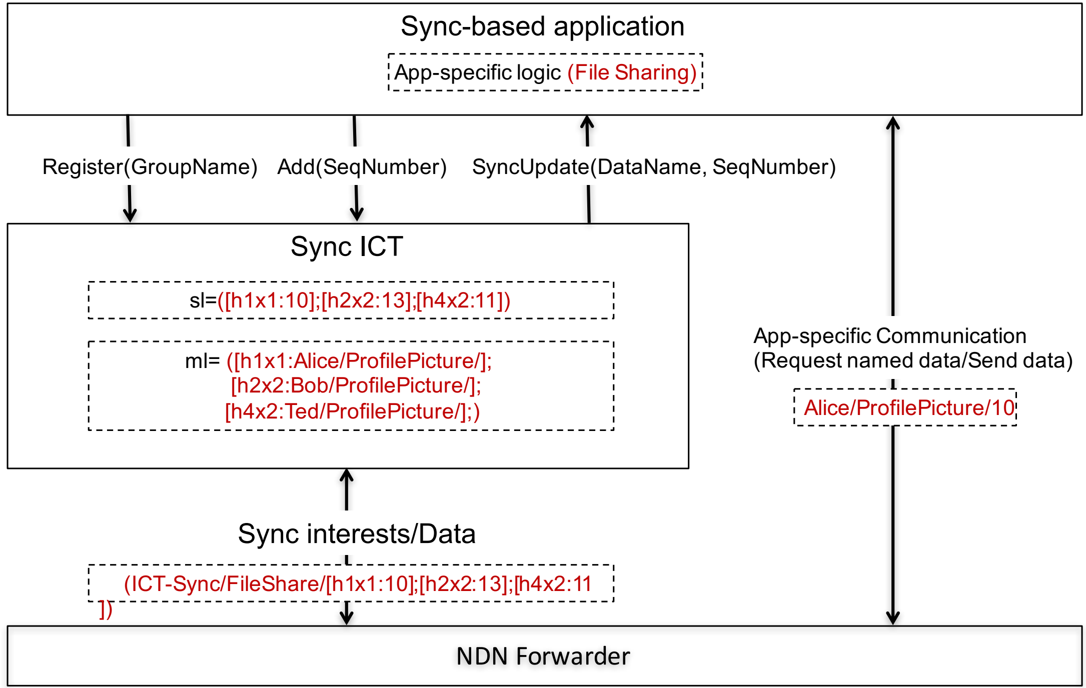


Figure 4.6: ICT-Sync API

Figure 4.7 demonstrate the high-level details of ICT-Sync protocol, and shows how the protocol handles simultaneous updates. Here, Alice, Bob and Ted form a sync group named Group1. The table next to each participant shows its sync state at different points in time. The first column indicates the step number, the second and third columns show the participant's sl and ml, and the fourth column shows the state of the incoming pending Interest. A name in the fourth column indicates that the participant received a sync Interests with this name, but was not able to satisfy it and therefore saved it for future processing. Red text in the table indicates a change between the steps.

At step 1, the three parties register their data names and UIDs, and each multicast an initial sync Interest: ICT-Sync/Group1/00. There sl and ml are still empty, but their incoming pending Interest column shows that each party aggregate all the incoming Interests for this initial sync state.

At step 2, both Alice and Bob use ICT-Sync API to add a new sequenced data to publish. Alice adds /Alice/1, and Bob adds /Bob/1. ICT-Sync updates sl and ml with the new sequenced name, and searches its incoming pending Interest table. In this case, ICT-Sync finds that both Alice and Bob have an incoming pending Interest for ICT-Sync/Group1/00, and each responds to the pending Interest with its new state. Alice's Data packet name is ICT-Sync/Group1/00, with content: <Alice:h1x1:1>. Bob's Data packet name is ICT-Sync/Group1/00, with content: <Bob:h2x2:1>.

At step 3, Alice receives Bob's Data packet, Bob receives Alice's Data packet, and Ted receives only one of them. In this example, we assume Alice's message consumes the PIT first, and therefore arrives at Ted while Bob's Data packet gets dropped. When receiving the Data packets, each party reconciles the differences according to the information it finds in the content.

At step 4, after the three updated their local state, each sends a new pending Interest to announce its new sync state. Here, both Alice and Bob receive two Interest packets, one from the other and another from Ted. ICT-Sync on both Alice and Bob identifies that the set-difference between the local state and Ted's state is not empty.

At step 5, both Alice and Bob respond to Ted's Interest with a Data packet, and Ted's ICT-Sync reconciles the difference and updates its local state.

4.3.3 The Intermediate ICT-Sync Component

Like other synchronization protocols, the goal of ICT-Sync is to maintain data consistency over time of a shared namespace. However, ICT-Sync was designed according to the ICT concept, and therefore, in addition to its endpoint library, it also consists of an optional in-network mechanism that can be deployed by the network operator.

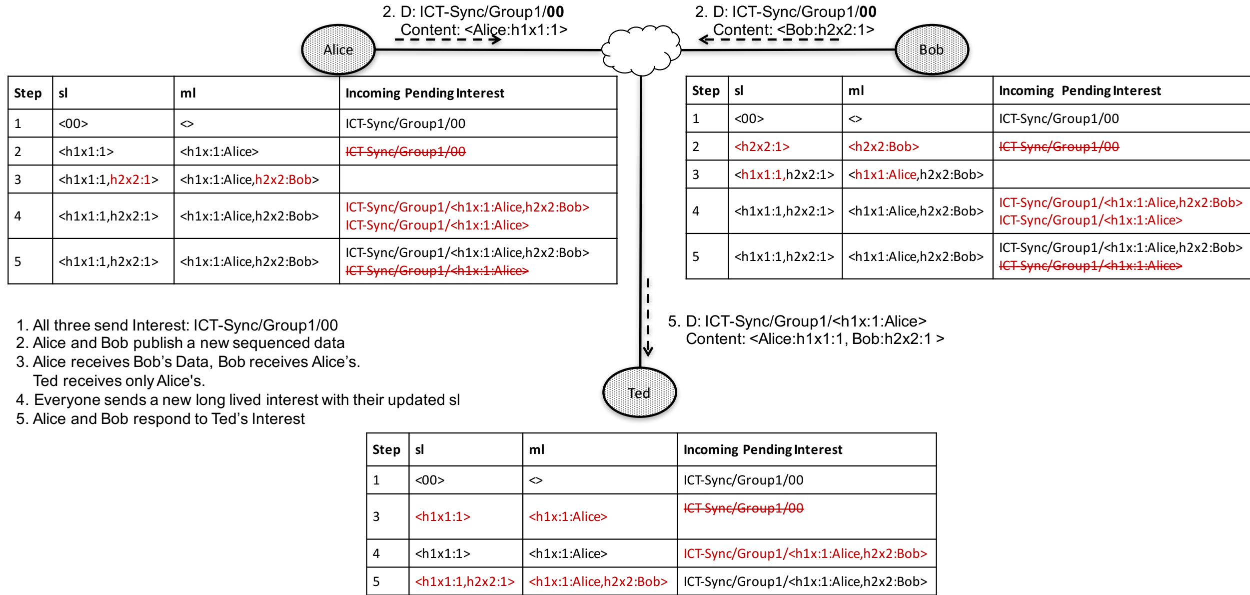


Figure 4.7: ICT-Sync Simultaneous Updates

The intermediate ICT-Sync process learns about the synchronized groups by classifying sync packets using their ICT-Sync prefix. When receiving a Sync packet, the intermediate ICT-Sync acts like an endpoint, and maintains the group state by looking at the sl name component and by sending Sync Interest and Data packet when identifying a set-difference. However, unlike an endpoint process, we implemented the intermediate process to automatically fetch the content associated with new name updates. On receipt of content, it validates the signer of the content, using its trust model. If the content is validated, it saves the full Data packet, including the original signature of the content and all headers. Then, it serves as a provider of the fetched data by registering the participant's prefix in its local NFD. Our implementation can be configured to use either persistent storage or the NDN CS to store the fetched data, depending on the characteristics of the network and the router.

4.3.4 Mapping Data Names to UIDs

As can be seen in Figure 4.7, ICT-Sync uses the information in its Data packet to map a UID into the participant data name. ICT-Sync does not choose the party's UID, and leaves the application to do that. This approach might be perceived as a weakness, because a party's UID must be kept unique for the correctness of the protocol. However, we argue that choosing a UID is not a hard task for an application with a finite number of group members. First, the application UID can simply be the party's Data name as this name must be unique for routing purposes. However, since the UID is expressed as the sync state in ICT-Sync packets, we recommend choosing a smaller representation of the name. This can be done by hashing the data name into a shorted UID, or by any other mechanism chosen by the application. If the application cannot guarantee the uniqueness of its parties' UIDs, it can use universally unique identifiers (UUIDs) [44].

When a party receives an Interest with a UID it doesn't recognize, it cannot map it to a Data name, and cannot update the application with the new update. The first time ICT-Sync can map a party name to a UID is when this party responds with a Data packet. This could create long convergence time when the update frequency of multiple parties is high, and when a Data packet from one party always wins the race and consumes the PIT. Therefore, as an optional enhancement to the protocol, ICT-Sync uses a Discovery Interest to map a UID to a Data name.

With 'Discovery' mode on, a party sends a Discovery Interest to request the Data name of a UID. The Interest name would consist of the ICT-Sync Interest, the group name, and the UID of the requested Data name. In the example described in Figure 4.7, the Interest name would be ICT-Sync/Group1/Discovery/h1x1.

Discovery Interests and Data packets have another benefit. It allows ICT-Sync to remove the Data name component from the Data packet, and therefore, results in smaller Data packets. We evaluate the discovery enhancement in Section 4.3.5.

4.3.5 ICT-Sync Evaluation

The goal of this section is to show the capabilities of ICT-Sync. In the next set of experiments, we focus on demonstrating how ICT-Sync provides transport for a file sharing application by decoupling it from connectivity characteristics. We conducted all our experiments on the Open Network Lab (ONL) [76], where real routers and links can be programmed to control the factors we used in our experiments: link delay, link bandwidth, packet drop rates, and link availability.

Experimental Setup

First, we tested ICT-Sync in the small network topology presented in Figure 4.8. Although it seems simple, this topology illustrates a use case in which endpoints communicate via a path of unreliable links and nodes, where links are lossy and an end-to-end communication is not guaranteed. For instance, consider a sensor network in which two sensors communicate via one or more intermediate nodes and links that can asynchronously move or fail. As we show in this section, without an ICT, the application is highly coupled with the network connectivity to the point it can completely break.

This topology consisted of six endpoints and four intermediary NDN routers running on 10 two-core machines. In addition, we used five Ubuntu Linux (16.04.4) software routers servers to interconnect the machines. All two-core machines ran NFD [5] and each experiment was repeated at least five times. The endpoints ran tested applications with ICT-sync APIs.

NDN routers were a combination of two machines - a Linux software router and a machine that ran the NFD code.

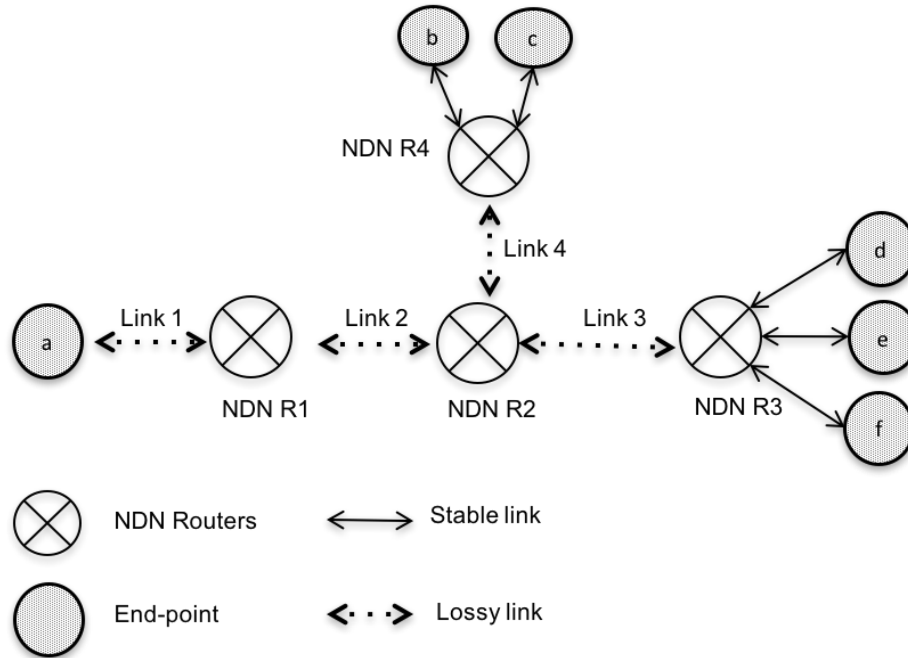


Figure 4.8: Tested NDN Topology

We configured each of nodes 'b'-'e' as producers and node 'a' as the consumer. Each producer encrypted data read from local 10MB files into 512KB chunks, and published five chunks per second using ICT-Sync API. For each published chunk, ICT-Sync updated its state list to represent the sequence number of each new chunks, and synchronized the entire group of endpoints 'a'-'f' with the new sequence numbers of each producer. ICT-Sync libraries on the endpoints exchanged sync Interest and Data packets to reconcile the new information, and notified the application of every new chunk name. In our experiments, we implemented the application to fetch every new chunk in order to measure its performance on top of a variety of connectivities. Once the chunk was received, the signer was validated, and the content

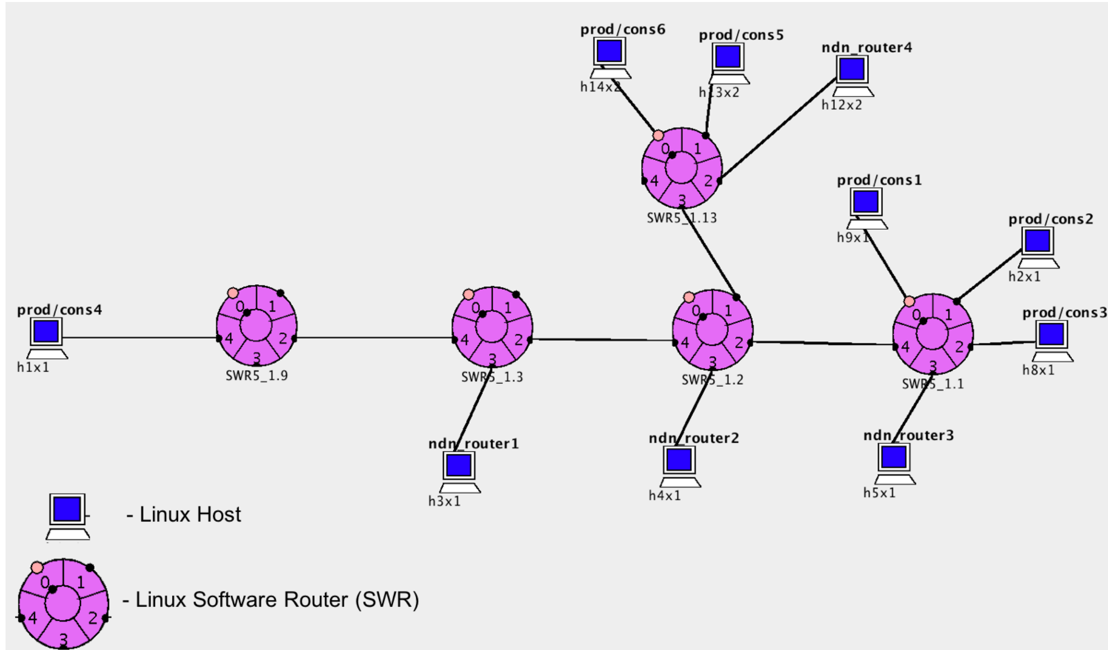


Figure 4.9: Mapping Topology onto Physical ONL Hardware

was decrypted, and concatenated into a single file. In our experiment, we preconfigured the nodes with the sync prefix for the content and the trust anchor needed for validation.

We used the producers' system clock to record the publish time in the Interest name — when the chunk was initially published by its producer — and the sync time – when the consumer learned about the chunk for the first time. We set the Link rate to 1000Mbps and changed the loss rate of the links to replicate an ad-hoc network with low bandwidth and intermittent links.

To compare ICT-Sync with ChronoSync, we implemented a file sharing application that can work with both APIs, and we measured the sync times of the file's chunks when ICT-Sync or ChronoSync are deployed on the endpoints. To evaluate the impact of in-network sync mechanisms, we deployed the intermediate ICT-Sync component (described in Section 4.3.3 on the network routers R1 and R2 in some of our experiments. Following the ICT concept, we ran the same application regardless of the deployment of the intermediate ICT component.

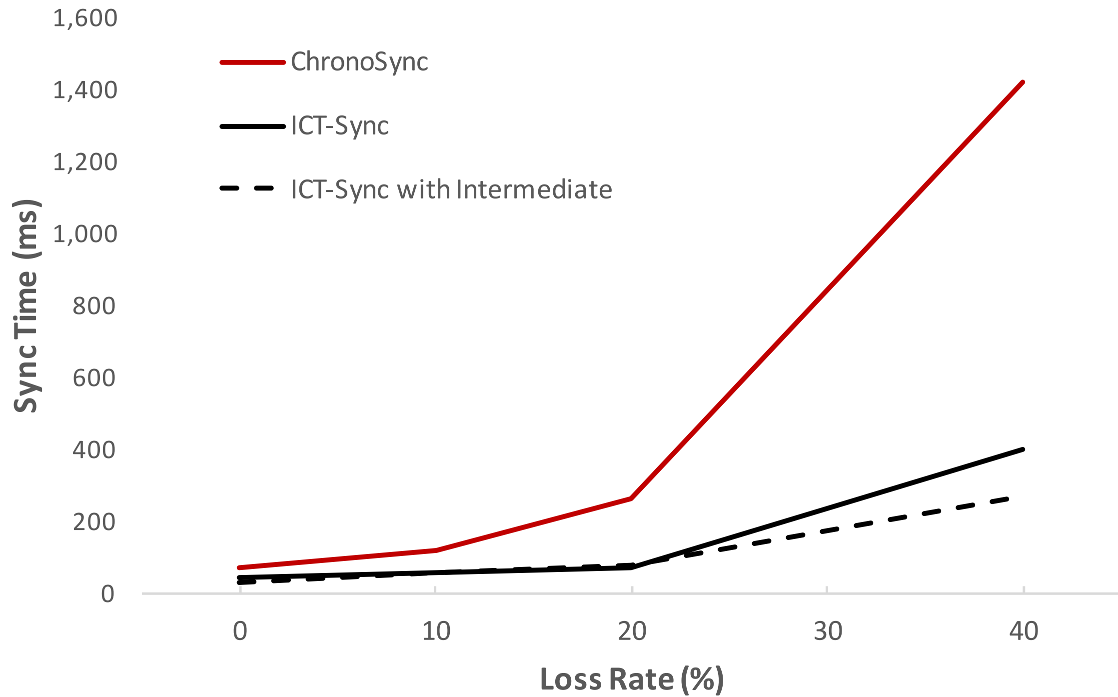


Figure 4.10: Average Sync Times over Low Loss Rates

In addition, we tested the same topology with a similar IP-based application by using ipref. We recorded all sync times and reported the average.

Sync Times over Different Loss Rates

In the first set of experiments, we evaluated ICT-Sync and ChronoSync over different loss rates. In each experiment we configured all the links in our topology to have the same loss rate, and we tested ICT-Sync with and without an intermediate component on NDN R1 and NDN R2. Figure 4.10 supports the following conclusions:

- ICT-Sync achieves faster sync times than ChronoSync over different loss rates.

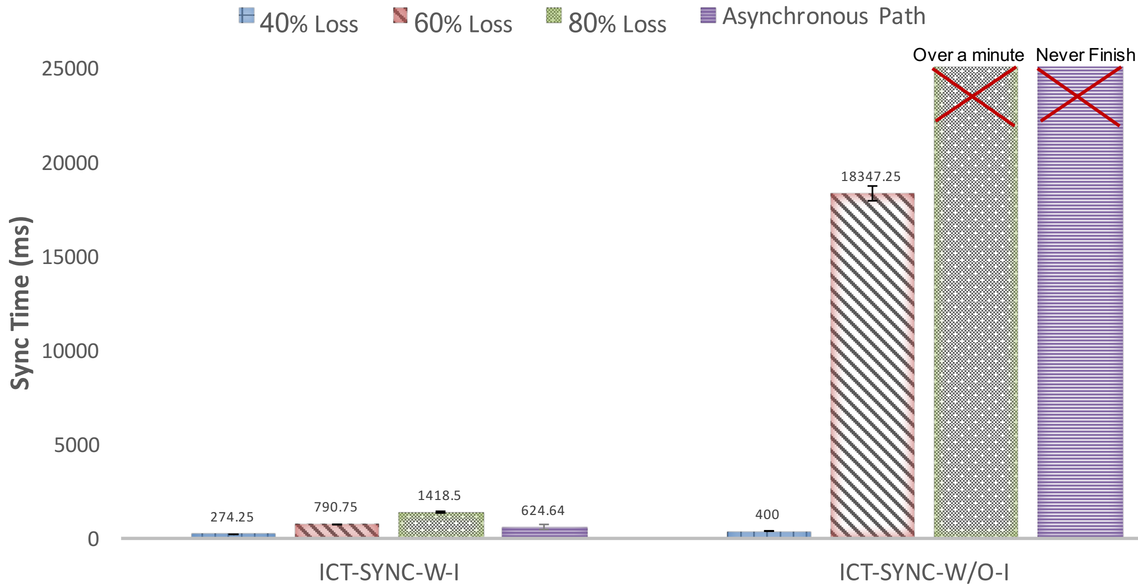


Figure 4.11: Average Sync Times over High Loss Rates

- In small loss rates (10%-20%), ICT-Sync with an intermediate component achieves similar sync times as ICT-Sync without an intermediate component deployed in the network.
- In loss rates greater than 20%, the deployment of an intermediate ICT-Sync achieves faster sync times.

To better understand the impact of an intermediate ICT-Sync component on synchronization times, we continued and tested our application over larger network losses. Due to the different scale, we present those results separately in Figure 4.11. Here, ICT-SYNC-W-I bars represent the sync times over high loss rates when an intermediate ICT component was deployed in the network, while ICT-SYNC bars show the sync times without the deployment of an intermediate ICT. As can be seen in Figure 4.11, an intermediate ICT-Sync component significantly improved sync times for 60% and 80% loss rates. Moreover, the results show that without an intermediate component, it took over a minute to synchronize each file chunk

when the network loss was set to 80%. Additionally, Figure 4.11 shows that the intermediate ICT-Sync component is essential when there is no SEEP, and without it no named data can be synchronized. We extend no SEEP experiments in the next subsection.

Synchronous End-To-End Path (SEEP)

Next, we evaluated ICT-Sync in a network with no SEEP. We ran the same file sharing application on nodes 'a' and 'e', with 'a' being the consumer and 'e' being the producer. We configured the producer to publish five chunks of file a second, and we stopped our experiment shortly after it published 1024 file chunks. During this time, we alternated link 1 and link 2 (as shown in Figure 4.8), for one, two or three seconds. Therefore, in this experiment there was never a SEEP between the consumer and the producer.

Figure 4.12 shows the percentage of the fetched file. Here, the x-axis indicates the number of seconds each link is up before being stopped. The results support the following conclusions:

- The application succeeded in fetching above 50% of the shared file when links 1 and 2 broke the end-to-end path only with an intermediate ICT component was running on router 1.
- Without an intermediate ICT component, the consumer failed to fetch any portion of file. Therefore, both ICT-Sync and Iperf failed.
- The results did not demonstrate statistical significance for the tested up and down times. This is a direct result of our experimental setup, in which we stopped the experiment shortly after the producer published 1024 chunks, and regardless of the link state at that time.

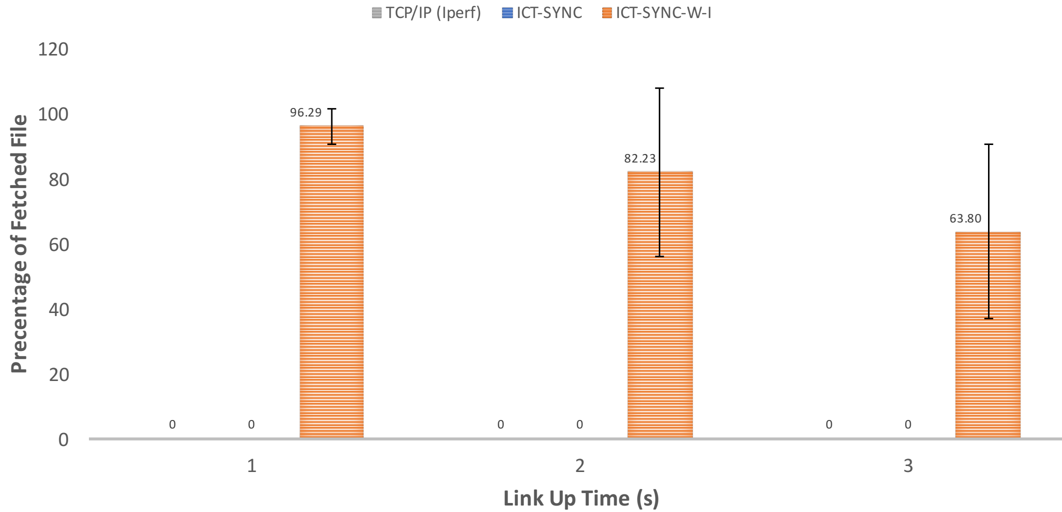


Figure 4.12: Percentage of Fetched File of Different Communications with Alternating Links

- When we let the experiment run after the producer published its 1024 chunk, ICT-Sync with an intermediate always fetched 100% of published chunks, while Ipref and ICT-Sync without the intermediate remained on 0%.

