UNIVERSITY OF CALIFORNIA
SANTA CRUZ

DIRECTORY ASSISTED ROUTING OF CONTENT IN INFORMATION-CENTRIC NETWORKS

A dissertation submitted in partial satisfaction of the
requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER ENGINEERING

by

Nitish John

September 2020

The Dissertation of Nitish John
is approved:

Prof. J.J. Garcia-Luna-Aceves, Chair

Prof. Martine Schlag

Prof. Brad R. Smith

Quentin Williams
Interim Vice Provost and Dean of Graduate Studies

Table of Contents

List of Figures v

List of Algorithms viii

Abstract ix

Dedication xi

Acknowledgments xii

1 Introduction 1

2 Related Work 6
- 2.1 Name Resolution and DNS . . . 6
- 2.2 Content-Oriented Network . . . 8
- 2.3 Information-Centric Network (ICN) . . . 9
- 2.4 Loop-free Scalable Routing in ICN . . . 16
- 2.5 Compact & Prefix Routing . . . 17
- 2.6 Information Naming & Lookup . . . 19
- 2.7 Content Retrieval . . . 22

3 DARCI: Directory Assisted Routing of Content in ICN 26
- 3.1 Basic Operation . . . 28
- 3.2 Name Resolution System . . . 31
  - 3.2.1 Information Stored & Exchanged . . . 32
  - 3.2.2 Directory Trie . . . 36
  - 3.2.3 Trie Maintenance . . . 41
- 3.3 Publish and Subscribe . . . 47
  - 3.3.1 Publish Operation . . . 47
  - 3.3.2 Subscribe Operation . . . 50
  - 3.3.3 Content Encoding . . . 52
- 3.4 Content Routing . . . 53

3.5 Simulation 57
3.6 Performance Comparison 59
3.7 Summary 67

4 MIDAR: Multi-Instantiated contents with Directory Assisted Routing 68
4.1 Basic Operation 69
4.2 Multi-Instance Publish & Subscribe 73
4.2.1 Information Stored & Exchanged 73
4.2.2 MIDR Example 75
4.3 Simulation 81
4.4 Performance Comparison 83
4.5 Summary 91

5 ACED: Adaptive Cache Enabled Directory Assisted Routing 93
5.1 Basic Operation 95
5.2 ACED 96
5.2.1 Information Stored & Exchanged 98
5.2.2 ACED Example 98
5.3 Simulation 104
5.4 Performance Comparison 106
5.5 Summary 107

6 Conclusion 110

Bibliography 114

List of Figures

2.1 Name Resolution in Legacy DNS: Resolvers translate names to addresses by following a chain of delegations iteratively (2-5) or recursively (6-7) . . . 7
2.2 CDN servers . . . 8
2.3 Interval Routing . . . 17
2.4 Prefix Label Routing . . . 18
2.5 Example content name mapping in DHT with consistent hashing . . . 20
2.6 Example content name mapping in Hilbert Curve . . . 20
2.7 Example HTML content with sub-resources . . . 22
2.8 Content retrieval in Internet . . . 23
2.9 Content retrieval in a DHT network . . . 24
2.10 Content retrieval in NDN . . . 25

3.1 Content retrieval in DARCI . . . 27
3.2 Basic operation of DARCI . . . 29
3.3 Directory Router Initialization in DARCI . . . 34
3.4 Start of HELLO messages in DARCI . . . 35
3.5 Directory router discovery and initial trie formation in DARCI . . . 38
3.6 Stabilized trie in DARCI with root directory a . . . 40
3.7 Example link failure & recovery in DARCI . . . 44
3.8 Trie root a unreachable in DARCI . . . 45
3.9 New trie root i elected in DARCI . . . 46
3.10 Content publish in DARCI . . . 47

3.11 Content subscribe in DARCI 50
3.12 HCN and ICN naming architecture comparison and effect on performance 53
3.13 Content routing in DARCI 54
3.14 DARCI simulation in 59 node British Telecom NA topology . . . 57
3.15 DARCI simulation: Throughput (Mbps) with 20K Names 59
3.16 DARCI simulation: Throughput (Mbps) with 5K Names 60
3.17 DARCI simulation: Throughput (Mbps) with 500 Names 60
3.18 DARCI simulation: Throughput (Mbps) with 20 Names 61
3.19 DARCI simulation: Delay (μs) with 20K Names 61
3.20 DARCI simulation: Delay (μs) with 5K Names 62
3.21 DARCI simulation: Delay (μs) with 500 Names 62
3.22 DARCI simulation: Delay (μs) with 20 Names 63
3.23 DARCI simulation: Hop count with 20K Names 63
3.24 DARCI simulation: Hop count with 5K Names 64
3.25 DARCI simulation: Hop count with 500 Names 64
3.26 DARCI simulation: Hop count with 20 Names 65
3.27 DARCI simulation: Table size LT+NDT, FIB & PIT 66

4.1 Basic operation of MIDAR 70
4.2 Publishing in MIDAR 76
4.3 Subscribing in MIDAR 78
4.4 Content routing in MIDAR 80
4.5 MIDAR simulation: 59 node BT North America topology 81
4.6 MIDAR simulation: Throughput (Mbps) with 50K Names 83
4.7 MIDAR simulation: Throughput (Mbps) with 20K Names 84
4.8 MIDAR simulation: Throughput (Mbps) with 5K Names 84
4.9 MIDAR simulation: Throughput (Mbps) with 500 Names 85
4.10 MIDAR simulation: Throughput (Mbps) with 20 Names 85
4.11 MIDAR simulation: Delay (μs) with 50K Names 86
4.12 MIDAR simulation: Delay (μs) with 20K Names 86

4.13 MIDAR simulation: Delay (μs) with 5K Names 87
4.14 MIDAR simulation: Delay (μs) with 500 Names 87
4.15 MIDAR simulation: Delay (μs) with 20 Names 88
4.16 MIDAR simulation: Hop count with 50K Names 88
4.17 MIDAR simulation: Hop count with 20K Names 89
4.18 MIDAR simulation: Hop count with 5K Names 89
4.19 MIDAR simulation: Hop count with 500 Names 90
4.20 MIDAR simulation: Hop count with 20 Names 90

5.1 Basic operation of ACED 97
5.2 ACED Cache Progression 99
5.3 ACED simulation: 1998 World Cup Total Traffic (April 30-July 26) 105
5.4 ACED simulation: 1998 World Cup flash-crowd (April 30-May 5) 105
5.5 ACED simulation: Throughput (Mbps) 107
5.6 ACED simulation: Delay (μs) 108
5.7 ACED simulation: Hop count 108

List of Algorithms

1 DARCI: Process `HELLO` message at router u 37

2 DARCI: Process `ENROLL-DIRECTORY` at directory router t 39

3 DARCI: Process `ENROLL-DIRECTORY-ACK` at directory router u . . 39

4 DARCI: Process `UPDATE-DIRECTORY` message at directory router u 41

5 DARCI: Process `HELLO-DIRECTORY` message at directory router u 43

6 DARCI: Process `UPDATE-DIRECTORY` message at directory router u 45

7 DARCI: Transform function at directory a 49

8 DARCI: Process `PUBLISH` message at directory a 49

9 DARCI: Process `LOOKUP` message at directory o 51

10 DARCI: Process `DATA-REQUEST` message at directory o 55

11 DARCI: Process `DATA-REQUEST` message at router c 56

12 DARCI: Process `DATA-REPLY` message at d 56

13 ACED: Process `DATA-REQUEST` message at router i 102

14 ACED: Process `DATA-REPLY` message at i 103

15 ACED: CANOP pack-unpack at i 104

Abstract

Directory Assisted Routing of Content in Information-Centric Networks

by

Nitish John

The Internet's original architecture is on resource sharing by addressing hosts within a single network. As the number of addressable hosts within separate networks increased, a distributed Domain Name System (DNS) resolved hierarchical hostnames into the host's physical Internet addresses. Hence, we perceive the origins of the Internet to be host-centric. As the Internet evolved with the adoption of name resolution, application and user-generated information grew exponentially. The dire need to bring information closer and faster to users ushered in the implementation of the content distribution network (CDN). CDN works by redirecting users to closest hosts by assuming the user's location the same as resolver's - an unreliable assumption that would often cost in performance. User experience matters in delivering content. The demands of Quality of Service (QoS), security, and scalability from today's Internet, which has evolved into the implementation of patches on top of a fundamental host-centric architecture, introduces multi-fold challenges and complications in architectural evolution and robustness.

Recent advances in pivoting from host-centric to information-centric architecture ushers in a clean slate approach whereby information is treated as first-class addressable entity. This direction, known as Information-Centric Network (ICN), addresses efficiency, scalability, and robustness required to distribute and manipulate the ever-increasing information generated by applications and connected devices on the Internet. In this dissertation, we begin by studying the goals of an ICN. ICN proposes uniquely named data as the core Internet principle. We review various surveys on existing ICN architectural proposals and focus on Named

Data Networking (NDN) - which is one of the widely pursued architectures in the research community and industry. We introduce Directory Assisted Routing of Content in Information-centric network (DARCI), Multi-Instantiated contents with Directory Assisted Routing (MIDAR) in ICN and Adaptive Cache Enabled Directory (ACED) assisted routing in ICN.

DARCI is a new protocol for information addressing, discovery, and routing where information names are mapped to distributed directories in a trie with prefix labels using a space-filling curve (SFC). DARCI combines information name and mapping to intervals and prefix labels to create a stateless, scalable, reliable, and efficient name resolution and routing of information between publisher and subscriber. DARCI achieves lookup speed in the order of magnitude higher than the non-deterministic Longest Prefix Match lookup on variable-length content name string by transforming content names to locality preserving codes of fixed length integers. MIDAR adapts the DARCI protocol to enhance further and scale information name resolution and routing for multiple instances of information. Routers and directories use the compact routing in intervals and prefix labels to analytically decide on the next direction for the request or response by combining knowledge of intervals and prefix reachability at a given node. ACED furthers DARCI and MIDAR to achieve an unconventionally efficient caching methodology where caches opt to become authoritative owners of selective information temporarily. Such a cache node registers the information name with the directory trie. Subsequent information name resolution will receive the cache node as one of the publishers. Subscribers and intermediate nodes use this information to reach the closest copy of the requested content. DARCI, MIDAR, and ACED has been extensively evaluated against NDN, using real-world network topology and traffic data.

Dedication

To my beloved wife, Harriet, for her patience,
to our children Nathan, Evangelina & Daniel for their infinite forgiveness
and to our parents for their unwavering sacrifice in supporting my family as I
juggled graduate school and a full time job.

Acknowledgments

Returning to graduate school as a doctoral student while also working in the industry was not a trivial undertaking. Throughout this journey, Professor J. J. Garcia-Luna-Aceves has been my North star, keeping me on course and never letting me give up when challenges multiplied. His guidance, encouragement, and honest feedback pushed me to develop and deliver the best of my skills. The greatest gift I received is an invitation to celebrate 50 years of ALOHA System & the Future of Networking on January 24th, 2020. Through him, I met the founding members and inventors of "Wireless & Internet Communication" - Frank Kuo, Norm Abramson, Bob Kahn, Vint Cerf, Richard Binder, Don Nielson, and Fouad Tobagi. It was my honor to attend and witness the past, helping define the future. It is humbling to cherish this association with such esteemed founding members through Professor J. J. Garcia-Luna-Aceves.

I thank Professors Martine Schlag and Brad Smith, who, despite their busy schedules, found time to serve as committee members and provide feedback. I thank Professors Katia Obraczka and Roberto Manduchi for the warm welcome I received to the email I sent when searching for a graduate research school. I thank the past and present peers in Computer Communication Research Group (CCRG) - James Mathewson, Dhananjay Sampath, Turhan Karadeniz, Ehsan Hemmati, and Ramesh Srinivasan. I also thank Graduate Student Advising office, Emily Gregg, Alicia Haley, and Lisa Stipanovich for their patience in guiding me and Chantal Herrera, who made sure I received the code to register for Independent Study before the deadlines.

To my loving wife Harriet and our children, Nathan, Evangelina, and Daniel - it is your compassion and love that keeps me going. To our parents who traveled across the world to be with us and help support our family - Thank you.

Chapter 1

Introduction

The modern-day Internet architecture is a host-centric communication model. Early days of the Internet accessed valuable computing resources. With the successful testing of TCP, hosts within various mini-networks were able to communicate with each other. Domain Name Systems (DNS) helped users identify hosts using human-readable names. A DNS lookup process requested machine addressable information given a host's human-readable identification. This initial success in ease of host lookup and reachability of one host from another host ushered an era of exponential growth for user applications and user-generated information. Uses of the Internet shifted from using the Internet for accessing valuable computing resources to accessing information across hosts. Organically, this phenomenon produced an exponential growth in the number of connected hosts (or devices) and also in the amount of information created. For example, Cisco Visual Networking Index reports the number of devices connected to IP networks will be 28.5 billion by 2022[1]. In another report, Cisco predicts an annual global cloud IP traffic to reach 19.5 ZB (1.6 ZB per month) by the end of 2021; with the total amount of data created (and not necessarily stored) by any device reaching 847 ZB per year by 2021[2]. Data created is two orders of magnitude higher than the data stored.

Hence, a new architecture for the Internet of the future must put "information" at the core of its design to alleviate the modern problems seen with scaling content distribution, mobility, security, caching, and trust in host-centric networks.

To adapt the original Internet architecture to meet the demands of information distribution, various modifications to DNS systems were proposed and implemented [3][4][5][6][7][8][9][10][11]. It is quite clear to this point that all efforts are to expedite the name resolution to a host address in order to retrieve information for user. By the mid 1990's, distributed content stores known as Content Distribution Network (CDN) with content replicas were strategically placed in various geographies to induce proximity to users. CDN's rely on DNS for request routing and replica server selection. For this, CDN relies on the IP address of the client's local DNS resolver. This builds upon the flawed assumption that in the absence of knowledge regarding the client's actual network location, its resolver's location provides the best approximation. Such an assumption may yield a worse end to end web performance [12]. Users care about information and not its location. Users respond to faster access to information [13][14]. Unfortunately, today's application and network architecture for delivering faster content to users is one medley of patches on top of a fundamental Internet architecture that is not information-centric [15].

Recent advances in pivoting from host-centric to information-centric architecture ushers in a clean slate approach whereby information is treated as first-class addressable entity. This direction, known as Information-Centric Network (ICN), addresses efficiency, scalability, and robustness required to distribute and manipulate the ever-increasing information generated by applications and connected devices on the Internet. ICN proposes uniquely named data as the core Internet principle. Various ICN architectures proposed and studied[16][17][18]

have information requested, routed and delivered without addressing its host. These architectures address the common core principles of ICN but with different approaches. A widely adopted architecture is the Named Data Networking (NDN)[19][20]. NDN allows consumers to issue an Interest in the content object (CO) by addressing CO's name. NDN content routers maintain three tables: a stateful Pending Interest Table (PIT) that aggregates Interests for a CO, Forwarding Information Base (FIB) that maps the next hop to CO name prefix used to forward Interest towards publishers and Content Store (CS) to cache COs locally. Considering the future predictions on the growth of content created and on the number of devices connected [1][2], the vast size of the naming space imaginable poses a significant scalability challenge for name resolution, routing and caching.

In this thesis, we propose a new name resolution and routing protocol that adheres to ICN's goals. The results of extensive simulation with on-path caching and no caching comparing NDN and DARCI validates that DARCI scales efficiently as the number of uniquely addressable named information within the network increases. In chapter 2, we summarize a previous survey on ICN architectures and briefly summarizes key points of the design. We focus on the architecture of NDN and the role of FIB & PIT in the routing of contents. We highlight previous research that points out the challenges in scaling FIB and PIT sizes at internet scale. Such prior work has laid the foundation for a stateless design of routing protocol that can scale as the number of uniquely addressable content increases in ICN. We also describe the building blocks that make up the *sense of direction* ($\mathcal{SD}$) used by request and response packets in DARCI. ICN treats information as first-class citizens, as addressable and routable. Hence, merely implementing a simple lookup of named information lacks contribution to optimized and efficient range queries. To the best of our knowledge, there is no approach where name

resolution systems provide an elementary form of range query in ICN. However, unlike various literature in research that pairs the use of directory with *Distributed Hash Table* (DHT) for mapping content/services to identifiers, DARCI uses *Space Filling Curve* (SFC)[21] for mapping names to code space. DARCI configures a set of specialized routers called Directories into a trie. Instead of DHT, DARCI Directory trie uses the code space for a given space-filling curve and distributes it among trie members. The intervals allocated to trie members and their prefix labels create a compact routing direction for the request and response packets.

Chapter 3 introduces Directory Assisted Routing of Contents in ICN (DARCI) and describes the details and operation of the DARCI routing protocol. Specialized routers called directories form a trie to partition code space. The trie acts as the de-centralized communication backbone in DARCI. We assume that the overall system is aware of the space used for SFC code. All routers maintain multiple loop-free routes to Directories using sequence numbered distances. Directories use Trie Access Protocol (TAP) to maintain membership and to distribute code spaces. Publishers, subscribers, and Directories use Object Publish Update and Subscribe (OPUS) protocol for information name resolution.

Chapter 4 introduces Multi-Instantiated contents with Directory Assisted Routing in ICN (MIDAR) and describes the adaptation and working of DARCI name resolution and routing protocol for multi-instantiated information in ICN. MIDAR eliminates signaling overhead with zero-flooding and by embedding *sense of direction* for the requested copy of information instance.

Chapter 5 introduces Adaptive Cache Enabled Directory Assisted Routing of contents in ICN (ACED) and describes an unconventional approach to caching popular information in ICN. ACED solves quite a few problems seen with various cache implementations such as complex computations for determining eviction

strategy, sub-optimal eviction policy-induced churn, and lack of support for multi-instantiated copies of content in a network. ACED builds upon DARCI and MIDAR, uses intervals and prefixes for compact routing in name resolution and request of contents in ICN.

Chapter 2

Related Work

2.1 Name Resolution and DNS

As more hosts connected to the Internet, the Domain Name System (DNS) was introduced to scale and distribute the resolving of hierarchical hostnames into an Internet address. As the number of user and application generated contents increased, the number of virtual neighborhoods addressable by 32 or 128-bit numbers called Internet Protocol(IP) addresses increased. DNS became the signpost[4] in the virtual world, serving as the navigation infrastructure across the Internet. Fig. 2.1 shows the working of a legacy DNS [7] based name resolution. A client querying for "soe.ucsc.edu" sends a lookup message to its local resolver. If the local resolver cannot find the requested name, it forwards the query to the upstream server. If this server is unable to determine the authoritative name server for the requested hostname, it sends the query to the root server. The root server will refer the resolver to the appropriate authoritative server for the domain immediately below the root and to which the zone belongs. The resolver will then query this server and thus stepping down the tree from the root of the desired zone. A shown in the figure, resolvers translate names to addresses by following a chain of

delegations iteratively (2-5) or recursively (6-7). Following a chain of delegations to resolve queries incurs delay. The legacy DNS uses aggressive caching in order to reduce the latency of query resolution.

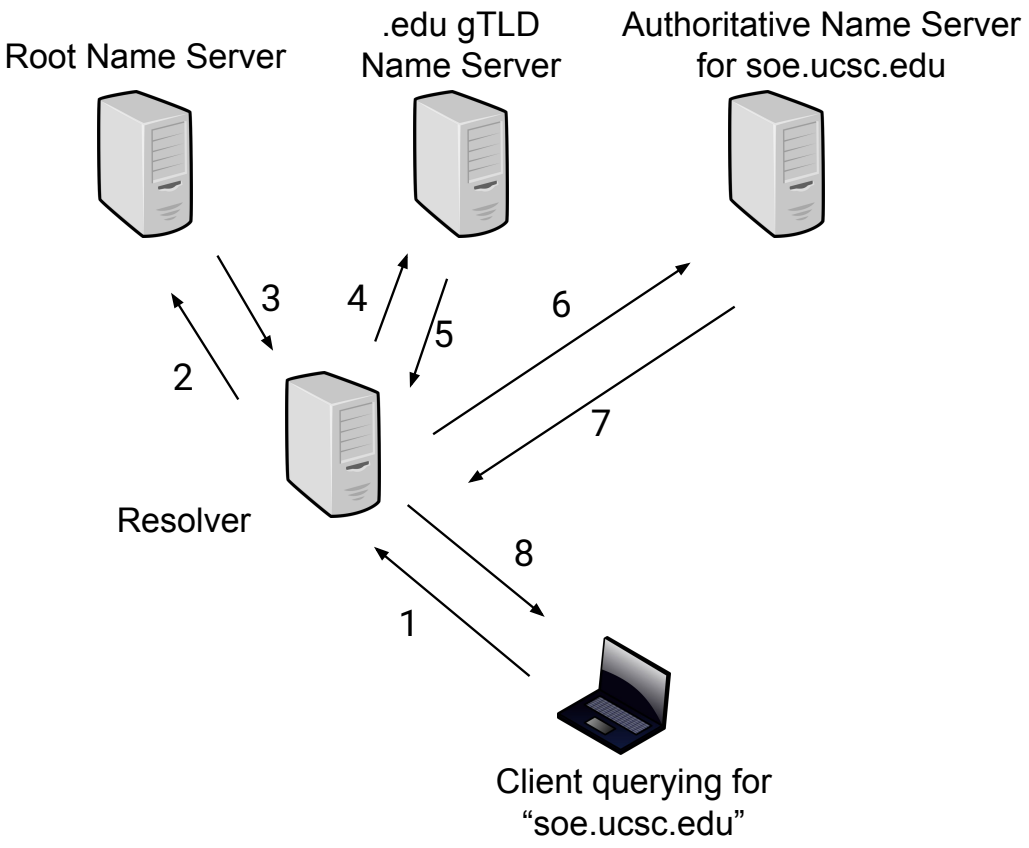

Figure 2.1: Name Resolution in Legacy DNS: Resolvers translate names to addresses by following a chain of delegations iteratively (2-5) or recursively (6-7)

In a host-centric network, the Zipf-like distribution is the analytical foundation for Web traffic analysis. Craig and Shang [22] has found that DNS lookup time takes away more than 20% of the web object retrievals in Web traffic. Low cache hit rates cause this low performance due to heavy-tailed Zipf-like query distribution in DNS [5]. There are various research work proposed to improve the latency incurred in resolving names to the host address have. Some of the improve-

ments are CoDNS[7], Disposable Domains[8], DNS Lie[9], Domain Sharding[10] and EDNS to send client subnet in DNS queries[23]. The complexity introduced by such after-thought architectural additions in the form of patches further hinders the path to efficiency improvements.

2.2 Content-Oriented Network

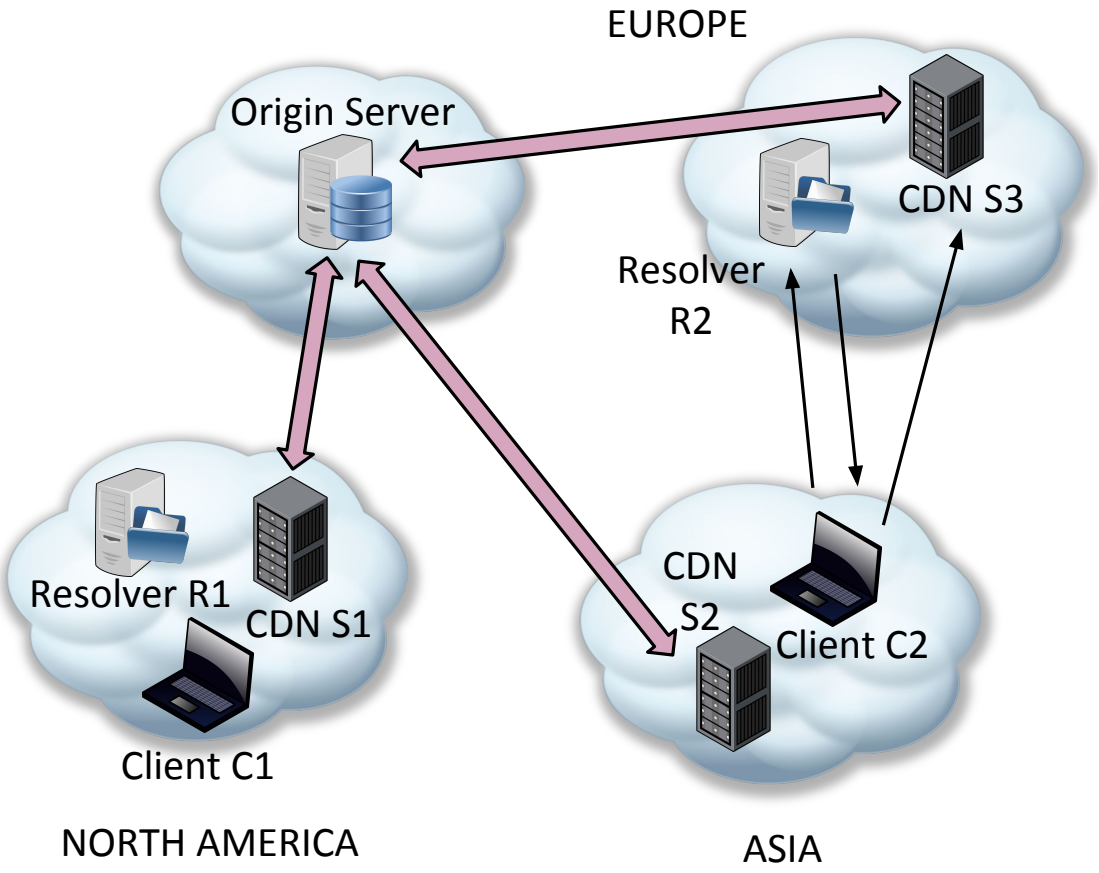

Figure 2.2: CDN servers

DNS systems increased reachability to a plethora of user and application generated content. Web protocols such as HTTP enabled access to contents with reduced latency with improved caching. However, traditional caching is limited in

the way it scales for various content types such as audio, video, and images. The inefficiency of basic caching overloaded content providing servers at busy times. Content Distribution Network (CDN) was created to alleviate the load on origin servers by delivering content on behalf of the origin server. The servers in CDN are either colocated in the same site as origin server or at different global locations with either some of all of the origin server's content cached among the servers. For client requests for content, the CDN uses the address of the resolver as a hint to the client's location and chooses the server close to the client - geographically or topologically. In Fig. 2.2, client C2 in ASIA uses resolver R2 in Europe to reach the closest CDN server. R2 redirects C2 to S3 in Europe, assuming C2 is in the same region as R2. Krishnamurthy et al. [12] studied the cost in performance by assuming the user's location to be that of its resolver's address. To address this issue, Chen, Sitaraman, and Torres [24] proposed scalable mapping using the EDNS0 [23] client-subnet extension of the DNS protocol. In EDNS0 client-subnet extension, recursive resolvers provide the client subnet as part of the DNS query. Resolvers use this information to make optimized routing decisions. A disadvantage of EDNS0 client-subnet extension, as mentioned in [23] Section 11.3, is where an arbitrary resolver or malicious client provides false information to reverse-engineer the algorithms in nameservers to calculate tailored responses.

2.3 Information-Centric Network (ICN)

It is evident that users care for contents irrespective of its location. Users experience matters when it comes to delivering faster access to information [13][14]. This demanded Quality of Service (QoS), security and scalability from Internet. Today's application and network architecture for delivering faster contents to users is built using patches on top of a fundamental Internet architecture that is not

scalable and not information-centric [15]. Information-centric Network (ICN) addresses improved efficiency, scalability and robustness by treating information as uniquely addressable entity.

Several ICN architectures [16][17][18] propose to replace the current host-centric model in order to address the challenges and shortcomings mentioned in our Introduction1. Some of the ICN projects include DONA [25], PURSUIT [26] and its predecessor Publish-Subscribe Internet Routing Paradigm (PSIRP) [27], Scalable & Adaptive Internet soLutions (SAIL) [28] and its predecessor 4WARD [29], COntent Mediator architecture for content aware nETworks (COMET) [30], CONVERGENCE [31], the US funded projects Named Data Networking (NDN) [19] and its predecessor Content Centric Networking (CCN) [32] and Mobility-First [33], as well as the French funded project ANR Connect [34] which adopts the NDN architecture. One of the architectural differences among most of these is how name resolution and data routing are done. We see two approaches: *coupled* and *decoupled*. In the coupled approach, the information requested is sent to the information provider, which returns the information to the host by following the reverse path over which request was forwarded. In the decoupled approach, the name resolution function does not influence the data routing path. Another key functionality that differentiates various implementation of ICN is in the approach taken to *name* information. The two approaches used are *hierarchical structured naming* and *flat naming*. *Naming*, *name resolution*, *data routing* and *caching* mechanism contributes as key architectural differentiators for ICN implementations.

DONA [25] uses flat names that are self-certifying to identify information. The name resolution mechanism is implemented as an overlay that maps its flat names to corresponding information. Each piece of information is assigned a flat name

consisting of the cryptographic hash of the host's public key and a label that uniquely identifies the information with respect to the host. DONA assumes that clients interested in a piece of information will learn about its name through some trusted external mechanisms (e.g., a search engine). DONA supports coupled and decoupled name resolution and data routing. Publishing host sends REGISTER message with objects name to its local *Resolution Handler* (RH). The local RH stores a pointer to the object and propagates this registration to RHs in its parent and peering domains based on established routing policies. This causes registrations to be replicated in all RHs to tier-1 providers. Since tier-1 providers are peers, RHs located at tier-1 providers are aware of all registrations in the entire network. Subscribers trying to locate information send out FIND message to its local RH, which also propagates this message to its parent according to its routing policies until a matching registration entry is found. After that, future requests follow the pointers created by the registrations in order to reach the publisher eventually. In a coupled option, the publisher on receiving FIND returns data on reverse path generated by FIND. In the decoupled option, the publisher on receiving FIND causes data to be directly sent to the subscriber using regular IP routing and forwarding. DONA supports on-path caching via RH infrastructure.