Scaled Up Topology

To better compare the performance of ICT-Sync and ChronoSync, we scaled up our topology and evaluated the two protocols using the ONL topology presented in Figure 4.13. We used 40 producers that served also as consumers, each publishes 5 data chunks per second, with a total of 401 data chunks published by each producer.

Table 4.1 presents the comparison of the two protocols. The results demonstrate that ICT-Sync improves average sync times by 97% while using about 20% of ChronoSync traffic. In addition, the results show that ICT-Sync packets are about 3 times larger than ChronoSync packets.

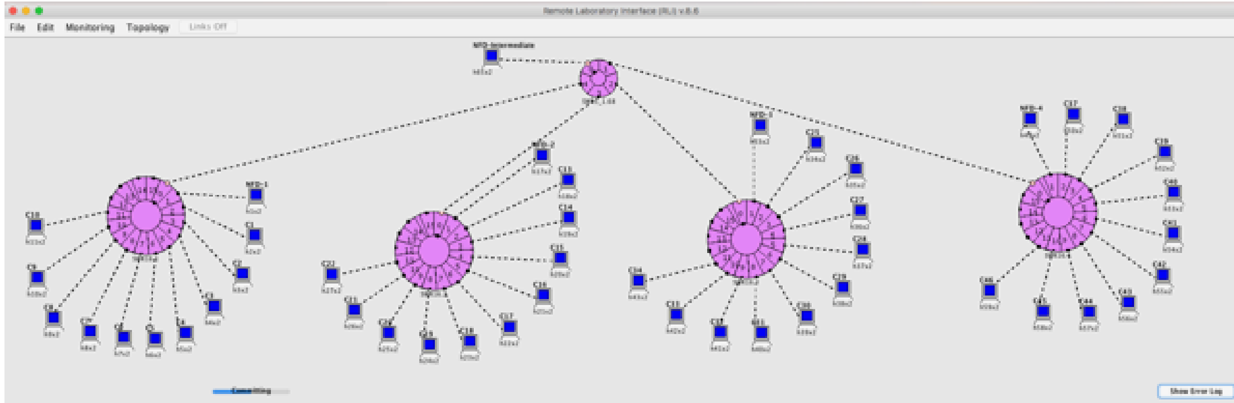


Figure 4.13: Scaled Up ONL Topology

	ICT-Sync	ChronoSync
Average Sync Time (ms)	26.96	1033.64
Standard Deviation	4.11	154.89
Total Number of Packets	149508	773131
Bytes/Packet	624.36	160.36

Table 4.1: Large Scale Comparison of ICT-Sync and ChronoSync

To evaluate the overhead added by the intermediate ICT-Sync component, we counted the number of packets sent and received by an endpoint in the tested topology with and without an intermediate ICT component. The results shown in Figure 4.14 demonstrate that the overhead caused by the intermediate ICT-Sync component over different loss rates.

4.3.6 ICT-Sync Summary

The characteristics of ICT-Sync as Information-Centric Transport are as follows:

- ICT-Sync is a primitive mechanism and it can be used by any application that is looking for sync-based services.
- The existence of the intermediate ICT-Sync component does not introduce any change to the application running at the endpoints.

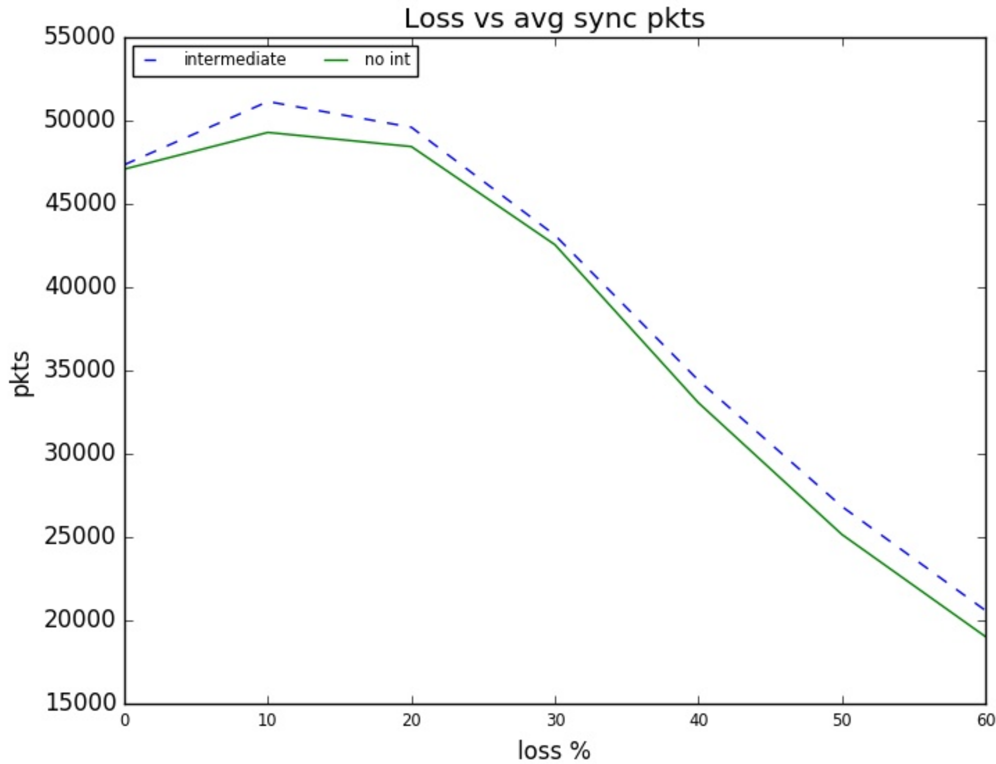


Figure 4.14: Traffic Overhead of ICT-Sync

- The intermediate ICT-Sync component has no knowledge of any application-specifics property or the content it handles. It builds its local sync state from looking at the names of Interest and Data packets and without decrypting the content.

- Because an application uses the same API in both reliable and challenged networks, its information plane is fully decoupled from the actual connectivity plane.

- If required, ICT-Sync can maintain existing NDN trust schema mechanisms to fetch keys and validate the data. It does this by using the existing NDN tools and by looking at the packet's name and key-locator fields, without decrypting the content.

4.4 iSync: Synchronizing Namespaces With Invertible Bloom Filters

In this section, we describe another synchronization mechanisms, named iSync. The goal of this section is to explore how invertible Bloom filters (IBFs) can be exchanged by synchronization mechanisms to support a large number of sync updates of arbitrary names. Therefore, while ICT-Sync synchronizes sequenced names using a state list, iSync synchronizes arbitrary application names using IBFs. Unlike ICT-Sync, we did not implement or evaluate an intermediate iSync component, but we focused on implementing and testing alternative sync mechanisms that can form an ICT for sync-based applications.

We designed and implemented iSync for the CCN architecture, with the goal to provide full data synchronization to CCN applications, hence synchronization of the names and their content. While iSync and CCNx Sync — the default synchronization protocol of CCN — share the same motivation, their infrastructures are fundamentally different. CCNx sync used bounded logs of namespaces, as discussed in 4.2 and iSync uses invertible Bloom filters. The goal of this subsection is to explore another sync mechanism that can support a large number of distributed updates of arbitrary names.

4.4.1 IBF Background

The invertible Bloom filter (IBF), introduced in 2011 as an extension of the original Bloom filter [32], is a simple and space-efficient probabilistic data structure. The Bloom filter answers whether an item exists in a dataset or not. The implementation of the Bloom filter consists of a simple array and a number, k, of hash functions. The k functions map an item into k cells in the array. The mapped cells are marked as occupied upon the insertion of an

item. A Bloom filter tests for the existence of an item by looking into its hashed cells, and answers yes if all the cells are marked as occupied. The original design of the Bloom filter does not support either deletion of an item or querying the stored items.

IBF was designed to address those limitations. As in the counting Bloom filter, IBF uses *count* to indicate the number of items that have been indexed to each cell. In addition, the IBF algorithm introduces two new values in each cell: *idSum* and *hashSum*, to represent key-value pairs in each cell. The insertion process of an item is similar to the original insertion process of the Bloom filter. A set of k hash values is generated to map the item into k cells in the array. However, unlike the standard Bloom filter, the value of the inserted item is added, using a XOR operation, to the value of *idSum* in each of the mapped cells. Also, another hash function adds the hash value of the inserted item to the value of *hashSum* in each of these cells. This addition is also achieved by XORing the hash value of the inserted item with the value of *hashSum*. An item insertion increments *count* in each of the k mapped cells.

In a similar way, we can delete an item from an IBF by subtracting it and its hash value from *idSum* and *hashSum* respectively, and by decrementing *count* in each of k hashed cells.

We can retrieve the items from an IBF by looking for pure cells. A pure cell is a cell that contains only one item, and the hash value of the cell's *idSum* equals the value of *hashSum*. To discover additional pure cells, and therefore additional stored items, we subtract the items found in the pure cells from the other cells indexed to store those items The subtraction process can be repeated as long as there are pure cells to retrieve.

IBF not only supports insertion and deletion of items, but also can obtain the difference set of two IBFs by subtracting one IBF from another, followed by decoding the resulting IBF to retrieve the stored items.

We present an example to illustrate the encoding process of an IBF: Upon the insertion of a new element S into a cell i, the value of *idSum* of the ith cell is XORed with S, while the value of *hashSum* is XORed with the hash value *H(S)*. Given two IBFs, A and B, we can compute the set difference (A - B) by XORing the values of *idSum* and *hashSum*, and decreasing the value of B's *count* from the value of A's *count*. In case the set includes at least one *pure* cell, we can start the encoding process and retrieve the elements in the difference set. Again, the pure cell must satisfy two requirements: Its *count* value must be equal to 1 or -1, and the hash value of its *idSum* must be equal to its *hashSum*. The encoding process can list the IBF's elements as long as there are pure cells to subtract from other cells.

4.4.2 iSync Synchronization Model

As shown in Figure 4.15, iSync consists of a repository, a repository API, and a sync agent. iSync can be regarded as an additional data synchronization layer in the CCNx stack, and can serve as a synchronization service to CCNx applications. The repository offers an interface for CCNx entities (i.e., applications) to insert files and publish sync collections. Like in CCNx Sync, iSync defines a collection as the set of content items sharing a common prefix. To synchronize a collection, an application is required to declare the same collection in each of the participating nodes. An example of a collection can be a set of music items that share a common prefix, such as John'sDocuments/Music. Possible content items in such a collection might include
John'sDocuments/Music/MichaelJackson/song1 or
John'sDocuments/Music/BestOf2010/song32. iSync repository provides an API to declare a sync collection. Applications can declare multiple sync collections, which will be synchronized independently.

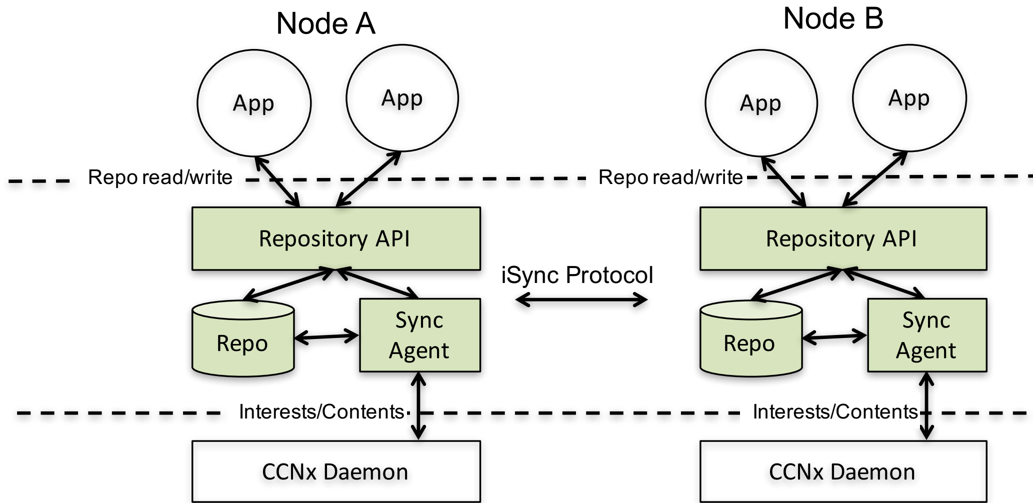


Figure 4.15: iSync Data Synchronization Model

The iSync repository notifies the sync agent when a new content name matches one of the local declared collections. Then, the sync agent indexes the inserted content name and updates a digest that reflects all the names of the collection. The sync agent notifies the remote nodes of its local digests by sending periodic broadcasts, while receiving remote ones. Like CCNx Sync, iSync identifies whether the collections are up-to-date by comparing local with remote digests. When a remote collection digest does not match the local collection digest, a reconciling process starts, and the set difference is found by repeatedly requesting, receiving, and comparing remote IBF tables against local and global IBF tables. The notation of global IBF will be described in section 3.4.

It is important to note that, like other applications built on top of CCNx, the sync agent communicates with its peers using CCNx packets.

4.4.3 iSync's Protocol and Data Structure

Different types of data collections may have different update frequencies. For example, a chat application generates a few messages per second, while video streaming can produce hundreds

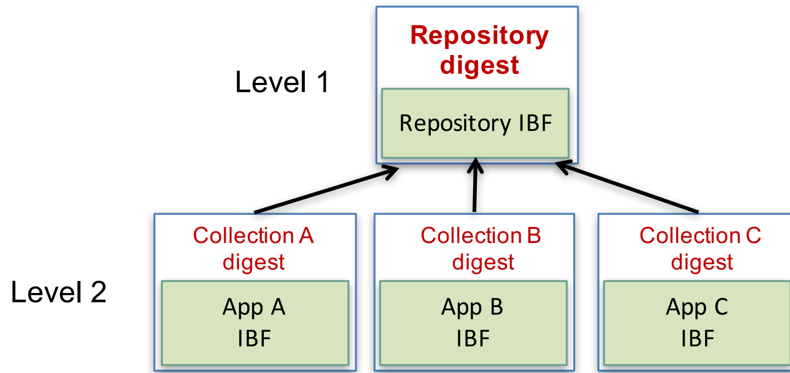


Figure 4.16: Hierarchical Synchronization Data Structure.

of video chunks per second. The iSync protocol was designed to offer synchronization as a service to different applications simultaneously, and therefore is required to support different types of update frequencies. We designed the protocol data structure to address three key tasks: 1) Efficiently maintain a digest to reflect the status of the entire repository, and hence, the status of all the local collections; 2) Efficiently distinguish between up-to-date and out-of-date collections; 3) Obtain set differences quickly, and with minimal traffic overhead.

As shown in figure. 4.16, iSync uses a hierarchical two-level IBF: *Digest sync IBF* and *collection sync IBFs*. The higher level records the status of the entire repository, while the lower level logs file insertions or deletions of each sync collection separately. An update to a collection changes the collection's IBF digest by hashing the new name into the corresponding *collection sync IBF*. The change in a collection digest invokes an update to the repository IBF and digest in the first level.

Thus, the three key tasks are all achieved: 1) The first level digest holds the status of the entire repository. 2) Currency is maintained by subtracting a remote first level IBF from the local first level IBF. While traditional NDN synchronization protocols iteratively send the digests of each collection, iSync requires only one data exchange to discover all the out-of-date collections. 3) In a similar way, the set difference of the changed collection is obtained

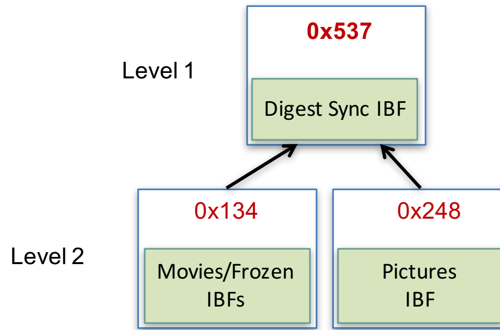


Figure 4.17: Alice's Hierarchical Data Structure.

by subtracting a corresponding remote IBF from the local second level *collection sync IBF*. iSync sequentially requests all the remote out-of-date *collection sync IBFs*, subtracts them from the local ones, obtains the updated namespaces, and fetches the content.

We illustrate the operation of the protocol using an example in which two applications declare collections to be synchronized. The first application is a dropbox style application that requests synchronizing all the files in a specific directory. To achieve this, the application declares a collection using the namespace "MyFiles/Pictures" in all the participating hosts. The second application is a media streaming application that requests synchronizing all the chunks of a video. This application declares a collection using the namespace "Movies/Frozen/". When a new file with the prefix "MyFiles/Pictures" is added to one of the hosts, iSync automatically indexes the file name in the IBF corresponding to the dropbox style applications. In a similar way, the name of the video chunk "Movies/Frozen/chunk12" is indexed to the corresponding media streaming IBF. A collection digest is calculated to represent each of the modified IBFs and the repository's IBF.

Figure 4.17 shows Alice's hierarchical data structure after adding new contents, while Figure 4.18 presents the timeline of our discussed example.

In our example Alice and Bob are the participating hosts, and Alice adds ten new chunks of the movie Frozen to her local repository. All the chunks have the same shared prefix, Movies/Frozen/. In addition, Alice uploads her camera photos and names the uploaded files using the "MyFiles/Pictures" prefix. As will be explained in section 3.3, the names of the movie chunks are indexed and added to the 'Frozen' IBF, while the pictures are indexed to the 'Pictures' IBF. The digests of the two IBFs are changed according to the added names, while the digest of the repository IBF is changed according to the 'Frozen' and 'Pictures' digests. Upon the expiration of the sync timer, Alice sends an interest packet consisting of her repository digest. Bob doesn't recognize the digest in the incoming interest, and therefore he requests Alice's *Digest sync IBF*. When it is received, Bob subtracts the incoming *Digest sync IBF* from its local one, and discovers all the out-of-date collections. Bob then iteratively asks for the IBFs of the outdated collections, the IBFs of the 'Frozen' and 'Pictures' collections. As he did for the *Digest sync IBF*, Bob decodes and requests the added file names by subtracting the remote collections' IBF from the local ones. As will be described in section 3.4, when the subtraction fails to resolve all the differences, Bob requests Alice's previous (local) IBFs. After Bob receives and decodes the missing names, he indexes the changes in his *collection sync IBFs* and updates the collections and the repository digests. To store the content of a video chunk or a picture, Bob sends an interest with the chunk or picture name.

We emphasize that while traditional synchronization protocols require multiple data retrievals and comparisons for a single update, iSync can reconcile multiple differences using a single data request. In the example, Alice uses a single interest packet to notify Bob that multiple changes were made in two collections. Also, a single request for a collection IBF results in the discovery of all the updates made in the collection. Therefore iSync's workload is concentrated in computation, which reduces time and traffic overheads.

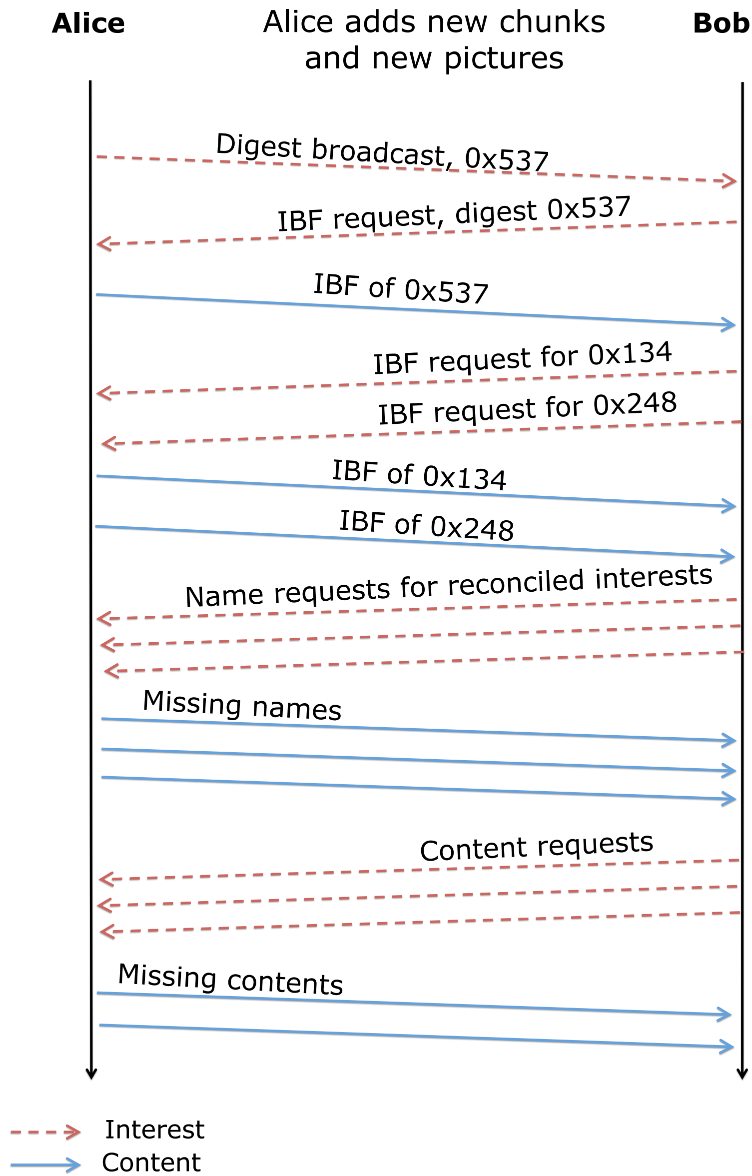


Figure 4.18: iSync Timeline Example.

It is important to note that while a single packet is used to notify a participant about multiple changes, and a single packet is used to reconcile multiple differences, the retrieval of the content names as well as the content items is done by sending one packet per name. Therefore, iSync can leverage the NDN characteristics and receive the requested content from a cached router or from a third user who already holds the missing content.

Name Encoding and Recording Scheme

The design of IBF handles fixed-length item names and does not support content lookup very well [32]. To support namespace synchronization in NDN, iSync introduces a name encoding and recording scheme.

The name encoding and recording scheme is part of the second level table described in section 3.2 and in Figure 4.16. The scheme is responsible for two tasks: mapping variable-length file names into fixed-length IDs, and recording what items have been inserted. Mapping is carried out by using a hash-indexed table to support bidirectional mapping relations between file names and file IDs; Recording is based on an invertible Bloom filter to support file insertions, deletions, and queries. This IBF represents the collection digest. During the entire synchronization process, file names are replaced by fixed-length file IDs. Moreover, the scheme can be regarded as a name compression scheme that reduces traffic overhead. The insertion process of the name encoding and recording scheme is described in Algorithm 1.

To simplify the discussion, we refer to content name as a File name in Algorithm 1. However, the name could represent different types of data, as explained in the Introduction. Algorithm 1 distinguishes between two errors: Already existing content names and hash collisions. The first error implies that a content item with the same name already exists in the collection.

Algorithm 1 Name Encoding and Recording Scheme

$File_ID \leftarrow Hash_Function(File_Name)$
if $File_ID$ *is not found in Member Recording Table* **then**
  Adding $File_ID \rightarrow$ *Member Recording Table*
  Insert $File_Name \rightarrow$ *Name Encoding Table*
else if $File_Name$ *Found In Name Encoding Table* **then**
  Report File_Aleady_Inserted_Error
else
  Report Hash_Collision_Error
end if

The second error indicates that the content name is new, but its hash value is already mapped by another content name. Both errors could be easily fixed by prompting a renaming request to the user. False positive errors such as the hash collision error could occur when mapping a long namespace into a shorter one. However, this probability is well studied and can be controlled by choosing the right algorithms and length of content IDs [25, 50]. In addition, by considering relevant application requirements and the potential number of names, collision resolutions can be maximized [42, 47]. The 'name encoding and decoding' scheme is implemented in the second level of the iSync hierarchical design, and therefore each application could make its own decision regarding the length of the encoded IDs.

Difference Size Control Scheme For Collection Sync IBFs

Using the name encoding and recording scheme, the iSync protocol utilizes IBFs to hold the set of name IDs for each sync collection. As described in the background section, it is efficient to compute differences between two IBFs by subtracting them and decoding the resulting IBF. However, the decoding process can compute the differences only as long as there are pure cells in the resulting IBF. Therefore, there is no guarantee that all the differences can be decoded. For a fixed-size IBF, the more updates it holds, the less likely it can be perfectly decoded (recover all item IDs). Moreover, the number of updates varies among different

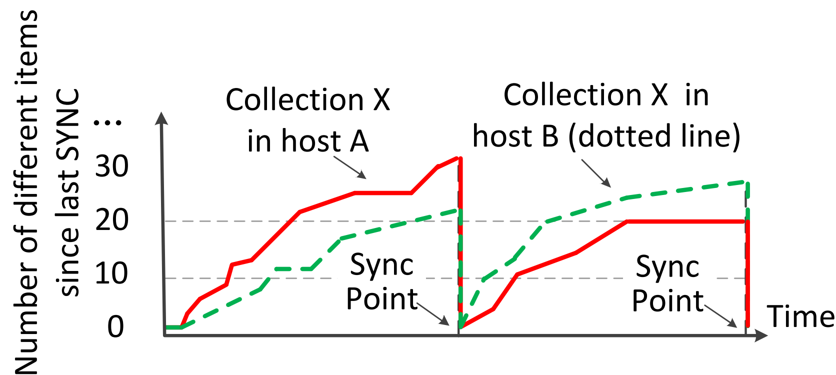


Figure 4.19: Periodical Synchronization.

sync collections, and does not follow a predictable pattern. Thus, a fixed size IBF can be inefficiency large in one application scenario, but small enough to cause decoding errors in another. To optimize the IBF size for various applications, we designed a difference size control mechanism.

First, as shown in Fig 4.19, hosts that have declared the same sync collections periodically confirm the consistency of their data sets. This periodic operation guarantees bounded delay of file shares and limit the potential size of differences between nodes.

Second, for any sync collection in one host, iSync creates multiple IBFs to hold the changes produced during a sync period. This scheme offers a flexible capability for recording and recovering the latest updates. As described in Fig 4.20, two types of IBFs are used: global and local. The global IBF can be regarded as a public version of the data collection. It is the latest local IBF in a sync cycle and the foundation of the first local IBF in the next synchronization cycle. The local IBFs support the process of reconciling the set difference at the end of a sync cycle. To find all changes made in a sync cycle, iSync first subtracts a remote global IBF from a local IBF. If it fails to obtain the complete set difference, it uses the local IBFs.

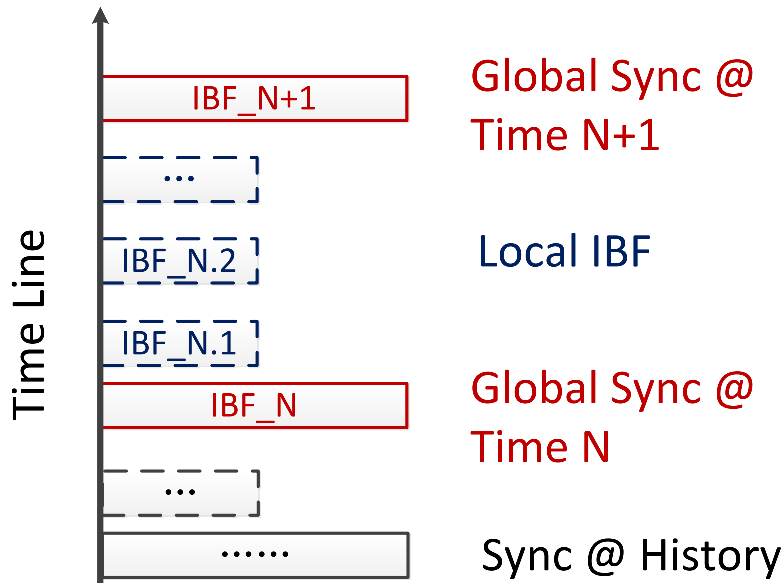


Figure 4.20: Local and Global IBFs for a Sync Collection.

Algorithm 2 presents the details of the described difference size control scheme. When there is an update to a collection, and therefore to its current IBF, iSync checks whether the total number of updates exceeds the configured maximal number. If exceeds, the current IBF is marked as local, stored as a backup table, and the number of updates is reset to 0. In addition, to limit memory consumption, iSync checks if the number of stored IBF exceeds the maximal number allowed. When iSync timer expires the current IBF is marked as global. Each application configures its own constants for the maximum number of updates and the number of history IBFs.

The combination of the local and global IBFs makes differences between two IBFs traceable. To support the optimal decoding of different update frequencies, an application can tune the periodic sync time and the IBF size according to its specific requirements.

Algorithm 2 Difference Size Control Algorithm

```
if updates_count > max_updates then
    if stored_IBFs_count > max_stored_IBFs then
        Dequeue Backup_Queue
        stored_IBFs_count ← stored_IBFs_count + 1
    end if
    Enqueue Current_IBF_Table As Local
    updates_count ← 0
    max_stored_IBFs ← max_stored_IBFs + 1
end if
if Time_since_last_sync > Time_sync_period then
    Globalize Current_IBF_Table
    Enqueue Current_IBF_Table As Global
end if
```

Recovery Scheme

False positive errors will occur if two different name IDs are mapped into the same IBF cells. Even though the possibility can be controlled, a recovery scheme is inevitably needed to provide disaster relief. In case a newcomer or returner jumps in with an empty or outdated sync collection, the original IBF design has a limited capability to recover.

To solve this problem, iSync uses a blacklist based scheme. After two hosts have gone through all their IBFs and still do not have a common view of the sync collections, the Bloom filters of the local and remote data sets (maintained by the name encoding and recording scheme) are exchanged. Each host reconciles local data sets and sends the list of files to remote nodes. Thus, local computing sources bear most of the overhead of the recovery scheme.

4.4.4 iSync Evaluation

We built a prototype of iSync on top of CCNx and compared its performance with the CCNx Sync protocol. iSync was written in C to ensure its compatibility and performance. The

prototype consists of a repository server, a sync agent and a file retriever. All components are connected by system FIFOs to ensure the protocol's potential for scaling.

The sync agent consists of the three schemes described in section 3: name encoding and decoding, difference size control, and recovery mode. To reduce false positive errors, we used a hash indexed table with 2^{24} entries to build bidirectional name-to-ID mapping, and a counting Bloom filter of 2^{24} cells. Each cell in the counting Bloom filter contained an 8-bit counter to count the number of the inserted files. As shown in Figure 4.20, we used an IBF FIFO to store a history of 32 IBFs for each sync collection. Each IBF consisting of 160 cells, and guaranteed to hold 128 IDs by default. The capacity of each IBF could be configured when declaring sync collections. All communications followed the pattern of the NDN architecture, which means that all data chunks (contents) were retrieved by exchanging interests and data packets.

Experimental Setup

We used the open network laboratory (ONL) [76] to measure the synchronization time and traffic overheads of CCNx Sync and iSync. Our experiments explored the impact of four factors: 1) file name length, 2) file size, 3) the number of hosts, and 4) topology type. Each host in the ONL system ran Ubuntu 12.04.2 LTS and CCNx version 0.7.2 on a Xeon CPU @ 2.5GHz with 4 GB memory.

In each experiment, we inserted a file into a local host, and waited until this file was replicated by iSync or CCNx Sync, and hence, synchronized, in all the participating hosts. We used tcpdump to capture the exchanged traffic during the experiments, and analyzed the traces to measure the traffic overhead. To measure the time overhead, we modified a small section in the CCNx code to record the content insertion timestamp. Then we calculated the differences

between the insertion timestamp in the local host and the insertion timestamp in the remote hosts. We reported the average synchronization time as the time overhead.

As described in [27], the ratio between the number of inserted items and the number of IBF cells impacts the capability to reconcile the differences between two IBFs. To ensure a successful decoding rate of more than 99%, we limited this ratio to 60%. Therefore, a new backup of current IBF was created when the number of new updates exceeded 60% of its holding capacity.

In the next subsections, we first describe the results of a simple topology consisting of two nodes, and show the impact of the name length and file size factors on sync time and traffic overhead. Then we show how the topology type and the number of hosts affect the sync time. Last, we present the measured performance of iSync's recovery scheme.

While measuring the performance of the CCNx Sync protocol, we discovered that the API used to insert a content item into the CCNx repo is not optimized, and took about 2.8 seconds. Due to this large overhead, we reported two different times: CCNx Sync and CCNx Sync Data. The former records the time overhead of the entire synchronization process, while the latter does not include the API overhead.

Time Overhead of iSync in a Simple Topology

Figure 4.21 shows the synchronization times of a 128 KB file with different name length using the CCNx Sync and iSync protocols. The file *song1* consist of one component, while the file
John'sDocuments/Music/Maroon5/song1 consists of four components. This experiment was performed on a two-node network to provide a basic measurement.

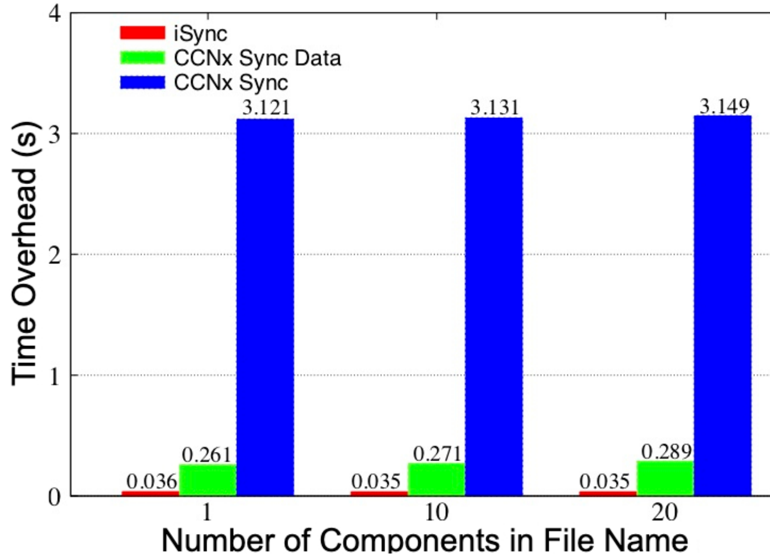


Figure 4.21: Impact of Number of Components in File Name on Synchronization Time.

The figure shows that synchronizing one file with one name component takes about 3.1 and 0.036 seconds by CCNx Sync and iSync respectively. This time overhead includes file insertion time, reconciliation time, and file retrieval time. CCNx Sync Data time is approximately 0.261 seconds. Therefore, iSync as an end-to-end system is about 86 times faster than CCNx Sync, while it is 8 times faster than the CCNx Sync Data when ignores the CCNx insertion overhead. We found that the number of components in a file name does not impact the sync time results.

File insertion and resolution play critical roles in iSync. We evaluated the time overhead of the two operations under different intensities to further understand their effect on the synchronization times. As shown in Figure 4.22 it takes about 3 ms and 330 ms to insert 80 and 20480 files respectively. In these experiments, we increased the IBF table sizes to ensure all file names could be perfectly resolved. Therefore, the time overhead was affected by the IBF table size, number of inserted files, and computing power of the host.

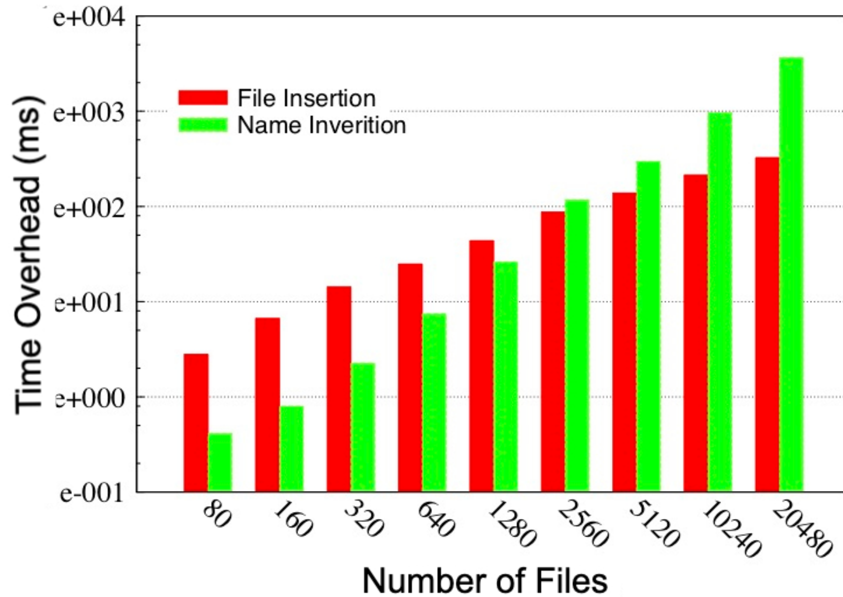


Figure 4.22: File Insertion and Recovery Time for iSync Protocol.

Figure 4.23 shows the effect of the file size on the synchronization time consumptions. The left figure shows the measured results, while the right figure shows the ratio between CCNx Sync and iSync. As expected, for both protocols, the synchronization time increases as file size increases. Synchronizing a file of 65536 KB took about 9 seconds for iSync and 63 seconds for CCNx Sync. Moreover, iSync was about 86 times faster than CCNx Sync for small files, and about 8 times faster for large files. The figure also shows that iSync is still 8 to 10 times faster than CCNx Sync, even if we do not consider file insertion time.

Synchronization performance can be divided into two tasks: reconciling content names and retrieving the file contents. As file size grows, the weight of data retrieval grows. iSync uses a more efficient scheme to find the set difference and optimized size of packets to deliver data, and therefore is faster than even CCNx Sync data, for large files.

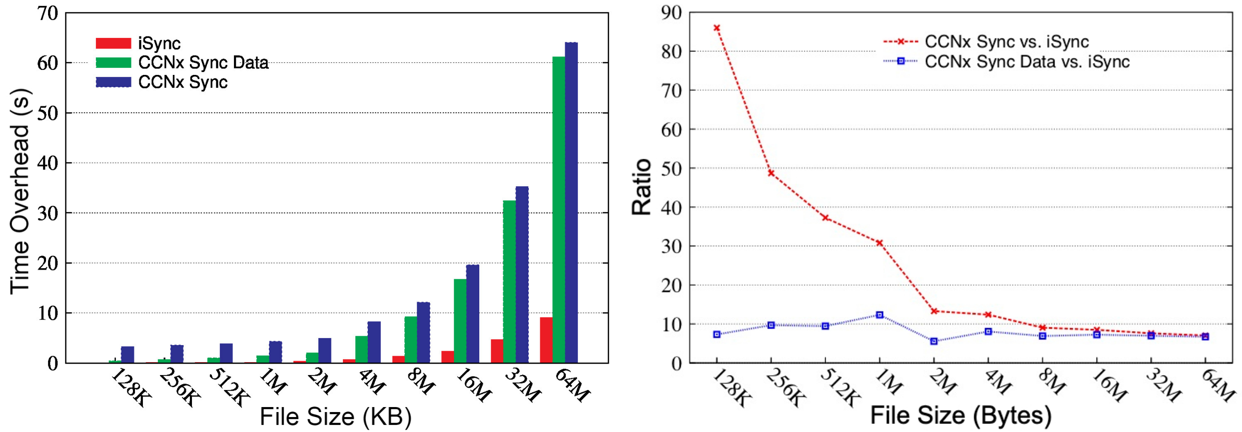


Figure 4.23: Impact of File Size on Time Cost (left) and Ratios of CCNx Sync vs. iSync (right).

Traffic Overhead

Traffic overhead impacts the scalability and efficiency of a synchronization protocol. We synchronized files of different sizes (from 128 KB to 64 MB) and measured the traffic overhead by capturing the network traffic, using tcpdump traces.

Figure 4.24 shows the number of packets transmitted by CCNx Sync and iSync for different files sizes, and the ratio between the protocols' overheads. In synchronizing a small file of 128 KB, iSync and CCNx Sync transferred 10 and 182 packets respectively. The numbers increase to 1032 and 49589 when the file size increases to 64 MB. From the ratio point of view, iSync is about 18 and 48 times more efficient than CCNx Sync on number of packets while sharing files of 128 KB and 64 MB respectively. We explain this ratio by looking at the number of packets sent by CCNx per one update as shown in the example demonstrated in the background section.

Figure 4.25 shows the number of bytes exchanged during the synchronization process. For synchronizing a file of 128 KB, iSync and CCNx Sync exchange about 160 KB and 204 KB. For a file of 64 MB, the traffic overheads increase to 65.5 MB and 83.6 MB. In all tests,

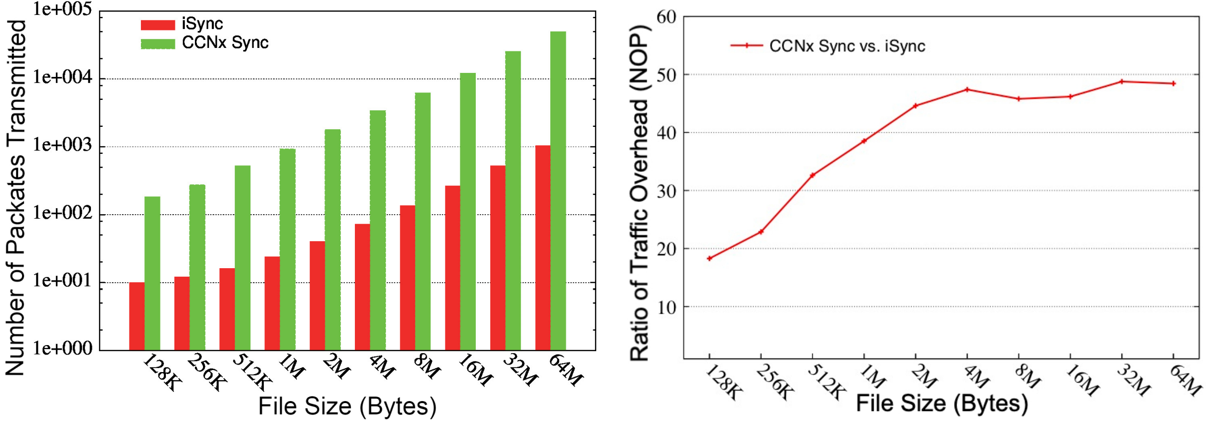


Figure 4.24: Traffic Overhead for Various File Sizes.

iSync exchanges a smaller number of bytes than CCNx Sync. However, the advantage of iSync over CCNx Sync in traffic amount is not as significant as in the exchanged number of packets. We believe that this might be the result of smaller packets sent by CCNx Sync compared to iSync.

The main reasons for the advantages of iSync over CCNx Sync can be summarized as follows. First, while CCNx Sync concentrates more on exchanging node information, iSync consumes more local computing resources for the IBFs' decoding process. It also causes the inconsistent differences between the number of exchanged packets and bytes in the experiments (most packets transmitted by CCNx Sync are smaller than 80 bytes). Second, the max data unit limitations of the protocols are different. Both CCNx Sync and iSync use jumbo packets to deliver the files. However, the max size of CCNx Sync content packets is about 22KB (which might be disassembled according to the MTU limit of switches and routers). The smaller the size limit, the more packets that need to be sent. To improve the performance, iSync uses content packets of 64KB, which is the optimal value the current CCNx daemon can handle. It significantly reduces the number of packets, as is shown in Figure 4.24. Third, extra interest and content packets are exchanged by the CCNx repository, and therefore we can expect a larger traffic overhead.

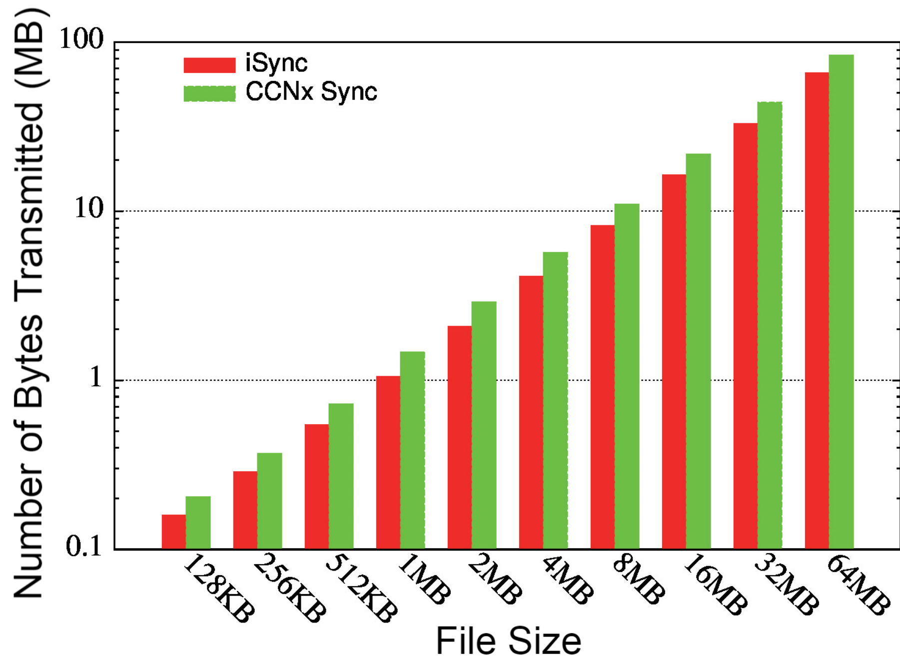


Figure 4.25: Number of bytes vs. File Size.

Scalability Test

In this section, we evaluated and compared the performance of the iSync and CCNx Sync protocols in multiple network topologies with the number of nodes ranging from 2 to 32. As described in the background section, interest packets are satisfied by a cached content or a producer. To verify the correctness of the protocol in the NDN architecture, we tested it in chain, ring, star and full mesh topologies, representing weak to strong connectivity. Files were inserted into the start node of the chain topology, and the core node (which has direct connections to all other nodes) of the star network.

Figure 4.26 shows the scalability results. We found that the synchronization times of CCNx Sync were very unstable; therefore, we plotted the average results in the bar charts and added annotations for the worst results on the top of each bar. To provide a better display

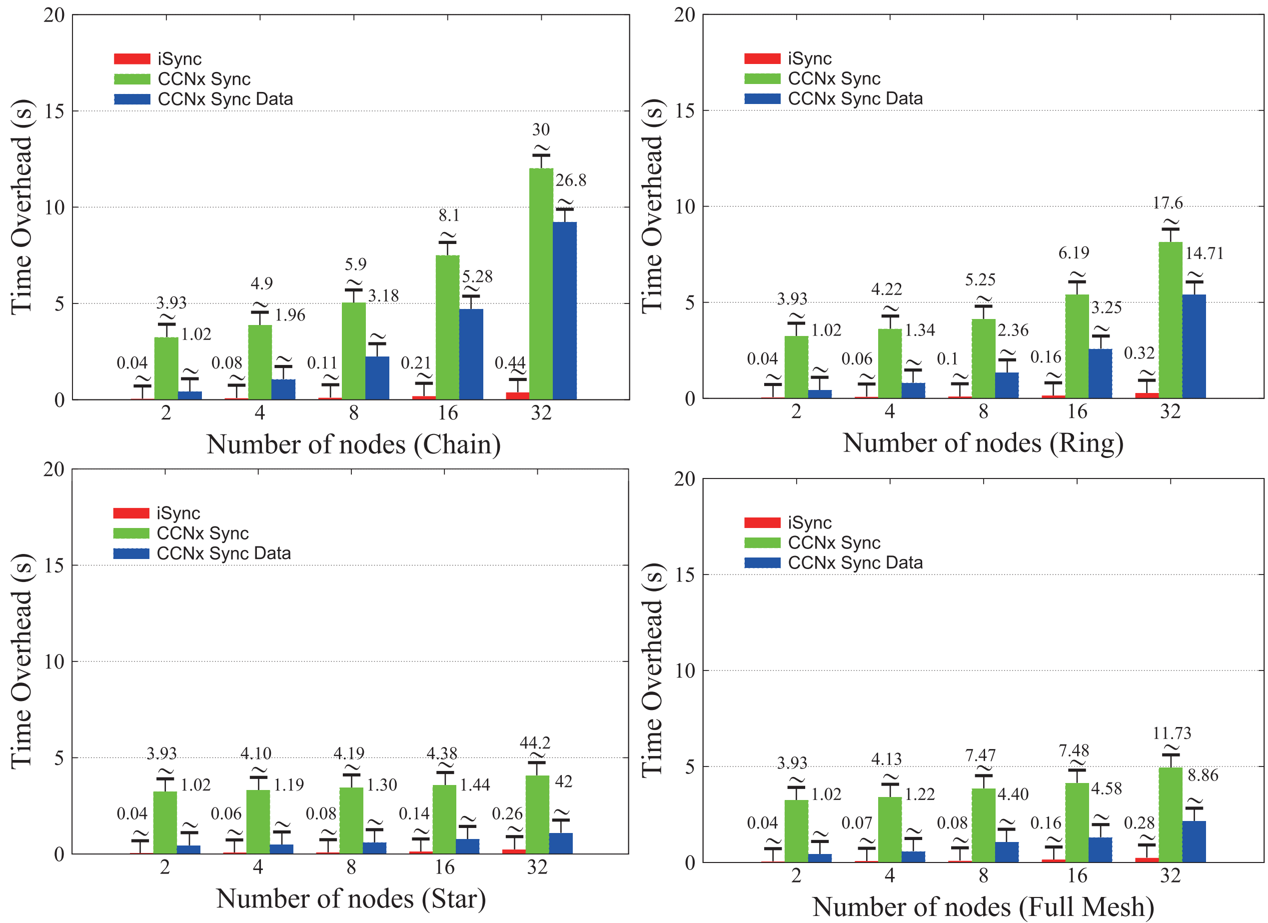


Figure 4.26: Average Synchronization Time of iSync and CCNx Sync in Networks of Various Topology Types (with max recorded results on top of each bar).

of the protocols' performance, we also plotted the results of CCNx Sync data to ignore the API overhead.