The *Named Data Networking* (NDN) [20] project funded by the US Future Internet Architecture program is further developing what PARC pioneered ICN as the *Content Centric Networking* (CCN) [32]. Names in NDN are hierarchical and may be similar to URLs used on the Internet today. Unlike Internet URL where the prefix identifies a DNS name, each name component can be anything, including a dotted human-readable string or a hash value. A request for named information is considered a match if any information has the requested name as a prefix. For example, a request for `/newsmedia.info/today/index.html` can be matched

to `/newsmedia.info/today/index.html/stream_1/segment_4`. All messages in NDN are forwarded hop-by-hop by *Content Routers* (CRs). Name resolution and data routing in NDN uses three data structures at CRs: *Forwarding Information Base* (FIB), the *Pending Interest Table* (PIT) and the *Content Store* (CS). The FIB maps information names to output interfaces that should be used to forward an INTEREST message towards the publisher. The PIT tracks incoming interfaces from which INTEREST messages arrive. The CS serves as a local storage cache for information objects that have passed through the CR. When an INTEREST arrives, the CR looks for the longest matching prefix in its CS. If a match is found, it is immediately sent back through the incoming interface in a DATA message. If a match is not found, the CR performs a longest prefix match on its FIB in order to decide the direction for this INTEREST message to be forwarded. On finding an entry in FIB, the CR pushes the INTEREST message to the corresponding interface and records the incoming interface in PIT. When the INTEREST message reaches the publisher node, the INTEREST message is discarded, and the information is returned in a DATA message hop-by-hop manner based on the state maintained in the PITs. A CR on receiving the DATA message first stores it in its CS, then looks up the longest prefix match in its PIT. On finding a match in PIT, the CR forwards the DATA message packet to interfaces and deletes the entry from the PIT. For cases where there is no entry in PIT, the CR drops the DATA packet. Name resolution and routing in NDN are coupled. The DATA messages follow the pointers left in the PITs by INTEREST messages, thus making routing symmetric. NDN natively supports on-path caching.

PURSUIT [27], [26] names object flatly by a unique pair of IDs, the *scope ID* and the *rendezvous ID*. The scope ID groups related information objects while the rendezvous is the actual identity for a piece of information. An information object

can belong to multiple scopes, but they must always belong at least one scope. PURSUIT architecture consists of three separate functions: *rendezvous, topology management* and *forwarding*. Name resolution is handled by rendezvous function by a collection of *Rendezvous Nodes* (RNs) and *Rendezvous Network* (RENE) implemented as a hierarchical DHT. PURSUIT uses Bloom filter to encode delivery paths into source routes. Name resolution and data routing are decoupled in PURSUIT, since name resolution is performed by RENE, while data routing is organized by the Topology Managers and executed by the Forwarding Nodes. PURSUIT supports both on-path and off-path caching.

SAIL [28], [29] architecturally combines elements present in the NDN and PURSUIT. The naming of information in SAIL is flat. Although they follow a defined pattern such as `ni://A/L` UTI scheme in which names consist of an authority part `A` and a local part `L`. Thus, SAIL names can be considered flat for comparison purposes at the router, and it can also be considered hierarchical when used for routing (similar to NDN). Name resolution and routing in SAIL can be done as coupled or decoupled or hybrid operation. In coupled operation, a routing protocol is used for advertising object names and populating the routing tables in CRs, as in NDN. A distinction from NDN is that instead of PIT, the GET messages accumulate routing directions along their path, which are simply reversed at the publisher or cache to reach the subscriber. In decoupled operation, a *Name Resolution System* (NRS) is used to map object names to locators that can be used to reach the corresponding information object, such as IP addresses. A subscriber sends a GET message to its local NRS, which consults the global NRS in order to return a locator for the object. Finally, the subscriber sends a GET message using the returned locator, and the publisher responds with DATA message.

COMET [30] allows for optimizing information source selection and distribution by mapping information to appropriate hosts or servers based on transmission requirements, user preferences, and network state [35]. The *Content Mediation Plane* (CMP) bridges information and infrastructure by mediating between network providers and information servers. COMET has two distinct designs for CMP: a coupled design called *Content-Ubiquitous Resolution and Delivery Infrastructure for Next Generation Services* (CURLING)[36], which is an ICN architecture with coupled name resolution and routing. A decoupled approach enhances information delivery without fundamentally changing the underlying Internet[37]. COMET does not have a defined naming nomenclature for contents; however, all contents are assigned names when registered with the *Content Resolution System* (CRS). CRS aggregates names for related information. In a coupled approach, a publisher sends REGISTER message to CRS. CRS assigns a name to the incoming content and records metadata, such as IP address. This information is propagated upstream in the AS hierarchy using PUBLISH messages so that each CRS ends up with a pointer to its child CRS that sent the PUBLISH message. PUBLISH message can be restricted by the publisher. Subscriber interested in content sends CONSUME message to its local CRS, which is similarly propagated upwards in the CRS hierarchy until it reaches a CRS that has information about that name. When a match is found, CONSUME message follows pointers in the CRSs to reach the actual publisher. A forwarding state is installed at each CRS as the CONSUME message travels to the publisher. The publisher uses these pointers to return data to the subscriber. COMETs decouple approach is very similar to DNS. The publisher sends REGISTER message to its local CRS. The local CRS assigns a name within its allocated namespace and prevents propagation of REGISTER further upstream. A subscriber sends CONSUME message to its local CRS, which sends

to the root CRS for resolving towards publisher's CRS. The subscriber uses *Path Configurator* (PC) to contact the publisher's CRS to retrieve the publisher's location. Subscriber's PC contacts publisher's PC requesting a source route from the subscriber to publisher. This source route is returned to the subscriber who uses it to request information. The publisher uses the reverse source route to return information requested by the subscriber. The decoupled approach has a dependency on DNS like architecture for CRS and PC - thus making names location-dependent.

CONVERGENCE [31] assigns a unique *Versatile Digital Item* (VDI) container to all kinds of digital information. It uses *Content Network* (CONET) [38] to allow publishers to publish VDIs and for subscribers to express interest in those VDIs. CONVERGENCE object names consist of a `namespace ID` and a `name` part, whose format is determined by the `namespace ID`. Naming can be flat, hierarchical, or URLs. The exact properties of the names depend on the namespace used. CONVERGENCE architecture has many similarities with NDN. Subscribers issue INTEREST messages requesting an information object, which are forwarded hop-by-hop by *Border Nodes* (BN) to publishers or *Internal Nodes* (IN) that provide caching. Publishers respond with DATA messages which follow the reverse path. Name resolution and routing is coupled; however, each step of the path taken by INTEREST and DATA message may not be a single hop but an entire IP path. Hence path in CONVERGENCE followed by INTEREST and DATA message are not necessarily symmetric. CONVERGENCE tries to reduce the state it maintains in BNs by reducing the number of name-based routing information it stores. This resembles a routing cache. For INTEREST message that cannot be forwarded, an external *Name Resolution System* (NRS) is consulted. As INTEREST message propagates, they accumulate the network address of the

BNs they pass, allowing the publisher to route the DATA message by reversing this path information. The name-based routing tables at BNs may be populated by routing protocol for name prefixes, e.g., OSPF, as in NDN.

2.4 Loop-free Scalable Routing in ICN

NDN is a widely adopted ICN architecture. The developers of this architecture [20][39][40] have argued that Interests can be forwarded correctly towards an intended node advertising a name prefix covering the content name, that routers can aggregate Interests so that a router can forward an Interest for the same content only once, and that Interest loops are detected whenever they occur. This claim has been researched and proven incorrect in SIFAH[41][42] and CCN-ELF[43]; it shows that Interest loops may go undetected when Interests from different consumers for the same content aggregates and Interests forwarded along routing loops, which may occur due to rankings of routes, failures, mobility, or congestion. SIFAH proved that there exists no correct Interest forwarding strategy that allows aggregation and detects Interest looping by identifying Interests uniquely. Loop-free routing is crucial in ICN. CORD [44] and ODRV [45] guarantees multiple loop-free routes to publisher directories by using sequence-numbered distances.

Considering the future predictions on the growth of content created and on the number of devices connected [1][2], the vast size of the naming space imaginable poses a significant scalability challenge for name resolution. Several results have reported regarding the large PIT sizes required for NDN to operate at Internet scale in benign scenarios[46][47][48][49][50][51]. This calls for alternatives where stateless routing without large routing table (like PIT and FIB in NDN) but with compact routing metadata can work decoupled from locating the source of the requested content. New approaches like DART[52][53], CCN-GRAM[54] and

CCN-RAMP[55] have been proposed as alternatives to PIT for Information-centric network.

As users of the Internet transition from an "always-connected" to an "information-sharing" society, addressing information retrieval for efficiency and scalability is critical. A name resolution system decoupled from data plane in a network scales better than coupled approach because: (1) The growth in content and an unbounded number of data names will result in uncontrollable growth of stateful table like FIB (2) Unlike name-based routing that does not guarantee discovery of content, name resolution system guarantees discovery in a limited number of hops [18] and (3) Name resolution routing performs optimization separately from the data plane. The architects of the widely used ICN architecture, NDN, validates the merits of decoupled approach as they propose SNAMP [56], an enhancement to scale the coupled NDN forwarding. SNAMP maps data names to set of globally routable names to retrieve data at times when routers do not know from where to retrieve data using application data names alone. Through SNAMP, NDN tries to make up for the deficiency in converging routing to a limited number of hops - a drawback of coupled name-based routing as highlighted in [18].

2.5 Compact & Prefix Routing

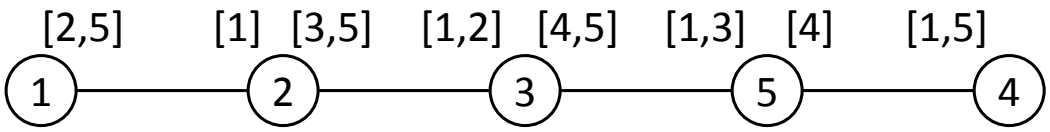

Figure 2.3: Interval Routing

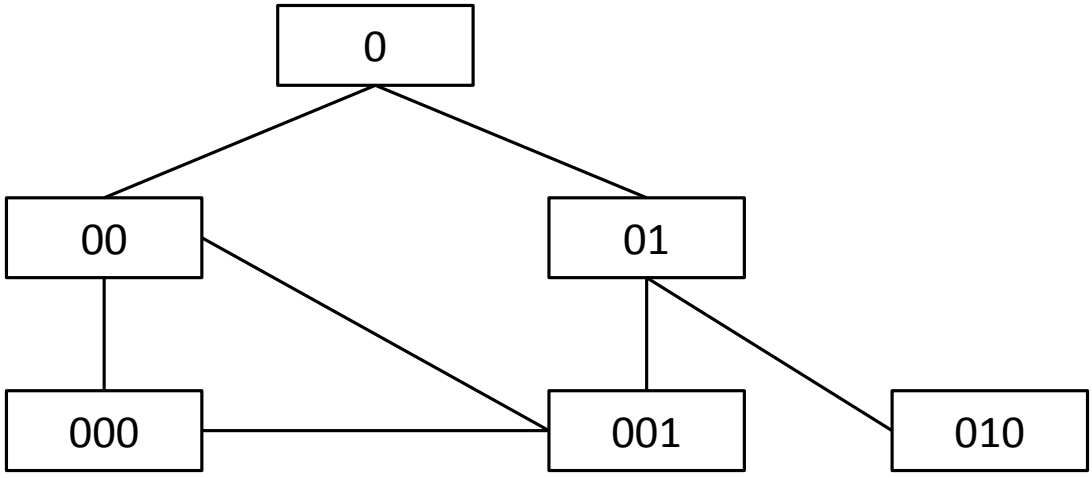

Figure 2.4: Prefix Label Routing

In the compact Interval Routing Scheme (IRS)[57], routing tables are designed to be minimal. IRS assigns each node a suitable address from the interval $[1...n]$. Each link is assigned a label which represents a unique interval from $[0...n]$. Fig. 2.3 is an example network with interval assignment from $[1, 5]$. A packet at node 2 destined for an address 5 in the interval selects the link with interval $[3, 5]$ to forward. The size of the routing table at a node is the number of links incident on that node.

Prefix Routing Scheme (PRS)[58] in a network assigns a label consisting of a string over some alphabet Σ, to each node. In PRS, each node has a routing table with a distinct label l_i consisting of a string of symbols. For a message at node u destined for v, where $u \neq v$, the routing table of labels $l_1, ..., l_d$ is searched for the longest string l_i such that l_i is prefix of v. The message is routed via the link that matched the longest prefix. Fig. 2.4 is an example network with nodes labeled with prefixes. A message at node 00 destined for 010 searches 00's routing table to discover node $(0 \bigcap 010) = 0$ as the longest prefix match. Node 00 forwards the message to node 0. At node 0, longest prefix match is $(01 \bigcap 010) = 01$, the message is forward to node 01. Node 01 forwards message to destination neighbor

010.

2.6 Information Naming & Lookup

Whether flat or hierarchical, an information name in its simplest form is a unit content or a collection of various kinds. For example, "soe.ucsc.edu/darci/paper" is a unit content whereas "soe.ucsc.edu/darci/media" is a collection of contents like "soe.ucsc.edu/darci/images", "soe.ucsc.edu/dari/videos" and "soe.ucsc.edu/darci/audio". Rather than considering each information name separately, there is an implicit understanding of the chances that two or more names are spacially closer to each other. Prior proposals like Chord[59], CAN[60], Pastry[61], and Tapestry[62] that mapped the content name to publishers used Distributed Hash Table (DHT). DHT is the building block for scalable Peer-to-peer (P2P) systems were given a key, an efficient lookup to locate the peer node storing the key can be done. While DHT has benefits of being distributed, defines simple primitives, is scalable and self-organizing, it has drawbacks that make it unsuitable for use in optimizing name resolution systems for ICN. They are ill-equipped in supporting range queries. This is primarily due to loss of data locality. It is destroyed by uniform hashing used in DHTs. The hashing distributes keys uniformly and hence cannot rely on any structural properties of the keyspace, such as an ordering among keys[63][64]. DHT's are mostly implemented with a host-centric networking mindset. Because DHTs perform a lookup of keys with lost locality, content retrieval mechanism delivers specific data and not a range of data. In a DHT overlay network, each node in the DHT keeps track of resources that fall within its key range. This solution has load-balancing problems. A large number of resources may have the same key value, which could lead to overburdening some nodes in DHT. Another issue with DHT systems is that the path on the overlay network between two nodes can be

significantly different from the unicast path between those two nodes on the underlying network. As a result, lookup and routing latency in such overlay systems can be quite high and can adversely impact the performance of applications built on top of such systems for Information-centric networks.

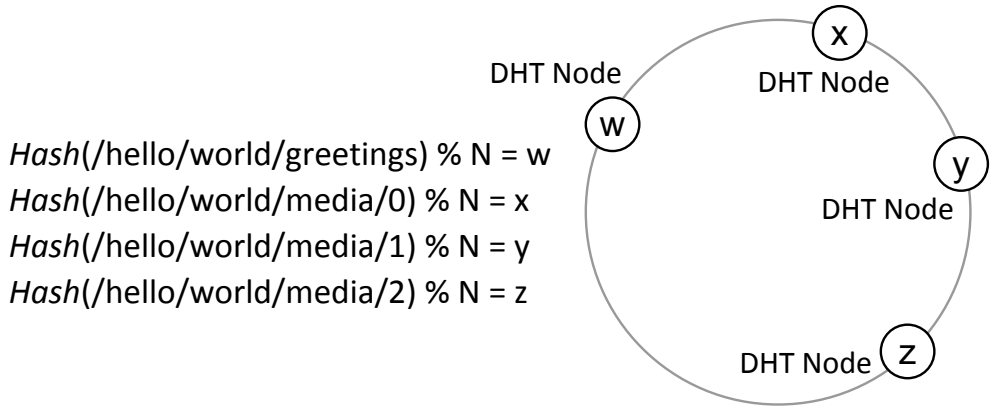

Figure 2.5: Example content name mapping in DHT with consistent hashing

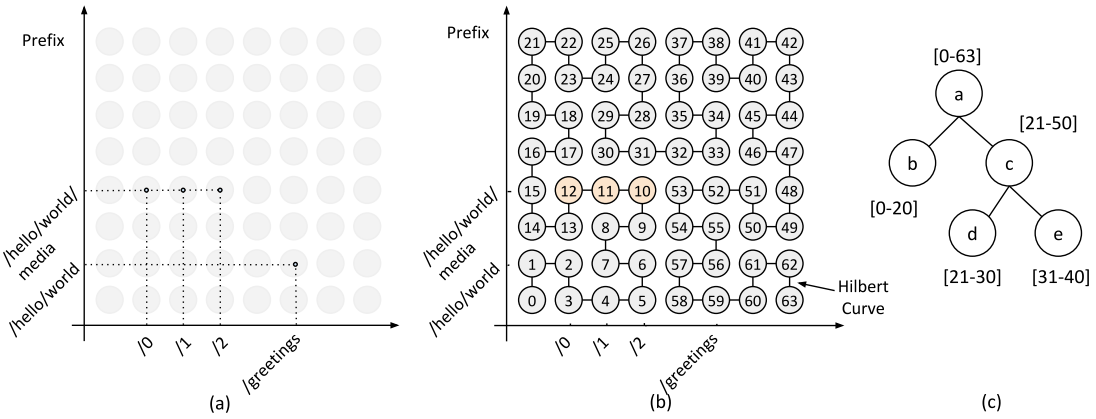

Figure 2.6: Example content name mapping in Hilbert Curve

In ICN, contents are uniquely addressable and routable. Names in NDN, unlike fixed-length IP addresses, have variable length without a limit on the upper bound. Without a fixed length, the name lookup on a variable-length string cannot achieve wire-speed. The cost of lookup time is linearly proportional to the variable-length content name, rather than a deterministic time using hashing techniques. The

Longest Prefix Matching (LPM) algorithms used in a host-centric network with fixed length numeric IP address at wire speed are rendered inefficient for variable-length content name strings. In NDN, the PIT size estimate [49] is given by $\lambda * T$, where λ is the wire speed, and T is the time each entry exists in the PIT. For $\lambda = 40$ Gbps and 500 ms $\leq$ T $\leq$ 1 second, a worst-case scenario where average Internet latency is substantially greater than 80 ms can result in PIT containing between 30 and 60 million entries. PIT is subject to 3 basic operations: *Insert*, *Update*, and *Delete*. These operations on an already large PIT exponentially adds to the delay in lookup at wire speed.

We consider an encoding function that is locality preserving and one that produces a code of fixed length integer. Space-filling curve (SFC) such as Hilbert Curve transforms a 2-D input of information name and its metadata into a 1-D code preserving locality, i.e., two neighboring points in one dimension are neighbors in the other dimension after folding. SFC supports a direct lookup (a DHT functionality) and a range query. It is possible to achieve wire-speed with the discrete approximation of the Hilbert Curve implemented as iterations rather than recursion. Lookup based on a fixed-length integer is memory-efficient as these codes are fetched in one call to the DRAM. A fixed-length integer code allows for cheaper routers with less power consumption.

Fig.2.5 is a simplified example of DHT nodes arranged in a ring. Consistent hashing for content and sub-resources in Fig. 2.7 are mapped randomly to distinct nodes. DHT's hash partitioning loses spatial locality of the keys and hence is inefficient for answering *ad hoc* range queries, e.g., prefetching all contents with a given prefix. Fig.2.6(a) is an example transform that maps name prefixes of content and sub-resources in Fig.2.7 to identifiers in a 2-D plane. Hilbert Curve in Fig.2.6(b) transforms the 2-D code space to 1-D code space. Fig.2.6(c) is a trie

with Hilbert Curve code intervals. Sub-resources that are neighbors and mapped to code 10-12 is assigned to node b.

2.7 Content Retrieval

/hello/world/media/greetings

```
<html>
    <head>
        <title>Hello World !!</title>
    </head>
    <body>
        A view of the world !!
        <img src="/hello/world/media/0">
        The sound of the world !!
        <audio controls>
            <source src="/hello/world/media/1" type="audio/mpeg">
        </audio>

        A movie of the world !!
        <video width="320" height="240" controls>
            <source src="/hello/world/media/2" type="video/mp4">
        </video>
    </body>
</html>
```

Figure 2.7: Example HTML content with sub-resources

Fig. 2.7 is an example content named /hello/world/media/greetings. To render this content for a user, sub-resources /hello/world/media/0, /hello/world/-media/1 and /hello/world/media/2 has to be retrieved. This is an example of content with chained dependency on other contents in order for it to be utilized.

Fig. 2.8 is a simplified example of content retrieval in the current Internet. A subscriber first resolves the name for the content by issuing a DNS query for /hello/world/media/greetings. On receiving the hosting server's address, the subscriber issues a GET request to fetch the content for /hello/world/media/greetings. On receiving the content (content data in Fig. 2.7), the subscriber decodes

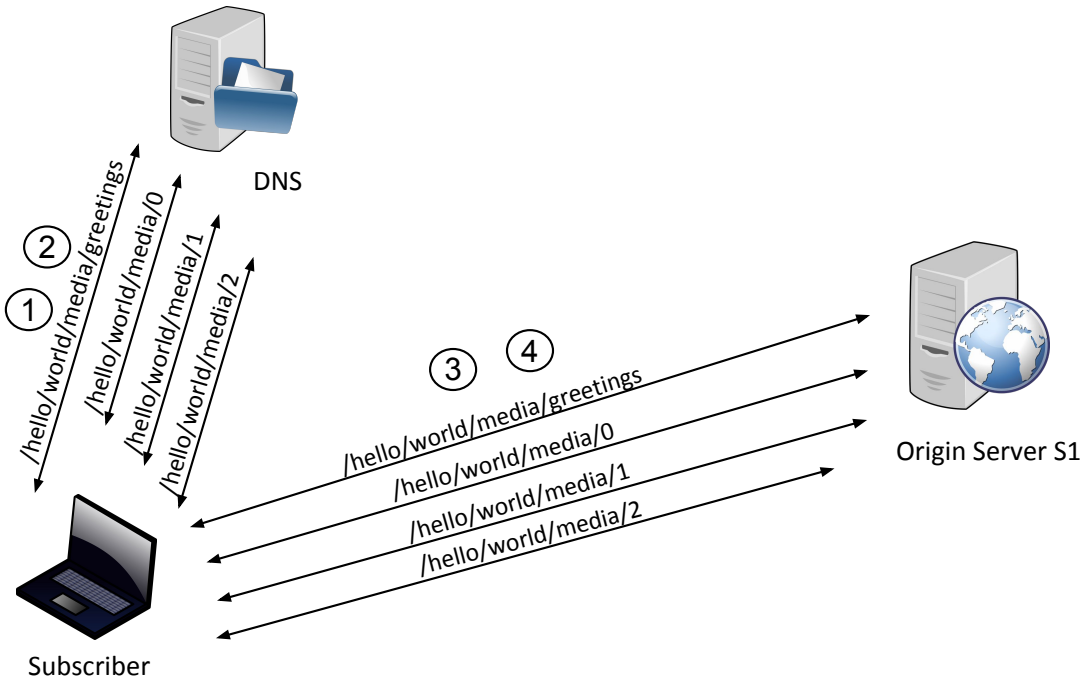

Figure 2.8: Content retrieval in Internet

the data to create a list of sub-resources required to render the content for the user. For each sub-resource, the subscriber issues a separate DNS query. On receiving the host address for the hosting server, the subscriber issues separate requests to fetch each sub-resource content. Typically, the subscriber has to wait till it receives all sub-resources before it can compile and render the content for the user. In this example, it took eight requests for all contents to be received.

Fig. 2.9 is a simplified example that uses DHT for name resolution. A subscriber first resolves the name for the content by issuing GET for /hello/world/-media/greetings towards H1 in the DHT overlay. H5 responds to H1 with the value containing the address of the hosting server. H1 replies to the subscriber with the address to the server. On receiving the address for the hosting server, the subscriber issues a GET request to fetch the content for /hello/world/media/greetings. On receiving the content (content data in Fig. 2.7), the subscriber

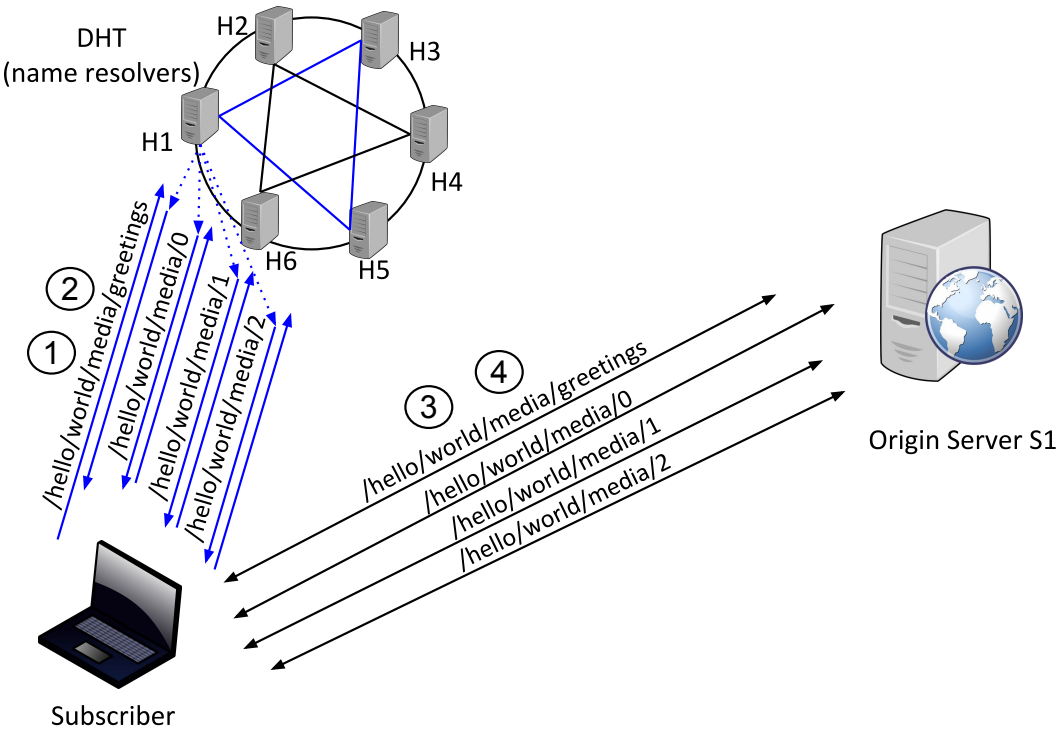

Figure 2.9: Content retrieval in a DHT network

decodes the data to create a list of sub-resources that are required to render the content for the user. For each sub-resource, the subscriber issues a GET query. On receiving the host address for the hosting server, the subscriber issues separate requests to fetch each of the sub-resource content. Typically, the subscriber has to wait till it receives all sub-resources before it can compile and render the content for the user. In this example, it took eight requests for all contents to be received.

Fig. 2.10 is a simplified example that uses NDN for content retrieval. The subscriber sends out data requests for named content /hello/world/media/greetings towards connected router R1. Assuming all nodes in NDN have learned about neighborhood prefixes, the content request is forwarded by FIB to reach S1. S1 replies with the content in the reverse path. On receiving the content (content data in Fig. 2.7), the subscriber decodes the data to create a list of sub-resources

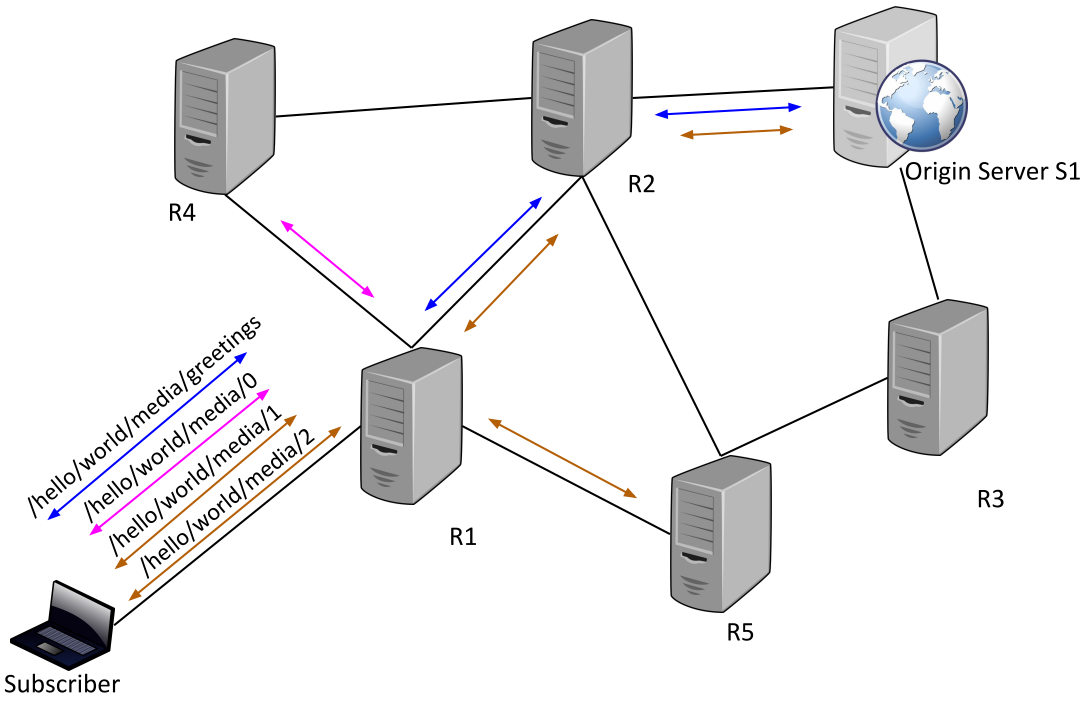

Figure 2.10: Content retrieval in NDN

that are required to render the content for the user. For each sub-resource, the subscriber sends a data request to router R1. At various routers in NDN, prior knowledge of prefix in the FIB is used to route the request to server S1. An intermediate router R4 or R5 with a locally saved cached copy of the requested content replies via the reverse path. Typically, the subscriber has to wait till it receives all sub-resources before it can compile and render the content for the user. In this example, it took four requests for all contents to be received.

Chapter 3

DARCI: Directory Assisted Routing of Content in ICN

As a design choice, DARCI decouples the name resolution system from the data plane. A name resolution system in a network guarantees the discovery of content within a bounded number of hops[18]. The building blocks of DARCI's name resolution system are (1) a trie of specialized routers called directory (2) a globally known linearly ordered code space and a locality preserving encoding function to encode content names and information to content publishers (3) a global labeling scheme for all connected directory routers in a trie (4) an interval assignment scheme using locality preserving function such as Hilbert Curve for all connected directory routers in a trie. The routing of content name resolution query and response is implicit. Labels and intervals of the directory router decide on forwarding to the next neighbor. The data plane uses information received in content name resolution to route content request and response hop by hop towards the directory router nearest to the destination.

Fig. 3.1 is a simplified example that uses DARCI for content retrieval using a content name range within a locality. The subscriber sends a LOOKUP

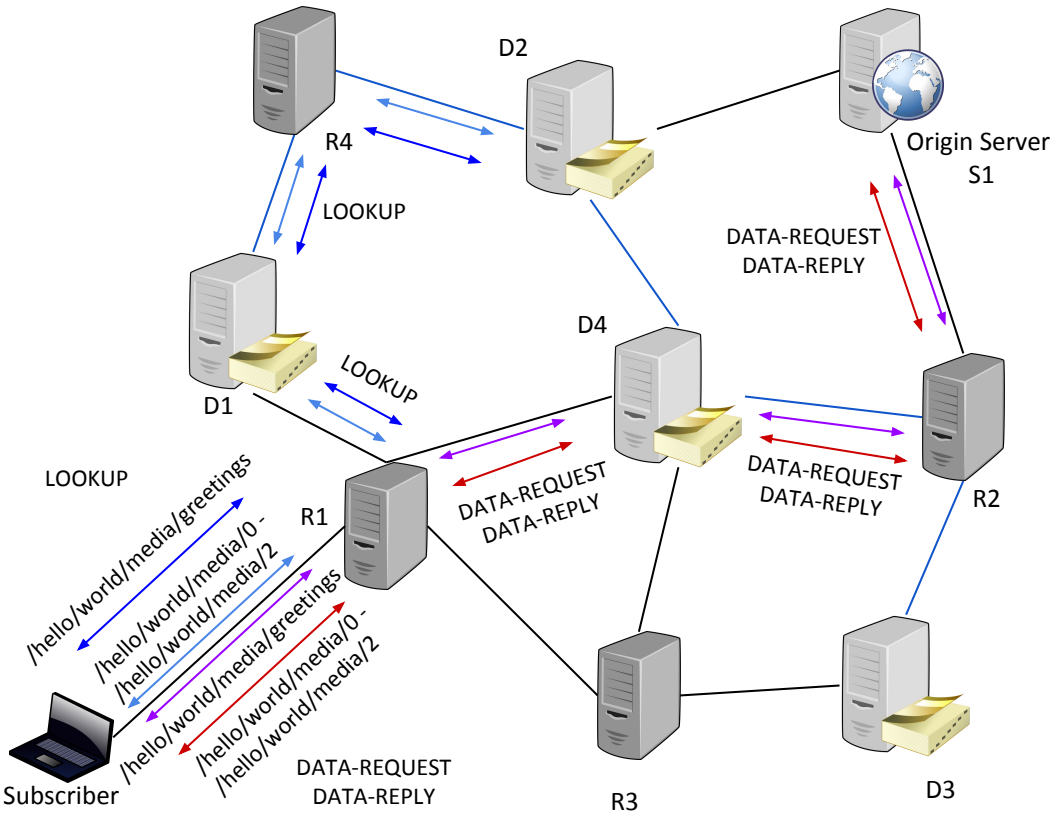

Figure 3.1: Content retrieval in DARCI

message for /hello/world/media/greetings to the connected router R1 towards subscriber's anchor directory D1. Directory D2 replies with the address of the hosting server. The subscriber sends DATA-REQUEST for /hello/world/media/-greetings towards S1's anchor directory D4. On receiving the content, as shown in Fig. 2.7, the subscriber decodes the data to create a list of sub-resources. These sub-resources are required to render the content for the user. For each sub-resource, subscriber resolves names with LOOKUP message containing name range /hello/world/media/0 to /hello/world/media/2. Directory D2 replies with all known mapping within the code range for /hello/world/media/0 to /hello/-world/media/2. Subscriber on receiving the mapping of content name to location

for all sub-resources issues DATA-REQUEST for the content name ranges /hello/world/media/0 to /hello/world/media/2 towards the anchor directory D4 of S1. S1 replies with the requested range of contents (in an application-specific delimited format) towards the anchor directory D4 of the subscriber. In DARCI, for this example, it took four requests for all contents to be received. However, as the number of sub-resources that share the same prefix increases, DARCI scales better than NDN due to the capability of locality preserving SFC such as Hilbert Curve allowing compressing of name ranges within a locality.

3.1 Basic Operation

DARCI assumes that (1) all routers and hosts in the network have a position independent globally unique identifier, which can be flat or hierarchical (2) parent directory router in the trie assigns prefix label to its child directories (3) all directory routers use a global locality preserving encoding function $\mathscr{F}$ and an interval of code space $[\alpha, \omega]$ where $\alpha < \omega$ (4) named objects NO are uniquely addressable with names that are flat or hierarchical (5) routers cache name resolution responses and contents opportunistically.

Fig. 3.2 illustrates how DARCI operates. DARCI first builds a content name resolution system. The name resolution system in DARCI is a trie of specialized routers called directory router. Publishers publish content to the network by registering it with the trie. Subscribers learn about the available content in a network by looking up the content name in the trie. In the simplified example, a, c and g are directories in a trie with a as the root. a is a parent directory to directories c and g. Directories maintain an authoritative interval of code that is either global if it is a root or a sub-interval from within a parent's interval. In the simplified example, a being the root is authoritative for a global interval

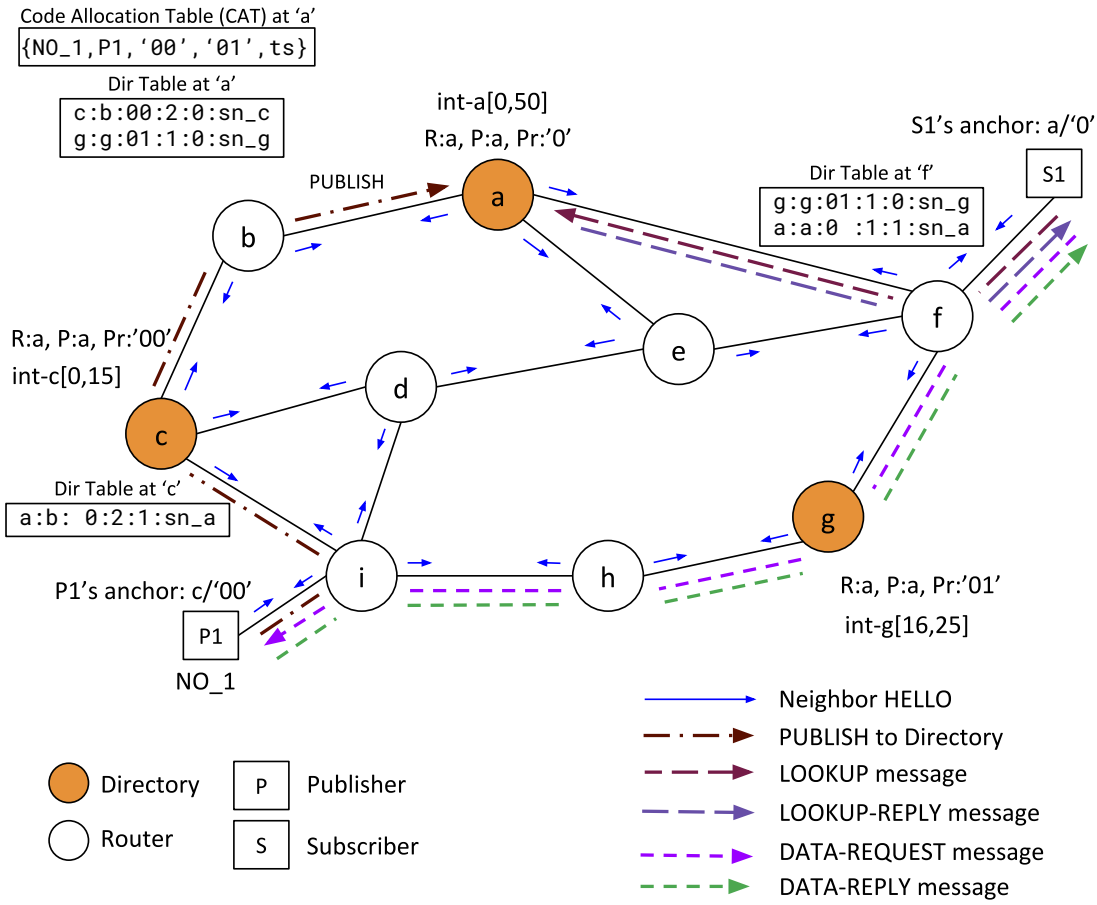

Figure 3.2: Basic operation of DARCI

[0,50]. a allocates sub-intervals [0,15] and [16,25] to directories c and g respectively. Directories use a globally known locality preserving encoding function like Hilbert Curve to encode content name to one of the codes in the global interval. A consistent global prefix labeling scheme labels the directories in a trie. The root directory a begins with prefix "0". It assigns labels "00" and "01" to its child directories c and g.

All routers send sequence numbered HELLO messages to its 1-hop neighbors. Routers and directories store their neighborhood information and sequence numbers in neighbor table NT. All routers maintain a sequence-numbered sn distance to directories within r hops. Only a directory can update the sequence num-

ber in its following updates. Such a route is loop-free, proved in CORD[44] and ODVR[45]. Routers and directories store sequence numbers in their persistent storage, sequencing increments with every update. On a reset or routing state initialization, $sn = 0$ and distance to directory is ∞. Routers and directories store shortest distance to neighbor directories in diretory table DT. HELLO messages periodically send all updates to a router's neighbor table to neighbors.

Publisher P1 publishes content NO_1 with code $\mathscr{F}(NO\_1) = 40$ to its anchor directory c. Anchor c forwards publish message towards authoritative directory a with interval $[0, 50]$ using interval routing. A CAT^a entry consists of the content name prefix NO_1, local publisher ID P1, prefix label of the publisher's anchor (00) and landmark directory (01), and a timestamp (ts). Subscriber S1 requesting content NO_1 sends a name resolution LOOKUP message to anchor directory a. Anchor directory a replies to S1 with the mapping of content name to the local ID of the publisher and prefixes of its anchor and landmark directories.

Content routing in the data plane is assisted by the name resolution system of DARCI and the publish-subscribe signaling between routers and directories in a trie. Subscriber S1 derives the direction for the shortest path by comparing the known prefix labels for content NO_1 with the prefix labels in its DT. S1 sends a DATA-REQUEST with the content name, prefix labels of its anchor, and landmark directories towards publisher P1 via directory g with the prefix "01". Publisher P1 replies with DATA-REPLY with requested content towards subscriber S1. P1 derives the direction for the shortest path to S1 by sending DATA-REPLY in the direction of a directory with the longest prefix match. In the example, P1 sends DATA-REPLY towards g. Local router f delivers the content to subscriber S1.

The operation of DARCI in the control plane is independent of the data

plane after a router learns of the prefix labels of anchor and landmark directories. Routers forward data requests and reply messages in the path of the longest prefix match to reach the destination without the need to maintain long stateful tables.

3.2 Name Resolution System

DARCI first builds a name resolution system to store mapping of content names to its publishers in the network. Routers exchange sequence numbered HELLO messages with its immediate neighbors. All routers maintain loop-free routes to neighboring directory routers within r hops away using sequence numbered distances. Neighboring directories connect to form a trie with a root that is elected by a globally known policy. Forwarding chooses a neighbor router with the shortest distance to directory router on the trie or to a directory router with the most recent sequence number reported. Fig. 3.3 is an example network with a, i, o, t and u as directory routers. HELLO messages between neighboring routers send the distances to all known neighbor directory routers. Directory routers on discovering neighboring directories select its parent by using a globally known policy. For directory routers t & u without a parent directory, t is a parent to u if lexicographically $|t| < |u|$ within r hops of u, and if u learns of t before knowledge of any other lexicographically smaller directory router. A directory router in a trie has one or all of the following roles:

As an *anchor* directory ($\mathcal{A}$) to which subscribers and publishers register their presence.

As an *authoritative* directory ($\mathcal{U}$) that owns a contiguous range of codes, this range is delegated to $\mathcal{U}$ by its parent directory.

As a *landmark* directory ($\mathcal{L}$) that a publisher or subscribe learns through neighbor *HELLO* messages.

3.2.1 Information Stored & Exchanged

A router directory a maintains the following tables: (a) *a neighbor table* $[NT]^a$ listing router a's immediate neighbors; (b) a *directory table* $[DT]^a$ to store routing information to known directory routers as learned from neighbor HELLO; (c) a *child directory table* $[CDT]^a$ to store child directories of a (d) a *named object routing information* $[NORI]^a$ table to store named object NO routing information for each known named object seen by router a; (e) a *cached named object pointer* $[CANOP]^a$ at router a to store the mapping of NO names to the pointer to the cached data store; (f) a *code allocation table* $[CAT]^a$ to store space-filling curve code allocation for registered NO at router a.

For router a, the entry in $[NT]^a$ has neighbor ID and sequence number sn. The information stored in $[DT]^a$ for each discovered directory router i consists of router ID $|i|$ for router i, next hop neighbor $b \in [NT]^a$ that has shortest path to directory router i, the prefix label $\langle i \rangle$ for router i, distance to directory router i via b, a boolean field to store root status of the directory and sequence number sn_i. The tuples in $[CDT]^a$ consists of columns for an ordinal, child directory router ID, code interval allocated, and last updated timestamp. The entry in $[NORI]^a$ stores routing information for each known named object that passes through router directory a. Each row specifies the prefix of named object $NO^\star$, directional vector, a boolean showing whether the prefix $NO^\star$ for named object is cached in a's $[CANOP]^a$ and timestamp t_a. Directional vector for each $NO^\star$ consists of publisher ID, prefix label of publisher's anchor directory, set of prefix labels of publisher's landmark directories, and a timestamp. $[NORI]^a$ entries are

purged after a pre-configured time to live (TTL). Routers opportunistically cache contents in the network. The $[CANOP]^a$ database keeps track of named object mapping to its corresponding storage pointer. A $[CANOP]^a$ entry consists of named object prefix $NO^\star$, a pointer to content storage PTR and a timestamp t_a. A directory router a maintains its authoritative range of code in code allocation table $[CAT]^a$. In a given code space, directory router a is assigned a range by its parent. Directory a is the authoritative directory for named objects that maps to one of the codes in the range. The two scenarios where $[CAT]^a$ stores mappings for named objects that map to codes outside of the assigned authoritative range are: (1) when directory a is the local directory to a nearby publisher and (2) when a decides to cache a copy of a named object and become an authoritative cached copy for that named content. In $[CAT]^a$, a code points to Named Object Profile Set (NOPS). A NOPS for a given named object $NO^\star$ published by host H is a set of one or more Named Object Profile (NOP), which consists of the following tuples:

$$(NOP^{NO^\star})_a = (H, \mathcal{A}^H, \mathcal{L}^H_{[0...\gamma]}, sn_H, t_a)_{[0...\lambda]}$$

Where $\mathcal{A}^H$ is the anchor directory for publisher node H, $\mathcal{L}^H_{[0...\gamma]}$ is a list of γ landmark directory $\mathcal{L}$ prefixes known by publisher node H at the time of registering and publishing with its anchor directory $\mathcal{A}^H$, sn_H is a sequence number generated by publisher node H and t_a is the timestamp generated for this record by directory a. For a given named object, up to λ Named Object Profiles can exist, where λ is bounded by storage resource at directory router a.

Non-directory routers are an integral part of directory trie. They facilitate efficient and robust routing of name resolution requests and contents by using knowledge of surrounding directory routers. A router b maintains (a) *a neighbor*

table $[NT]^b$ listing router b's immediate neighbors; (b) a *directory table* $[DT]^b$ storing routing information to known directory routers as learned from neighbor HELLO; (c) a *named object routing information* $[NORI]^b$ table to store named object NO routing information for each known named object seen by router a; (d) a *cached named object pointer* $[CANOP]^b$ at router b to store the mapping of NO names to a pointer to the cached data store.

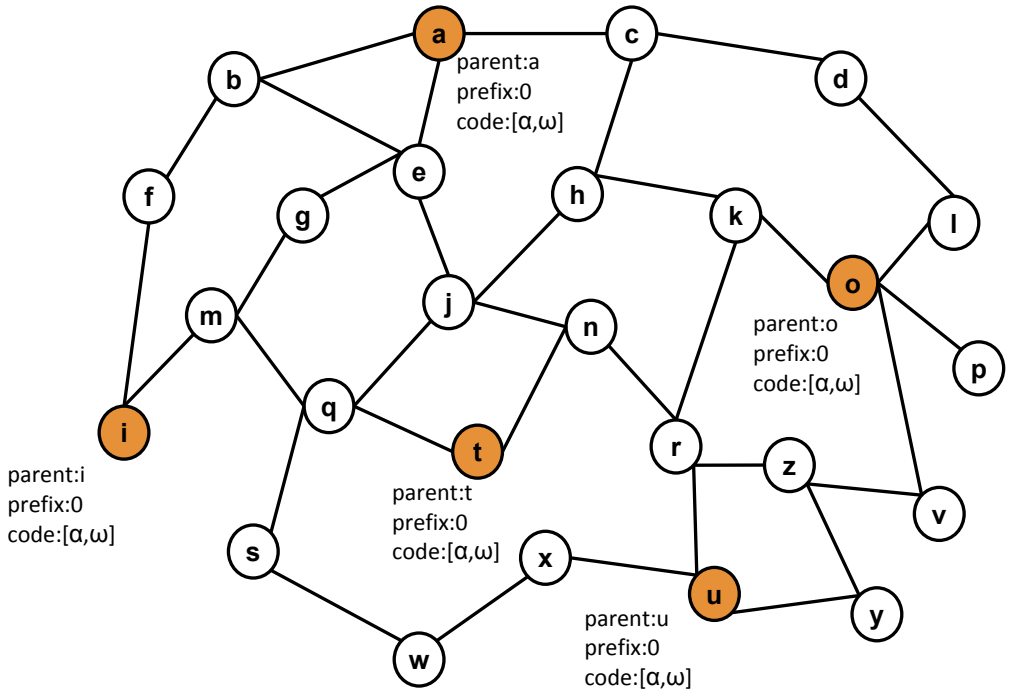

Figure 3.3: Directory Router Initialization in DARCI

All directory router initializes by assigning itself as its parent and root, a root prefix label "0" and assigns the global code space $[\alpha, \omega]$ where $\alpha < \omega$. An example initialization for directory routers a, i, o, t and u is shown in Fig. 3.3 . Every router in the network periodically sends sequence numbered HELLO to its neighbor. For directory router a the HELLO message contains the router ID a, ID of root directory a, parent of a as a, label prefix "0" and sequence number sn_a.

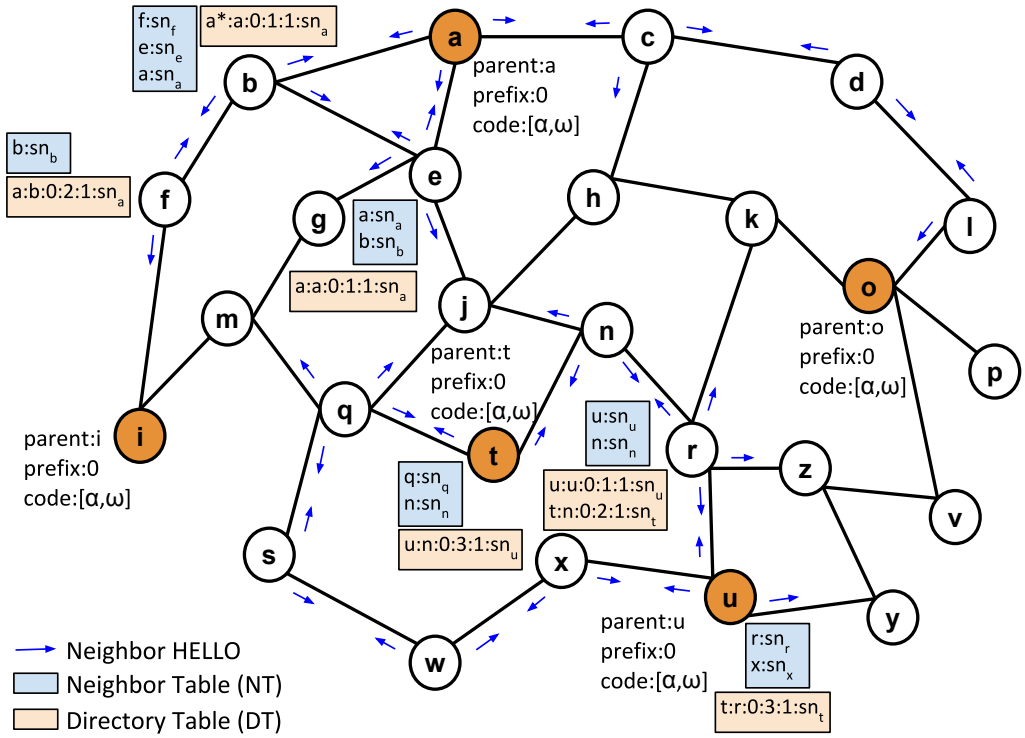

Figure 3.4: Start of HELLO messages in DARCI

For router b the HELLO message contains the router ID b, ID of root directory known, list of all known directory routers from its DT, and sequence number sn_b. An example instant showing the start of neighbor HELLO in some part of the network is shown in Fig. 3.4. Routers populate their NT and DT as they receive HELLO from their neighbors. Every router builds the shortest sequence numbered loop-free route to all known neighbor directory routers. In Fig. 3.4, on receiving HELLO from directory router a, b saves a tuple `{a:a:"0":1:1:`sn_a`}` in its DT, where the columns of this tuple are directory ID, next hop, prefix label, distance, root status of a and the latest known sequence number for the directory respectively. Router b enters a tuple `{a:`sn_a`}` in its NT, where the columns of this tuple are neighbor router ID and the latest known sequence number from that neighbor. Router e discovers two ways to reach directory router a: (1) via router

b with a distance of 2 hops and (2) directly to a with a distance of 1 hop. For a given sequence number, e selects the shortest path towards a. Router e saves a tuple {a:a:"0":1:1:sn_a} in its DT. Directory router t receives HELLO from neighbors n and q. Router t discovers two ways to reach directory router u: (1) via neighbor router n with a distance of 3 hops to u and (2) via neighbor router q with a distance of 5 hops to u. For a given sequence number, t selects the shortest distance offered by neighbor router n as the next hop towards u, saves a tuple {u:n:"0":3:1:sn_u} in its DT. Algorithm 1 describes the processing of HELLO messages at directory routers. Lines 27 to 34 of the algorithm are skipped for routers that are not a directory.

A router b updates its DT^b for the next hop towards a directory if it receives a HELLO message with the latest sequence number but with shorter distance than what it has currently stored in DT^b. A link failure to next hop towards a directory triggers an update for the next hop. The router deletes all entries for the failed next hop router, discovers from HELLO messages the next router with the most recent sequence number that offers the shortest distance to a directory. In Fig. 3.4, a link failure to a from b deletes entries to all directories in DT^b with a as the next hop. Router b discovers from e via HELLO a route to directory a with the most recent sequence number. It adds an entry in DT^b towards directory a and with e as the next hop.

3.2.2 Directory Trie

DARCI uses a trie of directory routers to partition global code space. Unlike a tree that partitions data set, nodes in trie represent a particular set of keys. Because a trie uses space, which is a constant independent of the actual data set, there is some implicit knowledge about the location of a key. DARCI routes name

Algorithm 1 DARCI: Process `HELLO` message at router u

```
1: input: NT^u, DT^u, NORI^u, CANOP^u, message;
2: procedure ProcessHello
3:     nbr[..] ← getAllNeighborsFrom(message);
4:     dir[..] ← getAllDirectoriesFrom(message);
5:     for all n ∈ nbr[..] do                                   ▷ Update NT
6:         if n ∈ NT^u AND n.seqNum > FindInNT(n).seqNum then
7:             removeNeighborFromNT(n);
8:         end if
9:         addNeighborToNT(n, n.seqNum);
10:    end for
11:    for all d ∈ dir[..] do                                   ▷ Update DT
12:        entry ← findDirectoryInDT(d);
13:        if entry ≠ ∅ then
14:            if entry.seqNum = d.seqNum then
15:                if entry.distance > d.distance then
16:                    removeDirectoryFromDT(d);
17:                    addDirectoryToDT(d);
18:                end if
19:            else if entry.seqNum < d.seqNum then
20:                removeDirectoryFromDT(d);
21:                addDirectoryToDT(d);
22:            end if
23:        else
24:            addDirectoryToDT(d);
25:        end if
26:    end for
27:    if self = DIRECTORY then                    ▷ For directory router only
28:        if self.ID ≡ myParent().ID() then                    ▷ Check for parent
29:            parentID ← getLexicallySmallestAndClosestDirectoryFromDT();
30:            if parentID ≠ ∅ then
31:                sendEnrollDirectory(parentID);
32:            end if
33:        end if
34:    end if
35: end procedure
```

resolution packets using the location of partition or intervals and prefix label of the directory router. The hop by hop routing eliminates stateful routing table lookups. In Fig. 3.4, directory router u discovers neighboring directory t. Directory u

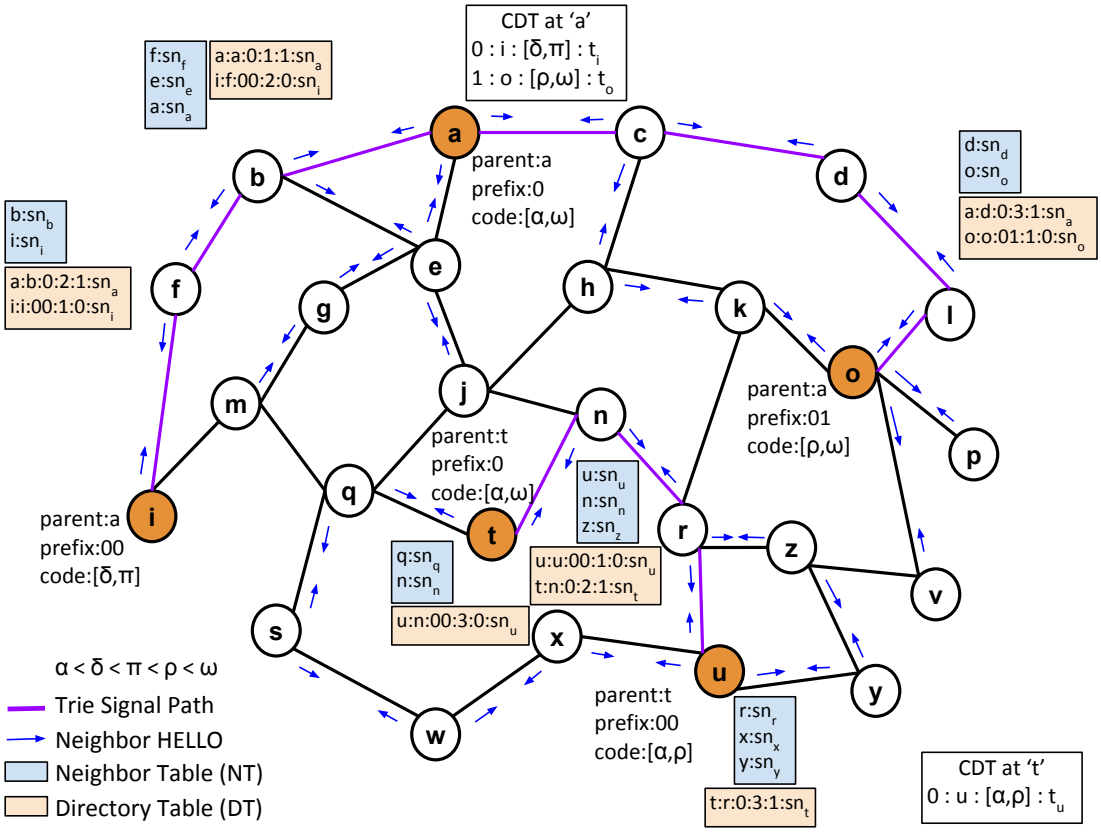

Figure 3.5: Directory router discovery and initial trie formation in DARCI

initiates an ENROLL-DIRECTORY message to notify t an intent to be its child directory. r forwards ENROLL-DIRECTORY message towards t using loop-free routing. On receiving ENROLL-DIRECTORY, directory t updates $[CDT]^t$ with the ID of the child directory u, assigns a subset of code interval from its interval and inserts a timestamp. Directory t replies to u with ENROLL-DIRECTORY-ACK. The ENROLL-DIRECTORY-ACK message contains a prefix label and a code interval for u.This acknowledgment travels a loop-free route to u via n. Fig. 3.5 shows prefix label and interval assignments by directory t to u. Algorithm 2 and 3 describes the processing of directory enrollment and acknowledgment messages at directory routers t and u respectively.