As expected, the results show that the chain topology produces the slowest synchronization times in both protocols. Since we inserted the files into the start of the chain, each file should be delivered one by one until all nodes have received it. Moreover, the time overhead shows an approximately linear relationship with the number of nodes. The variation in the results of the chain topology is large, especially for CCNx Sync. The average and max times for iSync to synchronize a 128 KB file in a 32 nodes network are 0.37 and 0.44 seconds

respectively, and CCNx Sync results are 12 and 30 seconds. Therefore, the max time is about 2.5 times larger than the average time. Ratio similar to that is found in the results for iSync and CCNx Sync Data (about 2.9 times).

The results show that synchronizing a file in a ring topology is much faster than in a chain topology, because the parallel synchronization starts from two directions. For example, the synchronization time in a ring network of 32 nodes is nearly half of that in a chain network. The variation is also much smaller.

The star topology achieves the fastest synchronization time. In our experiment, we inserted the file into the central node that has direct access to all the other nodes; therefore, the data could be delivered to all nodes at the same time. During the experiments, we found that large variations occasionally occurred in the results of CCNx Sync in a 32 node network. After careful consideration, we believe that this variation is caused by the forwarding plane of CCNx and the design of CCNx sync. As illustrated in the example in the background section, CCNx Sync is a pairwise protocol that operates between neighbors. Therefore, when adding a file into the central node in a star topology, the protocol operates on every two neighbors separately. Moreover, the behavior of the CCNx forwarding strategy delays the RootAdvise notification sent to a subset of the neighbors.

Compared to other types of topologies, full mesh topology networks have the strongest connectivity. However, the time overhead of data synchronization in full mesh topology is no better than that in a star topology. All nodes publish digests after receiving any updates, and each of them has to process redundant data units from synchronization protocols (iSync or CCNx Sync).

Recovery Overhead

The recovery overhead of iSync from false positive errors is mainly concentrated in local computing. Specifically, the effectiveness of this scheme is highly dependent on the hash algorithm and local computing power. To evaluate the effectiveness and efficiency of the recovery scheme, we randomly deleted a few file contents in the defined collections and measured the time it took the protocol to reconcile the differences. We manually deleted up to 10% of the collection files, and repeated each test ten times.

As shown in Fig 4.27, the time overhead of the recovery scheme shows a linear relationship with the size of the collection. It costs about 10 ms for the iSync host to recover missed file names from a data collection of 1K files. The time overhead grows to about 3.5 seconds for recovering all names from a collection of 1024K files. In those experiments, and as guaranteed in theory, iSync could always recover and reconcile discarded items.

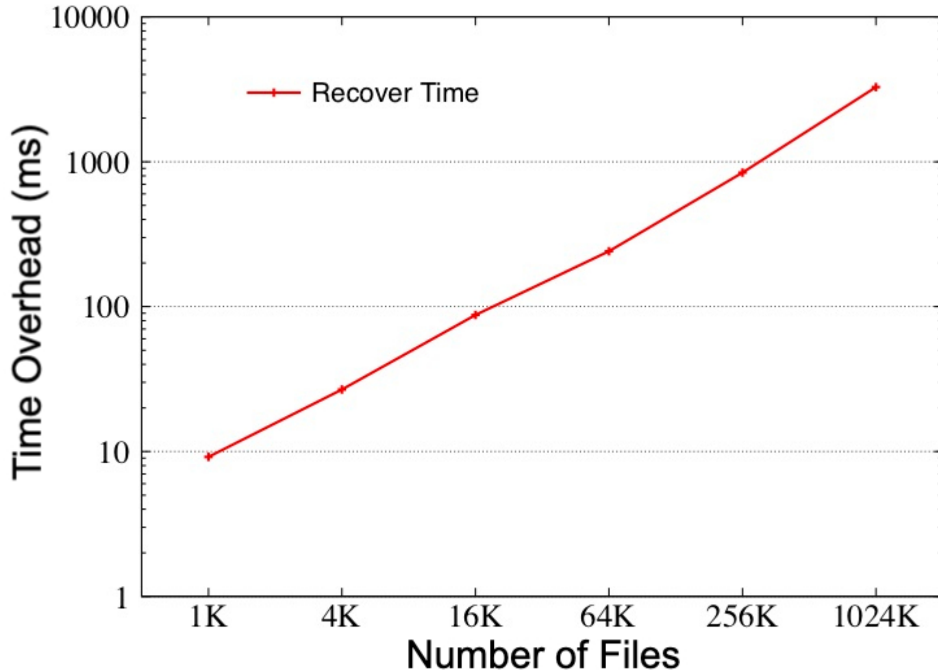


Figure 4.27: Recovery Time vs. Number of Items after 10% of items have been deleted randomly

4.5 Conclusions

This chapter discussed the task of data synchronization in ICN, and explored general-purpose sync mechanisms that can be used to form the basis of sync ICTs. Our work has illustrated two different sync methods that represent different tradeoffs: ICT-Sync was designed to represent a set of sequenced namespaces and can synchronize an update within 1.5 RTT, while iSync synchronizes a set of arbitrary names, and requires an additional RTT to map IBF hashes into names. We implemented iSync for in the CCN architecture, and compared it with the CCNx synchronization protocol, CCNx Sync [20], and evaluated its scalability and performance over different network topologies.

We implemented ICT-Sync as a two-component ICT, an endpoint library and intermediate sync component, we evaluated it in both reliable and lossy environments, and we compared

it to ChronoSync[85]. Our experiments demonstrate how a Sync ICT can keep applications in the information plane, decoupled from connectivity characteristics, without a change to the application or its libraries. Therefore, applications with sync requirements can rely on ICT-Sync to solely operate in the information plane.

Table 4.2 summarizes the properties and considerations of sync as an ICT.

ICT property	**ICT-Sync**
Broadly applicable needs	Synchronize a set of namespaces
Application examples	File sharing Group chat applications Key management tools Routing protocols
Known challenges	Dataset representation, and algorithms to find the set-differences
ICT API	Register: App registers a prefix to be synchronized Add: App inserts a new name into the dataset SyncUpdate: Application callback notifying a change in the dataset
Intermediate ICT	Acts as a sync consumer Stores all or selected Data packets according to the network policy

Table 4.2: Sync Properties as an ICT

Chapter 5

Push Notifications

This chapter explores push notifications as a new application abstraction in ICNs. Like data synchronization, this chapter also aims to address this abstraction for a group communication model, in which a consumer can also be a producer. Unlike Sync, here, a consumer in the group is not interested in maintaining data consistency over time, but wants new data to be pushed to him as soon as possible, even at the cost of dropping previously pushed data.

Pushing data in ICNs is a complicated problem due to the following challenges:

1. NDN is a pull-based paradigm, and it does not natively support pushing Data packets unless someone expressed an Interest packet first.

2. Naming future and unknown data is a complex task, especially in a group communication where multiple endpoints act as data producers.

We elaborate on these challenges before we discuss related works in the area of push data in ICNs.

First, in ICNs, no Data packet can be sent without an Interest packet requesting it. A common approach to address this challenge is to use long-lived Interests. In ICNs, every Interest packet carries a lifetime value that determines how long an unsatisfied Interest should be kept in the PIT before it is removed. To extend the lifetime of an Interest packet, a new Interest for the same name must be sent. A long-lived Interest is an Interest packet that is sent periodically in intervals that are shorter than the packet lifetime, and therefore maintains a continuous valid PIT entry. Using long-lived Interests, a producer can push Data packets at any given time.

This leads to the second challenge: what name should be used in the pending interest? In pub-sub systems, where a single producer pushes content to a group of consumers, any routable name can be used to push one Data packet to all the consumers. However, in the ICN communication model, in which endpoints act as both consumers and producers, any node can push notifications to others, and therefore the namespace design plays a crucial role.

There are two approaches. First, a long-lived Interest can carry a producer identifier in the notification's routable prefix, and every participant in the group can send a separate Interest to each one of the other group members. While this approach is straightforward, it consumes more network overhead (state and bandwidth), and it requires that a consumer know all the producers in its group. Moreover, a consumer would be required to maintain this knowledge and be updated when a new producer joins the group or when other leaves.

The second naming approach uses one general name multicasted to all group members in one long-lived Interest. This approach consumes one entry in the network PITs, and anyone in the group can satisfy the pending Interest and push Data packet. However, this approach does

not support simultaneous multi-source Data packets because the first Data packet consumes the PIT entry, so the second one will be dropped.

Both naming approaches must address another challenge which emerges with long-lived Interests — how to differentiate one long-lived Interest from another? Clearly, the naming convention must guarantee that a new long-lived Interest is different from the one satisfied before it, so a producer or the CS would not push the same Data again. An intuitive way to achieve it is by adding a sequence number to every long-lived Interest. For instance, if a consumer sends a producer-specific long-lived Interest, then /Alice/1 is a long-lived Interest sent to Alice requesting Alice's first notifications, and /Bob/3 is a long-lived Interest requesting Bob's third notification.

Another approach is adding a timestamp a long-lived Interest to ask for data generated after the timestamp. For instance, a long-lived Interest with name /notification/t3 asks for data generated after t3. However, this approach does not support unordered data when names are used without a producer identifier. To see why, consider that consumer C sends a long-lived Interest looking for notifications that occur after t1. Producer P1 responds with a notification that occurred at t2, and a little after, producer P2 responds with another notification occurred at t3. If t3 arrives at C before t2, then C would ask for notifications that occurred after t3, and therefore will miss the notification occurred at t2.

As the preceding demonstrates, supporting push notifications in ICNs is a real challenge. As we will see following a brief discussion of related work, the use of an ICT represents a promising path forward.

5.0.1 ICN Push in Related Works

Recent ICN works have studied the problem of handling push-based notifications. [9] discusses and evaluates three schemes to support push-based traffic in NDN: Interest notification, unsolicited data, and virtual Interest Polling. Interestingly, due to the trade-off between a device's efficiency and the network overhead, it was found that no scheme is better than the others, and that the selected scheme should be decided according to the expected traffic load and the constraints of devices. [10] discusses a framework for multi-source data retrieval in IoT networks. This framework uses exclude filters to allow selective data retransmissions, and controls PIT deletions to support multiple Data packets for the same Interest. [58] proposes a new ICN packet type for push notifications.

The work in [10] and [58] propose changes to the ICN architecture to better support push notifications. In contrast, our work focuses on designing and developing a proof-of-concept multi-source notification ICT without any modifications to the architecture. Our goal is to show that such an ICT can be a primitive in the current implementation of NDN, without any modification to the PIT or the forwarder, and without adding a new packet type. We do not argue that our work is more efficient than the alternatives. Rather, we argue that a mechanism for push-based notifications can evolve as an ICT because it supports common application needs, and it can allow NDN applications to stay in the information plane, free from connectivity concerns.

5.1 ICT for Push Notifications

The works mentioned in Section 5.0.1 discuss push notifications as an essential mechanism in ICN, but present it as a hard problem. To the best of our knowledge, there is currently

no mechanism that solves the problem of push notifications in a general-purpose way, and without introducing modifications to the ICN architecture. In this section, we propose to identify push notifications as a problem that requires an ICT abstraction.

The goal of an ICT abstraction for push notification is to provide different applications with a clear and easy API to push notifications, and to implement the network mechanisms required to achieve this behavior. The high-level role of the ICT component is to decouple applications from the details of connectivity. Therefore, an ICT abstraction for push notifications is required to decide what to do when unreliable connectivity, such as intermittent links or network delays, prevents a reliable delivery of a pushed notification.

For instance, consider a GPS-tracking application in which the movement of multiple end-points is tracked on a grid. If a user moves from point 1 to point 2 and then to point 3, and the Data packets pushing the two movements are lost, should the producer or the network keep retransmitting both notifications? Or should it drop the packet announcing point 2 and move on to deliver only the one announcing on point 3? The goal of defining a clear ICT abstraction is to address this question, and to guarantee that an application can choose an ICT that addresses its application-level needs.

We discuss possible abstractions to this question in section 5.1.1, but we first summarize and say that the role of an ICT for push notifications is two-fold: Simplify the process of push notifications in the pull-based ICN paradigm, and implement in-network information-oriented mechanisms to comply with the selected application abstraction.

5.1.1 Abstractions for Push Notifications

As mentioned at the beginning of this chapter, ICT for push notifications is not required to maintain data consistency over time, and therefore, an intermediate ICT component

deployed in the network should not simply store all Data packets. In this section we explore two abstractions for push notifications in ICN: a producer-oriented notification abstraction, and a time-oriented notification abstraction. Each of these abstraction addresses different application needs, and therefore, each requires different in-network mechanisms.

1. Producer-Oriented Notifications Abstraction: A consumer requests to be notified on the latest data produced by each producer in the group, regardless of the specific time it was produced.

   For instance, consider a group chat application, such as Slack [72] or Google hangouts [70], in which a connectivity sign is presented for each participant: A green mark when a participant is 'online', a yellow mark when he is 'away' and a red mark when he is 'offline'. Here, a consumer using the notification abstraction requests to be notified on the latest status of each member in the group. In this case, a lost notification is relevant only if it is the most recent notification. This way, when a new consumer joins the group he receives only the latest and most accurate connectivity status of every participant.

   Another example of an application that can use this abstraction could be a GPS-tracking application, in which participants report their locations as they move. A consumer in this application wants to be notified whenever a producer is moving. However, if the producer made multiple steps while the network was disconnected, then a consumer only wants to get the latest one.

2. Time-Oriented Notifications Abstraction: A consumer requests to be notified on the latest data produced in the group, only if this data was produced within a specific (recent) interval of time. This abstraction does not guarantee any producer-specific

notification, but only that the latest set of content items generated in the group is pushed.

For instance, consider a news feed that presents the headlines of the latest hour. Using this abstraction, the consumer gets all the headlines produced within that hour. It may present multiple headlines pushed by the same news resource within this hour, and zero headlines from others if they didn't produce new data during that time.

Another application example that can use this abstraction is a sensor network application, in which sensors report their measurements only if they pass a pre-defined threshold. In this example, a consumer requests to get all the measurements that passed their thresholds recently, even if some of them were pushed by the same sensor. This abstraction promises that in case of packet loss, a measurement would not be lost unless it expired. This can be helpful if a consumer wants to identify patterns of multiple measurements pushed by the same sensor within a short amount of time, even if some of them were initially lost due to bad connectivity.

5.1.2 Push Notifications Vs. Data Synchronization

One could say that these two abstractions can be simply satisfied by Sync, because if Sync synchronizes the entire set of names, it synchronizes the latest. However, we argue that the mechanisms required to support full namespace synchronization are heavyweight for applications that need only the latest update(s). Moreover, all existing Sync mechanisms require at least 0.5 Round-Trip-Time (RTT) to synchronize the name of the data, and an additional RTT to fetch it [20, 29, 84, 85]. The goal of this chapter is to explore whether Data in the group can be pushed to all consumers immediately, with the best case scenario of within 0.5 RTT.

To illustrate the differences between push notifications to Sync, consider the example of the Slack-like application, where a connectivity sign represents if a participant is 'Online', 'Offline' or 'Away'. Using mechanisms for data synchronization the application gets all the sequence numbers of connectivity events, and can then fetch the latest to find out the current status it needs to present. Using a push mechanism, an application can simply get the latest status to be presented. Moreover, in the case of a disconnected or lossy environment, or when a new consumer joins the group, there is no point in sending notifications for previous events that show that a participant was 'Away', 'Offline' and 'Away' again, but he is now 'Online'. Instead, it is much more useful for the application to simply get the latest status rather than getting all the statuses pushed when a link was down.

In addition, one could say that the abstraction for push notification can be achieved by pub-sub mechanisms already implemented in NDN [84]. However, we argue that the communication model of the two is different, and therefore, each requires different mechanisms. First, pub-sub mechanism focus on retrieving large streams of data from specific known sources. However, the goal of our proposed abstraction is to push relatively small notifications, fit a single Data packet, from multiple dynamic sources in group communication.

To conclude the objectives of this chapter, we reiterate that the goal of this chapter is to explore mechanisms for dynamic and multi-source event-driven applications, with the goal of pushing small notifications within a one-way delay latency.

5.2 ICT-Notify

In this section we describe ICT-Notify, which is an implementation of the push notification abstractions we presented in Section 5.1.1. A detailed flow of ICT-Notify will be presented

in Figure 5.2. But first, we define the high-level requirements from ICT-Notify to ensure that it is a primitive ICT that can be used by different types of ICN applications:

- Supports multi-source data retrieval.
- Supports scenarios in which the exact number and identity of the group members are unknown.
- Does not require changes to the ICN architecture or to its forwarders.
- Introduces an API for applications.
- Maintains trust relationships as defined by applications.
- Its intermediate component must not understand application-specific semantics, and is not required to decrypt the application's content.

As mentioned, the goal of ICT-Notify in not to support data consistency over time, but instead, to push only the relevant data as quickly as possible, and preferably, within one-way delay latency. Therefore, applications that use ICT-Notify must tolerate notification loss, if it is no longer relevant as defined by the abstraction.

Like ICT-Sync, ICT-Notify consists of an application library and an intermediate process that can be deployed by the network operator. The implementation of ICT-notify follows the long-lived Interest scheme. As discussed in [9, 58], this scheme presents challenges in namespace design. It is important to mention that our goal was to implement a proof-of-concept ICT mechanism, and therefore, this work does not study the cost of long-lived Interests on the PIT, and does not evaluate the tradeoffs between long-lived Interests and architectural modifications to include a new type of push packets.

To address our requirement for dynamic environments with unknown identities of producers, we choose the general-purpose naming approach, in which a consumer sends one long-lived Interest for future push notifications, and not a separate long-lived Interest for each producer in the group. ICT-Notify supports simultaneous data delivery through its namespace design, with the penalty of additional latency.

The major challenge in sending long-lived Interests is to distinguish one Interest from another. In the past, this has been done by adding a sequence number or a timestamp to the Interest name. As discussed in the first section of this chapter, a sequence number is used to request the next sequenced notification, and a timestamp is used to request a notification that occurred after that time. However, none of these options provide a complete solution. Sequenced Interests require the producer's identity in the namespace, and timestamps do not support unordered data delivery. To address this challenge, ICT-Notify follows the sync approach. Instead of simply naming future data to be pushed using a timestamp or a sequence number, ICT-Notify names the state of the data.

5.2.1 ICT-Notify API

To participate in a notification group, application parties use ICT-Notify API and *Register* an application name. All the parties that register the same name join the same notification group. To push notifications in the group, parties call the *Push* function and provide an ***event*** to be pushed.

We define an event to be the content of the notification pushed in a Data packet. Each event is in the form of a hierarchical name, for instance: "/sensorA/temperature/50" or "/User/Alice/Location/X/Y". Since the event is the Data payload, and not the Interest or Data name, it can be of any form, and it can contain producers' identities if desired

by the application. When an application pushes an event, ICT-Notify API generates a corresponding event identifier.

The event identifier depends on the specific notification abstraction. When ICT-Notify implements the producer-oriented abstraction, in which a consumer requests to be notified on the latest notification per producer, the event identifier is represented by a tuple of producer identifier and a sequence number, similarly to ICT-Sync representation. When ICT-Notify implements the time-oriented abstraction, in which a consumer requests to be notified on all the events occurred in the last X interval, then the event identifier is represented by a timestamp. ICT-Notify uses nanosecond accuracy to minimize collisions of frequently generated events.

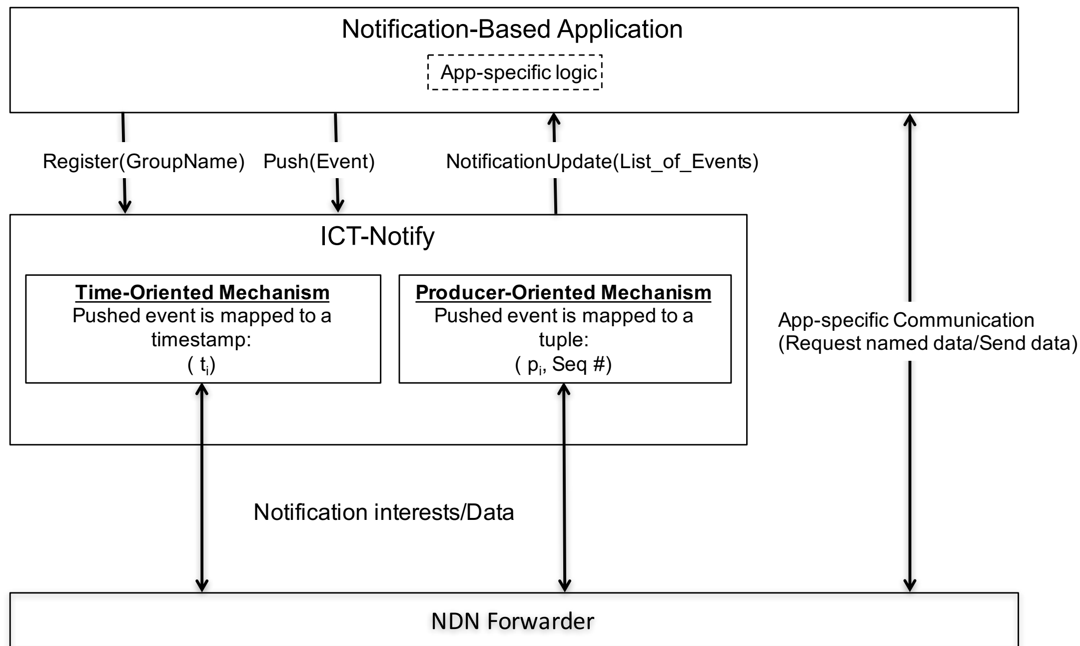


Figure 5.1: ICT-Notify API for the Two Abstractions

Figure 5.1 presents the high-level API of ICT-Notify, and shows that ICT-Notify implements mechanisms for the two different push notifications abstractions: The producer-oriented abstraction and the time-oriented abstraction.

To illustrate the interaction between the application and ICT-Notify, consider the following GPS-Tracking application that follows the producer-oriented notification abstraction: Using ICT-Notify API, Alice, Bob and Ted register the same application name to participate in a notification group, which triggers multicast of three long-lived Interests, one from each participant, for the same group name. When Alice moves to a new location X,Y, she uses the push call in ICT-API to create an event and push it to others. The event content can look like "/User/Alice/Location/X/Y". ICT-Notify API will then create the event identifier is (Alice,1). The ICT-Notify API notifies Bob and Ted about Alice's notification using the NotificationUpdate Callback.

5.2.2 ICT-Notify Protocol and Namespace Design

ICT-Notify follows the next namespace design for its long-lived Interest: *<ICTName>/<AppName>/<ConsumerState>*. <ICTName> identifies ICT-Notify packets in the network. <AppName> is the group name registered by the participants to differentiates one group from another, and it also supports multicasting to only application parties. <ConsumerState> consists of a ***notification list*** which is the list of relevant events.

As mentioned in the previous subsection, the specific representation of notifications in the notification list depends on the abstraction ICT-Notify implements: a tuple of producer identifier and sequence number in the producer-oriented abstraction, or a timestamp in the time-oriented abstraction. When ICT-Notify implements the producer-oriented notification

abstraction, then the notification list consists of a list of tuples, each pair represents a member identification and its latest sequence number. When ICT-Notify implements the time-oriented notification abstraction, then the notification list consists of the list of timestamps generated within the relevant time frame.

The name of a notification Data packet consists of the name of the long-lived Interest and an additional name component – <ProducerState> – that lists the latest notifications known by the producer. Before a new long-lived Interest is sent, old notifications are removed from the <ConsumerState> name component. In the producer-oriented abstraction, events of previous sequence numbers are removed, while in the time-oriented abstraction, expired timestamps are removed. In our current implementation, the expiration of a timestamp is configurable, and can be determined by the application.