In Fig.3.5 more HELLO messages from routers reach various routers. Di-

Algorithm 2 DARCI: Process `ENROLL-DIRECTORY` at directory router t

```
1: input: NT^t, DT^t, CDT^t, NORI^t, CANOP^t, CAT^t, message;
2: procedure ProcessEnrollDirectory
3:     x ← getDirectoryIDFrom(message);
4:     p ← assignPrefix(x);
5:     [s_min : s_max] ← assignCodeAllocation(x);
6:     sendEnrollDirectoryAck();
7:     updateCDT();
8:     updateCAT();
9: end procedure
```

Algorithm 3 DARCI: Process `ENROLL-DIRECTORY-ACK` at directory router u

```
1: input: NT^u, DT^u, NORI^u, CANOP^u, CAT^u, message;
2: procedure ProcessEnrollDirectoryAck
3:     x ← GetDirectoryIDFrom(message);
4:     myPrefix ← GetPrefixFrom(message);
5:     [s_min : s_max] ← GetCodeAllocationFrom(message);
6:     updateCAT();                          ▷ Update code allocation table
7:     for all childDirectories ∈ CDT^u do   ▷ Update child directories if any
8:         reallocateCodeSpace();
9:         reassignPrefixLabels();
10:        sendUpdateDirectory();
11:    end for
12: end procedure
```

rectories a, i and o store distance towards root directory. Directory routers in their default initialized state where it is the root and the parent periodically scans its DT to find a parent directory. Directory router o selects directory a as the lexicographically smallest and closest parent directory and sends ENROLL-DIRECTORY towards a via neighbor i. The CDT^a for directory a contains entry for directory router i and o at ordinal 0 and 1 respectively. Directory a assigns prefix label to its child directory i using the following transform:

$$\langle i \rangle = \mathscr{T}(\langle a \rangle \oplus \mathscr{S}(i))$$

where $\langle i \rangle$ is the label for directory i, $\langle a \rangle$ is the label for directory a, $\mathscr{S}$ is a

transform function to convert ordinal of entry for directory i in CDT^a to a label and $\oplus$ is a label concatenation operator. In Fig. 3.5, directory router a assigns label "00" and "01" to directories i and o respectively. Each child directory in CDT^a is assigned a non-overlapping interval of code by directory a satisfying the following condition: for any two interval $[\delta, \pi]$ and $[\rho, \psi]$ out of $[\alpha, \omega]$

$$\{[\delta, \pi] \cap [\rho, \psi] = \emptyset \mid \delta < \pi, \rho < \psi, [\delta, \pi] \subseteq [\alpha, \omega] \text{ and } [\rho, \psi] \subseteq [\alpha, \omega]\}$$

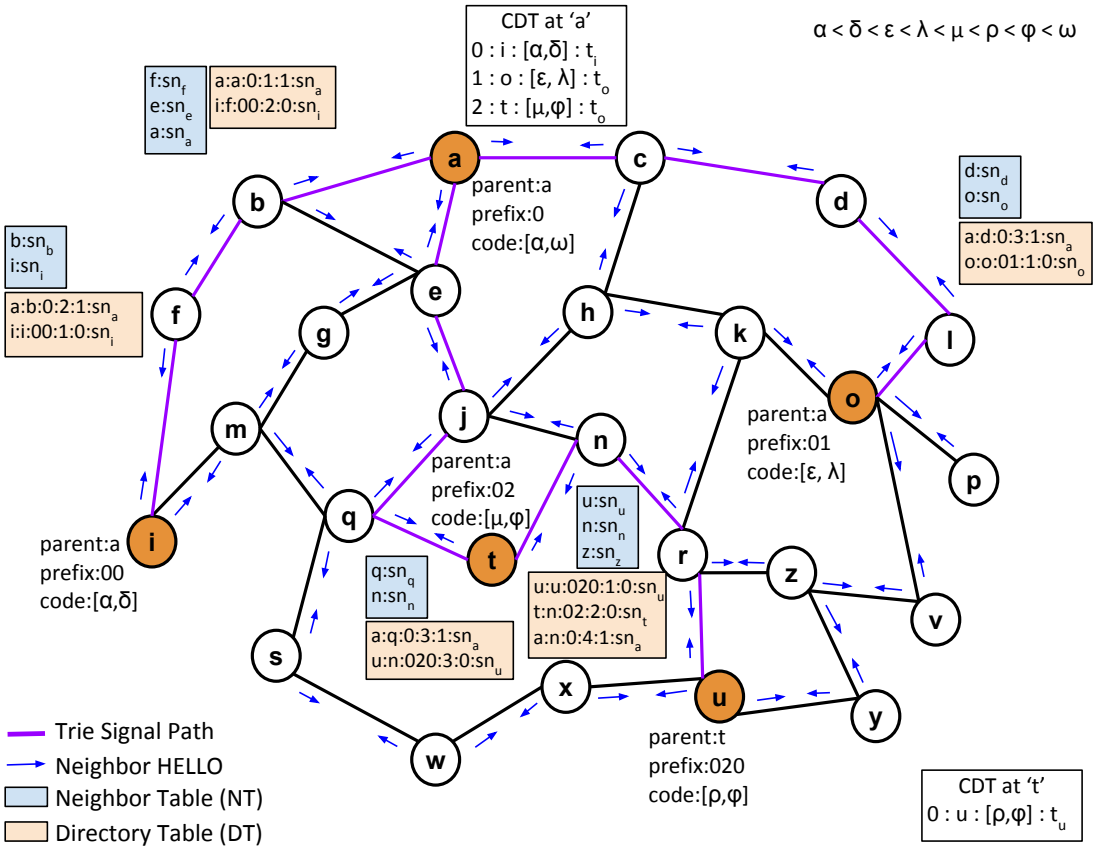

Figure 3.6: Stabilized trie in DARCI with root directory a

In Fig. 3.6, directory router t learns about directory i and a new root a. It also learns from neighbor router q a path to root a within r hops. Directory router t sends ENROLL-DIRECTORY towards a via neighbor q. Root directory

a allocates interval for t, generates a prefix label, and creates a new tuple in CDT^a. Root a replies to t with sequence numbered ENROLL-DIRECTORY-ACK message via router e. Router t updates its root and parent directory, its label prefix, and code interval. All children of directory t is reassigned new label prefix and interval. Updated child directory information is sent to all children in sequence numbered UPDATE-DIRECTORY message. In Fig. 3.6, u receives a new prefix label of "020" from t. Algorithm 6 describes processing of UPDATE-DIRECTORY message at directory u.

Algorithm 4 DARCI: Process `UPDATE-DIRECTORY` message at directory router u

```
1: input: NT^u, DT^u, NORI^u, CANOP^u, CAT^i, message;
2: procedure ProcessUpdateDirectory
3:     x ← GetDirectoryIDFrom(message);
4:     [s_min : s_max] ← GetCodeAllocationFrom(message);
5:     updateDT();
6:     for all code in message do
7:         updateCAT();
8:     end for
9:     for all childDirectories do
10:        reallocateCodeSpace();
11:        reassignPrefixLabels();
12:        sendUpdateDirectory();
13:    end for
14: end procedure
```

3.2.3 Trie Maintenance

The directory trie in DARCI acts as a communication highway for name resolution and content routing. Directories maintain trie membership with frequent exchange of signaling between directories and with neighbor routers.

Inter Directory Signaling

Directories signal each other by sending HELLO-DIRECTORY messages. Such a message from t to u contains parent of t, its label prefix, and interval. If u is a child of t, on receiving HELLO-DIRECTORY, it verifies the parent-child lineage by way of t's prefix label check. For any other router directory that is in its initialized state with itself as its parent and not as the root of trie will initiate an ENROLL-DIRECTORY message towards t. To reduce the churn caused by reactive nature of directories to find better parent directory as it receives neighbor HELLO updates, DARCI prevents a directory router from sending ENROLL-DIRECTORY if it already has connectivity to a parent. The only exception is when the directory router fails to receive HELLO-DIRECTORY message within a configured time Δ and its neighbors report no route to parent. Such a directory initializes itself as its parent, updates the interval to $[\alpha, \omega]$, assigns prefix label 0, and sends re-allocations to its child directories. For non-root directory i, it selects the next lexicographically smallest directory with the shortest distance from i in $[DT]$ to send ENROLL-DIRECTORY. Algorithm 5 describes the processing of the HELLO-DIRECTORY message at directory router u.

Link Failures and Recovery

Neighbor HELLO messages detect link failures. Routers update the route to root directory from other neighbors. Directory routers proactively initiate HELLO-DIRECTORY towards its parent directory on detecting link failures. If the root is reachable, but the distance to root is greater than r hops, the directory router sends ENROLL-DIRECTORY to the lexicographically smallest directory router in its CDT. The new parent allocates interval for the child directory, generates a prefix label, and creates a new tuple in its CDT. The parent directory

Algorithm 5 DARCI: Process `HELLO-DIRECTORY` message at directory router u

```
1: input: NT^t, DT^t, CDT^t, NORI^t, CANOP^t, CAT^t, message;
2: procedure ProcessHelloDirectory
3:     x ← GetDirectoryIDFrom(message);
4:     if self ≡ myParent() then
5:         if |x| < |self| then                          ▷ A new parent?
6:             sendEnrollDirectory(x);
7:         end if
8:     else if x == myParent() then
9:         if myPrefix ⊊ prefixOf(x) then
10:            sendEnrollDirectory(x);
11:        end if
12:    else if isChildDirectory(x) then
13:        if self ≠ parent(x) then
14:            removeChildDirectory(x);
15:            reallocateCodeSpace();
16:            reassignPrefixLabels();
17:            for all childDirectories do
18:                sendUpdateDirectory();
19:            end for
20:        end if
21:    end if
22:    updateNT();
23:    updateDT();
24: end procedure
```

replies to the child directory with a sequence numbered ENROLL-DIRECTORY-ACK message. The child directory updates its root and parent directory, its label prefix, the code interval, and reassigns prefixes and intervals for all its child directories. The directory sends updated child directory information to all children in a sequence numbered UPDATE-DIRECTORY message. Parent directory routers purges entries with expired timestamp. In Fig. 3.7, the loss of link between m and g, q and j, and j and n triggers ENROLL-DIRECTORY message from directory router t to the parent i. Directory router i assigns prefix label 000 with an interval $[\alpha, \lambda]$ where $\alpha < \lambda$. Directory router u receives new prefix label 0000 and an interval $[\alpha, \delta]$ from its parent t. Directory router a purges entry for t from its

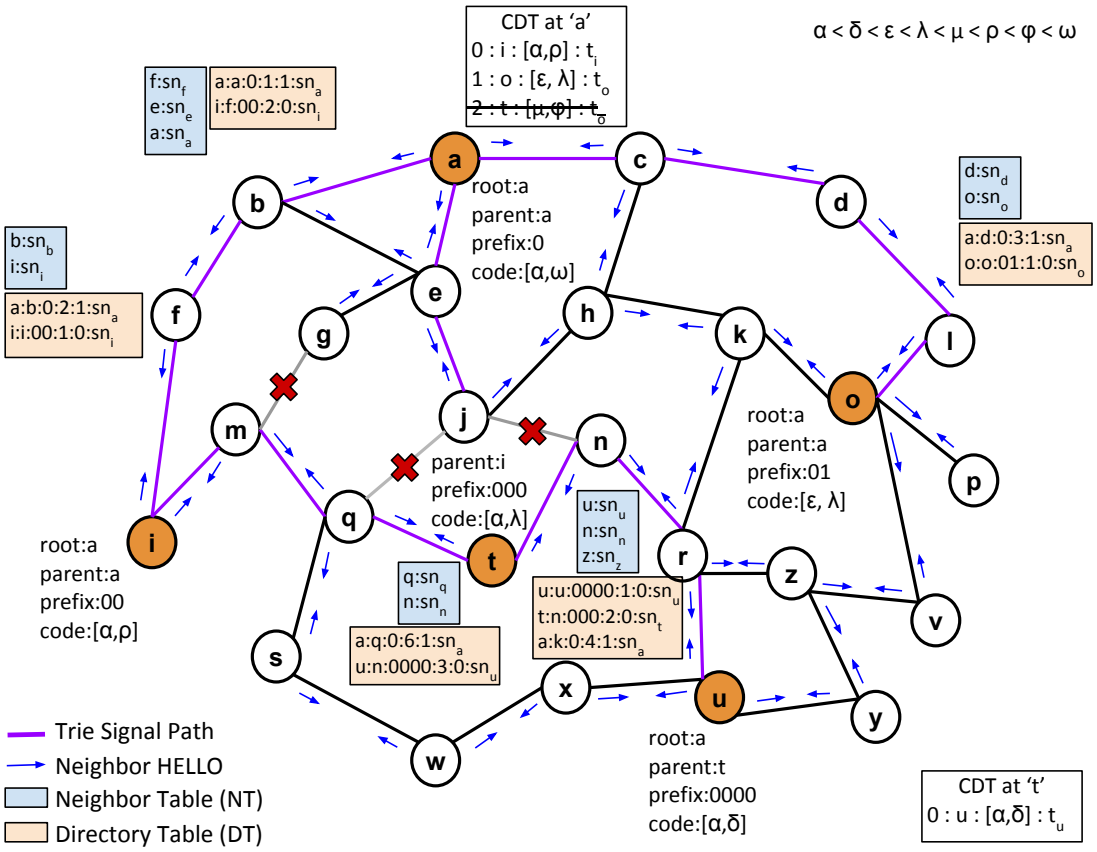

Figure 3.7: Example link failure & recovery in DARCI

CDT. Fig. 3.7 shows NT and DT for routers b, f, t, r, and l.

Alternate Root Election

Neighboring directory routers detect loss of root directory in HELLO and HELLO-DIRECTORY messages. Child directories update NT and DT and initialize itself as the root and parent directory. Neighbor routers update NT and DT with the new root directory ID. Newly transitioned root directories reassign label prefixes and intervals to all child directories and send UPDATE-DIRECTORY message. More than one newly transitioned root directories converge to one root when each directory knows of other roots. A transitioned root directory selects the lexicographically smallest known root directory as its parent. Fig. 3.8 and

Algorithm 6 DARCI: Process `UPDATE-DIRECTORY` message at directory router u

```
1:  input: NT^u, DT^u, NORI^u, CANOP^u, CAT^i, message;
2:  procedure ProcessUpdateDirectory
3:      x ← GetDirectoryIDFrom(message);
4:      [s_min : s_max] ← GetCodeAllocationFrom(message);
5:      updateDT();
6:      for all code in message do
7:          updateCAT();
8:      end for
9:      for all childDirectories do
10:         reallocateCodeSpace();
11:         reassignPrefixLabels();
12:         sendUpdateDirectory();
13:     end for
14: end procedure
```

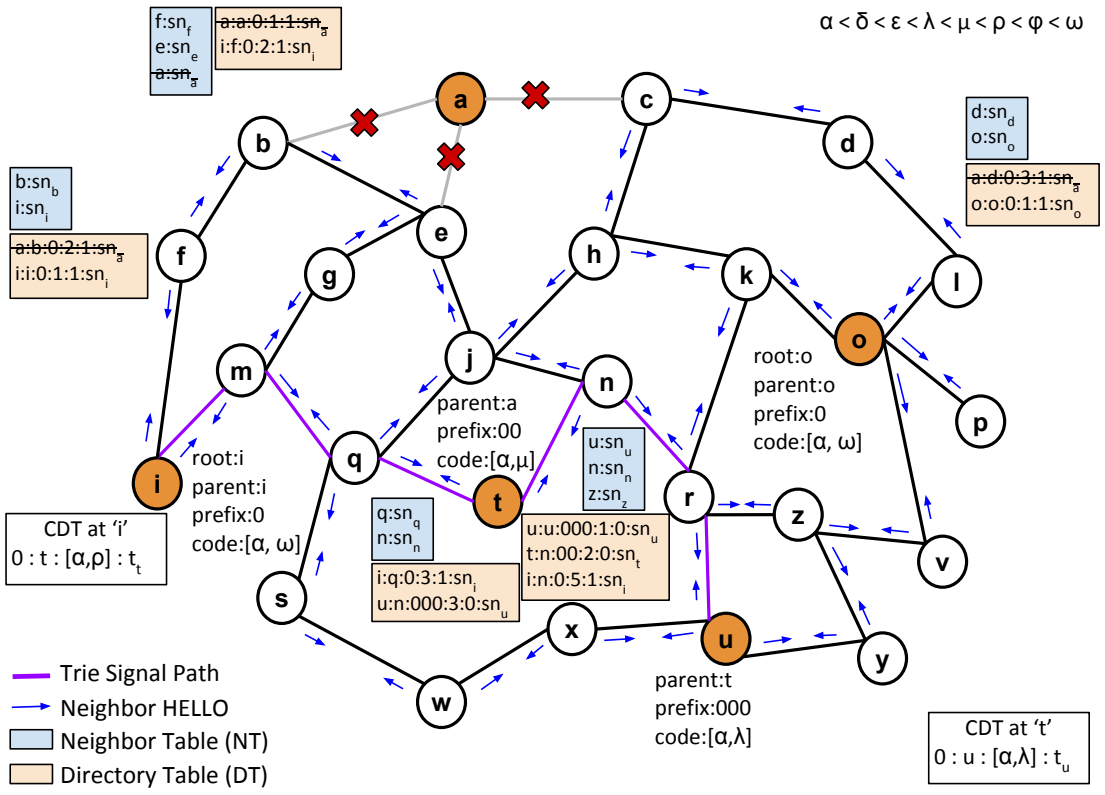

Figure 3.8: Trie root a unreachable in DARCI

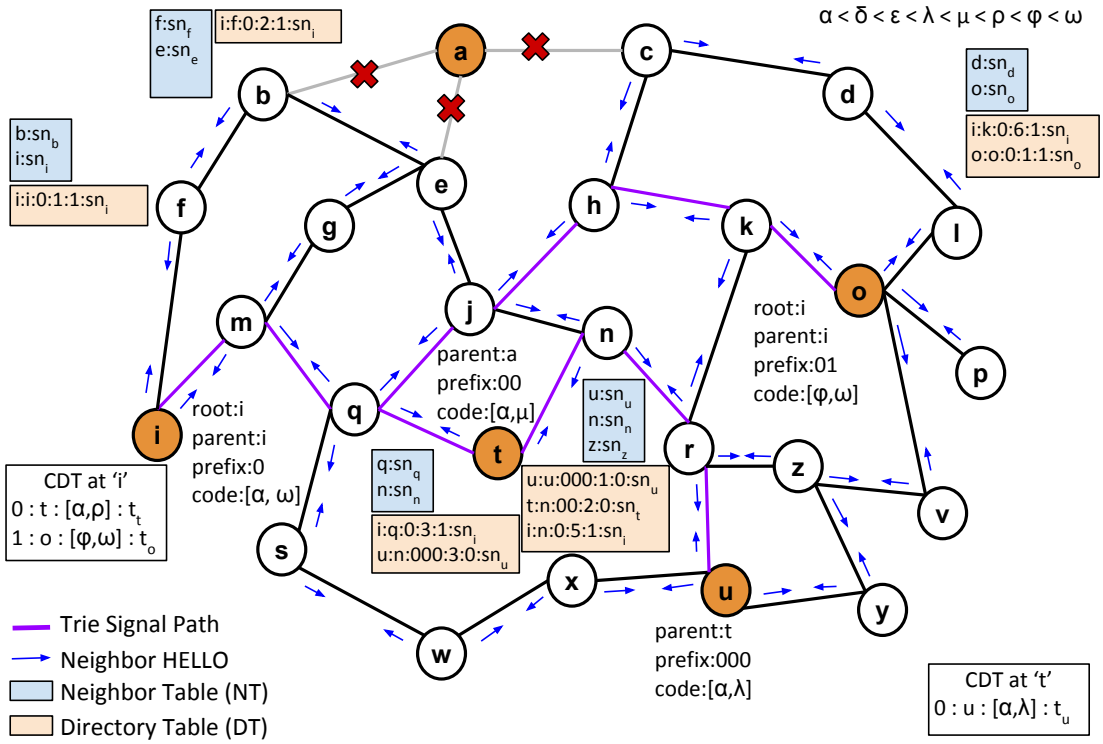

Figure 3.9: New trie root i elected in DARCI

3.9 exemplifies alternate root election. At the loss of root a, neighbor routers b and f updates i of unreachability. Similarly, router c, d, and l updates directory o. Directories i and o initializes as its root and parent and claims the global code space $[\alpha, \omega]$. Directory t discovers router i as the nearest root and send ENROLL-DIRECTORY message. Directory t updates its prefix label and code interval, reassigns new prefix label and code interval to all its children, and sends UPDATE-DIRECTORY message to child directory u. In Fig. 3.9, root directory changes are propagated to directory o. Directory o sends ENROLL-DIRECTORY to the lexicographically smallest root i. Directory router i assigns prefix label and code interval for o. Directory o updates its prefix label and code interval, and points to i as the new root.

3.3 Publish and Subscribe

3.3.1 Publish Operation

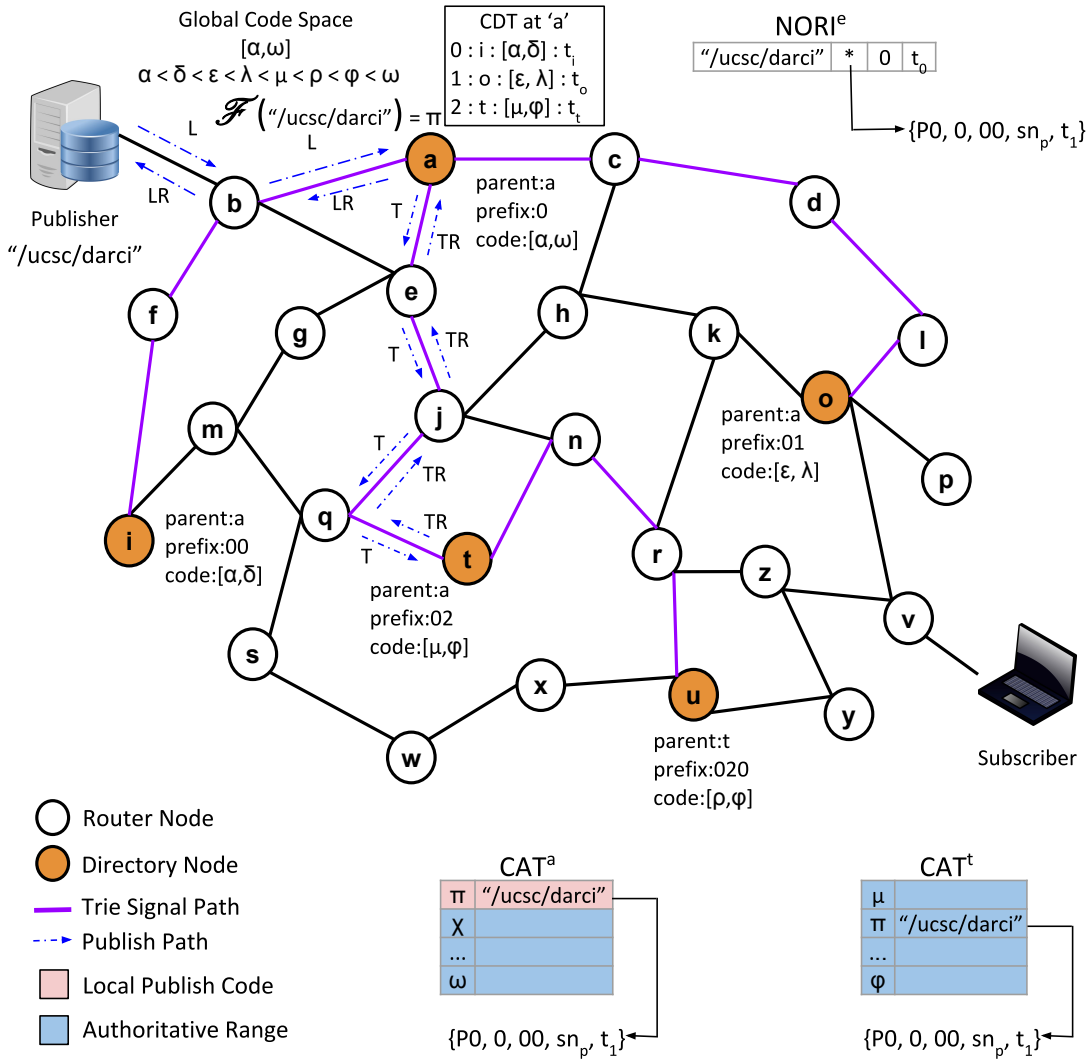

Figure 3.10: Content publish in DARCI

Publishing in DARCI consists of publishers forwarding PUBLISH messages to their neighbor router. The PUBLISH message contains the content name prefix, a local ID of the publisher, and a sequence number. Fig. 3.10 is an example of publisher P0 connected to router b. Neighbor router b forwards the local publish

message to the closest directory router a in the trie by adding labels of directory routers within its r hops. The local publish path towards directory a is labeled with L in Fig. 3.10. The directory router a on receiving the local publish message becomes the anchor directory $\mathcal{A}$ for the publisher P0. Directory a replies to publisher P0 with PUBLISH-ACK. The local reply path for PUBLISH-ACK is labeled LR in Fig. 3.10. Local routers and directory for a publisher maintain a loop-free path to the publisher using neighbor HELLO. These routes are periodically refreshed based on HELLO intervals. Anchor directory a computes the space-filling curve code for the content name using the transform $\mathscr{F}(NO) = \pi$, where $\mathscr{F}$ is as described in Algorithm 7. The mapping of content name to a publisher in the local publish message is registered in anchor directory a's CAT^a. Directory a retrieves the interval for π by looking in the CDT^a. In the example in Fig. 3.10, $\pi \in [\mu, \phi]$ is in an interval assigned to directory t. The publish message is forwarded in the trie towards the prefix 02 of authoritative directory t. Fig. 3.10 shows the trie path for publish and reply to t and a with label T and TR respectively. Each router in the trie path from a to t forwards the PUBLISH message and caches the mapping (e.g., $NORI^e$ at router e). Fig. 3.10 shows the local and authoritative registration of the mapping in CAT^a and CAT^t pointing to named object profile (NOP). Publishers periodically refresh the content they own by sending PUBLISH messages towards the trie via their neighbor router. For a set of already published and unchanged contents, publishers send a compressed message that contains the signature to identify the set of unchanged content names. Periodic refresh by publishers addresses updates in mapping to anchor directory prefixes when trie structure changes. Algorithm 8 describes the processing of PUBLISH message at anchor directory router a.

Algorithm 7 DARCI: Transform function at directory a

input: M; ▷ Hilbert Curve Index
input: Σ; ▷ Set of symbols and corresponding codes Ξ
input: T; ▷ A tranform, convert string of symbols to string of code from Σ
input: `message`;
procedure GETCODE
    $C \leftarrow$ GetContentName(`message`);
    $X = 0$ ▷ X coordinate in 2D
    $Y = 0$ ▷ Y coordinate in 2D
    $Y \leftarrow$ STRLEN(C);
    $X \leftarrow$TO_NUMBER($T(C)$) ▷ $X < 2^M, X \in \mathbb{N}$
    $H = HC(M, X, Y)$;
    return H;
end procedure

Algorithm 8 DARCI: Process `PUBLISH` message at directory a

input: NT^a, DT^a, CDT^a, $NORI^a$, $CANOP^a$, CAT^a, `message`;
procedure PROCESSPUBLISH
    $a \leftarrow$ GetAnchorDirectoryFrom(`message`);
    $l[..] \leftarrow$ GetLandmarkDirectoriesFrom(`message`);
    $PubID \leftarrow$ GetPublisherLocalID(`message`);
    $c \leftarrow$ GetContentNameFrom(`message`);
    $s_c \leftarrow \mathscr{F}(c)$; ▷ Compute code in SFC
    updateNORI(c, $PubID$, a, $l[]$); ▷ Cache in local routing table
    $NOP \leftarrow$ CreateNamedObjectProfileFor(c, $PubID$, a, $l[]$);
    if $a = \emptyset$ **then** ▷ A local publisher
        updateCAT(NOP); ▷ Update local code allocation table
        sendLocalPublishAck($PubID$); ▷ Always ACK local publisher
        $message \leftarrow$ UpdateMessageWithSourceAnchor(`message`, MyPrefix());
    end if
    if $s_c \in [s_{min} : s_{max}]$ **then** ▷ Is this my authority?
        updateCAT(NOP);
        $if \leftarrow$ GetInterfaceTowardsAnchorPrefix(a);
        sendPublishAck(if, a); ▷ Ack to Anchor directory
    else
        $if \leftarrow$ GetInterfaceForIntervalWith(s_c);
        forwardPublish(if, `message`); ▷ Interface with interval s_c
    end if
end procedure

3.3.2 Subscribe Operation

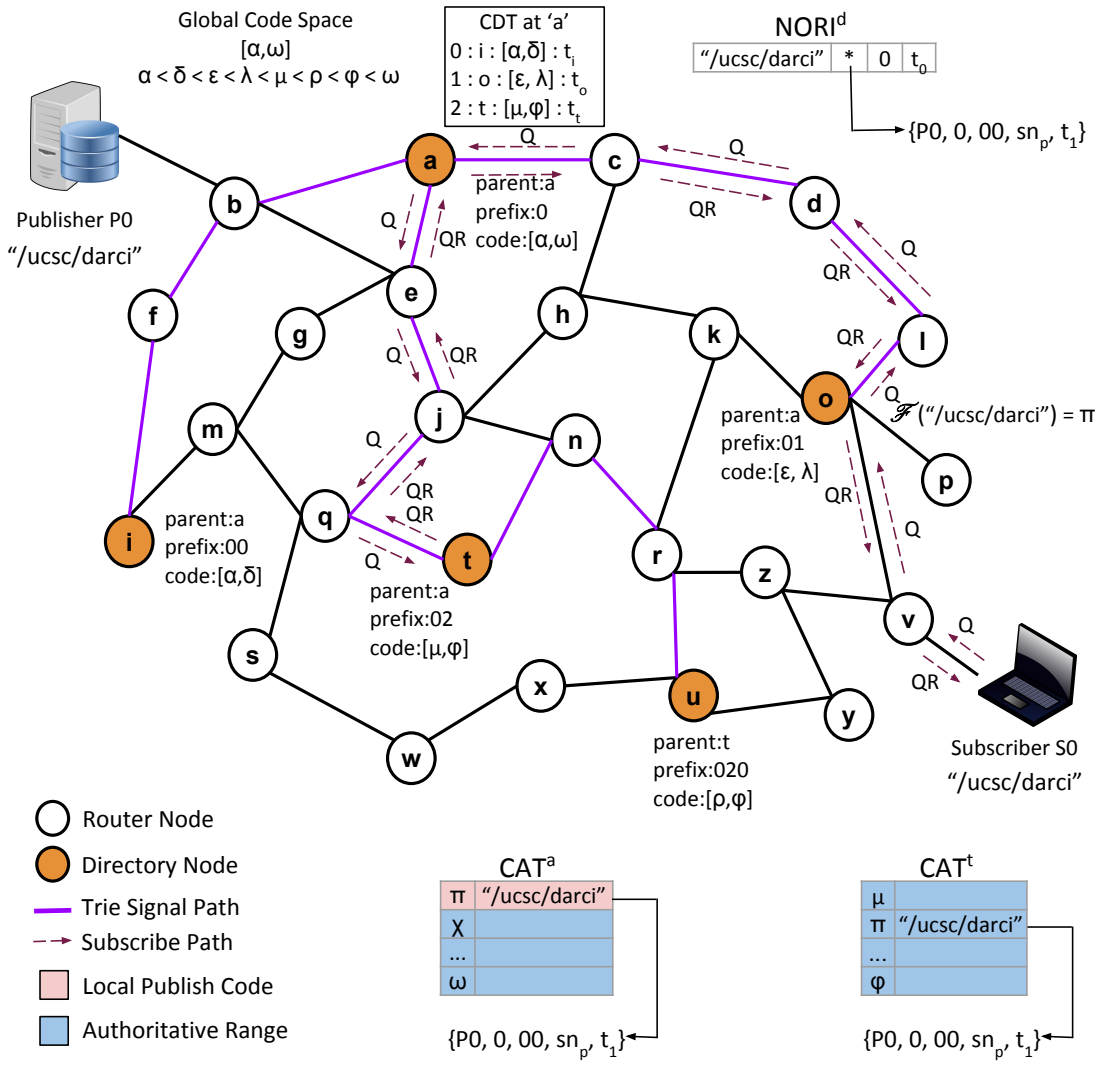

Figure 3.11: Content subscribe in DARCI

Subscribers register with neighbor routers using HELLO messages. All neighbor and directory routers within "r" hops maintain a loop-free sequence numbered route to subscribers. A subscriber interested in content sends a LOOKUP message containing the name of the content, a local ID of the subscriber, and a sequence number to its neighbor router. The neighbor router forwards the LOOKUP message to the closest directory. This directory is the anchor direc-

Algorithm 9 DARCI: Process `LOOKUP` message at directory o

```
1: input: NT^o, DT^o, CDT^o, NORI^o, CANOP^o, CAT^o, message;
2: procedure ProcessLookup
3:     a ← GetAnchorDirectoryFrom(message);
4:     SubID ← GetSubscriberIDFrom(message);
5:     c ← GetContentNameFrom(message);
6:     s_c ← ℱ(c);                                   ▷ Compute code in SFC
7:     if c ∈ NORI^o then                                        ▷ In NORI?
8:         NOP ← CreateNOPFromNORIFor(c);
9:         if a = MyPrefix() then                    ▷ Is this from local host
10:            if ← GetInterfaceForDestination(SubID);
11:            sendLookupReply(if, NOP);
12:            return;
13:        else
14:            if ← GetInterfaceTowardsAnchorPrefix(a);
15:            sendLookupReply(if, NOP);
16:            return;
17:        end if
18:    end if
19:    if s_c ∈ CAT^o then
20:        NOP ← LookupInCAT(c);
21:        if ← GetInterfaceTowardsAnchorPrefix(a);
22:        sendLookupReply(if, NOP);
23:        return;
24:    end if
25:    if s_c ∈ [s_min : s_max] then                ▷ Am I the Authoritative?
26:        if s_c ∈ CAT^o then
27:            NOP ← LookupInCAT(c);
28:            if ← GetInterfaceTowardsAnchorPrefix(a);
29:            sendLookupReply(if, NOP);
30:            return;
31:        end if
32:    end if
33:    if ← GetInterfaceTowardsAnchorPrefix(a);
34:    sendLookupReply(if, CONTENT-UNAVAILABLE);
35: end procedure
```

tory for the subscriber. At the anchor directory, the code for requested content is computed and checked for its availability in the local cache. The anchor directory sends LOOKUP-REPLY with the mapping to the subscriber. Anchor

directory determines the direction for the interval in which the code is. It marks the LOOKUP message with its label prefix as the source of the request and forwards the LOOKUP message using interval routing in the trie towards the authoritative directory that owns the interval. The authoritative directory replies with LOOKUP-REPLY containing the mapping for the requested content. The LOOKUP-REPLY follows the trie path towards the source anchor directory prefix using prefix label routing. Intermediate routers in trie that forward the LOOKUP-REPLY opportunistically cache the mapping information in its $NORI$ table. Anchor directory on receiving LOOKUP-REPLY updates its $NORI$ table and forwards it to the subscriber. Fig. 3.11 is an example of subscriber S0 connected to router v and requesting content "/ucsc/darci". Anchor directory o computes the space-filling curve code for the content name using the transform $\mathscr{F}(NO) = \pi$, where $\mathscr{F}$ is as described in Algorithm 7. $\pi \notin [\epsilon, \lambda]$, anchor directory o forwards the LOOKUP message to its parent directory a. The LOOKUP message path from anchor directory o to authoritative directory t through trie using interval routing is labeled Q in the figure. The LOOKUP-REPLY message path from t to anchor directory o via trie using prefix label routing is labeled QR in the figure. Intermediate router d in the trie stores the mapping information in its $NORI^d$ table. Algorithm 9 describes processing of LOOKUP message at directory o.

3.3.3 Content Encoding

DARCI's naming can support any naming nomenclature - whether flat or hierarchical. Fig. 3.12 compares information domain naming in host-centric network (HCN) architecture and the corresponding optimal way of naming in an ICN architecture. Column three in the figure shows the Hilbert Curve code for each of the names in DARCI's ICN architecture. For an HCN, resolving the address

from “0041.ace.darci” to “0050.ace.darci” takes 10 DNS requests and responses. Range queries in DARCI LOOKUP can request name resolution by specifying the start and end of the name prefix. For example, a LOOKUP can request all mappings between “0041.ace.darci” to “0050.ace.darci” so that a web page can request resources to compile and render in a browser. Column 4 lists the compute time to generate Hilbert Curve code on a machine with Dual-core processors at 1GHz clock frequency and memory of 512MB. DARCI thus inherently supports the basic range query of named objects.

HCN Architecture	ICN Architecture	Hilbert Curve Code in 1D	Hilbert Curve Coding Time (µS)
0041.ace.darci	ace/0041	18360399985699435	6
0042.ace.darci	ace/0042	18360399985699436	7
0043.ace.darci	ace/0043	18360399985699439	7.5
0044.ace.darci	ace/0044	18360399985699440	5
0045.ace.darci	ace/0045	18360399985699441	6
0046.ace.darci	ace/0046	18360399985699454	5
0047.ace.darci	ace/0047	18360399985699455	6.5
0048.ace.darci	ace/0048	18360399985699370	7
0049.ace.darci	ace/0049	18360399985699369	6
0050.ace.darci	ace/0050	18360399985704560	7

Figure 3.12: HCN and ICN naming architecture comparison and effect on performance

3.4 Content Routing

A subscriber receives mapping to requested content via LOOKUP-REPLY. The subscriber sends a DATA-REQUEST message to its neighbor router by embedding the mapping of content name to its locator and landmark prefixes. At every hop, routers inspect the destination prefixes and determine the shortest route to reach them. Routing prioritizes on reaching anchor prefix. If no path exists, the

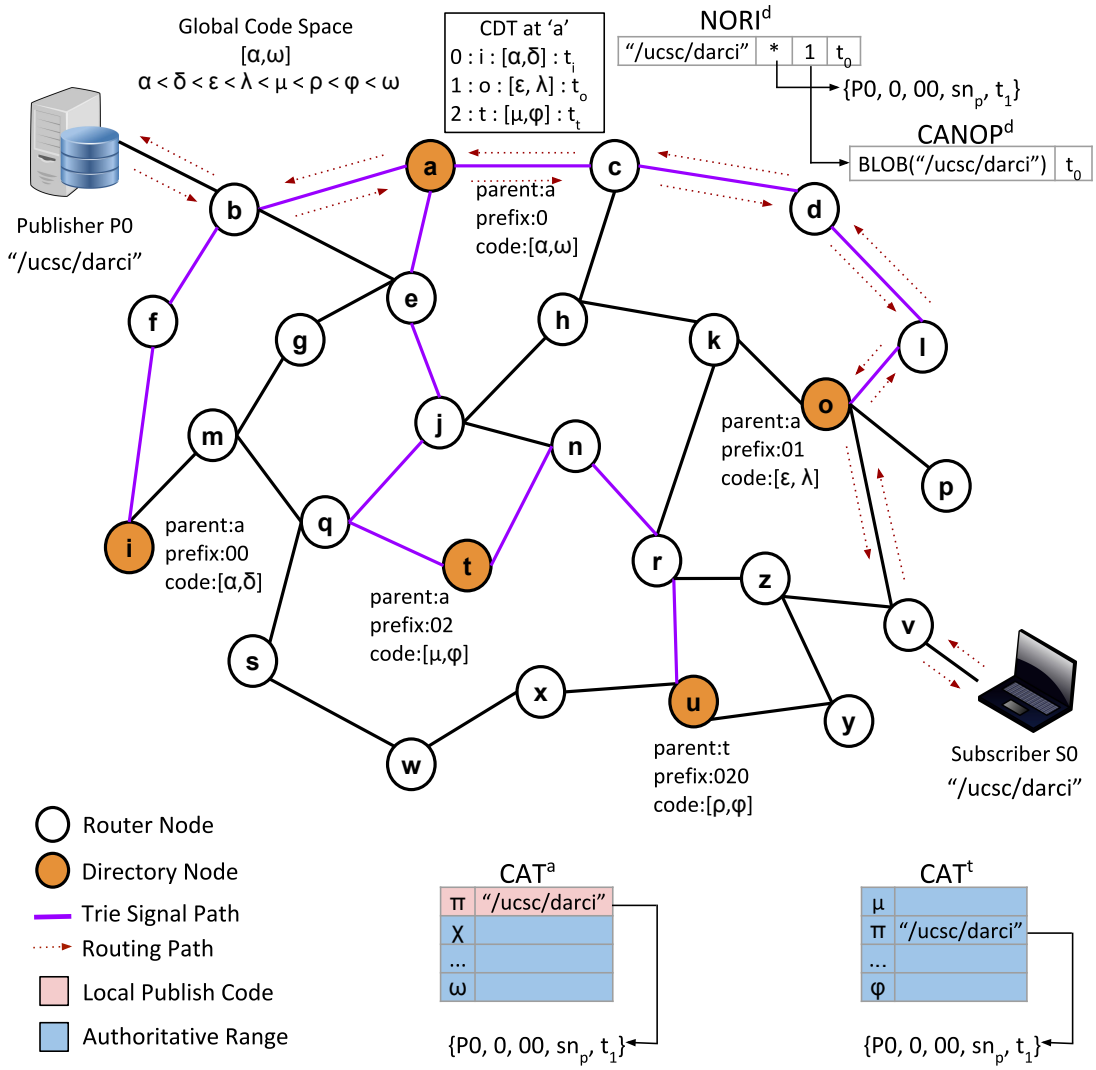

Figure 3.13: Content routing in DARCI

default path is to forward the DATA-REQUEST message to the neighbor directory router with the longest prefix match towards a publisher's anchor directory prefix. Intermediate routers check the cache CANOP for a local copy of the requested content. Routers reply with a cached copy in the DATA-REPLY message towards the subscriber's anchor prefix. If trie is the shortest path derived to reach the publisher's anchor directory, then the DATA-REQUEST message is forwarded hop by hop using prefix label routing. An intermediate router or directory can

Algorithm 10 DARCI: Process `DATA-REQUEST` message at directory o

```
1: input: NT^o, DT^o, CDT^o, NORI^o, CANOP^o, CAT^o, message;
2: procedure ProcessDataRequestAtDirectory
3:     a_p ← GetPubAnchorDirectoryFrom(message);
4:     l_p[] ← GetPubLandmarkDirectoriesFrom(message);
5:     a_s ← GetSubAnchorDirectoryFrom(message);
6:     l_s[] ← GetSubLandmarkDirectoriesFrom(message);
7:     c ← GetContentNameFrom(message);
8:     s_c ← ℋ(c);                                  ▷ Compute code in SFC
9:     if a_p = GetPrefix(self) then                ▷ Am I the Anchor
10:        if s_c ∈ CAT^o then                      ▷ Locally aware?
11:            NOP ← GetNOPFor(c);
12:            if c ∈ CAT^o then                    ▷ In cache?
13:                data ← GetFromCANOP(c);
14:                sendDataReply(data, a_s, l_s[]);
15:            else
16:                if ← RouteToNextHopTowardsPublisher(NOP);
17:                forwardToPublisher(if);
18:            end if
19:        else
20:            sendError(CONTENT-UNREACH, a_s, l_s[]);
21:        end if
22:    else
23:        if ← RouteToNextHopTowardsAnchor(a_p, l_p[]);   ▷ Get interface
24:        forwardDataRequest(if);
25:    end if
26: end procedure
```

redirect the DATA-REQUEST to a shorter path if one is available after consulting its NT and DT. DATA-REQUEST message on reaching the publisher's anchor directory is forward to the publisher via the local routers. Publisher replies with requested content in the DATA-REPLY message. The neighbor router determines the next hop to reach the prefix of the subscriber's anchor directory. The default next hop is towards the closest directory router with the longest prefix match towards a subscriber's anchor directory prefix. Intermediate routers and directories opportunistically cache content data in their CANOP cache. The anchor directory

Algorithm 11 DARCI: Process `DATA-REQUEST` message at router c

```
1: input: NT^c, DT^c, NORI^c, CANOP^c, message;
2: procedure ProcessDataRequestAtRouter
3:     a_p ← GetPubAnchorDirectoryFrom(message);
4:     l_p[] ← GetPubLandmarkDirectoriesFrom(message);
5:     a_s ← GetSubAnchorDirectoryFrom(message);
6:     l_s[] ← GetSubLandmarkDirectoriesFrom(message);
7:     c ← GetContentNameFrom(message);
8:     if c ∈ CANOP^c then                                  ▷ In cache?
9:         data ← GetFromCANOP(c);
10:        sendDataReply(data, a_s, l_s[]);
11:    else
12:        if ← RouteToNextHopTowardsAnchor(a_p, l_p[]);    ▷ Get interface
13:        forwardToAnchor(if);
14:    end if
15: end procedure
```

Algorithm 12 DARCI: Process `DATA-REPLY` message at d

```
1: input: NT^d, DT^d, NORI^d, CANOP^d, message;
2: procedure ProcessDataReply
3:     SubID ← GetSubscriberIDFromMessage(message);
4:     c ← GetContentNameFrom(message);
5:     data ← GetDataFrom(message);
6:     if SubID = MyID() then
7:         sendToApplication(c, data);
8:         return;
9:     end if
10:    UpdateCANOP(c, data);                                ▷ Add or update cache
11:    a_s ← GetSubAnchorDirectoryFrom(message);
12:    l_s[] ← GetSubLandmarkDirectoriesFrom(message);
13:    if ← RouteToNextHopTowardsAnchor(a_s, l_s[]);        ▷ Get interface
14:    forwardDataReply(if);
15: end procedure
```

on receiving DATA-REPLY forwards to the subscriber via local routers. Fig. 3.13 shows an example of subscriber S0 requesting content "/ucsc/darci". The DATA-REQUEST and DATA-REPLY, in this example topology, follow the trie path, as shown in the figure. Router d on receiving DATA-REPLY updates its $NORI^d$

table with a pointer to the cache location in $CANOP^d$ that stores a copy of "/ucsc/darci". Router d replies to any future LOOKUP or DATA-REQUEST messages for content "/ucsc/darci" and replies with DATA-REQUEST with a cached copy from its $CANOP^d$. Algorithm 10 describes processing of DATA-REQUEST at directory router o. Algorithm 11 describes processing of DATA-REQUEST at an intermediate router c. Algorithm 12 describes the processing of DATA-REPLY at intermediate router d where it updates the local cache with a copy of requested content.

3.5 Simulation

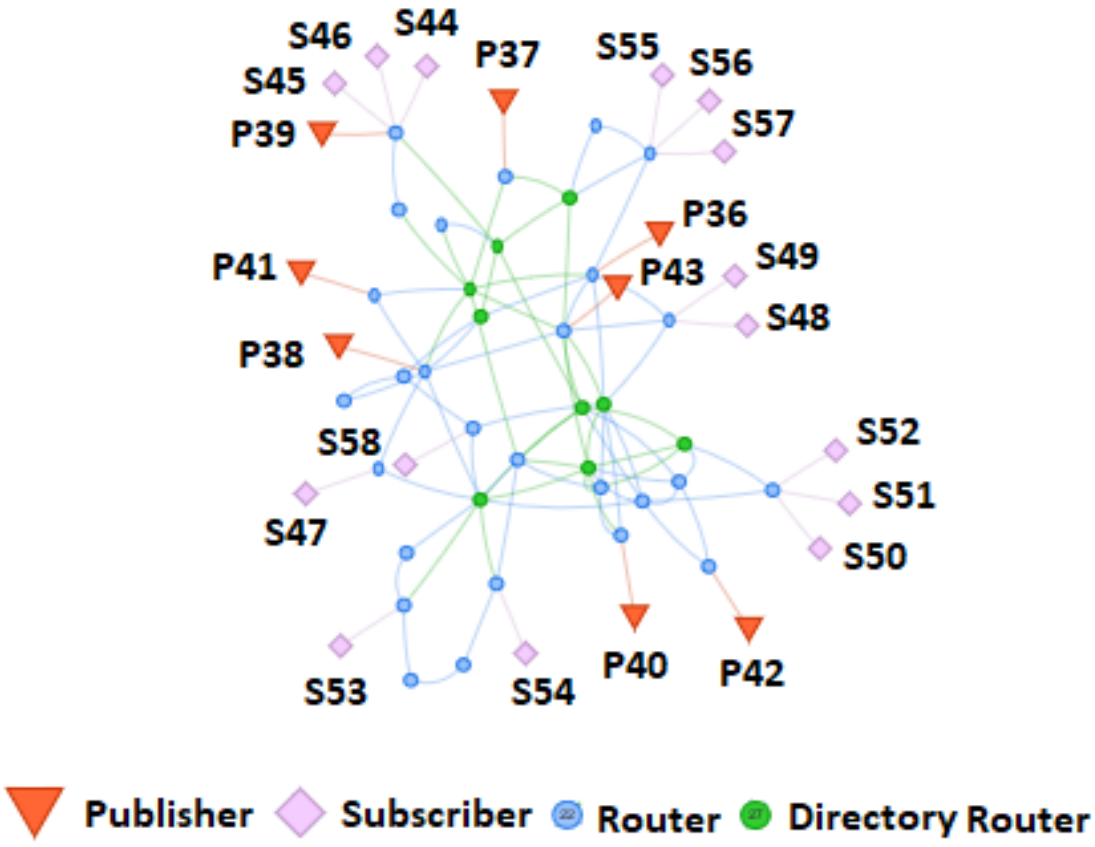

Figure 3.14: DARCI simulation in 59 node British Telecom NA topology

We implemented DARCI in ns-3 version 3.27[65] and used ns-3 implementation of NDN[66] for comparing performance metrics of average throughput, average delay, and average round trip hop count as seen by subscriber nodes for an increasing number of content names. We derived a 59 node fixed topology as shown in Figure 3.14 from "BT North America" dataset[67]. Six unique information name object files each 200K, 50K, 20K, 5K, 500 and 20 names were derived from web traffic dataset[68]. The simulation ran for 3 hours with 5 different seed values, with subscribers requesting 25 names per second. Publisher applications on publishers 36-43 were allowed to start publishing the contents they owned before subscriber applications began requesting information. Publishers are assigned the list of unique names such that a given publisher in DARCI and NDN had the same set of names. Subscribers 44-58 were aware of the complete name list, out of which each subscriber randomly selects names. We compare performance metrics for name data files 20K, 5K, 500, and 20. We evaluate the two protocols in a wired network with fixed topology. In DARCI, we use a maximum distance of 2 hops for a publisher or a subscriber to select an anchor directory. Nodes in DARCI and NDN are connected using point-to-point CSMA links with a data rate of 1Gbps, delay of 6.5 ms, and MTU size of 1400. For NDN, the size of the content store is 100 for on-path caching and set to a size of 1 for no caching scenarios. The caching policy for NDN is LRU. For DARCI, the size of CANOP is set to 100 for on-path caching and set to 0 for no caching scenarios. The caching policy in DARCI is LRU, and the routing strategy is "/localhost/nfd/strategy/multicast". The subscribers and publishers are nodes with no caching enabled. We perform two sets of simulations. The first set uses name data files 20K, 5K, 500, and 20 at 25 requests per second to measure average throughput, delay and hop count for on-path and no caching in DARCI and NDN. The second set of simulations profiles

the growth in the size of PIT and FIB in NDN against LT and NDT in DARCI as the uniquely addressable content names in the network increase for datasets 200K, 50K, 20K, 5K, 500 and 20 with subscribers requesting at 25 requests per second.

3.6 Performance Comparison

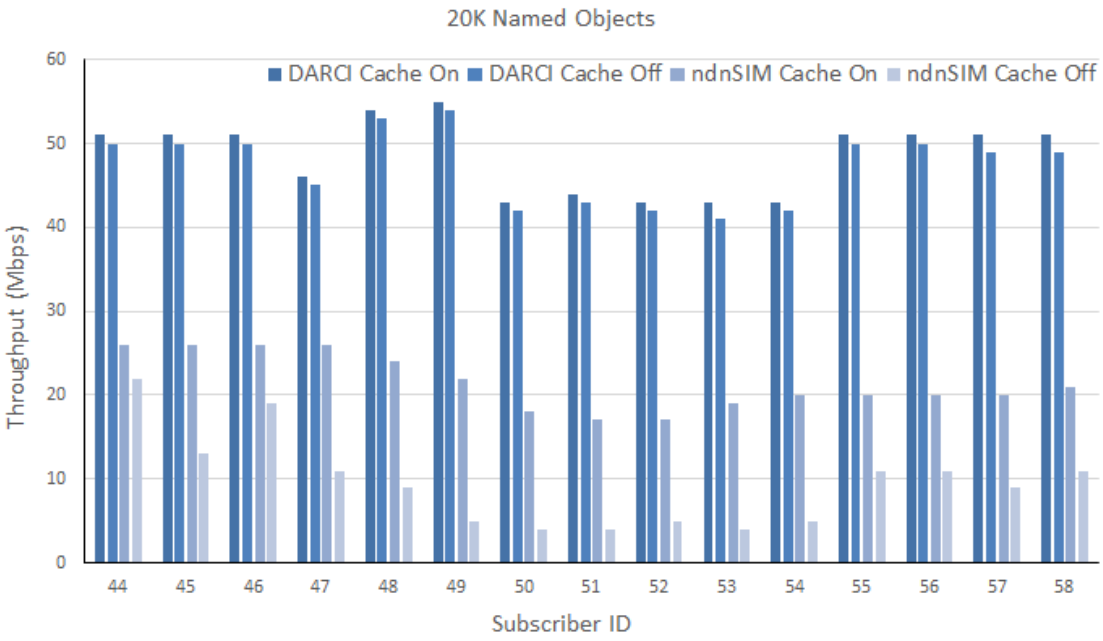

Figure 3.15: DARCI simulation: Throughput (Mbps) with 20K Names

Figures 3.15-3.18 summarizes average throughput as seen by subscribers for 4 name datasets with on-path and no caching for DARCI and NDN. We see that as the number of uniquely addressable names in the network increases, average throughput decreases for DARCI and NDN with on-path and no caching. However, DARCI produces better throughput compared to NDN. For Figure 3.18, no caching in DARCI produces better throughput than NDN with no caching. For on-path caching, our analysis of DARCI implementation revealed that since traffic

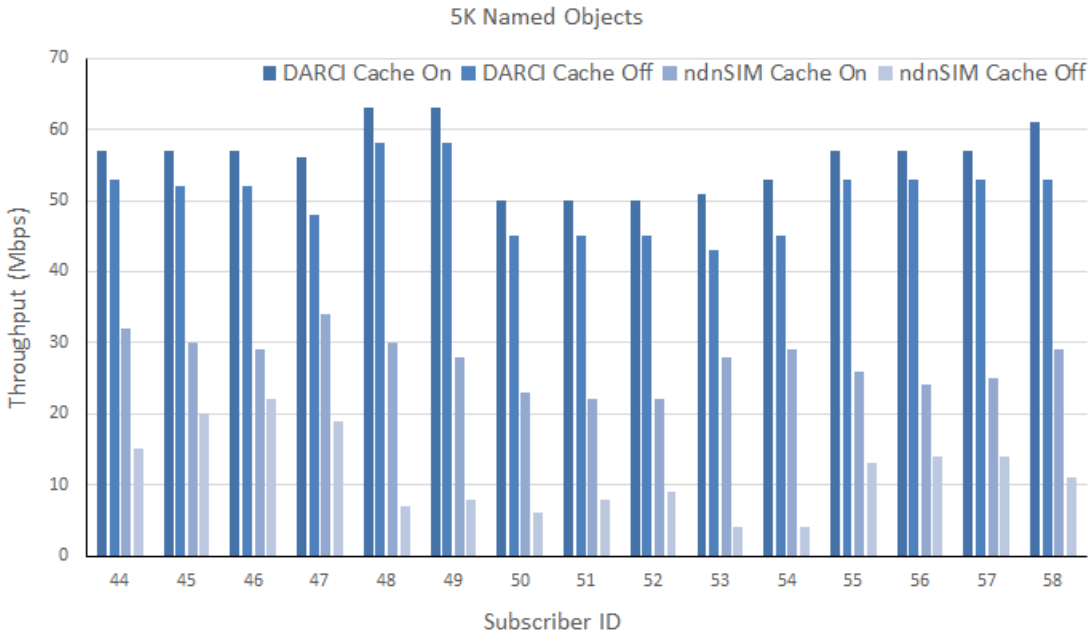

Figure 3.16: DARCI simulation: Throughput (Mbps) with 5K Names

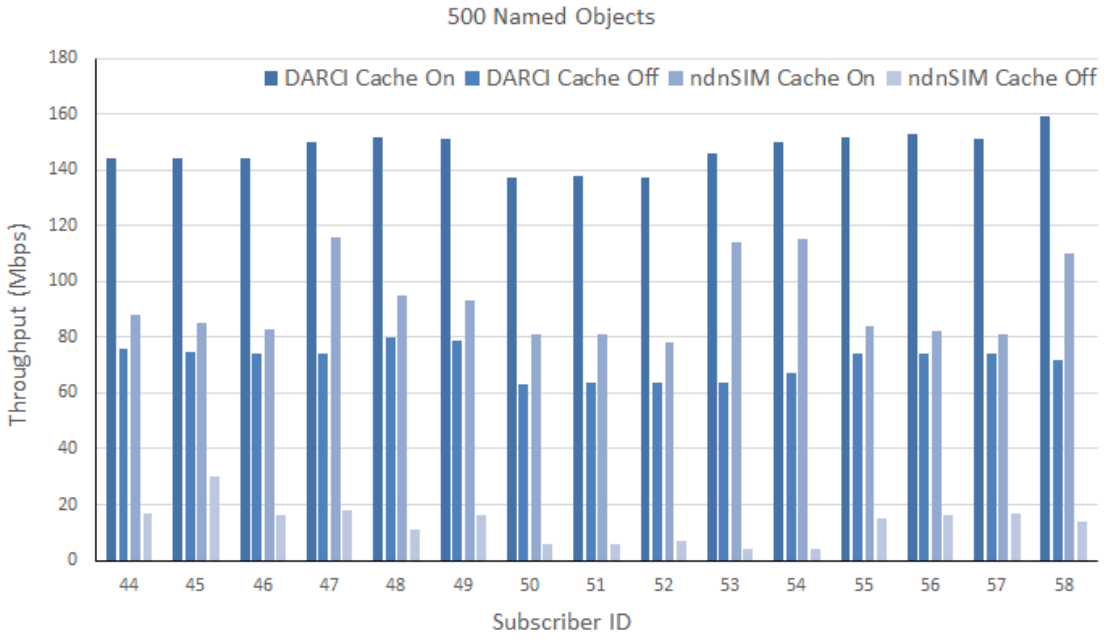

Figure 3.17: DARCI simulation: Throughput (Mbps) with 500 Names

in DARCI is always from the cache, our cache implementation required optimization. With only 20 uniquely know global names, contents are all cacheable within