When receiving an Interest, a producer compares its local notification list of known event identifiers with the identifiers in the received <ConsumerState> name component, and determines if it has new events unknown to the consumer. If so, the producer responds with a Data packet that consists of 1) its local notification list in the <ProducerState> name component, and 2) the events that correspond to the missing identifiers in the payload. Therefore, the Data name contains two notification lists: the consumer's out-of-date list from the Interest name, and the latest up-to-date producer's list. The Data payload contains the set-difference of the two lists, which is a list of the missing events.

Since the long-lived interest scheme guarantees that there is always an Interest in the PIT, a producer can push Data notifications whenever the application generates an event. Having <ConsumerState> in the Interest name allows every participant, including the intermediate ICT process, to find the relevant set-difference and to respond quickly with Data. It also enables relevant data retrieval from NDN caches.

For the time-oriented notification abstraction, we explored two implementations of <ConsumerState> and <ProducerState>: an invertible Bloom filter (IBF) [27], and a simple vector. We encoded each of these data structures in a name component and compressed these names using the Bzip algorithm. We found that although IBFs are considered to be efficient data structures, they consume more memory than vectors when supporting small numbers of items (in the hundreds). Our experiments showed that Interest names with vector representations are six times smaller than Interest names with IBF representations. A quantitative evaluation of vectors is presented in Section 5.2.3. Since the goal of ICT-Notify is to push only the latest notifications, old names are removed from the vector and therefore we do not anticipate a large number of events encoded in the name components.

Furthermore, ICT-Notify API allows applications to define filters on the events they request to receive. This way, different instances of the applications can choose to be notified about specific events, and not about every event generated by a producer. For instance, a sensor in an IoT network can wake up and send its temperature periodically, but an application can choose to be notified only when the temperature is above X or below Y. ICT-Notify follows the implementation of the schematized trust model [82] to enable regular expressions as configurable filters, as is done to define trust relationships.

The intermediate process of the ICT works similarly to an end-point consumer. It maintains a <Consumer State> list according to Data names it sees, and it generates long-lived Interests to notify others of its state. However, it does not produce notifications, and unlike ICT-Sync, it does not need to store all Data packets. Instead, the intermediate ICT component keeps tracking the system's latest events: either the latest sequence number per producer in the producer-oriented abstraction, or the latest timestamps in the time-oriented abstraction. Then, the intermediate ICT-Notify component stores only the corresponding payloads of the unexpired events, and can responds to Interests just like any end-point party, without

the need to decrypt the data. This way, ICT-Notify supports applications even when the network is disturbed, with the worst case of no Synchronous End-to-End Path (SEEP).

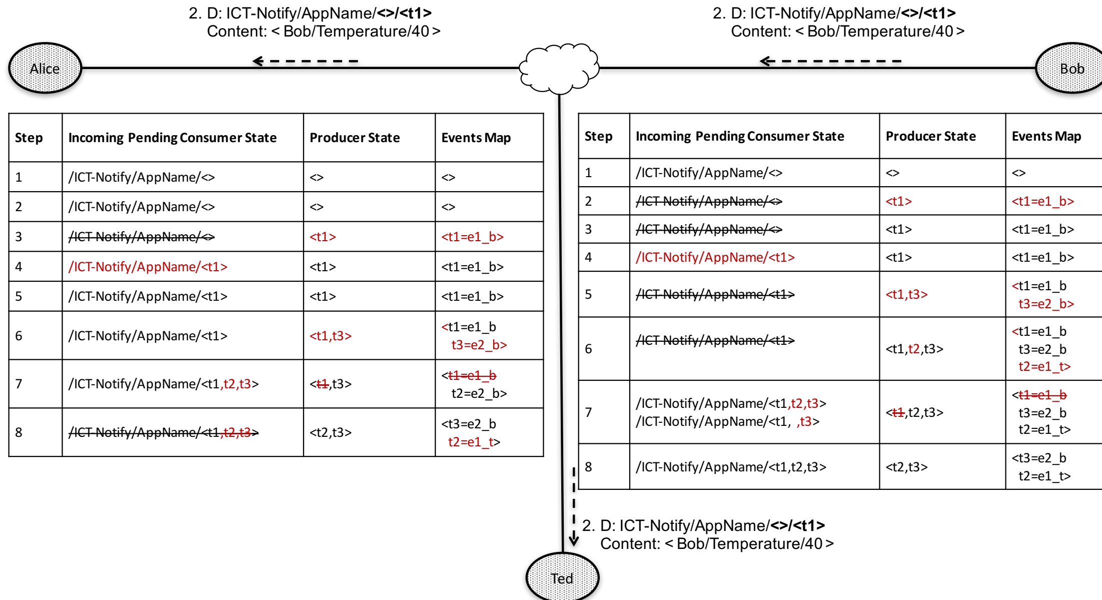
Figure 5.2 summarizes the high-level flow of ICT-Notify, when it implements the time-oriented push mechanism. The red text represents changes between the steps. The example illustrates how ICT-Notify pushes data immediately to all the consumers, and how it deals with simultaneous pushes. Here, an intermediate is not deployed in the network, however, in network with intermittent links, the intermediate ICT-Notify component could be deployed and act as a consumer.

5.2.3 Evaluation

The experiments discussed in this section focus on showing the capabilities of the time-oriented ICT-Notify abstraction rather than its scalability. Similarly to ICT-Sync, we conducted all our experiments on the Open Network Lab (ONL) [76] to control network connectivity factors such as link delay, packet loss rates, and link availability.

Experimental Setup

Our ICT-Notify experimental setup is similar to the setup we used to evaluate ICT-Sync. The NDN topology, presented in Figure 5.3, is mapped to the ONL topology presented in Figure 5.4. The topology consisted of six endpoints and four intermediary NDN routers running on 10 two-core machines, and five Ubuntu Linux (16.04.4) software routers. All two-core machines ran NFD [5] version 0.6.1, and each experiment was repeated three times. NDN routers were a combination of two machines - a Linux software router and a machine that ran the NFD code.



Step	Incoming Pending Consumer State	Producer State	Events Map
1	/ICT-Notify/AppName/<>	<>	<>
2	/ICT-Notify/AppName/<>	<>	<>
3	~~/ICT-Notify/AppName/<>~~	<t1>	<t1=e1_b>
4	/ICT-Notify/AppName/<t1>	<t1>	<t1=e1_b>
5	/ICT-Notify/AppName/<t1>	<t1>	<t1=e1_b>
6	/ICT-Notify/AppName/<t1>	<t1,t3>	<t1=e1_b t3=e2_b>
7	/ICT-Notify/AppName/<t1,t2,t3>	<~~t1~~,t3>	<~~t1=e1_b~~ t2=e2_b>
8	~~/ICT-Notify/AppName/<t1,t2,t3>~~	<t2,t3>	<t3=e2_b t2=e1_t>

Step	Incoming Pending Consumer State	Producer State	Events Map
1	/ICT-Notify/AppName/<>	<>	<>
2	~~/ICT-Notify/AppName/<>~~	<t1>	<t1=e1_b>
3	~~/ICT-Notify/AppName/<>~~	<t1>	<t1=e1_b>
4	/ICT-Notify/AppName/<t1>	<t1>	<t1=e1_b>
5	~~/ICT-Notify/AppName/<t1>~~	<t1,t3>	<t1=e1_b t3=e2_b>
6	~~/ICT-Notify/AppName/<t1>~~	<t1,t2,t3>	<t1=e1_b t3=e2_b t2=e1_t>
7	/ICT-Notify/AppName/<t1,t2,t3> /ICT-Notify/AppName/<t1, ,t3>	<~~t1~~,t2,t3>	<~~t1=e1_b~~ t3=e2_b t2=e1_t>
8	/ICT-Notify/AppName/<t1,t2,t3>	<t2,t3>	<t3=e2_b t2=e1_t>

Step	Incoming Pending Consumer State	Producer State	Events Map
1	/ICT-Notify/AppName/<>	<>	<>
2	/ICT-Notify/AppName/<>	<>	<>
3	~~/ICT-Notify/AppName/<>~~	<t1>	<t1=e1_b>
4	/ICT-Notify/AppName/<t1>	<t1>	<t1=e1_b>
5	~~/ICT-Notify/AppName/<t1>~~	<t1,t2>	<t1=e1_b, t2=e1_t>
6	~~/ICT-Notify/AppName/<t1>~~	<t1,t2,t3>	<t1=e1_b t2=e1_t t3=e2_b>
7	/ICT-Notify/AppName/<t1,t2,t3> /ICT-Notify/AppName/<t1, ,t3>	<~~t1~~,t2,t3>	<~~t1=e1_b~~, t2=e1_t t3=e2_b>
8	/ICT-Notify/AppName/<t1,t2,t3>	<t2,t3>	<t3=e2_b t2=e1_t>

1. All three send Interest: ICT-Notify/AppName/<>
2. a. Bob pushes a new event with content: Bob/Temperature/40 (marked e1_b)
   b. ICT-Notify maps the content to t1, and pushes Data packet
3. Alice and Teb receive Bob's Data, ICT-Notify updates their producer state.
4. All send a new long-lived Interest: ICT-Notify/AppName/<t1>
5. a. Ted pushes a notification at t2: Bob/Temperature/45 (marked e2_b),
   b. Bob pushes a message at t3: Ted/Temperature/43 (marked e1_t).
   c. ICT-Notify libraries on both Bob and Ted push their content simultaneously.
6. a. Bob's notification consumes the PIT first, so Alice only gets Bob's notification.
   b. Ted and Bob get each others notifications.
7. a. All send their new long-lived Interests.
   b. The notification pushed at t1 expires.
8. Bob and Ted respond to Alice's Interest, and push notification e1_t.

Figure 5.2: ICT-Notify Example

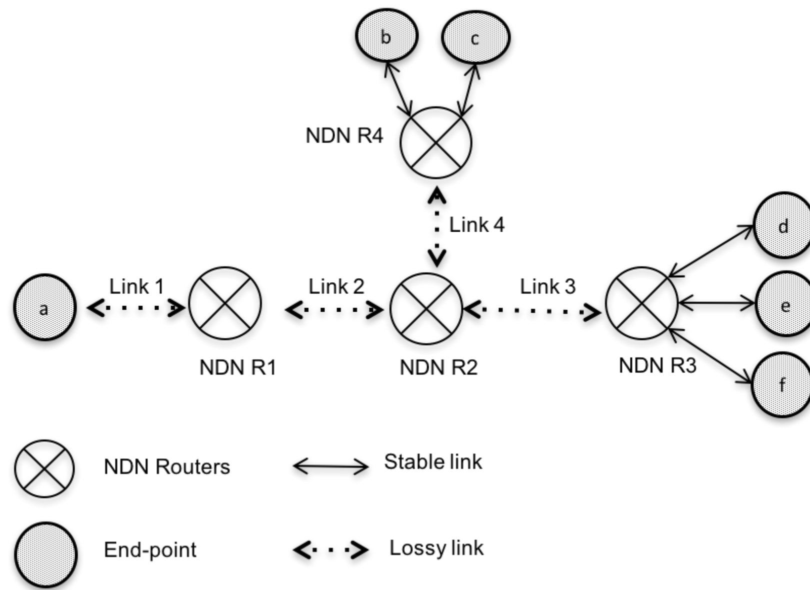


Figure 5.3: Tested NDN Topology

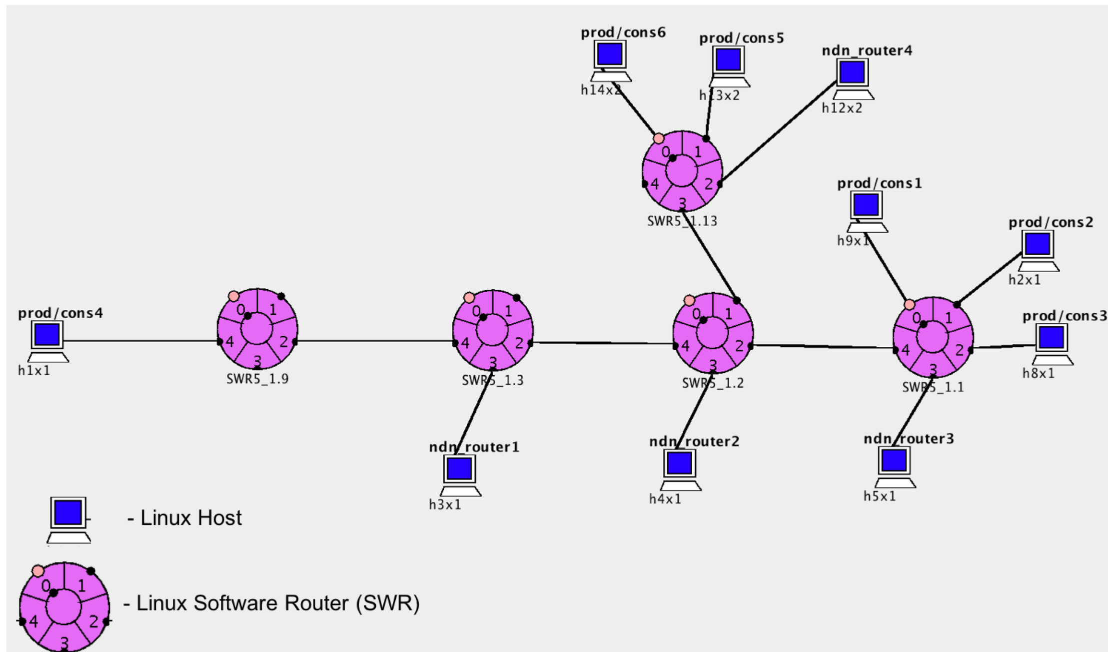


Figure 5.4: Mapping Topology onto Physical ONL Hardware

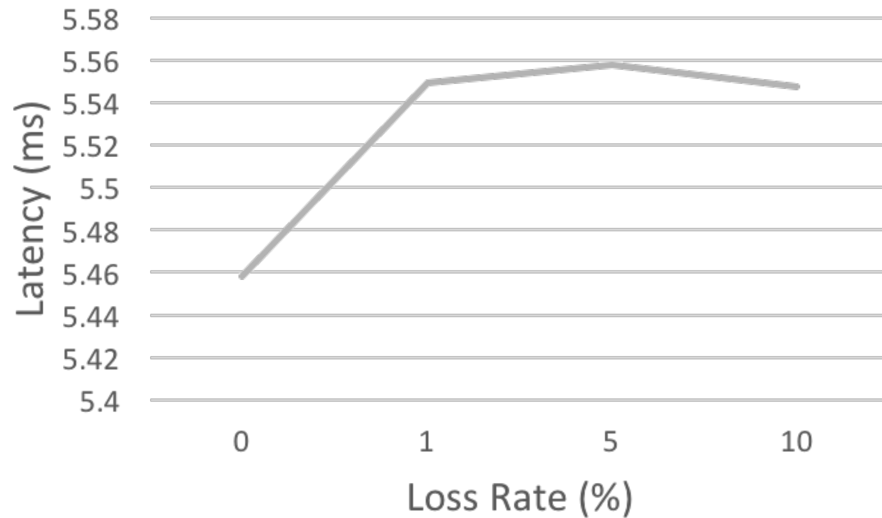


Figure 5.5: Average Push Latency over Different Loss Rates (ms)

The Network Time Protocol (NTP) was used on ONL machines to ensure the consistency of timestamps. We implemented a simple application that uses the time-oriented notification abstraction to push notifications to participants in the group.

ICT-Notify over Different Connectivities

First, we evaluated push latency of a single producer in the group. We configured node 'e' to push between 80-100 notifications, one every 1-4 seconds. The notification push time was recorded as the event identifier, and we collected the time the notification arrived at the consumer 'a'. Figure 5.5 shows the average notification times when the links were stable and reliable, and without an intermediate ICT-Notify at intermediate routers.

The results demonstrate how the latency of pushed notifications remained stable over a small percentage of loss rates. Here, we set notifications to expire one second after they have been sent. We could not evaluate larger loss rates because a large percentage of the pushed notifications expired before they reached the consumer.

Second, we evaluated push latency over different loss rates, when link 1 and link 2 alternated for different amounts of times, hence, there was never a SEEP. Here, we set notifications to expire after five seconds, and configured nodes 'b'-'f' to push 100 notifications in random intervals of 1-4 seconds. When running this experiment without the deployment of the intermediate ICT process, zero notifications received by the consumer. Therefore, we ran the same experiment with an intermediate ICT-Notify deployed on NDN R1 and NDN R2, and we present the results in Figure 5.6.

The x-axis of Figure 5.6 indicates the different loss rates we tested, and the y-axis indicates the latency for the different up and down times we tried. The results show that notifications are pushed faster when the links alternated for a short amount of time, with the exception of alternating the links for 500 ms. After a thorough investigation, we found that the ndn-cxx library, the library we use for ICT-Notify implementation, does not support multi-threaded applications, and can process incoming Interests and Data packets only from a single thread. However, the application we implemented for our tests consisted of two threads: a producer thread and a consumer thread. While the nodes in our tested topology sent and received Interests and Data in random intervals, nfd processed them all in one-second intervals as if they were all sent by a single thread. Therefore, alternating links for less than one seconds did not improve push latency.

Dynamic Name Size

Next, to understand the limitations of our namespace design, in which a list of timestamps or tuples are encoded in a name component, we evaluated the dynamic sizes of the Interest and Data names. Here, we deployed the intermediate ICT-Notify process on NDN R1, and we programmed the five producers to start sending simultaneous notifications approximately four seconds after we started each experiment. We set notifications to expire five seconds

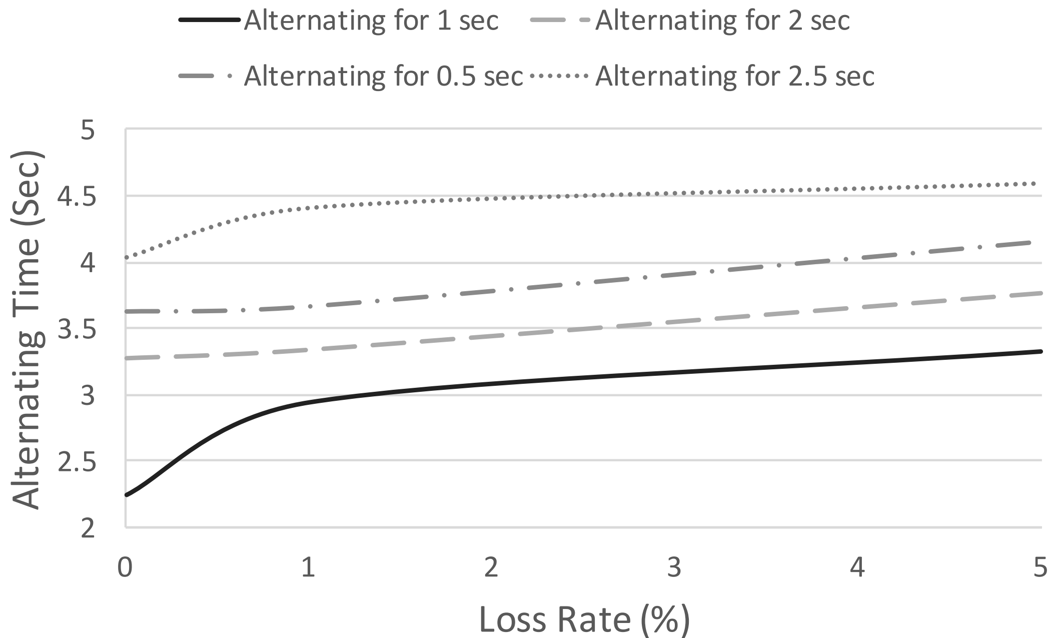


Figure 5.6: Push Time with no SEEP (sec)

after they have been sent. Each producer sent a total of 50 notifications every 1-3 seconds before we stopped the experiment. Figure 5.7 shows the size (in Bytes) of the name of the Interest and Data packets sent and received by the intermediate process.

As shown in the figure, in our the tested setup of 5 simultaneous producers, the Interest name size was around 400 Bytes, while the Data name size was around 800 Bytes. Data names are approximate twice the size of Interest names because they carry both the consumer and producer states. Interestingly, Figure 5.7 also shows how the name size drops when expired timestamps are removed, and how it grows back up as new notifications are pushed. The goal of this experiment was to verify that this size is bounded, and explore how expired notifications impact the name size. Future work should evaluate the name size as a function of different update rates for a larger amount of producers, and explore name enhancements such as name compressions.

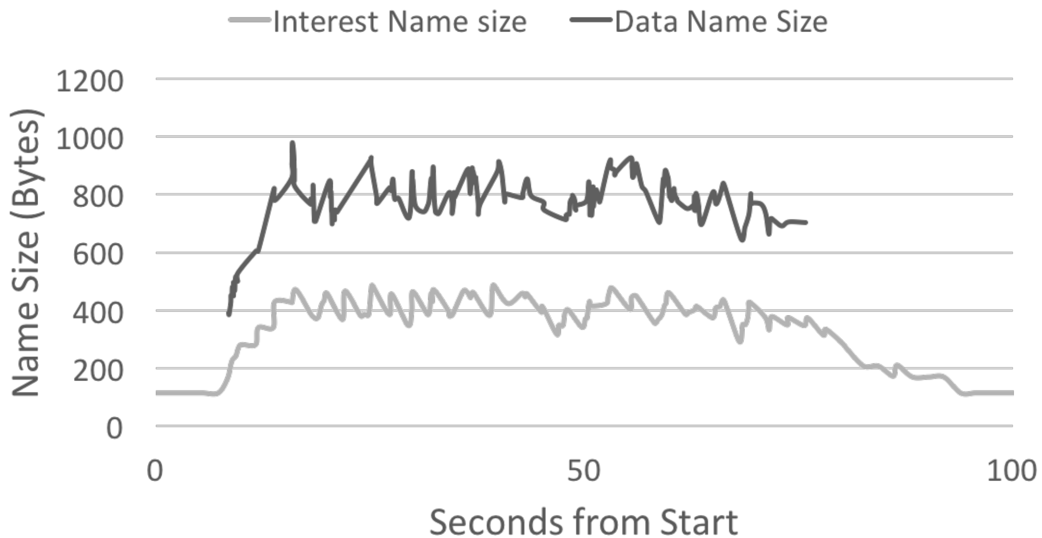


Figure 5.7: Name Size (Bytes)

Simultaneous Notifications

Last, we evaluated the latency of simultaneous push notifications in a reliable network, without an intermediate ICT component. In this set of experiments, we increased the number of producers that sent simultaneous notifications, and configured each producer to push between 80-100 notifications every 1 second. We measured the time elapsed from the moment a producer pushed a notification until it arrived at the consumer. We used node 'a' as the consumer, and nodes 'b'-'f' as the producers.

Figure 5.8 shows the latency of simultaneous notifications over a different number of producers. The results suggest a linear trend between the number of simultaneous updates and push latency. To understand the linear trend, consider that a notification can be lost if another notification consumes the PIT entries on R2, R3, or R4. In such cases, ICT-Notify resends the lost notification as a response to the next long-lived Interest, but this second

update can also lose the race to the router if another one was faster. Therefore, the push latency grows with the number of simultaneous producers.

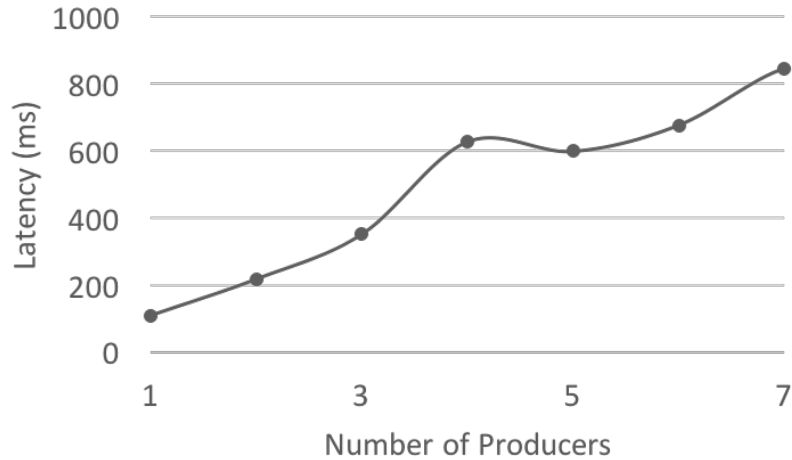


Figure 5.8: Push Time (ms) of Simultaneous Notifications

5.3 Conclusions

This chapter discussed the problem of push notifications ICN. We explored the challenges in supporting push mechanisms in the pull-based paradigm, and we illustrated a mechanism that can push Data packets based on two application abstractions: a time-oriented push abstraction and a producer-oriented push abstraction. We evaluated the basic push functionality of ICT-Notify in a simple topology with intermittent links. Our experiments show that with the time-oriented intermediate ICT-Notify component, applications were able to push notifications to others when the network links present small loss rates, and when there was never a SEEP between the members of the group.

To summarize the contributions of this chapter, we designed and implemented ICT-Notify to support applications in two aspects: 1) It proposes a push-based mechanism as a primitive

for ICN applications, without relying on any application specifics. 2) Like ICT-Sync, it decouples information from connectivity and supports applications in different connectivities, even if there is no SEEP. Although ICT-Notify illustrates a push-based mechanism that can form a future ICT, and therefore achieves the goals of this chapter, it is important to note that it was not evaluated for robustness and efficiency, and future research should explore enhancements to the mechanism.