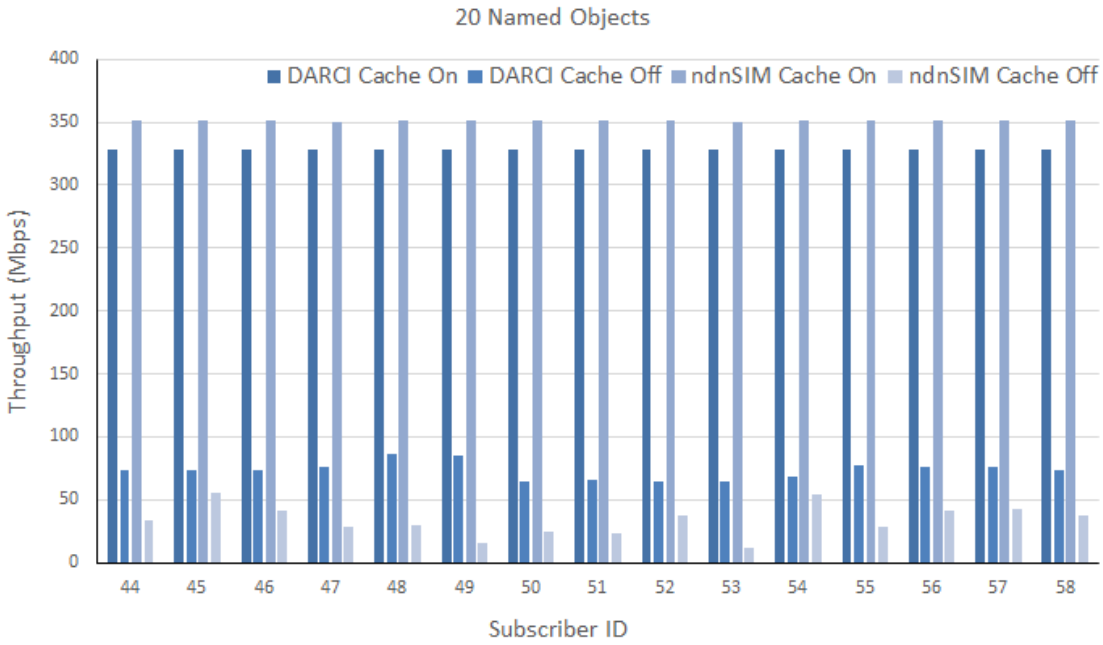

Figure 3.18: DARCI simulation: Throughput (Mbps) with 20 Names

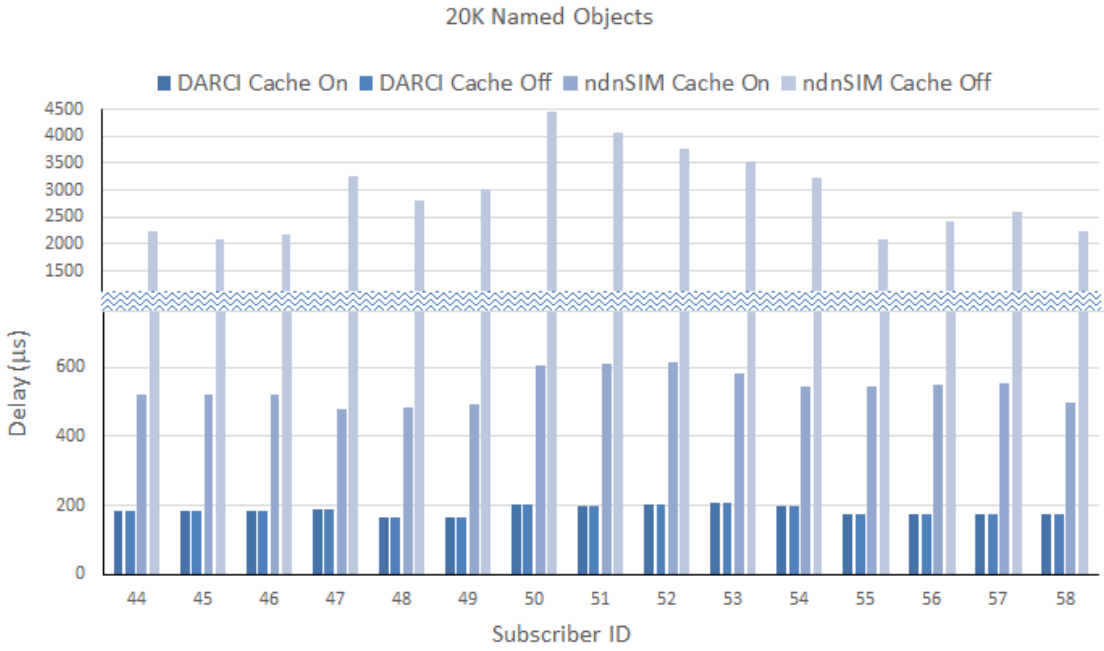

Figure 3.19: DARCI simulation: Delay (μs) with 20K Names

a content store of size 100. The results for no caching prove that interval routing in DARCI requires minimal lookup compared to FIB and PIT in NDN, it gives

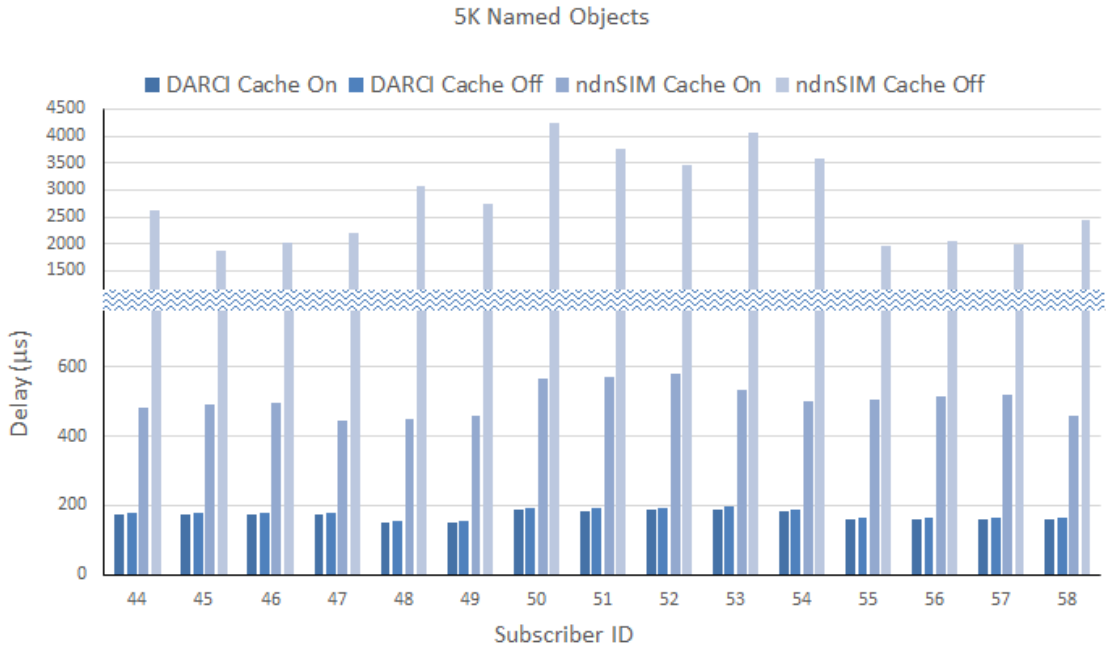

Figure 3.20: DARCI simulation: Delay (μs) with 5K Names

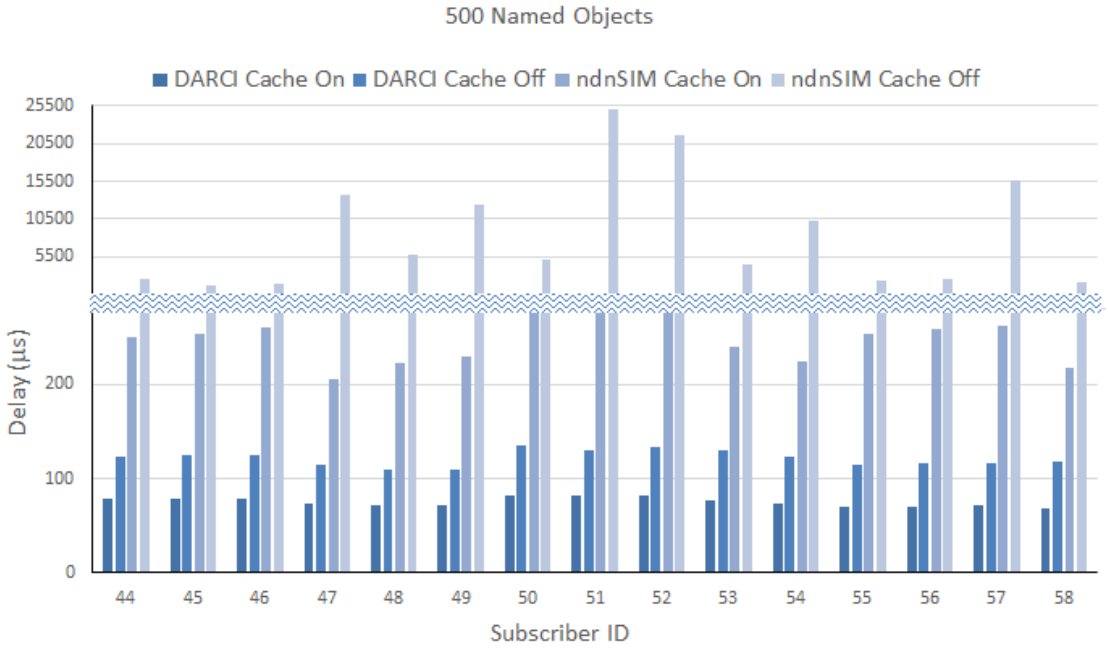

Figure 3.21: DARCI simulation: Delay (μs) with 500 Names

better throughput in DARCI for scenarios when requests have to be serviced by publishers. Figures 3.19-3.22 summarizes average delay as seen by subscribers for

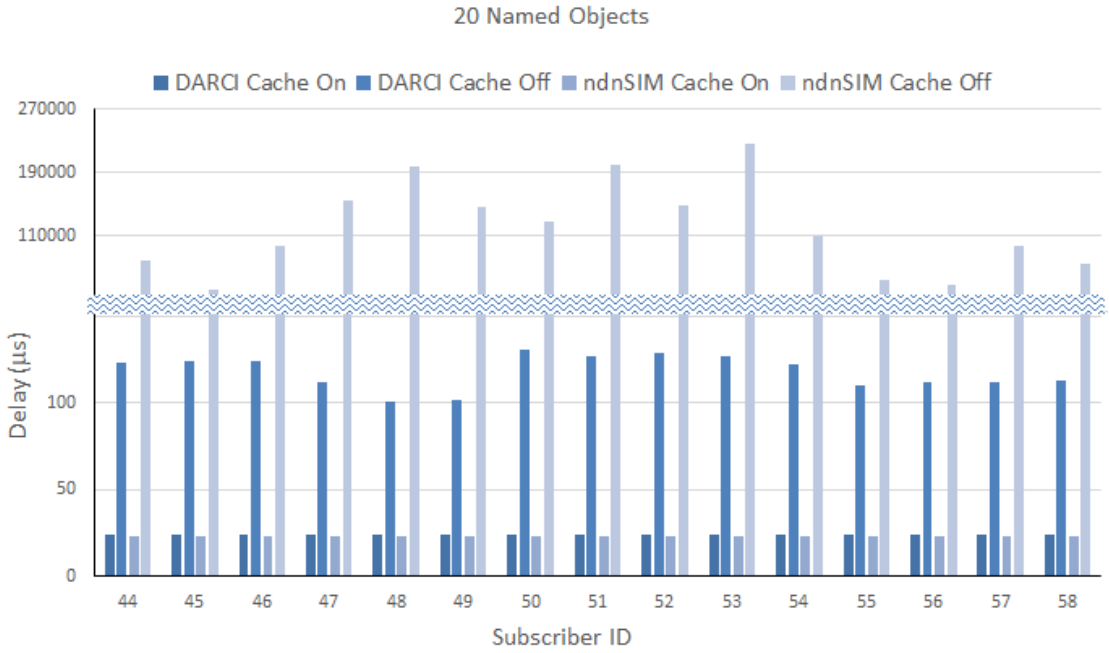

Figure 3.22: DARCI simulation: Delay (μs) with 20 Names

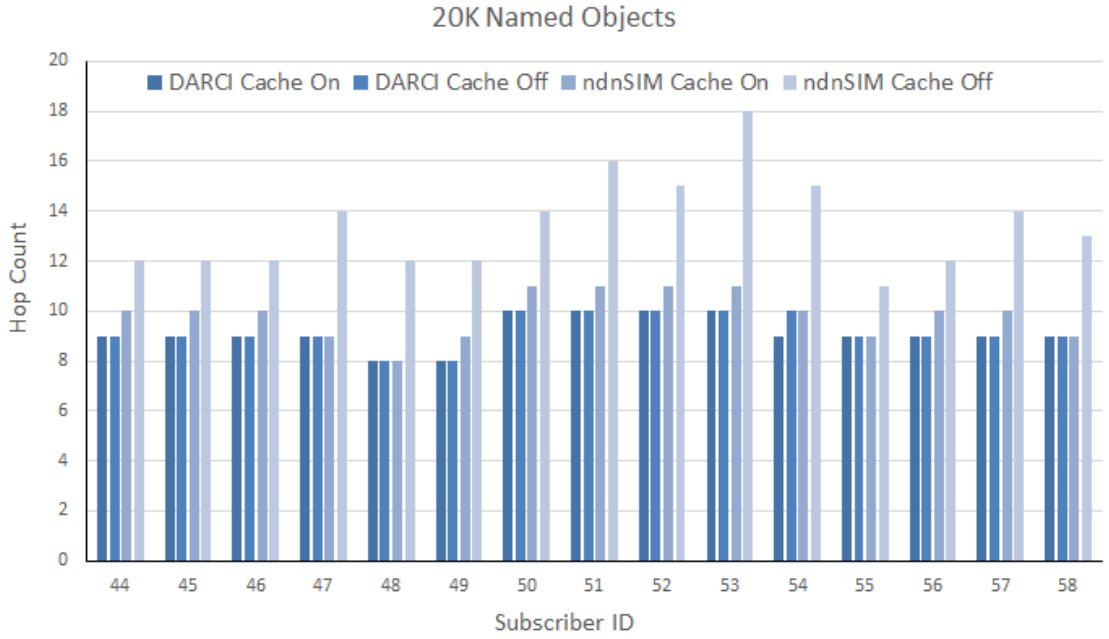

Figure 3.23: DARCI simulation: Hop count with 20K Names

4 name datasets with on-path and no caching for DARCI and NDN. For on-path caching, the average delay as seen by subscribers in NDN increases rapidly com-

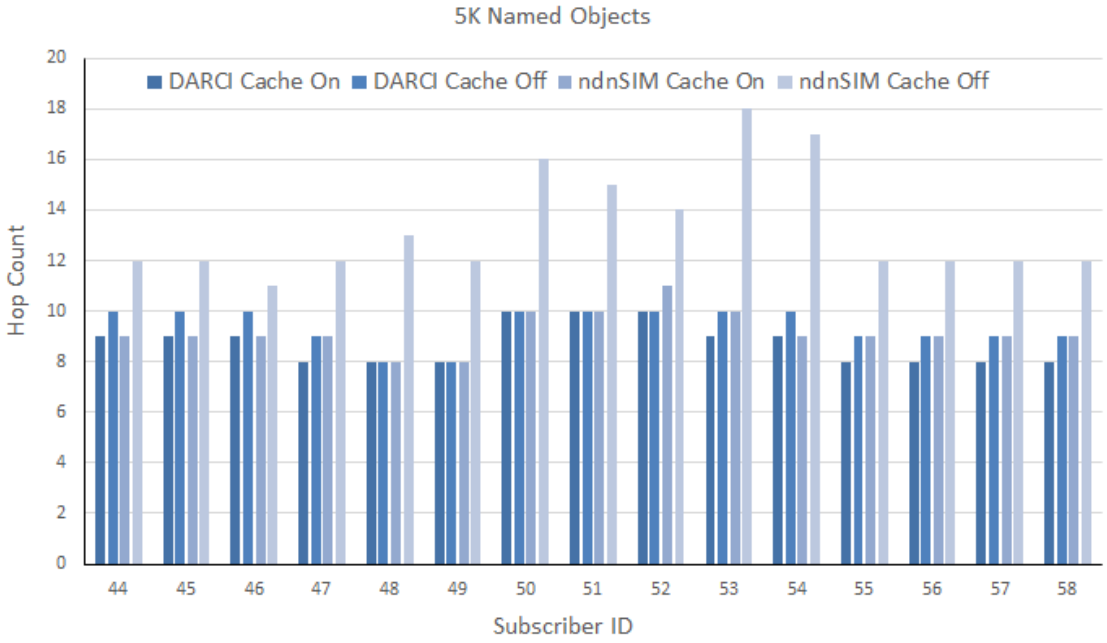

Figure 3.24: DARCI simulation: Hop count with 5K Names

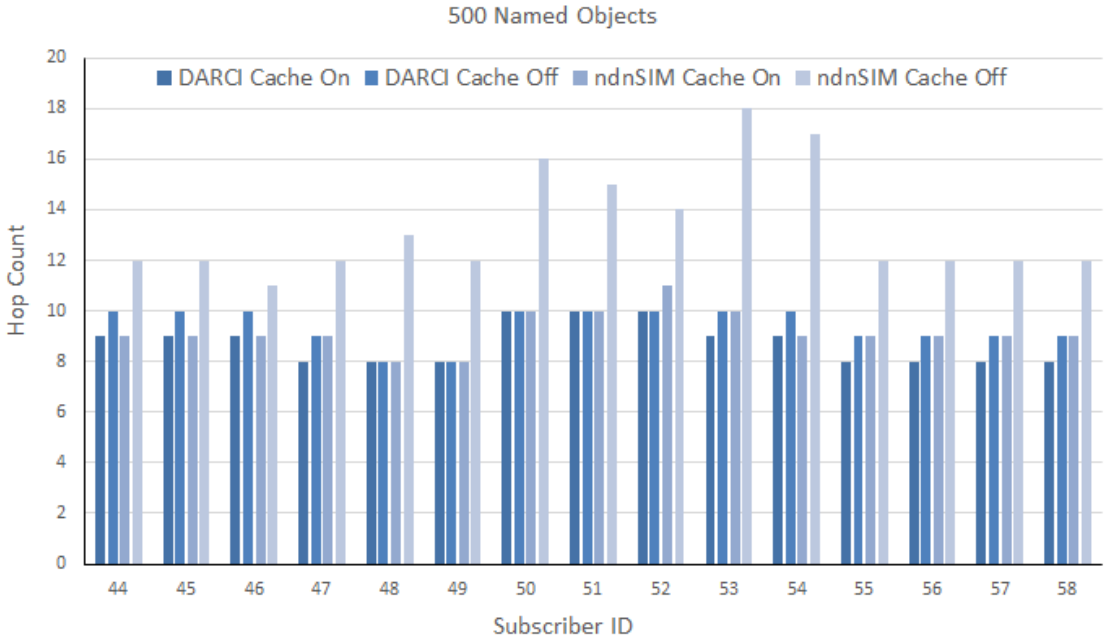

Figure 3.25: DARCI simulation: Hop count with 500 Names

pared to DARCI. The number of FIB and PIT lookups in NDN increases as the number of uniquely addressable contents increases while the requests per second

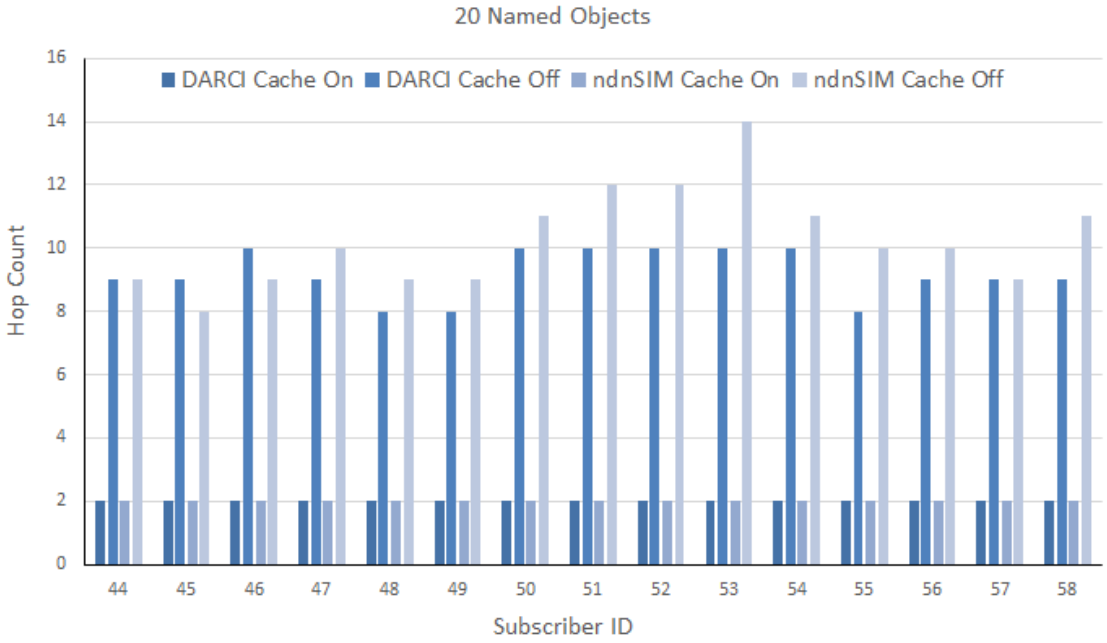

Figure 3.26: DARCI simulation: Hop count with 20 Names

and the size of the content store remain constant in the four scenarios. On the contrary, DARCI's interval routing and prefix labels provide a direction at each hop without any costly lookups. In Figure 3.22, on-path caching produces the lowest delay because contents are retrievable from immediate neighbors due to cache being big enough to hold the 20 unique content names. Figures 3.23-3.26 summarizes average round trip hop count as seen by subscribers for 4 name datasets with on-path and no caching for DARCI and NDN. For Figures 3.23,3.24,3.25 where cache size is smaller than globally addressable content names, DARCI almost consistently maintains the optimum hop count for on-path and no caching scenarios. On the contrary, NDN sees a higher hop count for no caching than on-path caching as the number of globally addressable content names increases. DARCI attributes this consistency to the embedded directional hints carried by the request and response packets. DARCI produces much better consistency in scalability, efficiency, and robustness as the number of addressable content names

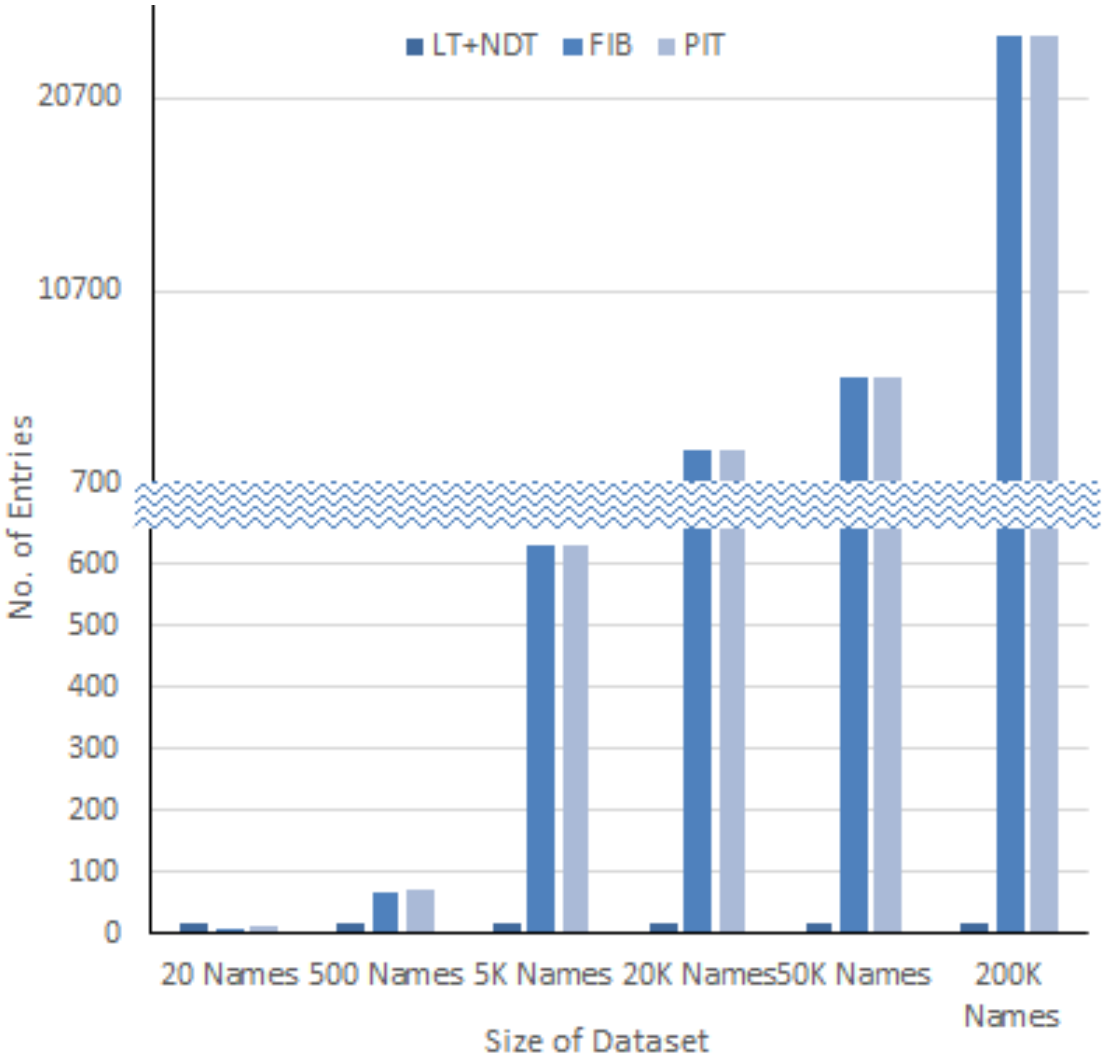

Figure 3.27: DARCI simulation: Table size LT+NDT, FIB & PIT

increases. Figure 3.27 compares the growth in table sizes as the number of addressable contents in the network increases. We monitor all nodes and capture the largest size of LT+NDT, FIB, and PIT table observed for the duration of the simulation. Because DARCI's LT and NDT are a constant size for all the runs, DARCI scales efficiently as the number of contents increases. We see that the FIB and PIT sizes grow dramatically compared to LT and NDT for datasets 5K and more. This correlates with Figures 3.15,3.16 and Figures 3.19,3.20 where

throughput decreases and delay increases for NDN for datasets 20K and 5K.

3.7 Summary

We introduced Directory Assisted Routing of Content in ICN. DARCI routes ICN packets to destination without the need for maintaining stateful tables. Subscribers and publishers issuing requests and responses for a content prefix carry content identifiers such as Landmark Directory prefixes and Anchor Directory prefixes that aide in scalable, robust, and efficient routing at each hop. By simulation using a real-world fixed topology, we see that DARCI performs and scales in order of magnitude larger than NDN.