Table 5.1 summarizes the properties of ICT-Notify as an ICT abstraction for push notifications mechanisms.

ICT property	**ICT-Notify**
Broadly applicable needs	Push the latest Data
Application Examples	Location-based applications Sensor network applications AR/VR applications Status applications
Known challenges	Pushing Data in the pull based paradigm Naming pending long-lived Interests Supporting simultaneous notifications in the same group
ICT API	Register: App registers to a notification group Push: push an event to others NotificationUpdate: ICT-Notify pushes a remote events to the application
Intermediate ICT	Acts as a consumer Remembers the latest relevant notifications Aggregates simultaneous Data packets and lost Notifications

Table 5.1: Push Properties as an ICT

Chapter 6

Data Partitioning

Like Chapters 4 and 5, this chapter also discusses a distributed group communication, but unlike the previous chapters, this chapter presents an ICN problem that has not been considered by others in related work. While data synchronization and push notifications have been discussed as essential ICN mechanisms in related works, and therefore were native candidates for ICT abstractions, mechanisms for fetching distributed partitioned data have not been explored. In this chapter, we not only argue that the task of fetching partitioned data is required by different types of ICN applications, but we also argue that this task is highly coupled with connectivity mechanisms, and therefore, requires an ICT abstraction.

We start the discussion with a definition of data partitioning in ICNs:

- An application with a dataset D, consists of N content items c_1,...,c_N with corresponding names p/n_1,...,p/n_N. All names share the same prefix p.
- D is partitioned into K subsets, and the subsets are distributed among different parties in the network.

- The names in each subset cannot be generalized and hierarchically represented in one partition name.

Figure 6.1 illustrates an example, in which N=8, K=3, and nodes A,B and C are the three distributed parties.

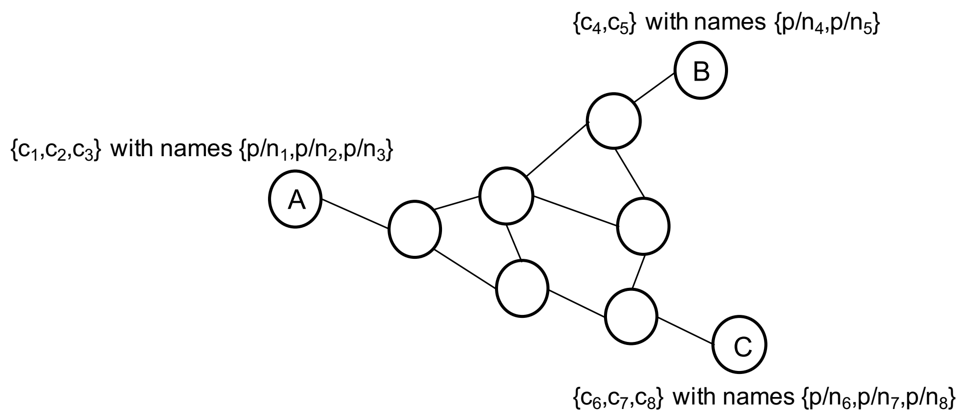


Figure 6.1: Distributed Producers with Partitioned Data

In theory, according to the request for named data abstraction, a consumer in a reliable network, with no intermittent or congested links, should be able to fetch any named data in the distributed dataset. However, in practice as we will show in the next subsection, a consumer must do more than simply request named data, and can only successfully fetch distributed partitioned data when specific connectivity mechanisms are present.

The following subsections of this chapter are organized as follows: We first demonstrate the problem of data partitioning in NDN by providing a simple application with partitioned data, and by showing empirically how NDN fails to satisfy the application requests unless there is a specific coupling of the application with forwarding strategies. Second, we discuss how this problem impacts different types of applications, and therefore, should be addressed by an ICT

abstraction. Third, we discuss existing and new mechanisms that can address the problem, and explore under which circumstances they can provide the desired abstraction. Fourth, we evaluate a simple abstraction that allows applications to control in-network information-oriented retransmissions, and show how it solves the problem of fetching partitioned data.

6.1 The Problem of Partitioned Data

In short, we argue that the problem of data partitioning arises when an application data is partitioned into disjoint subsets, and the data names in a subset cannot be generalized and hierarchically represented in a prefix. In such cases, ICNs cannot guarantee that an Interest for a named data would be satisfied, although the content exists and is reachable. Before we get to explain why, we prove our argument by emulating a distributed database application and show how NDN fails to retrieve the partitioned named data.

In our experiment, we created a dataset with 200 content items named p/n_i, in which p is the shared prefix and n_i is the name component that differentiates one name from the other. We partitioned the dataset into four equal and disjointed subsets S1-S4, and distributed them among four producers P1-P4. Therefore, each producer held 25% of the dataset. We emulated the distributed database application using ndn-traffic, and ran it on the ONL topology presented in Figure 6.2. We programmed a consumer (C) to randomly fetch all 200 content items, and collected all the Data packets it received.

We ran the experiment three times, and we found that in all repetitions, the consumer received exactly 25% of the requested named data, and only the data held by P2. When we checked the network logs, we found that all the requests arrived at P2, including the ones for data held by P1, P3 and P4. Therefore, P1, P3 and P4 were not able to respond to requests

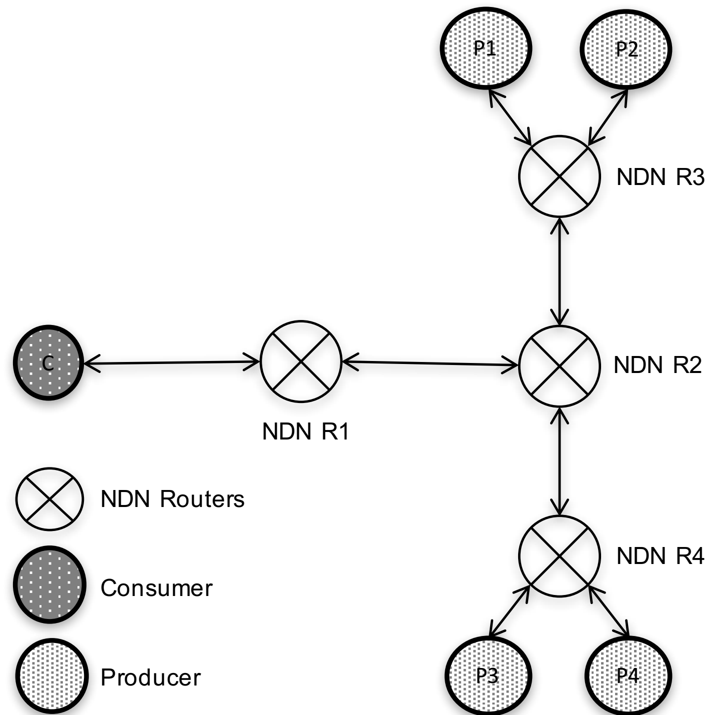


Figure 6.2: Distributed Database Use Case

for named data held by them. Hence, the problem lies in NDN's forwarding and/or routing mechanisms.

6.1.1 Where is the Failure?

To understand why NDN failed to satisfy 75% of the requests for named data, although the network was reliable and the links were configured to have routes between the consumer and all the producers, we must look into NDN's routing and forwarding planes. To announce existing named data, and to make it reachable to others, a producer in ICNs must publish the names of the data it holds. Then, a routing protocol is responsible to propagate those name in the network and to populate the FIB routes. Presently, the NDN architecture does not specify how an application should publish its names, and whether a producer should publish the full list of names it holds, or only prefixes that represent clusters of its data. However,

Prefix	Face Towards
/P/	P1, P2, R2

R3 FIB

Prefix	Face Towards
/P/	R3, R4

R2 FIB

Prefix	Face Towards
/P/	P3, P4, R2

R4 FIB

Figure 6.3: FIB Tables of R2, R3 and R4

for scalability, it was determined that NDN routing schemes should compute hierarchically structured routes based on name prefixes, and propagate those prefixes and not the full names [37]. In other words, NDN routing schemes aggregate similar names into shared prefixes, and populate the FIBs with the prefixes instead of full names.

In our tested application, the names in each partition could not intuitively be generalized into one hierarchical partition name. Therefore, similarly to the routes computed by routing schemes, the FIB tables in our tested network were configured to have the dataset prefix /P towards all the producers as shown in Figure 6.3. As discussed in Section 2.1.1, the NDN router looks for the FIB entry which has the longest prefix matching with the incoming Interest's name to determine the Interest's next hop. Here, all the requests for names in the dataset, regardless of the specific partition they belong to, matched the same FIB entry, and this FIB entry contained more than one possible next hop.

As also discussed in Section 2.1.1, when a FIB entry contains more than one next hop, the forwarding strategy determines which one(s) to use. Our experiments used the default NDN configuration which pairs the best-route strategy with all routable names. As discussed in Section 2.4, this strategy chooses the least expensive face when it forwards an Interest, and it does not change its selection unless the face cost has changed by a routing protocol or by the network operator. Here, all faces had the same cost, and the strategy chose the face with the lowest id to break the tie. This behavior caused R2 to choose R3 as the next hop for all

the Interests it processed, and caused R3 to choose P2 for all the Interest it processed. As a result, only the requests for named data in S2 were satisfied.

Strategy Choice

To better understand the impact of the strategy selection on the correctness of our distributed DB application, we changed the default configuration and paired our application prefix, P, with different NDN strategies. Table 6.1 presents the percentage of satisfied requests over different network strategies.

Strategy	% of Satisfied Requests
best-route	25
ncc	100
asf	27.5
multicast	100

Table 6.1: Percentage of Satisfied Requests over Different NDN Strategies

As can be seen in the table, NDN satisfied 100% of the requests only when two strategies were configured in our tested topology: The multicast and ncc strategies. The multicast strategy sent an Interest to all available next-hops found in the FIB entry, and therefore, each producer received all the requests, and responded to the requests for named data in its partition.

To understand why ncc successfully retrieved 100% of the requests, and why the asf strategy satisfied only 27.5%, we need to extend our discussion from Section 2.2.1, and discuss the details of the strategy's failure response mechanisms. When an Interest is not satisfied with a Data packet, the forwarding strategy can retransmit the Interest again on a different face,

or it can drop the Interest. Clearly, the best-route strategy drops the Interest, and does not try another face.

Like the best-route strategy, the ncc strategy also chooses one upstream face when the FIB entry contains multiple next hops. However, unlike the best-route strategy, the ncc strategy retransmits the Interest on a different face if a Data packet was not received within a specific amount of time. Therefore, the strategy guarantees that if there is a route to a producer that can satisfy the Interest, this producer would eventually receive the Interest and satisfy the consumer request.

Interestingly, the asf strategy satisfied 27.5% of the requests, 96.5% of them were satisfied by P2, and 3.5% were satisfied by P4. To see why, consider that like the ncc and the best-route strategies, the asf strategy sends an Interest to only one upstream face, and like the best-route strategy, asf does not retransmit an Interest if a Data packet was not received on time. However, after a few consecutive failures, the asf strategy changes its face selection, and forwards Interests for the same prefix on other faces in the FIB entry. The exact number of consecutive failures in which after the strategy changes its face selection depends on configuration parameters.

To conclude, our simple distributed database experiment demonstrated that in order to satisfy requests for named data, when the data is partitioned and the names in each partition cannot be represented hierarchically in an explicit partition name, an application must be coupled with a forwarding strategy that either multicasts Interests to all potential next-hops, or retransmits an Interests on different faces when a Data packet is not received on time. However, in Chapters 2 and 3, we explained why pairing applications with forwarding strategies break the ICN promise, and couples applications with connectivity mechanisms.

Therefore, choosing the 'right' forwarding strategy to solve the problem of data partitioning is not a sustainable solution.

6.2 ICT for Fetching Distributed Partitioned Data

We propose to identify data partitioning as a problem that requires an ICT abstraction. As discussed in Chapter 3, the goal of an ICT abstraction is to keep applications in the information plane while the ICT implements broadly applicable mechanisms to satisfy the requests for named data. In this section, we explore whether an abstraction for distributed partitioned data is required by different types of applications, and therefore, justifies an ICT abstraction.

Presently, mechanisms for fetching named data when the data is partitioned have not been recognized as essential mechanisms in ICNs. This is mainly because the problem could be solved by using topological names, in which the data name contains a location-oriented prefix. For instance, data items held by P1 producer would have /P1/ prefix while data items held by P2 would have /P2/ prefix. Although we agree that this is a valid mechanism in Internet-like infrastructures, such as the NDN testbed, we also argue that it is not a general-purpose solution for ICNs because it couples application with topological names.

In host-centric networks, the problem of efficiently locating the node that stores a particular data item when the data is partitioned was acknowledged as a fundamental problem that confronts peer-to-peer applications [65], and was addressed by decentralized overlay frameworks such as distributed hash table (DHT) [65] and Virtual cord protocols (VCP) [12].

In addition to traditional decentralized storage services, similar to the distributed database application we emulated in the previous section, our work on the NDN testbed encountered

two additional use cases that required a mechanism for fetching partitioned data. The first is a video streaming application [36], in which the producer had to respond to a large number of requests for video frames, in addition to requests for the key certificates. Due to long computation time of each Data packet, and in order to meet latency requirements, the application developers asked for a mechanism that stores every produced Data packet in a local or a remote repository, and then have that repository act like a 'cache' and respond to requests on the producer behalf. Hence, the producer would only work on generating Data for new requests, while the repository would satisfy previously generated Data packets. As a result, the application dataset was partitioned into two: previously generated data held by a repo, and newly generated data held by the application.

Here, the topological naming convention on the NDN testbed prevented the option to use a remote repository, because the application namespace was coupled with the testbed gateway. Moreover, similarly to the emulated distributed database application, the default testbed strategy, best-route, forwarded all the Interests to either the producer or the repository. Here, changing the strategy to multicast or ncc did not solve the problem because these strategies could not differentiate old requests from new requests, and forwarded a mix of old and new Interests to both the application and the local repository.

The second service with a requirement to fetch partitioned data we encountered on the NDN testbed was nTorrent [49], a BitTorrent-like [68] implementation for NDN. The naming convention of nTorrent identified the name of the torrent, the communication group, the files in the torrent, and the requested file chunk. A participant in a torrent, a peer, dynamically adds itself to a torrent by announcing the torrent prefix. However, announcing a prefix does not mean that the peer has all the torrent's data (e.g., the prefix of a file in the torrent does not mean the peer has all the file's chunk). Therefore, a mechanism for path exploration per name-based request is required. When deployed on the NDN testbed, the nTorrent prefix

was paired with the ncc strategy for successful data fetching. Alternatively, the nTorrent paper [49] suggests using name mapping services, such as SNAMP [3], to translate torrent names into topological names.

To conclude this section, we argue that the requirement to fetch distributed partitioned data is not an application-specific requirement. Instead, it is a requirement shared by different services with the goal to aggregate storage, computation or network resources, by using a number of distributed resources. Presently, those services must be coupled with either connectivity mechanisms, such as forwarding strategies and routing prefix aggregation, or with topological names. In other words, the Information plane is coupled with the connectivity plane. This coupling not only impacts the application complexity, but it also affects the performance and correctness of the application. To decouple the two planes and to simplify the process for fetching distributed partitioned data, we must use an ICT.

6.2.1 Considerations for ICT Mechanisms

In this subsection, we discuss mechanisms for fetching partitioned data, and we explore if and how they can form ICT abstractions.

NDN Map-and-Encap for Partitioned Data

The first mechanism we discuss is mapping data names into explicit routable names by an external service such as NDNS (DNS for NDN) [1]. For instance, if a data name p/n_i resides at Alice's laptop, and Alice is connected to WUSTL network, then every time a consumer sends a request for p/n_i it would be mapped into $/WUSTL/AliceLaptop/p/n_i$ by a service like NDNS. Then, the application can send Interest using the mapped name, and not the application-specific data name. The key property of this approach is to eliminate the FIB

ambiguity by mapping every data name to an explicit routable name. The mapped NDNs name can be topological, or arbitrary, depending on the network configuration.

The NDN Secure Namespace Mapping (SNAMP) [3] takes the Map-and-Encap approach, and adds a layer of indirection to NDN when names whose reachability do not follow topological hierarchy can be reached using other globally routed names. Hence, when the routing protocol does not propagate the data names published by the application.

Using SNAMP, the network responds with 'no route' NACK when an Interest for unreachable data name is expressed. When the NACK arrives at the consumer, SNAMP sends an Interest to NDNS to map the unreachable data name into a routable name. If found, NDNS responds with a routable name(s), and this routable name is added as a link object to the Interest. Then, SNAMP uses this link object to send the Interest to the specific data provider. The process of mapping unreachable application names into routable names is done by SNAMP, and is transparent to applications.

While the approach of mapping application-level names into explicit routable names can be considered as an ICT mechanism to fetch partitioned data, SNAMP does not present a complete solution. First, the goal of SNAMP is to retrieve unreachable named data, and not to retrieve partitioned data. Therefore, SNAMP relies on network NACKs to trigger the mapping operation. In contrast, the names of partitioned data are reachable, and can be found in ambiguous FIB entries. Hence, an ICT abstraction for fetching partitioned data cannot rely on network NACKs.

A potential ICT abstraction for fetching distributed partitioned data can follow SNAMP approach, with the following modifications:

- On the producer side, ICT-API maps every name in the local partition into an explicit routable name representing a route to the producer.
- On the producer side, ICT-API adds this mapping into NDNS, and publishes only the routable producer name.
- On the consumer side, When a consumer fetches named data, the consumer's ICT-API first fetches the routable name from NDNS, and then expresses an Interest for the data using the routable producer name and not the application data name.

While this ICT approach simplifies applications and can solve the problem of fetching partitioned data, it presents a few limitations. First, relying on a centralized server such as NDNS can result in failures when the server is down, or when connectivity prevents continuous access to the server. Therefore, it does not fully decouple information and connectivity. Second, mapping every request for an application name into a routable name using an external service results in additional delay, because it requires an additional RTT to translate an application-level name into an explicit routable name.

Decentralized Mapping Using Sync

To address NDNS limitations, we propose another possible ICT solution that follows a similar mapping approach, but does not require centralized services like NDNS. This ICT uses invertible Bloom Filters (IBFs) to encode data names into one partition name, and then requests named data by expressing an Interest carrying the IBF representation of the partition name. The following list discusses the high-level principles of our proposed ICT:

- ICT-API on the producer side encodes every name in the local partition into an IBF, and publishes the binary representation of the IBF as an explicit partition name. For instance: /ICT-DistributedFetch/<ApplicationName>/<PartitionIBF>.

- Routing protocols propagate the explicit partition names published by the ICT-API without aggregating them into a single prefix. As discussed in Chapter 3, while an application cannot express any routing requirements, such as name aggregation, an ICT is allowed to specify routing requirements because an ICT name represents broadly applicable needs.

- A sync mechanism, such as ICT-Sync, is used to synchronize all the partition names among the distributed producers.

- On the consumer side, a request to fetch named data causes ICT API to search the corresponding partition name in the synchronized list of IBFs. Then, the ICT API expresses an Interest for the named data by expressing an Interest for the data in a partition:
  /ICT-DistributedFetch/<ApplicationName>/<PartitionIBF>/<RequestedDataName>

- Thanks to the explicit partition name in the Interest, the Interest is forwarded in the network directly to the producer that holds the requested named data.

- On the producer side, when a producer receives an Interest for named data, the ICT library uses the <RequestedDataName> component to request the named data from the application. Then it responds to the Interest with the content retrieved from the application in a corresponding Data packet.

The advantage of this approach is that such an ICT abstraction adds a layer of indirection without modifying the application, without relying on a centralized server, and without

adding delays to translate application names into routable names. However, the synchronization of partition names is likely to add additional network overhead.

Theoretically, the proposed distributed ICT can implement an application abstraction for fetching distributed partitioned data. We think that an intermediate ICT component for fetching partitioned named data can synchronize partition names, store retrieved Data packets for future requests, and retransmit Interest packets when it is not satisfied by the network. This way, an intermediate ICT component can solve the challenge of false positive IBF decoding, and retransmits Interests towards additional data partitions. However, this assumed behavior should be verified and evaluated by an implementation.

Controlling Strategy Retransmissions

The third mechanism we consider is a simple application abstraction, that controls in-network retransmissions when FIB ambiguity implies that the data might be partitioned, and when an Interest packet is not satisfied by the originally selected face. We elaborate on this abstraction in Section 6.3.

6.3 Application Abstraction: Controlling Strategy Retransmissions

As shown in section 6.1.1, the problem of fetching distributed partitioned named data can be addressed by strategy retransmissions. However, coupling applications to a specific forwarding strategy is not a sustainable solution because it couples applications with network connectivity. Therefore, we present a simple abstraction that moves the decision to perform information-oriented in-network retransmission to the application, without the need to be

paired with a specific forwarding strategy. This way, the decision whether to retransmit in the network is decoupled from a variable strategy implementation, and made only by the application. Although we do not consider this abstraction as an ICT, it demonstrates how applications with partitioned data can be decoupled from connectivity mechanisms while controlling general-purpose in-network operations.

Our proposed abstraction for retransmission mechanism performs two independent yet complementary functions: retransmissions decoupling and retransmission differentiation.

6.3.1 Retransmission Decoupling

This abstraction adds a new TLV to the Interest packet to specify the application retransmission requirement. We name this Boolean type field the *'Interest Retransmission Policy'* (IRP) flag.

By setting the IRP to True or False, the application dictates whether the strategy should or should not retransmit an Interest as part of its failure response mechanism. A forwarding strategy can then support each option by providing the two failure response mechanisms as part of its implementation, one that performs in-network retransmissions and another that does not.

Algorithm 3 presents a simplified framework for a forwarding strategy that supports both retransmission mechanisms using the IRP flag.

The IRP flag does not determine the in-network retransmission algorithm, and not the strategy that should be used. The IRP flag only requires that a retransmission mechanism exists. Thus, the application decides whether an Interest should be retransmitted by the network, while the strategy determines the in-network retransmission algorithm, that is,

Algorithm 3 Strategy framework that decouples in-network retransmissions

```
Function ForwardInterst(interest):
    face_list = SelectNextHop(interest)
    IRP = GetIRP(interest)
    SendInterest(interest, face_list)
    if IRP then
        schedule retransmission at time x
    else
        wait for application retransmission
    end
    return
```

when to retransmit and which next hop(s) to choose. The retransmission and suppression timers presented in algorithm 3 are only placeholders for possible retransmission algorithms provided by a forwarding strategy. The strategy is free to choose any algorithm to support the two options. For example, a core network strategy might choose a retransmission algorithm that addresses congestion issues and relies on collecting round-trip-times, while an access strategy retransmission algorithm might simply follow a list of given faces and retransmit an Interest after a fixed time interval. The work in [57] proposes a statistical model to compute retransmission intervals.

6.3.2 Retransmission Differentiation

Our proposed strategy abstraction adds a second Interest TLV, the *'Network Retransmission Differentiation' (NRD)*, to differentiate application Interests from network retransmissions. Using the NRD TLV, strategies can support different mechanisms for controlling network traffic, and can collect performance measurements of alternative next hops in dynamic environments. We describe two scenarios in which the NRD field is required.

First, in dynamic networks, such as in a vehicular network [33] or in wireless networks[63], an adaptive forwarding strategy can probe faces to explore alternative next-hops. Such a

strategy might want to differentiate the probed Interests from the Interests generated by the application to support designated strategy mechanisms for control and data traffic.

The second scenario is an existing problem in the current ncc strategy and the NFD forwarding mechanism, in which loop detection caused by nonces can prevent better face exploration. As explained in Chapter 2, the ncc strategy adjusts its retransmission timer up whenever the best face upstream times out, and adjusts it back down whenever the face upstream successfully retrieves data. Adjusting the timer down for every successful data retrieval guarantees that at some point, the time period is less than the upstream RTT, and therefore allows ncc to explore other potential upstreams. However, because of the duplicated nonce mechanisms, ncc can fail to explore potentially better-performing upstreams. We illustrate this problem in Figure 6.4.

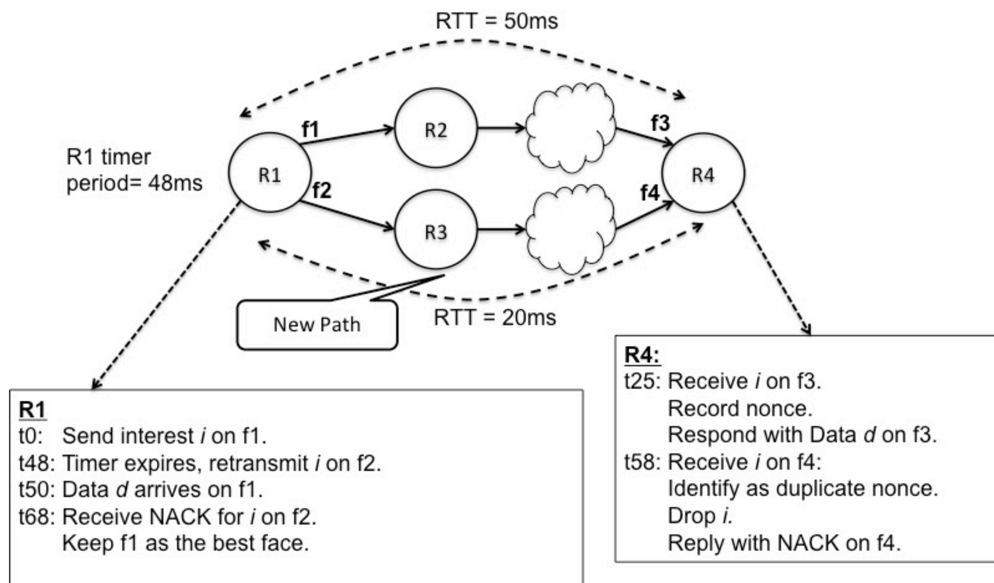


Figure 6.4: NACK problem

As presented in Figure 6.4, R1 has a new faster path to R4 through R3, but the ncc strategy has previously selected R1 as the (only) best performing upstream face, thus it sends all

Interests on face 1. When the R1 timer approaches 48ms, which is smaller than the actual 50 ms RTT through R2, the timer of Interest *i*, sent at t0, causes R1 to retransmit the Interest on face 2 at t48. Router R4, which received Interest *i* from R2 at t25, receives it again at t58. R4 recognizes the second *i* and its nonce as a duplicate Interest, and therefore drops it and replies with NACK. Here, ncc on node R1 does not receive a Data packet on face 2, and therefore continues to use face 1 as the best-performing face, although its upstream RTT is more than twice that of the other new path. The ncc strategy will switch to use face 2 only if a retransmitted Interest arrives at R4 before the original Interest. In other words, the strategy changes its best-face selection only if the "timer period" plus the "one way trip time through R3" is less than the "one way trip time through R2".

This problem could be solved by adding the NRD TLV to the retransmitted Interest, and differentiating the retransmitted Interest from the original one. This way, the strategy does not detect the Interest as a duplicate one, thus enabling better face exploration. However, by adding NRD TLV and processing Interests with the same nonce, we interrupt the core mechanism of loop detection in NDN. Therefore, using the NRD as a simple Boolean flag does not solve the problem.

In our implementation, we used a non-negative-integer to represent the NRD TLV. We set the initial value of the NRD TLV to 0, and increased it by one every time the Interest was retransmitted by the strategy to an additional face. In our experiments, we selected 10 as the maximum number of allowed retransmissions, and replied with NACK if the Interest's nonce was previously recorded and the NRD TLV was equal to 10. In addition, we implemented the strategy mechanism to reply with NACK when there were no unused upstream faces to use. Although our implementation provided us with the desired behavior, the NRD mechanism should be better explored as part of future work. We present a framework of our implementation in algorithm 4.

Algorithm 4 Retransmission Differentiation using NRD

```
Function DetectLoopAndRetransmissions(interest):
    if nonce previously recorded then
        if NRD == 'MaxAllowed' then
            send NACK
        else
            interest.NRD++ HandleRetransmission(interest)
        end
    else
        interest.NRD++
        ForwardInterst(interest) [algorithm 3]
    end
    return
```

Unlike NDN, CCN does not use nonces to detect loops, but uses an additional Time-To-Live(TTL) TLV to avoid infinite loops. While the problem described in Figure 6.4 might not occur in CCN, we suggest that the proposed differentiation can be useful in the CCN architecture to support more intelligent forwarding strategies that can differentiate an application Interest from an Interest injected by the network.

6.3.3 Empirical Results

We implemented the proposed retransmission mechanism in NFD 0.4 and added the two suggested TLVs to the Interest packet. We modified the loop-detection mechanism to follow algorithm 4, and tested the proposed in-network retransmission abstraction by running a set of experiments using the emulated NDN testbed [43] in the Open Network Lab (ONL) [76].

The emulated environment consisted of 26 dual-core machines that represent the testbed gateways, 26 virtual machines(VM) that represent end hosts, and four software routes. All these machines run Ubuntu 12.04.5 and our modified version of NFD 0.4. We configured each gateway to publish the same set of namespaces used by the corresponding world-wide NDN testbed [53] gateway, and ran NLSR 0.2.2 as the network routing protocol to distribute

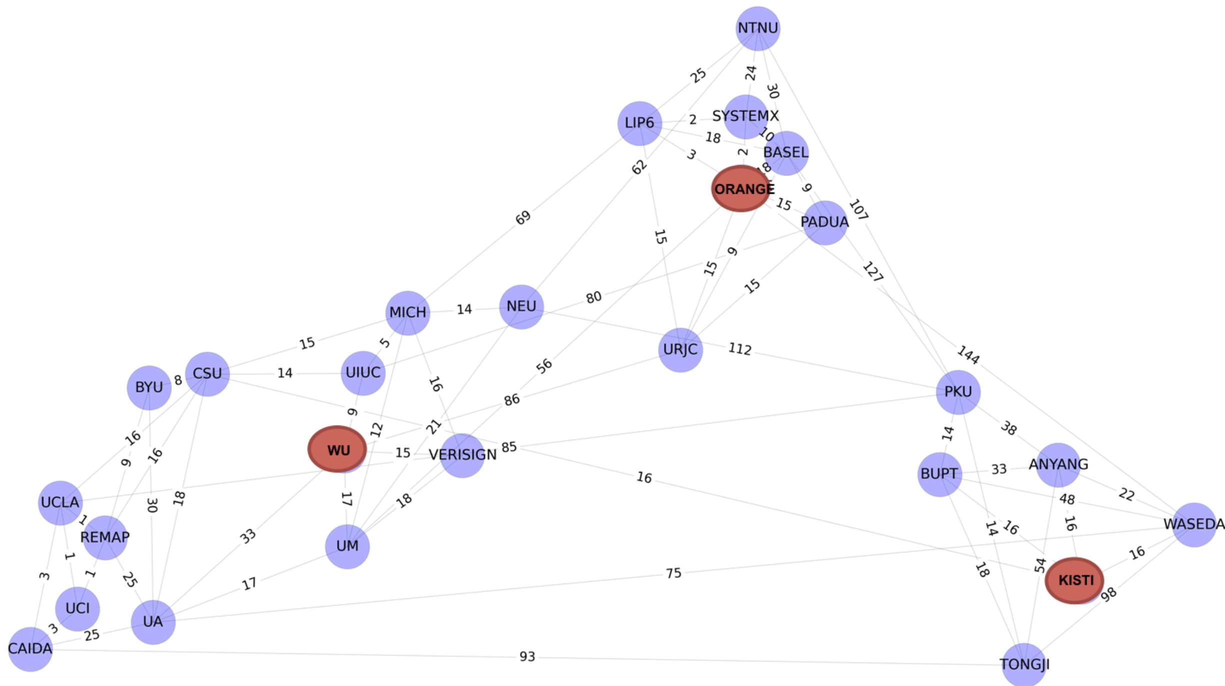


Figure 6.5: Emulated NDN testbed

the gateways' namespaces. The emulated testbed also had 66 links that were configured with costs that match world-wide NDN testbed costs.

We connected one VM to each of the gateway machines to emulate one end host connected to each gateway. Figure 6.5 presents our emulated gateways' topology. To simplify the presented topology, Figure 6.5 does not present the end-hosts connected to each gateway.

We modified the best-route and ncc strategies so they checked the IRP flag in order to determine if in-network retransmission was required by the application, and used NRD TLV to differentiate in-network retransmissions from application Interests. We used algorithm 4 to prevent infinite loops of retransmitted Interests. We named the modified best-route and ncc strategies the 'best-route-r' and 'ncc-r' strategies.

Three experiments in the scale of the NDN testbed demonstrated the impact of in-network retransmissions on the correctness of fetching partitioned data, and provided the cost of retransmissions in each of the evaluated scenarios.

Partitioned Data among Distributed Producers

In this experiment, we used modified versions of ndn-traffic and ndn-traffic-server to generate Interests and Data packets. We ran ndn-traffic consumer on the VM connected to the WU gateway, and two instances of ndn-traffic-server as multiple producers on the VMs connected to the ORAMGE and KISTI gateways. These three gateways are colored in red in Figure 6.5. We configured both servers to respond to the Interests for the same name, while ORANGE VM is the originator of the data, and KISTI serves as the data's repository by fetching all newly generated data to be stored in its local repo. This way, ORANGE can focus on generating new data and KISTI can satisfy future requests. To emulate a use case in which ORANGE is busy generating new data and cannot satisfy requests for named data, we configured the ORANGE producer to halt for 10 seconds during the run of the experiment. We set the consumer's Interests IRP flag to True, and thereby required in-network retransmission from the strategy. We did not provide any retransmission mechanism for unsatisfied Interests in the application scope. The total traffic sent over the network consisted of the traffic generated by our producer as well as the traffic generated by NLSR.

The details of the experiments can be summarized as follows: At the beginning of the experiment, we configured the consumer to start expressing Interest packets at the rate of 50 Interests per second, and the producers to respond with Data packets for each received Interest. We stopped the producer on ORANGE VM 10 seconds after the start point, and brought it back up again 10 seconds later for an additional five seconds.

Strategy	Unsatisfied Interest Rate(%)	Total Interest Sent by WU Gateway	Std Sample
best-route	42.55	1700	0.09
best-route-r	0.621	3563	0.00048
ncc	0.95	5322	0.044
ncc-r	0.93	5490	0.00073

Table 6.2: Multiple Producers Results Summary

We repeated the experiment five times with each of the following strategies: best-route, best-route-r, ncc, and ncc-r. We collected the total number of Interests sent by the consumer, the total number of Data packets received from each producer, and the number of Interests sent by WU gateway. The average results are presented in Table 6.2.

As shown in Table 6.2, when using best-route as the strategy paired with the application's namespace, an average of 42% of the expressed Interests remain unsatisfied. However, less than 1% of sent Interests remain unsatisfied when the application's namespace is configured with best-route-r, ncc, or ncc-r. In addition, Table 6.2 shows that the number of Interests sent by the WU gateway when using best-route-r was twice the number of Interests sent by WU when using best-route. This difference is explained by the specific implementation of best-route-r, in which the strategy retransmits an Interest after a fixed amount of time, which is shorter than the actual round-trip time in the used topology. This detail in the in-network retransmission mechanism should be better explored as part of future work. However, the experiment demonstrates that a simple change to the best-route strategy, supporting the IRP flag, can dramatically improve the unsatisfied Interest rate in the case of multiple producers with a congested node, and therefore supports a wider range of applications.

Our statistical analysis of the results did not indicate any statistical difference between ncc and ncc-r, Therefore, we can conclude that supporting the IRP flag does not change the performance and correctness of strategies that already support in-network retransmissions as part of their default implementation.

Partitioned Data among Distributed Producers with Intermittent Link

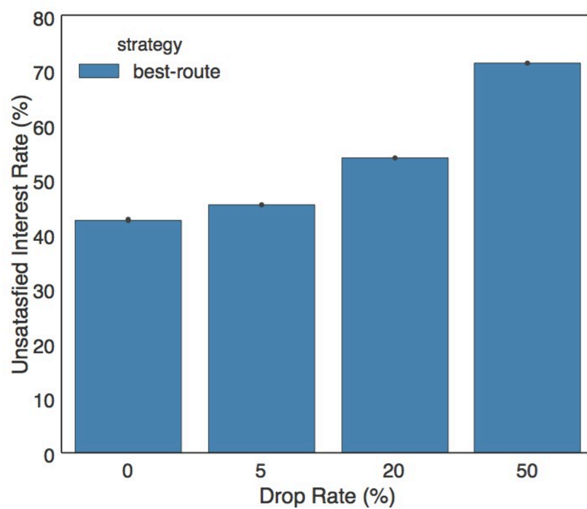
To evaluate this abstraction in different connectivities, we repeated the previous set of experiments and configured the links from WU gateway with a different drop rate each time. We used drop rates of 5%, 20%, and 50%. We present two figures due to the different scale of the results: Figure 6.6a presents the unsatisfied Interest rates of the best-route strategy and Figure 6.6b presents the unsatisfied Interest rates of the other strategies explored.

For the best-route strategy the rate of unsatisfied Interests reaches an average of 70% when the congested gateway drops 50% of the packets. However the unsatisfied Interests rate remains less than 1.5% when using best-route-r, ncc and ncc-r. This contrast again shows how simple support of in-network retransmission in the best-route strategy can improve the performance of a multiple producer application, even when one of the gateway nodes is congested and drops 50% of the packets.

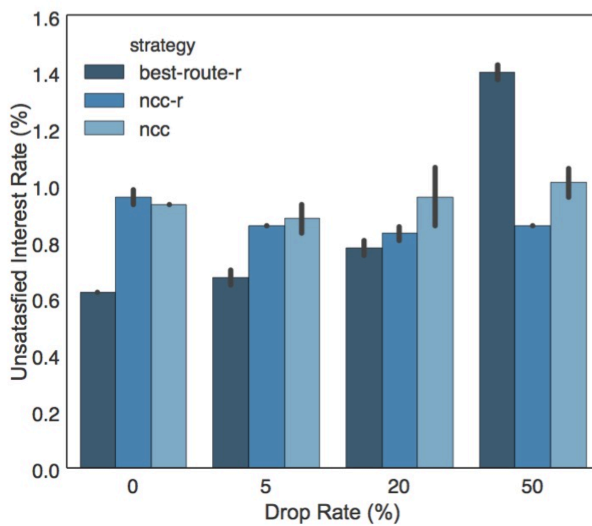
Abstraction Impact on Multipath Traffic with Failed Link

In this experiment, we used a simple consumer-producer service using ndn-traffic as the consumer running on the WU VM, and ndn-traffic-sever as the producer running on the KISTI VM. As before, we modified the consumer to set IRP to True, and did not support any application retransmission mechanism in the application's scope. To emulate a congested link, we set a drop rate of 100% on the link between WU and UIUC, which is the least expensive next-hop to reach the producer from the WU gateway. We collected RX and TX counters every 0.1 seconds on all participating links.

The details of the experiments can be summarized as follows: As before, we started the experiment by configuring the consumer to send 50 Interest packets per second. Ten seconds later, we configured the link between WU and UIUC to drop all application packets sent



(a)



(b)

Figure 6.6: Unsatisfied Interest Rates for Different Link Loss Rate: (a) best-route. (b) best-route-r, ncc-r, and ncc. Note the very different Y axis ranges

by the WU gateway. We recorded the traffic for 120 seconds before removing the dropping filter, and continued to record measurements for an additional 120 seconds before stopping the consumer's traffic. The total runtime of the experiment was 250 seconds.

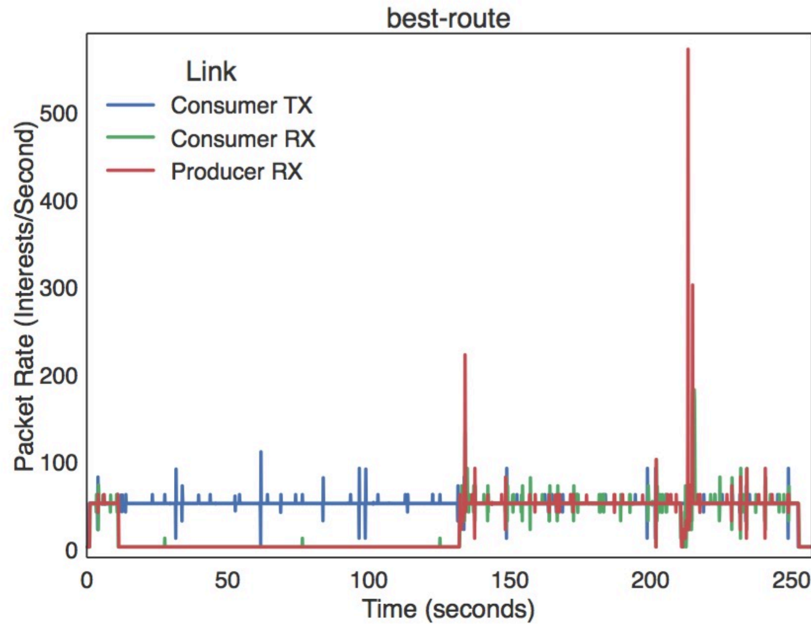


Figure 6.7: End Hosts Traffic over Time with best-route

Figure 6.7 shows the traffic recorded on the producer and consumer VMs when using the best-route strategy, and Figure 6.8 shows the traffic recorded using the best-route-r strategy. From these two figures we learn that all Interests sent during the dropping interval remained unsatisfied when using best-route, while the consumer-producer traffic remained unaffected when using best-route-r.

To better explore the strategy behavior, we recorded all the traffic transmitted on the WU gateway links to the following immediate hops: UIUC, UM, URJC, and VERISIGN. We show the results using the best-route strategy in Figure 6.9, and the results using the best-route-r strategy in Figure 6.10.

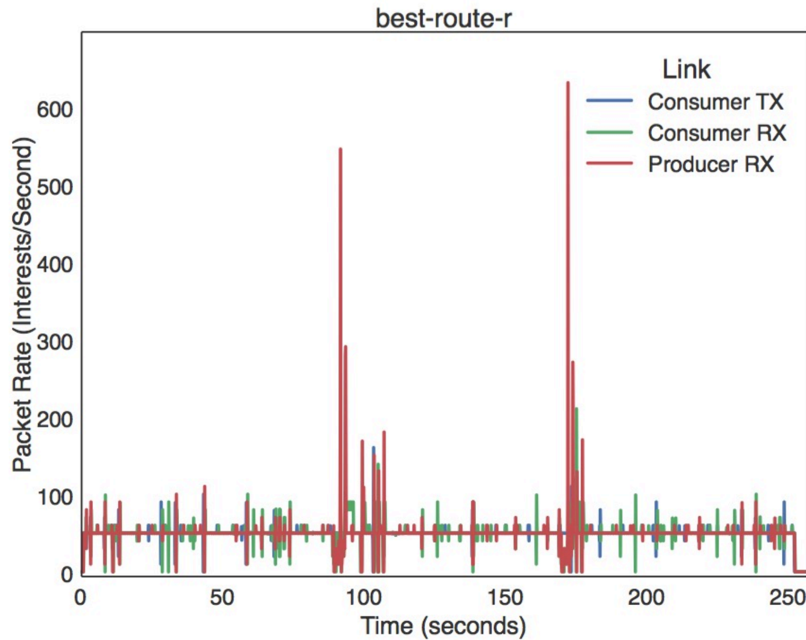


Figure 6.8: End Hosts Traffic over Time with best-route-r

As shown in Figure 6.9, due to the NLSR costs configured on the emulated testbed, UIUC was selected as the best next-hop by the best-route strategy. At t=10, when the link towards UIUC started dropping all the Interest packets transmitted by the consumer, the traffic on this link dropped to almost zero. Due to our overlay network setup on top of the ONL machines, the traffic reported in these figures contains NLSR traffic, and therefore the recorded TX counters on this link do not present an absolute zero. At t=135, NLSR determined the link to UM is the new least expensive nest-hop to the producer, and therefore the best-route strategy rerouted all the traffic to use this link. At t=210, 100 seconds after we stopped dropping UIUC packets, NLSR determined UIUC as the least expensive next hop again, and rerouted the traffic towards that link.

Figure 6.10 presents the measurements of the same WU links when using the best-route-r strategy. As before, the link to UIUC was first selected as the best next-hop towards the producer. At t=10, when the link towards UIUC started dropping the consumer packets, the

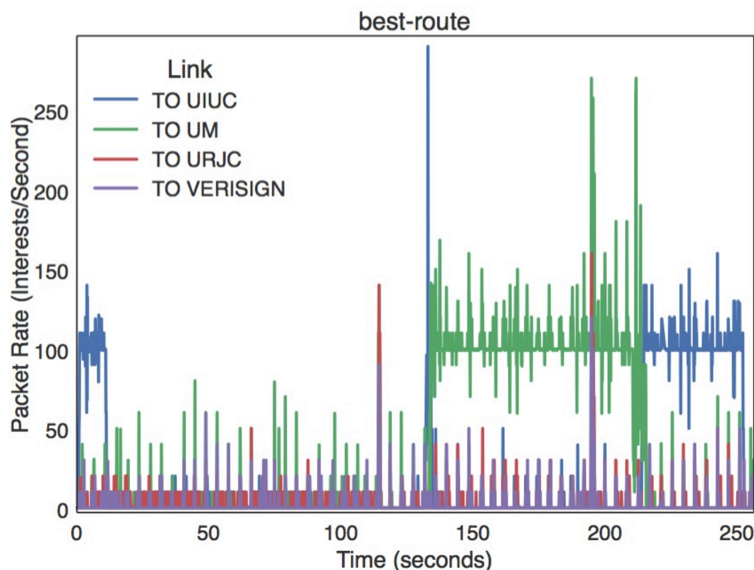


Figure 6.9: WU Traffic over Time with best-route

best-route-r strategy retransmitted all unsatisfied Interests towards UM, without waiting for NLSR to declare this face as the least expensive next-hop. The strategy continues to use UM link until UIUC becomes available again. This quick response was achieved because the application set the IRP flag to require in-network strategy retransmissions.

It is important to clarify that our intention in this experiment was not to compare forwarding recovery times to routing convergence times [79]. Instead we sought to demonstrate how a multipath consumer-producer service can maintain a continuous traffic flow even when the network is congested, and without forcing the application to implement a retransmission mechanism, as required by the existing best-route strategy. Moreover, as can be seen in Figure 6.10, the Interest rate sent by the best-route-r strategy on the WU gateway is on average twice the rate sent on WU using best-route. As in the previous experiments, this is a direct outcome of the fixed retransmission intervals we implemented in best-route-r, which is shorter than the actual round-trip time in the used topology, and will be better explored as part of future work.

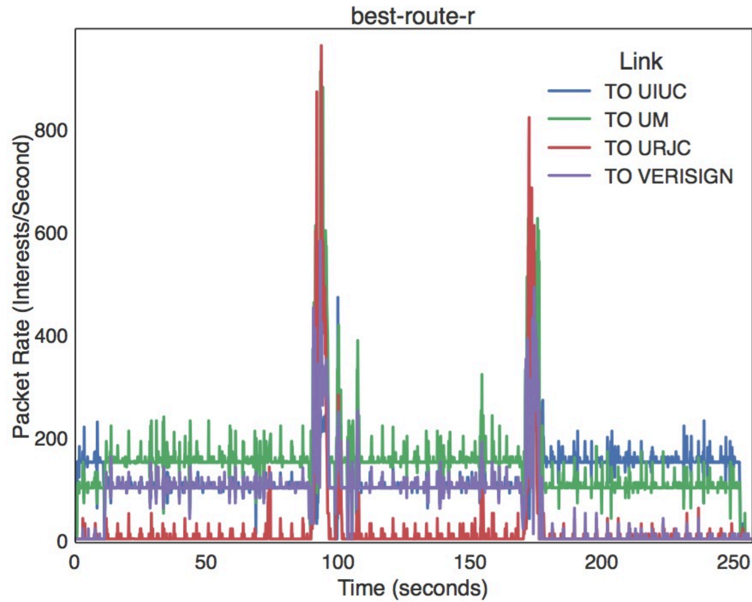


Figure 6.10: WU Traffic over Time with best-route-r

6.4 Conclusions

This chapter discussed the problem of fetching distributed partitioned data in ICN. We demonstrated how NDN fails to satisfy a request for named data, although the content exists and is reachable, and identified FIB ambiguity and forwarding mechanisms as the source of the problem. We provided examples to argue that the problem is not application-specific, and is shared by different types of applications asking to aggregate storage, network or computation resources. Therefore we proposed to address the problem using an ICT that decouples applications from the details of connectivity mechanisms, and simplifies the process of fetching distributed partitioned data.

To summarize the contributions of this chapter, we identified the problem of fetching distributed partitioned data, and we explored approaches to solve it by either using Map-and-Encap mechanisms, or by controlling in-network information-oriented mechanisms. We

implemented the second approach and evaluate an abstraction for controlling strategy retransmissions. We tested this abstraction on the emulated NDN testbed, and demonstrated how it allows applications to successfully fetch partitioned data regardless of the underlying forwarding strategy, and despite FIB ambiguity.

Table 6.3 suggests high-level properties of a future ICT abstraction to fetch distributed partitioned data in ICNs.