Chapter 4

MIDAR: Multi-Instantiated contents with Directory Assisted Routing

In this chapter, we introduce Multi-Instantiated contents with Directory Assisted Routing (MIDAR) protocol. As we discussed in chapter 1, the host-centric approach today focuses on reachability to a destination that hosts the contents. In doing so, various algorithmic approaches resort to computing path to a single destination using routing tables and overlays. Various examples of algorithmic design to route to multi-instantiated destinations based on routing to single instance destination have been proposed in [69],[70],[71],[72],[73],[74]. Routing to replicated copies of content in ICN has been surveyed in [16],[17],[18]. In MIDR[75], the authors assert that using algorithms designed for routing to single destination results in unnecessary signaling overhead and complexity, reduced scalability, and incorrectness. As the Internet evolves into an Information-centric network, name resolution and routing of contents with contents being uniquely addressable are

essential parts of ICN. In one of the first proposals for name-based routing, Directed Diffusion[76] uses sender-initiated approaches for multicasting, relying on the flooding of content requests due to requestor's lack of knowledge of the locations of copies of contents. Various ICN architectures as surveyed in [16] either use source routing by way of routers flooding link-state advertisements describing the status of links or prefixes of information for which routers have local copies or by adopting content routing modalities based on distributed hash tables (DHT). It is evident from prior works that re-using algorithms designed for single instance information copies to address multiple instances of information creates more signaling overhead and less reliable solution.

Following the definition in [75], a multi-instantiated destination is a non-empty set of entities denoted with the same unique identifier consisting of a string of alphanumeric symbols. Such an identifier can be drawn from either a flat or hierarchical naming space and can have fixed or variable length depending on the application. MIDAR achieves name resolution and routing to such multi-instantiated destinations by eliminating signaling overhead with zero-flooding and by deriving directions for routing using prefixes of neighbor directories for the publisher of the requested copy of information instance.

4.1 Basic Operation

MIDAR assumes that (1) all routers and hosts in the network have a position independent globally unique identifier, which can be flat or hierarchical (2) parent directory router in the trie assigns prefix label to its child directories (3) all directory routers use a global locality preserving encoding function $\mathscr{F}$ and an interval of code space $[\alpha, \omega]$ where $\alpha < \omega$ (4) named objects NO are uniquely addressable with names that are flat or hierarchical (5) routers cache name resolution

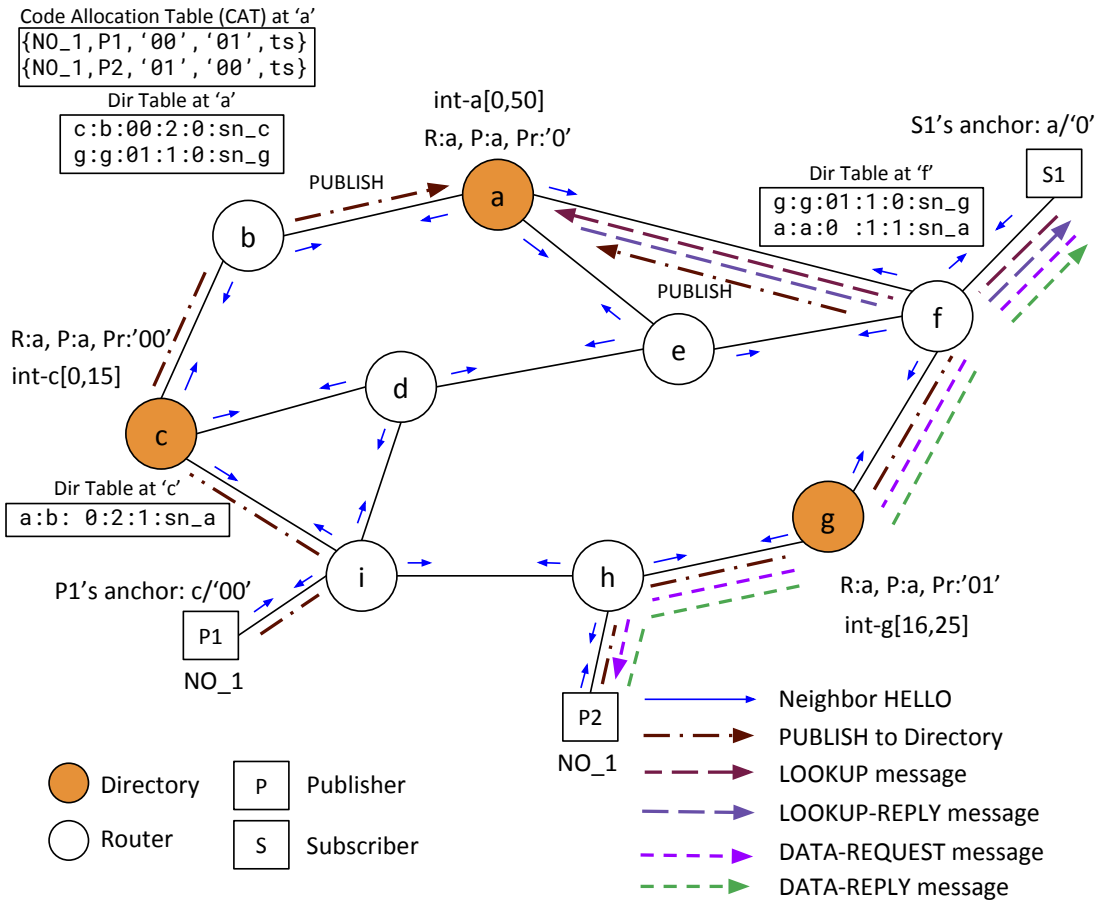

Figure 4.1: Basic operation of MIDAR

responses and contents opportunistically.

Fig. 4.1 illustrates how MIDAR operates. MIDAR first builds a content name resolution system. The name resolution system in MIDAR is a trie of specialized routers called directory router. Publishers publish content to the network by registering it with the trie. Subscribers learn about the available content in a network by looking up the content name in the trie. In the simplified example, a, c and g are directories in a trie with a as the root. a is a parent directory to directories c and g. Directories maintain an authoritative interval of code that is either global if it is a root or a sub-interval from within a parent's interval. In the simplified example, a being the root is authoritative for a global interval

[0,50]. a allocates sub-intervals [0,15] and [16,25] to directories c and g respectively. Directories use a globally known locality preserving encoding function like Hilbert Curve to encode content name to one of the codes in the global interval. A consistent global prefix labeling scheme labels the directories in a trie. The root directory a begins with prefix "0". It assigns labels "00" and "01" to its child directories c and g.

All routers send sequence numbered HELLO messages to its 1-hop neighbors. Routers and directories store their neighborhood information and sequence numbers in neighbor table NT. All routers maintain a sequence-numbered sn distance to directories within r hops. Only a directory can update the sequence number in its following updates. Such a route is loop-free, proved in CORD[44] and ODVR[45]. Routers and directories store sequence numbers in their persistent storage, sequencing increments with every update. On a reset or routing state initialization, $sn = 0$ and distance to directory is ∞. Routers and directories store shortest distance to neighbor directories in diretory table DT. HELLO messages periodically send all updates to a router's neighbor table to neighbors.

Publishers P1 and P2 connect to neighbor router i and h. P1 identifies the closest directory as its anchor directory. P1 sends a PUBLISH message to its router i with the content prefix and prefix label of its anchor directory and landmark directories. Similarly, publisher P2 publishes NO_1 to its anchor directory g. After registering the mapping in its CAT, g forwards the PUBLISH message to its parent directory a by interval routing. The CAT for a in the figure shows two entries for NO_1 exists - one for each publisher P1 and P2 with their respective anchor and landmark directory prefixes.

A subscriber S1 requesting content NO_1 sends a name resolution LOOKUP message to its closest directory a on the trie. If a is also the anchor directory for

the publisher of content NO_1 then a replies to S1 with the mapping of content name to the local ID of the publisher and prefixes of its anchor and landmark directories. Directory a replies to S1 with LOOKUP-REPLY with all known instances of NO_1. It returns from its CAT the local ID P1 and P2 of publishers P1 and P2 and their corresponding anchor and landmark directory prefixes.

Content routing in the data plane is assisted by the name resolution system of DARCI and the publish-subscribe signaling between routers and directories in a trie. In the example, S1 receives mapping of NO_1 to P1 and P2. S1 derives the direction for the shortest path by comparing the known prefix labels for content NO_1 with the prefix labels in its DT. S1 sends a DATA-REQUEST with the content name, prefix labels of its anchor, and landmark directories towards publisher P2 via directory g with the prefix "01". Publisher P2 replies with DATA-REPLY with requested content towards subscriber S1. P2 derives the direction for the shortest path to S1 by sending DATA-REPLY in the direction of a directory with the longest prefix match. In the example, P2 sends DATA-REPLY towards g. Local router f delivers the content to subscriber S1.

The operation of MIDAR in the control plane is independent of the data plane after a router learns of the prefix labels of anchor and landmark directories. Routers forward data requests and reply messages in the path of the longest prefix match to reach the destination without the need to maintain long stateful tables. MIDAR's support for multi-instantiated contents in the network automatically creates load-balancing of traffic as data request, and response gets routed to the closest prefix labels of destination anchors.

4.2 Multi-Instance Publish & Subscribe

MIDAR extends DARCI's use of directory trie to partition the global code space to map multiple instances of content. Unlike peer to peer or FIB mechanisms that does not scale when it comes to multiple instances of contents and for scenarios where contents are created as a result of flash-crowd at Internet scale[46][47][48][49][50][51], the directory trie retains the map of contents within the network with reduced signaling overhead by allowing publishers to periodically update its ownership of the content by using directed publish messages towards the trie as described in chapter 3.

4.2.1 Information Stored & Exchanged

We use the same notations as found in chapter 3. A router directory a maintains the following tables: (a) *a neighbor table* $[NT]^a$ listing router a's immediate neighbors; (b) a *directory table* $[DT]^a$ to store routing information to known directory routers as learned from neighbor HELLO; (c) a *child directory table* $[CDT]^a$ to store child directories of a (d) a *named object routing information* $[NORI]^a$ table to store named object NO routing information for each known named object seen by router a; (e) a *cached named object pointer* $[CANOP]^a$ at router a to store the mapping of NO names to the pointer to the cached data store; (f) a *code allocation table* $[CAT]^a$ to store space-filling curve code allocation for registered NO at router a.

The adaptation to DARCI's information storage to support MIDAR is in allowing $[NORI]^a$ and $[CAT]^a$ to support multiple instances of destination information object. One of the main aspects of MIDAR is the ability to distribute code spaces amongst its directory routers that form the trie. MIDAR uses space filling codes to map named objects and named object profiles. For a given code space, direc-

tory router a is assigned a range by its parent for which a is the authoritative responder for any query for named objects that maps to one of the codes in that range. For publisher nodes x and y as authoritative publishers of named object $NO^{\star}$, the code generated $\mathcal{C}$ at each directory router to determine the interval direction for publishing is respectively

$$\mathcal{C}_x^{NO^{\star}} = \mathcal{S}(NO^{\star}, \mathcal{A}^x, \mathcal{L}^x_{[0...n]})$$

$$\mathcal{C}_y^{NO^{\star}} = \mathcal{S}(NO^{\star}, \mathcal{A}^y, \mathcal{L}^y_{[0...n]})$$

Where for publisher nodes x and y, $\mathcal{A}^x$ & $\mathcal{A}^y$ are the anchor directory, $\mathcal{L}^x_{[0...n]}$ & $\mathcal{L}^y_{[0...n]}$ are a list of n landmark directories $\mathcal{L}$ prefixes known by publisher nodes at the time of registering and publishing with its respective anchor directory $\mathcal{A}^x$ & $\mathcal{A}^y$.

In $[CAT]^a$, a code points to Named Object Profile Set (NOPS). A NOPS for a given named object $NO^{\star}$ published by node p is a set of one or more Named Object Profile (NOP), which consists of the following tuples:

$$(NOP^{NO^{\star}})_a = (p, \mathcal{A}^p, \mathcal{L}^p_{[0...n]}, (sn)^p, t_i)_{[0...m]}$$

Where $\mathcal{A}^p$ is the anchor directory for publisher node p, $\mathcal{L}^p_{[0...n]}$ is a list of n landmark directory $\mathcal{L}$ prefixes known by publisher node p at the time of registering and publishing with its anchor directory $\mathcal{A}^p$, $(sn)^p$ is a sequence number generated by publisher node p and t_a is the time-stamp generated for this record by Directory a. For a given named object, up to m Named Object Profiles can exist, thus supporting multi-instances of named objects in network.

MIDAR allows opportunistic caching of contents in the network. The $[CANOP]^a$ database keeps track of named object mapping to its corresponding storage pointer. A $[CANOP]^a$ entry consists of named object prefix $NO^{\star}$, a pointer to content stor-

age PTR and a timestamp t_a. $[NORI]^a$ at node a for MIDAR supports multiple entries for Named Object Profile $NOP^{NO^\star}$ based on destinations.

Non-directory routers are an integral part of directory trie. They facilitate efficient and robust routing of name resolution requests and contents by using knowledge of surrounding directory routers. A router b maintains (a) *a neighbor table* $[NT]^b$ listing router b's immediate neighbors; (b) a *directory table* $[DT]^b$ storing routing information to known directory routers as learned from neighbor HELLO; (c) a *named object routing information* $[NORI]^b$ table to store named object NO routing information for each known named object seen by router a; (d) a *cached named object pointer* $[CANOP]^b$ at router b to store the mapping of NO names to a pointer to the cached data store.

4.2.2 MIDR Example

Publish Operation

Publishing in MIDAR consists of publishers forwarding PUBLISH messages to their neighbor router. The PUBLISH message contains the content name prefix, a local ID of the publisher, and a sequence number. Fig. 4.2 is an example of publisher P0 connected to router b and publisher P1 connected to the router. Neighbor router b and s for P0 and P1 forwards the local publish message to the closest directory router a and t respectively in the trie by adding labels of directory routers within its r hops. The local publishing path is labeled L in Fig. 4.2. The directory router a on receiving the local publish message becomes the anchor directory $\mathcal{A}$ for the publisher P0. Directory a replies to publisher P0 with PUBLISH-ACK. The local reply path for PUBLISH-ACK is labeled LR in Fig. 4.2. Local routers and directory for a publisher maintain a loop-free path to the publisher using neighbor HELLO. These routes are periodically refreshed based

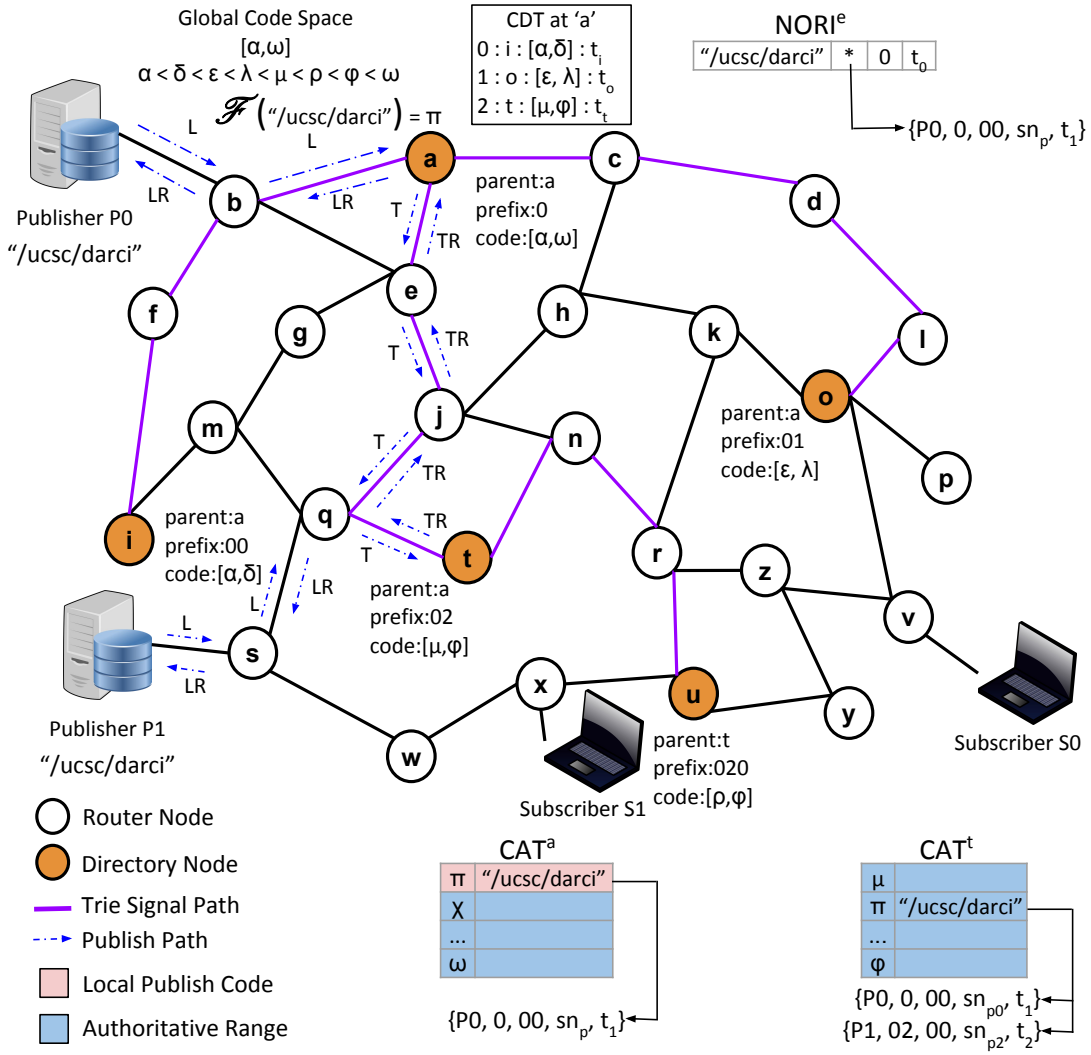

Figure 4.2: Publishing in MIDAR

on HELLO intervals. Anchor directory a computes the space-filling curve code for the content name using the transform $\mathscr{F}(NO) = \pi$, where $\mathscr{F}$ is as described in Algorithm 7. The mapping of content name to a publisher in the local publish message is registered in anchor directory a's CAT^a. Directory a retrieves the interval for π by looking in the CDT^a. In the example in Fig. 4.2, $\pi \in [\mu, \phi]$ is in an interval assigned to directory t. The publish message is forwarded in the trie towards the prefix 02 of authoritative directory t. Fig. 4.2 shows the trie path

for publishing and replies to t and a with label T and TR, respectively. Each router in the trie path from a to t forwards the PUBLISH message and caches the mapping (e.g., $NORI^e$ at router e). Fig. 3.10 shows the local and authoritative registration of the mapping in CAT^a and CAT^t pointing to named object profile (NOP). Directory router t is the anchor and authoritative directory for publisher P1. The $[CAT]^t$ with two NOP for multi-instantiated "/usc/darci" is shown in Fig. 4.2. Directory t replies to publisher P1 with PUBLISH-ACK. The local reply path for PUBLISH-ACK is labeled LR in Fig. 4.2. Publishers periodically refresh the content they own by sending PUBLISH messages towards the trie via their neighbor router. For a set of already published and unchanged contents, publishers send a compressed message that contains the signature to identify the set of unchanged content names. Periodic refresh by publishers addresses updates in mapping to anchor directory prefixes when trie structure changes.

Subscribe Operation

Subscribers register with neighbor routers using HELLO messages. All neighbor and directory routers within "r" hops maintain a loop-free sequence numbered route to subscribers. A subscriber interested in content sends a LOOKUP message containing the name of the content, a local ID of the subscriber, and a sequence number to its neighbor router. The neighbor router forwards the LOOKUP message to the closest directory. This directory is the anchor directory for the subscriber. At the anchor directory, the code for requested content is computed and checked for its availability in the local cache. The anchor directory sends LOOKUP-REPLY with the mapping to the subscriber. Anchor directory determines the direction for the interval in which the code is. It marks the LOOKUP message with its label prefix as the source of the request

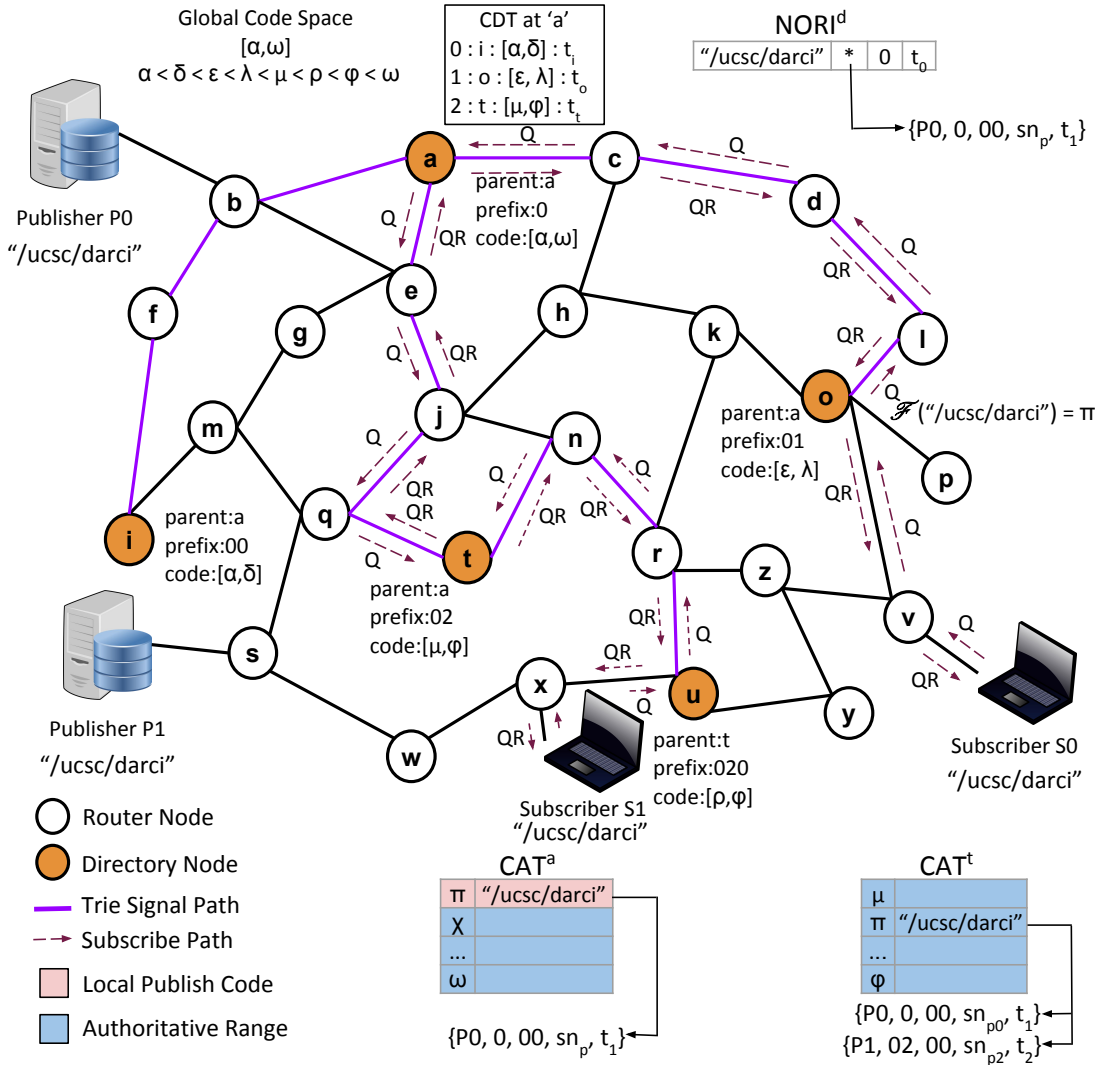

Figure 4.3: Subscribing in MIDAR

and forwards the LOOKUP message using interval routing in the trie towards the authoritative directory that owns the interval. The authoritative directory replies with LOOKUP-REPLY containing the mapping for the requested content. The LOOKUP-REPLY follows the trie path towards the source anchor directory prefix using prefix label routing. Intermediate routers in trie that forward the LOOKUP-REPLY opportunistically cache the mapping information in its *NORI* table. Anchor directory on receiving LOOKUP-REPLY updates its *NORI* table

and forwards it to the subscriber. Fig. 4.3 is an example of subscribers S0 and S1 connected to routers v and x, and requesting content "/ucsc/darci". Anchor directory o and u for S0 and S1 respectively computes the space-filling curve code for the content name using the transform $\mathscr{F}(NO) = \pi$, where $\mathscr{F}$ is as described in Algorithm 7. $\pi \notin [\epsilon, \lambda]$, anchor directory o forwards the LOOKUP message to its parent directory a. Anchor directory u for S1 forwards the LOOKUP message to its parent directory t. The LOOKUP message path from anchor directory to the authoritative directory through trie using interval routing is labeled Q in the figure. The LOOKUP-REPLY message path to subscriber's anchor directory via trie using prefix label routing is labeled QR in the figure. Intermediate router d in the trie stores the mapping information in its $NORI^d$ table.

Content Routing

A subscriber receives all mappings for requested content via LOOKUP-REPLY. Fig. 4.4 is a routing example for multi-instantiated content in the trie. Subscribers S0 and S1 finds the shortest path to reach the anchor directories of publishes P0 and P1. Subscriber S0 sends a DATA-REQUEST message to its neighbor router v by embedding the content name's mapping to its locator and landmark prefixes. Subscriber S1 learns from its NT that publisher P1 is within r hops; sends a DATA-REQUEST message to its neighbor router w by embedding the mapping of content name to its locator and landmark prefixes. At every hop, routers inspect the destination prefixes and determine the shortest route to reach them. Routing prioritizes on reaching anchor prefix. If no path exists, the default path is to forward the DATA-REQUEST message to the neighbor directory router with the longest prefix match towards a publisher's anchor directory prefix. Intermediate routers check the cache CANOP for a local copy of the requested content.

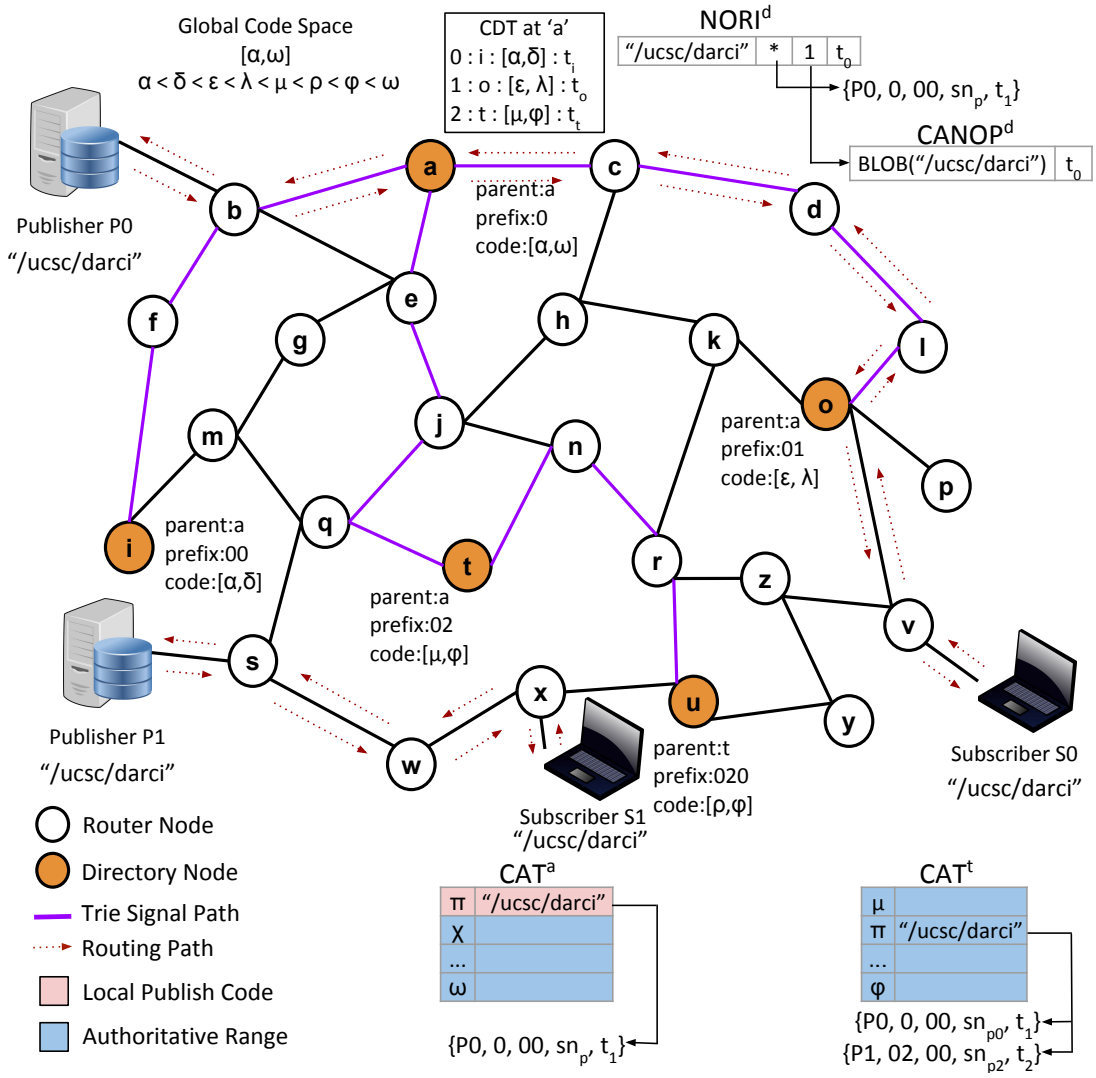

Figure 4.4: Content routing in MIDAR

Routers reply with a cached copy in the DATA-REPLY message towards the subscriber's anchor prefix. If trie is the shortest path derived to reach the publisher's anchor directory, then the DATA-REQUEST message is forwarded hop by hop using prefix label routing. An intermediate router or directory can redirect the DATA-REQUEST to a shorter path if one is available after consulting its NT and DT. DATA-REQUEST from S0 follows the trie path from *o* to publisher P0 via anchor directory *a*. However, DATA-REQUEST path from S1 follows local

routers x to reach publisher P1 via s. Publisher replies with requested content in the DATA-REPLY message. The neighbor router determines the next hop to reach the prefix of the subscriber's anchor directory. The default next hop is towards the closest directory router with the longest prefix match towards a subscriber's anchor directory prefix. The anchor directory on receiving DATA-REPLY forwards to the subscriber via local routers. DATA-REPLY is forwarded via local routers to destination if a subscriber is known to the publisher within r hops. Intermediate routers and directories opportunistically cache content data in their CANOP cache.