ICT property	**ICT for Fetching Distributed and Partitioned Content**
Broadly applicable needs	Application data is partitioned into disjoint subsets, and the content of a data subset cannot be generalized and hierarchically represented in a prefix
Application examples	Services with aggregated resources
Known challenges	Decoupling forwarding and routing mechanisms from applications, and FIB ambiguity
ICT API	Translate application needs by either: Mapping app-specific names into explicit routable names, Marking applications policy to control in-network retransmissions
Intermediate ICT	Decouple applications from connectivity by either: In-network mapping of data names and partition names Controlling information-oriented in-network retransmissions

Table 6.3: Fetching Partitioned Data Properties as an ICT

Chapter 7

Conclusions

In this dissertation, we explained how the network abstraction of ICNs — the request for named data — promises to decouple applications from the details of connectivity, and we identified that despite this promise, the information and connectivity planes are presently coupled through the forwarding strategy component. We showed how this coupling prevents applications from operating solely in the information plane, and we proposed to address this problem by adding a new architectural component to ICNs, named Information-Centric Transport (ICT).

We defined ICT to be an abstraction and a communication mechanism that consists of an API for application developers at the endpoints, and an intermediate network process for network operators. We defined the intermediate ICT process as a mechanism that implements broadly applicable needs, and we determined that it should not obtain any application-specific knowledge. To make the ICT abstraction practical and scalable, we defined high-level considerations of an ICT, and we argued that the number of ICTs implemented in ICNs must

be kept small. Moreover, we discussed how the concept of ICT does not preclude other applications from using traditional end-to-end transport mechanisms, but instead, it extends the concept of transport by allowing new in-network name-oriented mechanisms in ICNs.

To demonstrate the concept of ICT, we explored three general-purpose group communication mechanisms in ICN, and we discussed why these three implement broadly applicable needs, and therefore, can form ICT abstractions. The three explored general-purpose abstractions are: data synchronization, push notification, and fetching distributed partitioned named data. We discussed mechanisms for each abstraction and evaluated selected implementations in different network connectivities. We showed that when relying on these ICTs, ICN applications can operate in the information plane, and no longer need to rely on forwarding strategies. We demonstrated how the ICT approach could support the tested applications in intermittent environments, such as lossy links or lack of SEEP, where IP-based applications do not work well.

It may appear that the core contribution of this dissertation is to allow placing function-specific features at intermediate nodes. However, we argue that forwarding strategies already implement function-specific features in the network, and that the core contribution of this work is our attempt to decouple applications from connectivity. Our approach to the solution suggests providing ICN applications with a different architectural component to rely on in order to satisfy their application-level needs, and allowing this architectural component to operate at both the endpoint and in the network. Moreover, our contribution also includes a definition of the role of the forwarding strategy component, which was previously underspecified. We defined the forwarding strategy to be the architectural component that translates the request for named data abstraction into a practical connectivity-oriented network mechanism.

One can argue that the concept of ICT can be related to the concept of Delay-Tolerant Networking (DTN) [21]. Although both concepts address similar connectivity concerns, they serve fundamentally different roles. DTN is a protocol suite that aims to extend Internet capabilities by implementing routing mechanisms for disturbed networks. However, ICT is a broadly-applicable abstraction and a communication mechanism that aims to support ICN applications running on different network connectivities.

7.1 Future Research Directions

The approach taken by this dissertation aims to address the problem by decoupling information-oriented mechanisms from forwarding strategies, and by creating information-oriented abstractions and placing them in a new architectural component. While this dissertation demonstrates that the implemented ICT abstractions can decouple applications from connectivity in the tested topologies, this work does not formally prove that ICTs always decouple applications from connectivity. To justify the proposed addition to ICN architectures, formal proof should show that ICT can comprehensively solve the problem and decouple applications from connectivity.

Moreover, other non-architectural solutions should be explored and evaluated against ICT. For instance, can a smart ICN repository implement various information-oriented storage mechanisms that decouple applications from connectivity? As we showed in this work, relying on ICN storage is not sufficient for resilient data communication in all intermittent connectivities, and therefore, such a repo solution would have to implement different application-level mechanisms for both sending Interest and storing Data packets.

Moreover, although this dissertation made an effort to propose ICT abstractions that address real challenges of present ICN applications, determining the finite set of ICTs in ICNs remains

an open question for future work. We present the following research directions to further improve the concept of ICT and the ICT mechanisms we discussed in this dissertation.

The purpose of the implemented mechanisms for push notifications, and the purpose of the discussed mechanisms for data partitioning was to demonstrate the concept of ICT, and not to present robust and efficient ICT mechanisms. Therefore, the exploration and implementation of additional mechanisms for robust and efficient communication is a rich area for future work. For instance, future work can implement and evaluate the proposed ICT mechanism discussed in 6.2.1.

Specifically, the implementation of ICT-Notify presented in Chapters 5 was designed to address the challenges of a push mechanism, and to demonstrate the capabilities of the push abstraction in different network connectivities, and was not evaluated in a large scale topology. Future work should evaluate the implemented mechanisms in a larger scale, and test the performance of an intermediate ICT component under multiple notification streams.

Furthermore, the interaction between the ICT component and other network components must be further explored. For instance, how does routing interact with ICTs? Should QoS and congestion control mechanisms be implemented in the connectivity plane (forwarding strategy) or the information plane (ICT)? Are these mechanisms even relevant in the channel-less ICN architecture? Can one ICT implement different mechanisms for the same set of application needs? For instance, can we have one universal ICT API that can work with different in-network sync mechanism? Moreover, the exact placement(s) of an intermediate ICT should be explored and better understood. For instance, is it beneficial to deploy the intermediate ICT process everywhere in the network, or only in some specific network nodes?

Another rich area for future work is security and trust considerations of the ICT component. In this dissertation, we provided initial statements and determined that an ICT should

never rely on content decryption, but we did not sufficiently address validation and trust considerations. For instance, can an intermediate ICT validate packets in a general-purpose way? Or how can an endpoint trust traffic generated by an intermediate ICT component?

7.2 Publications of Dissertation Work

The work presented in this dissertation was previously published in the following:

- Our work presents the concept of Information-Centric Transport (ICT) has been published in [18]. This publication discusses the role of forwarding strategies, introduces the concept of ICT, and presents early versions of ICT-Sync and ICT-Notify. Chapters 2-5 include content published in this paper.
- The initial discussion about the coupling of ICN applications and forwarding strategies was published in [15]. Chapters 2 and 6 include content published in this paper.
- The iSync protocol, discussed in Chapter 4, was published in [29], and the initial performance measurements of CCNx Sync was published in [17]. As stated in the publication, the author of this dissertation was an equal contributor co-author of the work in [29].
- The abstraction to control strategy retransmissions, presented in Chapter 6, was published in [14].

References

[1] Alexander Afanasyev, Xiaoke Jiang, Yingdi Yu, Jiewen Tan, Yumin Xia, Allison Mankin, and Lixia Zhang. “NDNS: A DNS-Like Name Service for NDN.” In: *26th International Conference on Computer Communication and Networks (ICCCN)*. IEEE. 2017, pp. 1–9.

[2] Alexander Afanasyev, Cheng Yi, Lan Wang, Beichuan Zhang, and Lixia Zhang. *Scaling NDN Routing: Old Tale, New Design*. Tech. rep. 2013.

[3] Alexander Afanasyev, Cheng Yi, Lan Wang, Beichuan Zhang, and Lixia Zhang. “SNAMP: Secure Namespace Mapping to Scale NDN Forwarding.” In: *Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE. 2015, pp. 281–286.

[4] Alexander Afanasyev, Zhenkai Zhu, Yingdi Yu, Lijing Wang, and Lixia Zhang. “The Story of chronoshare, or how NDN Brought Distributed Secure File Sharing Back.” In: *12th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*. IEEE. 2015, pp. 525–530.

[5] Alexander Afanasyev et al. “NFD developer's guide.” In: *Dept. Comput. Sci., Univ. California, Los Angeles, Los Angeles, CA, USA, Tech. Rep. NDN-0021* (2014).

[6] Sachin Agarwal, Starobinski, and et al. “On the Scalability of Data Synchronization Protocols for PDAs and Mobile Devices.” In: *Network, IEEE* 16.4 (2002), pp. 22–28.

[7] Bander A Alzahrani, Vassilios G Vassilakis, and Martin J Reed. “Key Management in Information Centric Networking.” In: *International Journal of Computer Networks & Communications* 5.6 (2013), p. 153.

[8] Marica Amadeo, Claudia Campolo, and Antonella Molinaro. “Forwarding Strategies in Named Data Wireless Ad Hoc Networks: Design and Evaluation.” In: *Journal of Network and Computer Applications* 50 (2015), pp. 148–158.

[9] Marica Amadeo, Claudia Campolo, and Antonella Molinaro. “Internet of Things via Named Data Networking: The Support of Push Traffic.” In: *International Conference and Workshop on the Network of the Future (NOF)*. IEEE. 2014, pp. 1–5.

[10] Marica Amadeo, Claudia Campolo, and Antonella Molinaro. "Multi-Source Data Retrieval in IoT via Named Data Networking." In: *Proceedings of the 1st ACM Conference on Information-Centric Networking*. ACM. 2014, pp. 67–76.

[11] Marica Amadeo, Antonella Molinaro, Claudia Campolo, Manolis Sifalakis, and Christian Tschudin. "Transport Layer Design for Named Data Wireless networking." In: *Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE. 2014, pp. 464–469.

[12] Abdalkarim Awad, Christoph Sommer, Reinhard German, and Falko Dressler. "Virtual Cord Protocol (VCP): A Flexible DHT-like Routing Service for Sensor Networks." In: *5th International Conference on Mobile Ad Hoc and Sensor Systems*. IEEE. 2008, pp. 133–142.

[13] Florin Baboescu and George Varghese. "Scalable Packet Classification." In: *ACM SIGCOMM Computer Communication Review* 31.4 (2001), pp. 199–210.

[14] Hila Ben Abraham and Patrick Crowley. "Controlling Strategy Retransmissions in Named Data Networking." In: *Proceedings of the Symposium on Architectures for Networking and Communications Systems*. IEEE Press. 2017, pp. 70–81.

[15] Hila Ben Abraham and Patrick Crowley. "Forwarding Strategies for Applications in Named Data Networking." In: *Proceedings of the 2016 Symposium on Architectures for Networking and Communications Systems*. ACM. 2016, pp. 111–112.

[16] Hila Ben Abraham and Patrick Crowley. "In-Network Retransmissions in Named Data Networking." In: *Proceedings of the 2016 conference on 3rd ACM Conference on Information-Centric Networking*. ACM. 2016, pp. 209–210.

[17] Hila Ben Abraham and Patrick Crowley. "Performance Measurement of the CCNx synchronization protocol." In: *Architectures for Networking and Communications Systems*. IEEE. 2013, pp. 121–122.

[18] Hila Ben Abraham, Jyoti Parwatikar, John DeHart, Adam Drescher, and Patrick Crowley. "Decoupling Information and Connectivity via Information-Centric Transport." In: *Proceedings of the 2018 conference on 5th ACM Conference on Information-Centric Networking*. ACM. 2018, pp. 54–67.

[19] Chaoyi Bian, Zhenkai Zhu, Alexander Afanasyev, Ersin Uzun, and Lixia Zhang. "Deploying Key Management on NDN Testbed." In: *UCLA, Peking University and PARC, Tech. Rep* (2013).

[20] *CCNx Sync*. `https://www.ccnx.org/releases/latest/doc/technical/SynchronizationProtocol.html`. Accessed 2017.

[21] Vincent Cerf. "Delay Tolerant Networking." In: *Proceedings of the 10th Symposium on High Performance Interconnects HOT Interconnects*. IEEE Computer Society. 2002, p. 1.

[22] Raffaele Chiocchetti, Diego Perino, Giovanna Carofiglio, Dario Rossi, and Giuseppe Rossini. "Inform: a Dynamic Interest Forwarding Mechanism for Information Centric Networking." In: *Proceedings of the 3rd ACM SIGCOMM workshop on Information-Centric Networking*. ACM. 2013, pp. 9–14.

[23] David D Clark and David L Tennenhouse. "Architectural Considerations for a New generation of Protocols." In: *ACM SIGCOMM Computer Communication Review*. Vol. 20. 4. ACM. 1990, pp. 200–208.

[24] Alberto Compagno et al. "To NACK or not to NACK? Negative Acknowledgments in Information-Centric Networking." In: *arXiv preprint arXiv:1503.02123* (2015).

[25] Jean-Sébastien Coron, Yevgeniy Dodis, and et al. "Merkle-Damgård Revisited: How to Construct a Hash Function." In: *Advances in Cryptology–CRYPTO 2005*. Springer. 2005, pp. 430–448.

[26] Stephanie DiBenedetto and Christos Papadopoulos. "Mitigating Poisoned Content with Forwarding Strategy." In: *Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE. 2016, pp. 164–169.

[27] David Eppstein, Michael T Goodrich, Frank Uyeda, and George Varghese. "What's the Difference?: Efficient Set Reconciliation Without Prior Context." In: *ACM SIGCOMM Computer Communication Review*. Vol. 41. 4. ACM. 2011, pp. 218–229.

[28] Anja Feldman and S Muthukrishnan. "Tradeoffs for Packet Classification." In: *Proceedings of IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No. 00CH37064)*. Vol. 3. IEEE. 2000, pp. 1193–1202.

[29] Wenliang Fu, Hila Ben Abraham, and Patrick Crowley. "Synchronizing Namespaces with Invertible Bloom Filters." In: *Proceedings of the Eleventh ACM/IEEE Symposium on Architectures for networking and communications systems*. IEEE Computer Society. 2015, pp. 123–134.

[30] Massimo Gallo, Lin Gu, Diego Perino, and Matteo Varvello. "NaNET: Socket API and Protocol Stack for Process-to-Content Network Communication." In: *Proceedings of the 1st international conference on Information-Centric Networking*. ACM. 2014, pp. 185–186.

[31] JJ Garcia-Luna-Aceves. "A Fault-Tolerant Forwarding Strategy for Interest-Based Information Centric Networks." In: *IFIP Networking Conference (IFIP Networking)*. IEEE. 2015, pp. 1–9.

[32] Michael T Goodrich and Michael Mitzenmacher. "Invertible Bloom Lookup Tables." In: *49th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. IEEE. 2011, pp. 792–799.

[33] Giulio Grassi, Davide Pesavento, Giovanni Pau, Lixia Zhang, and Serge Fdida. “Navigo: Interest Forwarding by Geolocations in Vehicular Named Data Networking.” In: *16th International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*. IEEE. 2015, pp. 1–10.

[34] Pankaj Gupta and Nick McKeown. “Algorithms for Packet Classification.” In: *IEEE Network* 15.2 (2001), pp. 24–32.

[35] Pankaj Gupta and Nick McKeown. “Packet Classification on Multiple Fields.” In: *ACM SIGCOMM Computer Communication Review* 29.4 (1999), pp. 147–160.

[36] Peter Gusev and Jeff Burke. “NDN-RTC: Real-time videoconferencing over Named Data Networking.” In: *Proceedings of the 2nd International Conference on Information-Centric Networking*. ACM. 2015, pp. 117–126.

[37] AKM Hoque et al. “NLSR: Named-Data Link State Routing Protocol.” In: *Proceedings of the 3rd ACM SIGCOMM Workshop on Information-Centric Networking*. ACM. 2013, pp. 15–20.

[38] Mihaela Ion, Jianqing Zhang, and Eve M Schooler. “Toward Content-Centric Privacy in ICN: Attribute-Based Encryption and Routing.” In: *Proceedings of the 3rd ACM SIGCOMM workshop on Information-Centric Networking*. ACM. 2013, pp. 39–40.

[39] Van Jacobson, Diana K Smetters, James D Thornton, Michael F Plass, Nicholas H Briggs, and Rebecca L Braynard. “Networking Named Content.” In: *Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies*. ACM. 2009, pp. 1–12.

[40] Van Jacobson et al. “Custodian-Based Information Sharing.” In: *IEEE Communications Magazine* 50.7 (2012).

[41] Xiaoke Jiang and Jun Bi. *How To Design and Implement NDN Based Applications*. Tech. rep. Tsinghua University, 2015.

[42] Gary D Knott. “Expandable Open Addressing Hash Table Storage and Retrieval.” In: *Proceedings of the 1971 ACM SIGFIDET (now SIGMOD) Workshop on Data Description, Access and Control*. ACM. 1971, pp. 187–206.

[43] Ze’ev Lailari, Hila Ben Abraham, Ben Aronberg, Jackie Hudepohl, Haowei Yuan, John DeHart, Jyoti Parwatikar, and Patrick Crowley. “Experiments with the Emulated NDN Testbed in ONL.” In: *Proceedings of the 2nd International Conference on Information-Centric Networking*. ACM. 2015, pp. 219–220.

[44] Paul Leach, Michael Mealling, and Rich Salz. *A Universally Unique Identifier (UUID) URN Namespace*. Tech. rep. 2005.

[45] Vince Lehman, Ashlesh Gawande, Beichuan Zhang, Lixia Zhang, Rodrigo Aldecoa, Dmitri Krioukov, and Lan Wang. “An Experimental Investigation of Hyperbolic Routing with a Smart Forwarding Plane in NDN.” In: *IEEE/ACM 24th International Symposium on Quality of Service (IWQoS)*. IEEE. 2016, pp. 1–10.

[46] Jared Lindblom, M Huang, Jeff Burke, and Lixia Zhang. *FileSync/NDN: Peer-to-peer File Sync over Named Data Networking*. Tech. rep. NDN-0012, NDN, 2013.

[47] Witold Litwin. "Linear Hashing: A New tool for File and Table Addressing." In: *VLDB*. Vol. 80. 1980, pp. 1–3.

[48] Xuan Liu, Zhuo Li, Peng Yang, and Yongqiang Dong. "Information-Centric Mobile Ad Hoc Networks and Content Routing: a Survey." In: *Ad Hoc Networks* 58 (2017), pp. 255–268.

[49] Spyridon Mastorakis, Alexander Afanasyev, Yingdi Yu, and Lixia Zhang. "nTorrent: Peer-to-Peer File Sharing in Named Data Networking." In: *26th International Conference on Computer Communication and Networks (ICCCN)*. IEEE. 2017, pp. 1–10.

[50] Ralph C Merkle. "A Fast Software One-Way Hash Function." In: *Journal of Cryptology* 3.1 (1990), pp. 43–58.

[51] Ilya Moiseenko, Lijing Wang, and Lixia Zhang. "Consumer/Producer Communication with Application Level Framing in Named Data Networking." In: *Proceedings of the 2nd International Conference on Information-Centric Networking*. ACM. 2015, pp. 99–108.

[52] Marc Mosko. *CCNx 1.0 Protocol Specification Roadmap*. 2013.

[53] *NDN Testbed*. `http://ndnmap.arl.wustl.edu/`. Accessed 2019.

[54] *NFD - Named Data Networking Forwarding Daemon*. `http://named-data.net/doc/NFD/current/`. Accessed 2019.

[55] *Project CCNx*. `http://www.ccnx.org/`. Accessed 2017.

[56] *Project CICN*. `https://wiki.fd.io/view/Cicn`. Accessed 2019.

[57] Haiyang Qian, Ravishankar Ravindran, Guo-Qiang Wang, and Deep Medhi. "Probability-Based Adaptive Forwarding Strategy in Named Data Networking." In: *IFIP/IEEE International Symposium on Integrated Network Management (IM)*. IEEE. 2013, pp. 1094–1101.

[58] Ravi Ravindran, Asit Chakraborti, Syed Obaid Amin, and Jiachen Chen. *Support for Notifications in CCN*. Internet-Draft draft-ravi-icnrg-ccn-notification-01. Work in Progress. Internet Engineering Task Force, July 2017. 22 pp.

[59] Zeinab Rezaeifar, Jian Wang, Heekuck Oh, Junbeom Hur, and Suk-Bok Lee. "A Reliable Adaptive Forwarding Approach in Named Data Networking." In: *Future Generation Computer Systems* (2019).

[60] Matthew Roughan, Subhabrata Sen, Oliver Spatscheck, and Nick Duffield. "Class-of-service Mapping for QoS: A Atatistical Signature-Based Approach to IP Traffic Classification." In: *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement*. ACM. 2004, pp. 135–148.

[61] Jerome H Saltzer, David P Reed, and David D Clark. “End-to-End Arguments in System Design.” In: *ACM Transactions on Computer Systems (TOCS)* 2.4 (1984), pp. 277–288.

[62] Christos-Alexandros Sarros, Adisorn Lertsinsrubtavee, Carlos Molina-Jimenez, Konstantinos Prasopoulos, Sotiris Diamantopoulos, Dimitris Vardalis, and Arjuna Sathiaseelan. “ICN-Based Edge Service Deployment in Challenged Networks.” In: *Proceedings of the 4th ACM Conference on Information-Centric Networking*. ACM. 2017, pp. 210–211.

[63] Klaus M Schneider and Udo R Krieger. “Beyond Network Selection: Exploiting Access Network Heterogeneity with Named Data Networking.” In: *Proceedings of the 2nd International Conference on Information-Centric Networking*. ACM. 2015, pp. 137–146.

[64] Wentao Shang, Alexander Afanasyev, and Lixia Zhang. “Vectorsync: Distributed Dataset Synchronization over Named Data Networking.” In: *Proceedings of the 4th ACM Conference on Information-Centric Networking*. ACM. 2017, pp. 192–193.

[65] Ion Stoica, Robert Morris, David Karger, M Frans Kaashoek, and Hari Balakrishnan. “Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications.” In: *ACM SIGCOMM Computer Communication Review* 31.4 (2001), pp. 149–160.

[66] Andrew Tridgell and Paul Mackerras. *The Rsync Algorithm,”Australian National University*. Tech. rep. TR-CS-96-05, 1996.

[67] Christian Tschudin and Manolis Sifalakis. “Named Functions and Cached Computations.” In: *11th Conference on Consumer Communications and Networking (CCNC)*. IEEE. 2014, pp. 851–857.

[68] Bittorrent Website. *https://www.bittorrent.com/bittorrent-free*. Accessed 2019.

[69] Bittorrent Sync Website. *http://www.bittorrent.com/sync*. Accessed 2019.

[70] Google Hangouts Website. *https://hangouts.google.com*. Accessed 2019.

[71] NSF Future Internet Architecture Project Website. “http://www.nets-fia.net/.” In:

[72] Slack Website. *https://slack.com*. Accessed 2019.

[73] The Dropbox Website. *https://www.dropbox.com/*. Accessed 2019.

[74] The Google Drive Website. *http://www.google.com/drive/about.html*. Accessed 2019.

[75] Cedric Westphal. “Synchronizing State with Strong Similarity Between Local and Remote Systems.” In: *Proceedings of the Third ACM Workshop on Mobile Cloud Computing and Services*. MCS ’12. Low Wood Bay, Lake District, UK, 2012, pp. 15–20.

[76] Charlie Wiseman et al. “A remotely Accessible Network Processor-Based Router for Network Experimentation.” In: *Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*. ACM. 2008, pp. 20–29.

[77] Xipeng Xiao and Lionel M Ni. “Internet QoS: a big picture.” In: *IEEE network* 13.2 (1999), pp. 8–18.

[78] Cheng Yi et al. "A Case for Stateful Forwarding Plane." In: *Computer Communications* (2013).

[79] Cheng Yi et al. "On the Role of Routing in Named Data Networking." In: *Proceedings of the 1st international conference on Information-Centric Networking*. ACM. 2014, pp. 27–36.

[80] Cheng Yi, Alexander Afanasyev, Lan Wang, Beichuan Zhang, and Lixia Zhang. "Adaptive Forwarding in Named Data Networking." In: *ACM SIGCOMM Computer Communication Review* 42.3 (2012), pp. 62–67.

[81] Keping Yu, Mohammad Arifuzzaman, Zheng Wen, Di Zhang, and Takuro Sato. "A Key Management Scheme for Secure Communications of Information Centric Advanced Metering Infrastructure in Smart Grid." In: *IEEE transactions on instrumentation and measurement* 64.8 (2015), pp. 2072–2085.

[82] Yingdi Yu, Alexander Afanasyev, David Clark, Van Jacobson, Lixia Zhang, et al. "Schematizing Trust in Named Data Networking." In: *Proceedings of the 2nd International Conference on Information-Centric Networking*. ACM. 2015, pp. 177–186.

[83] Lixia Zhang, Alexander Afanasyev, Jeffrey Burke, Van Jacobson, Patrick Crowley, Christos Papadopoulos, Lan Wang, Beichuan Zhang, et al. "Named Data Networking." In: *ACM SIGCOMM Computer Communication Review* 44.3 (2014), pp. 66–73.

[84] Minsheng Zhang, Vince Lehman, and Lan Wang. "Scalable Name-Based Data Synchronization for Named Data Networking." In: *INFOCOM 2017-IEEE Conference on Computer Communications*. IEEE. 2017, pp. 1–9.

[85] Zhenkai Zhu and Alexander Afanasyev. "Let's ChronoSync: Decentralized Dataset State Synchronization in Named Data Networking." In: *ICNP*. 2013, pp. 1–10.

[86] Zhenkai Zhu, Chaoyi Bian, Alexander Afanasyev, Van Jacobson, and Lixia Zhang. *Chronos: Serverless Multi-User Chat over NDN*. Tech. rep. 2012.