4.3 Simulation

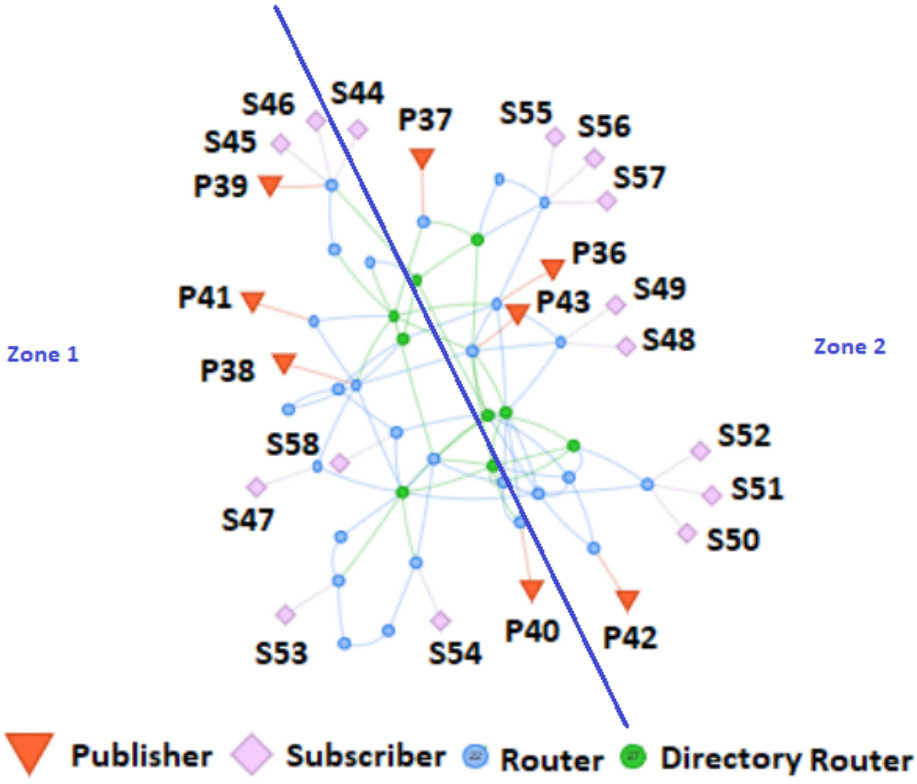

Figure 4.5: MIDAR simulation: 59 node BT North America topology

We implemented MIDAR in ns-3 version 3.27[65] and used the NDN implementation [66] for validating protocol correctness and to compare performance metrics of average throughput, average delay and average round trip hop count as seen by subscriber nodes for increasing number of content names. We derived a 59 node fixed topology as shown in Figure 4.5 from "BT North America" dataset[67]. Five unique information name object files each 50K, 20K, 5K, 500 and 20 names were derived from web traffic dataset[68]. The simulation ran for 3 hours with 5 different seed values, with subscribers requesting 25 names per second. Publisher applications on publishers 36-43 were allowed to start publishing the contents they owned before subscriber applications began requesting information. Publishers are assigned the list of unique names such that a given publisher in MIDAR and NDN had the same set of names. Additionally, as shown in figure 4.5, 8 publishers were virtually split into two zones such that for a given information name, there were at least 2 publishers one in each zone. Our intention here was to allow subscribers in either zone to reach the closest instance of the requested information dynamically. Subscribers 44-58 were aware of the complete name list, out of which each subscriber randomly selects names. We compare performance metrics for name data files 20K, 5K, 500, and 20. We evaluate the two protocols in a wired network with fixed topology. In MIDAR, we use a maximum distance of 2 hops for a publisher or a subscriber to select an anchor directory. Nodes in MIDAR and NDN are connected using point-to-point CSMA links with a data rate of 1Gbps, delay of 6.5 ms, and MTU size of 1400. For NDN, the size of the content store is 100 for on-path caching and set to a size of 1 for no caching scenarios. The caching policy for NDN is LRU. For MIDAR, the size of CANOP is set to 100 for on-path caching and set to 0 for no caching scenarios. The caching policy in DARCI is LRU, and the routing strategy is "/localhost/nfd/strategy/multicast".

The subscribers and publishers are nodes with no caching enabled.

4.4 Performance Comparison

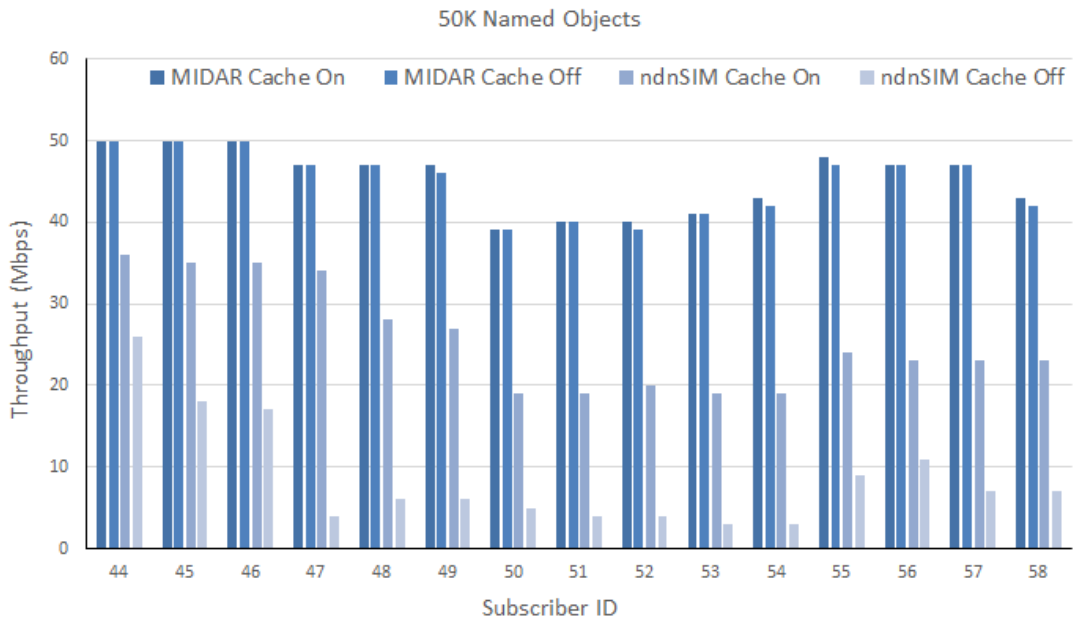

Figure 4.6: MIDAR simulation: Throughput (Mbps) with 50K Names

Figures 4.6,4.7,4.8,4.9,4.10 shows throughput seen by subscribers for 5 different data sets. For 20 names dataset, we see the throughput for MIDAR and NDN with cache enabled to be higher than for cache disabled. Since the cache is large enough to hold all the names, we see nearly consistent throughput with 20 names because subscribers are returned a cached copy of the information with a round trip hop count of 2. However, when the cache is disabled, the request for information has to reach the destination instance. MIDAR's throughput computation includes the round trip time taken for name resolution and round trip time to receive the requested information. For Figures 4.6,4.7,4.8,4.9 with cache enabled and disabled, the rate of decrease in throughput as the number of addressable

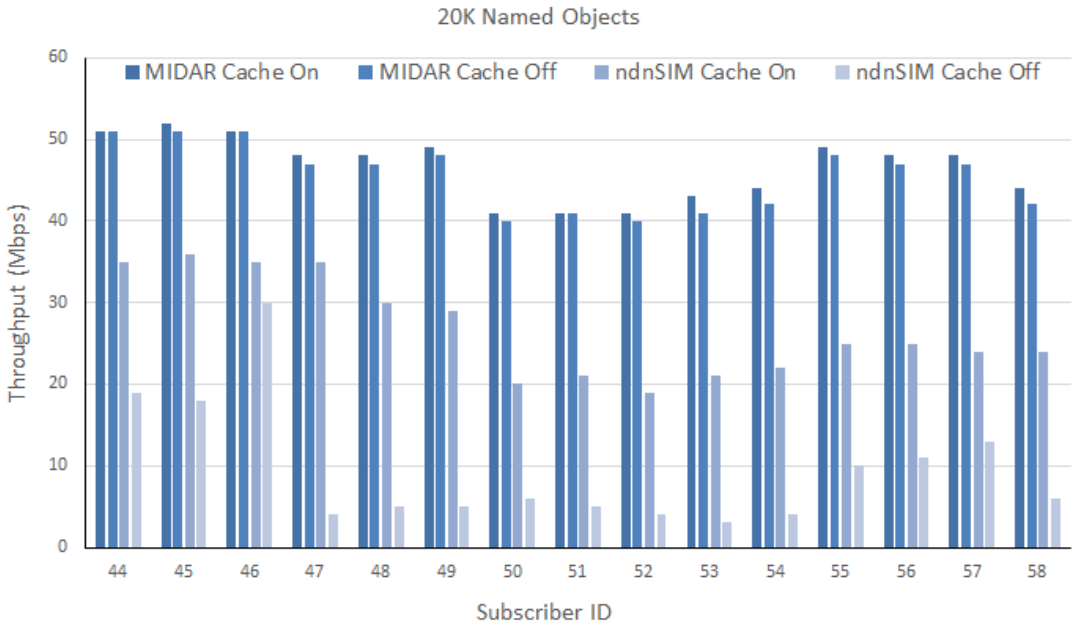

Figure 4.7: MIDAR simulation: Throughput (Mbps) with 20K Names

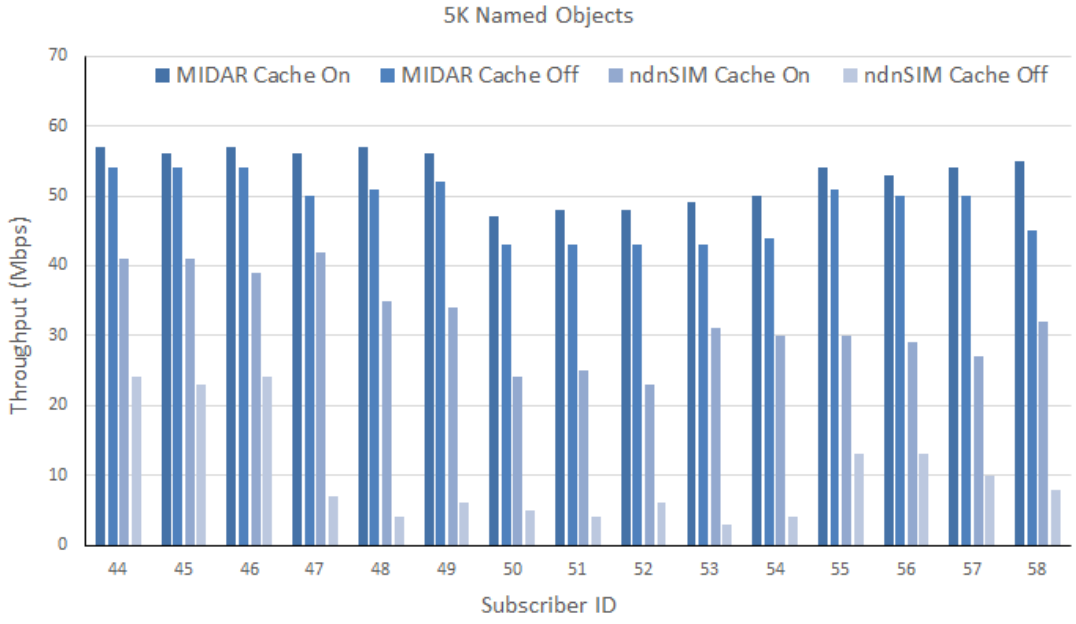

Figure 4.8: MIDAR simulation: Throughput (Mbps) with 5K Names

information names increase in the network is smaller in MIDAR compared to the large difference seen in NDN. This is because, in MIDAR, the name resolution

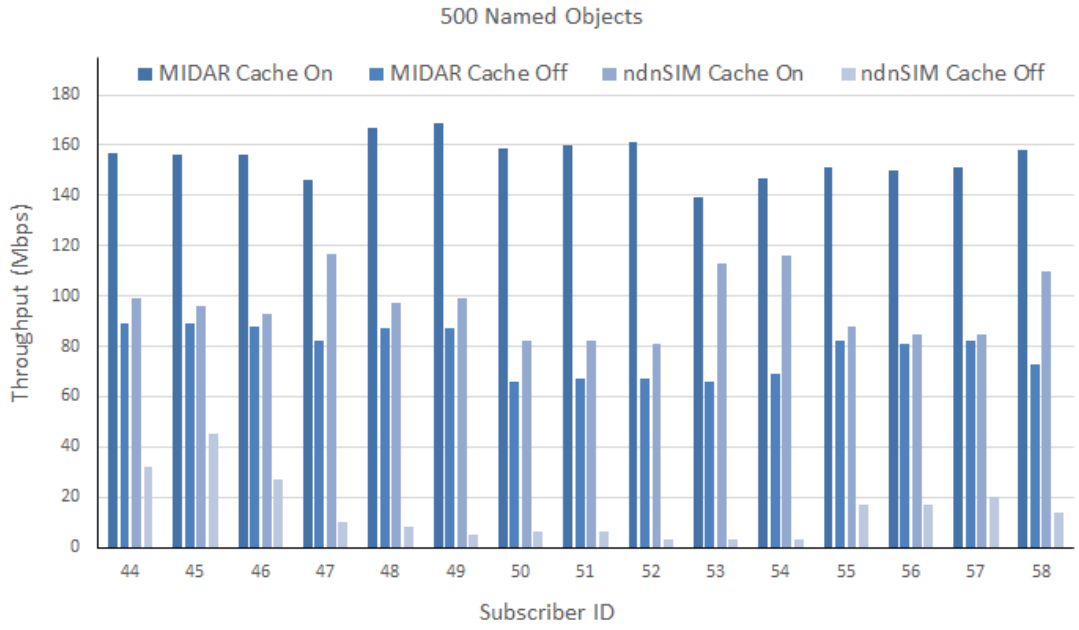

Figure 4.9: MIDAR simulation: Throughput (Mbps) with 500 Names

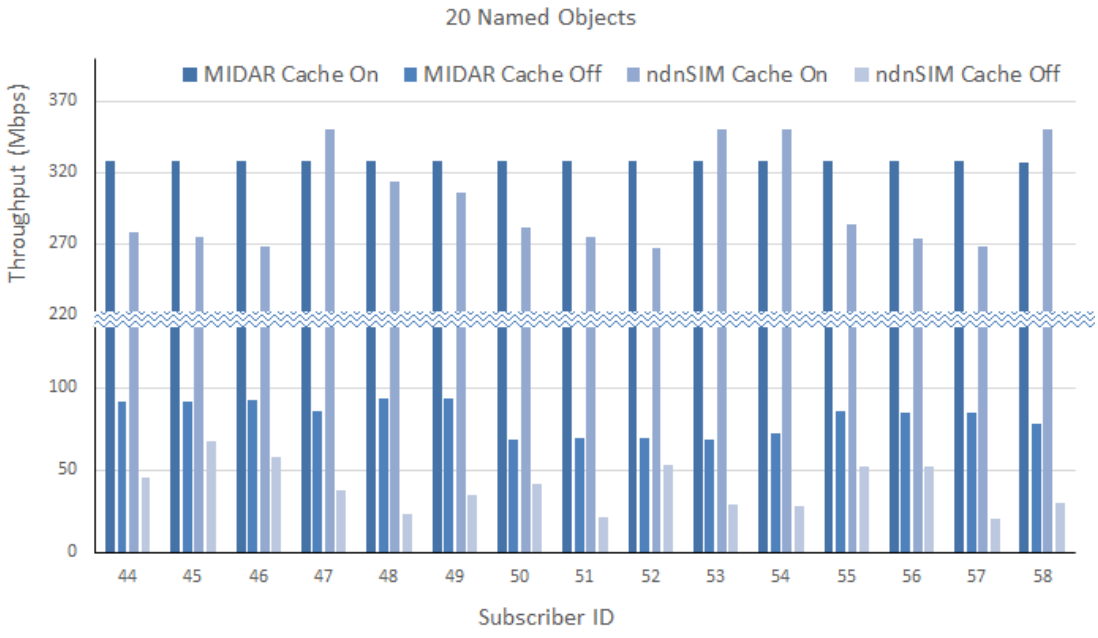

Figure 4.10: MIDAR simulation: Throughput (Mbps) with 20 Names

through `opus-lookup` returns information on all known multiple instances of information destination. Subscribers in MIDAR derives a direction for request by

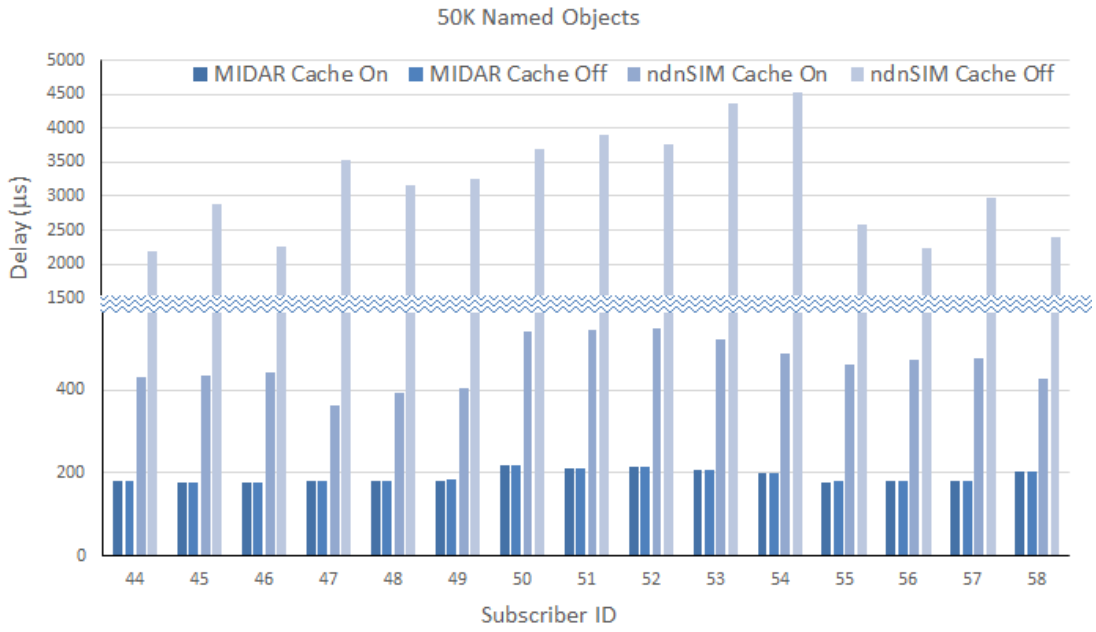

Figure 4.11: MIDAR simulation: Delay (μs) with 50K Names

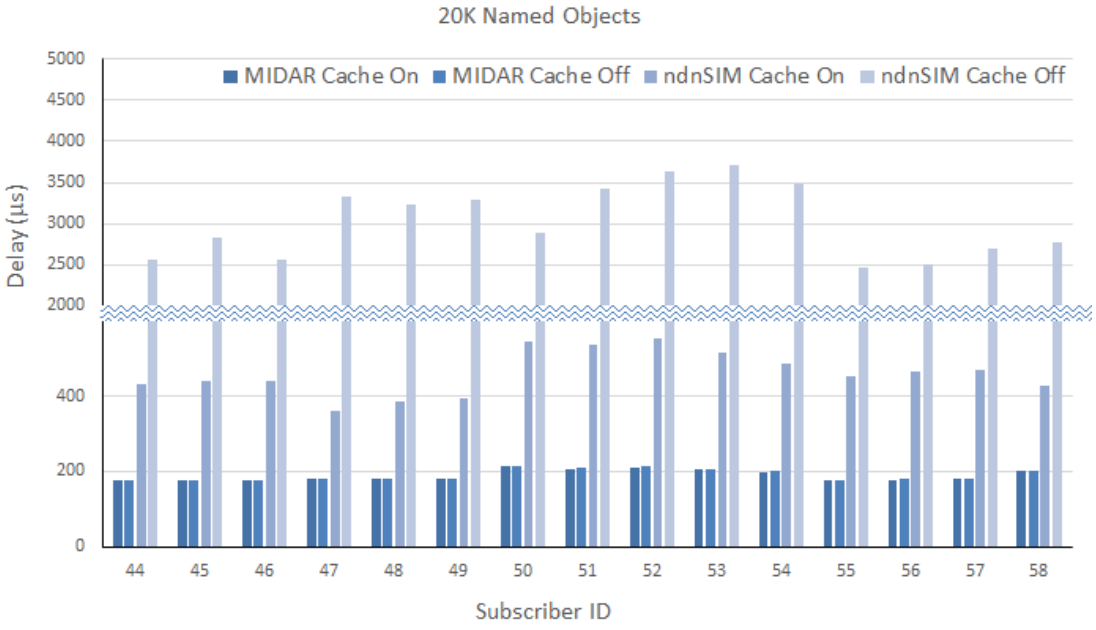

Figure 4.12: MIDAR simulation: Delay (μs) with 20K Names

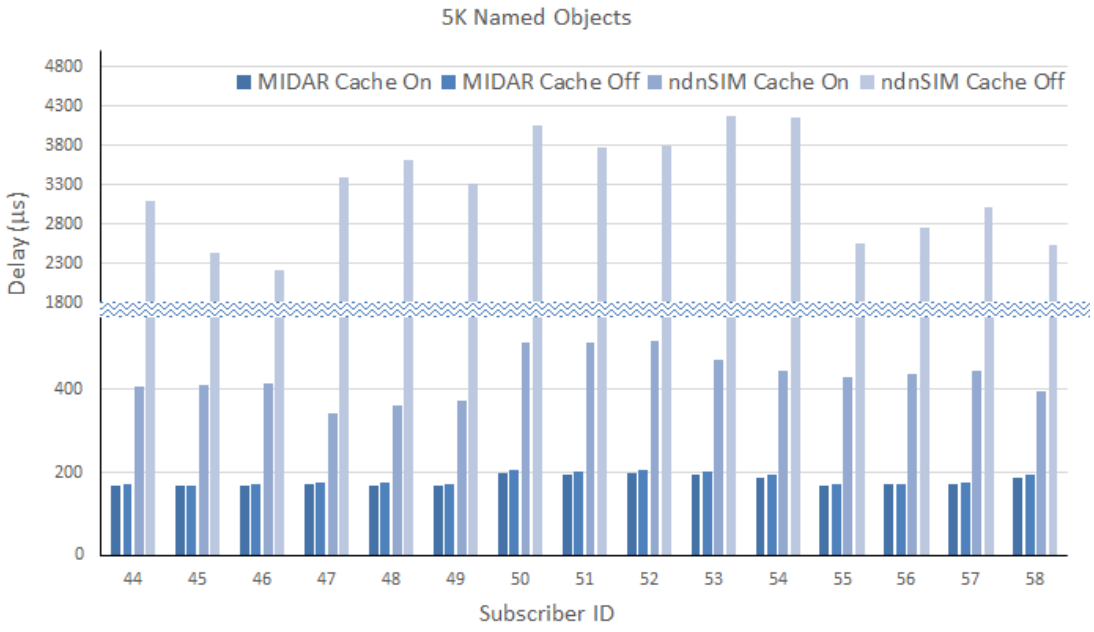

Figure 4.13: MIDAR simulation: Delay (μs) with 5K Names

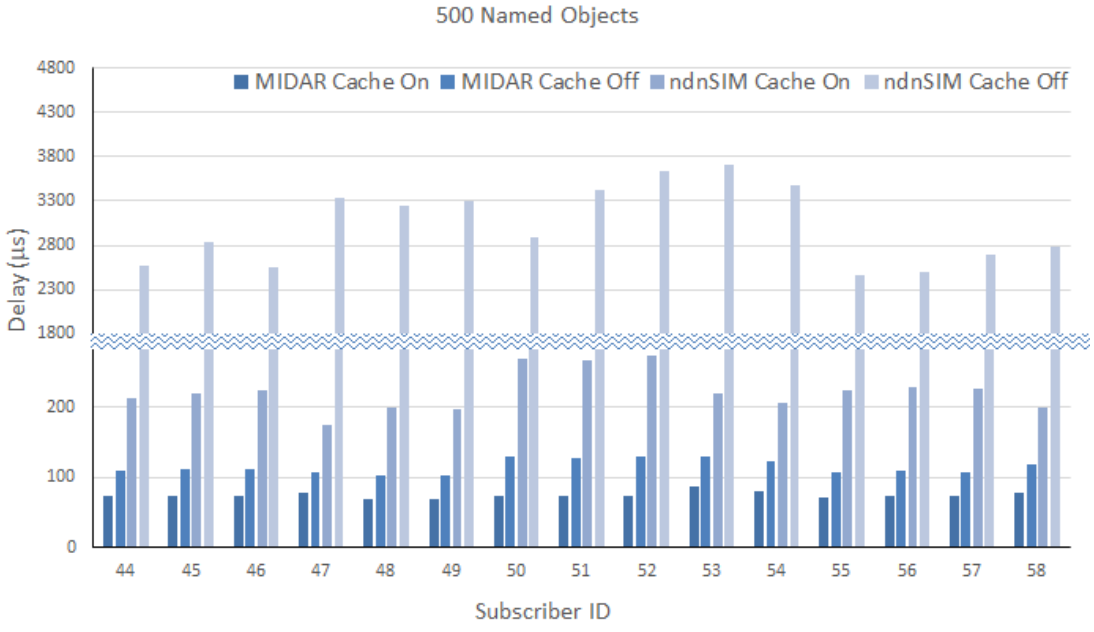

Figure 4.14: MIDAR simulation: Delay (μs) with 500 Names

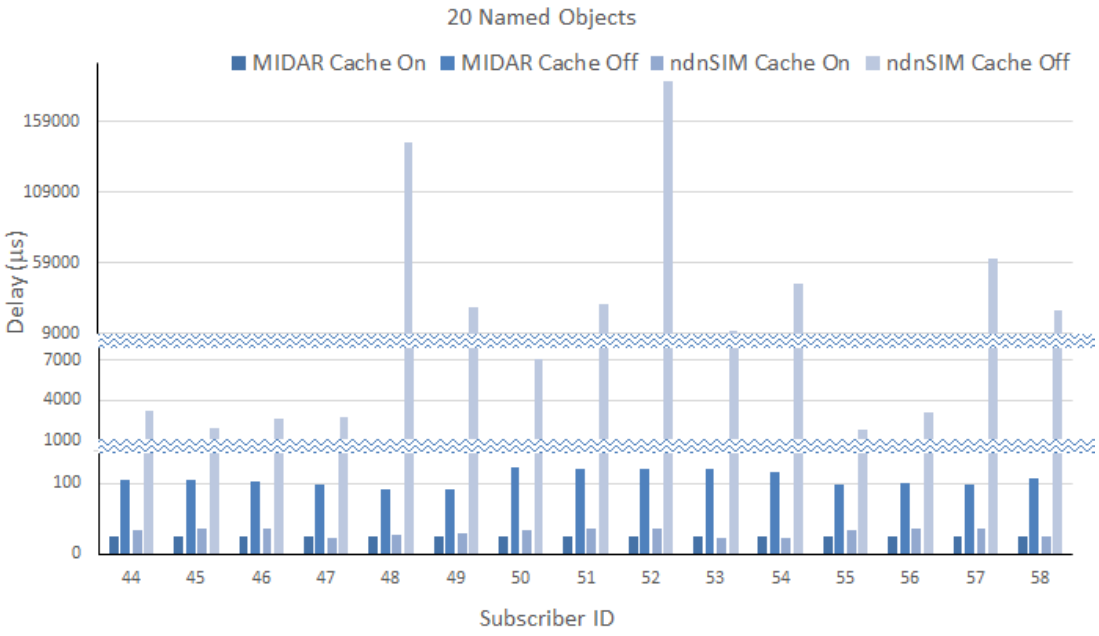

Figure 4.15: MIDAR simulation: Delay (μs) with 20 Names

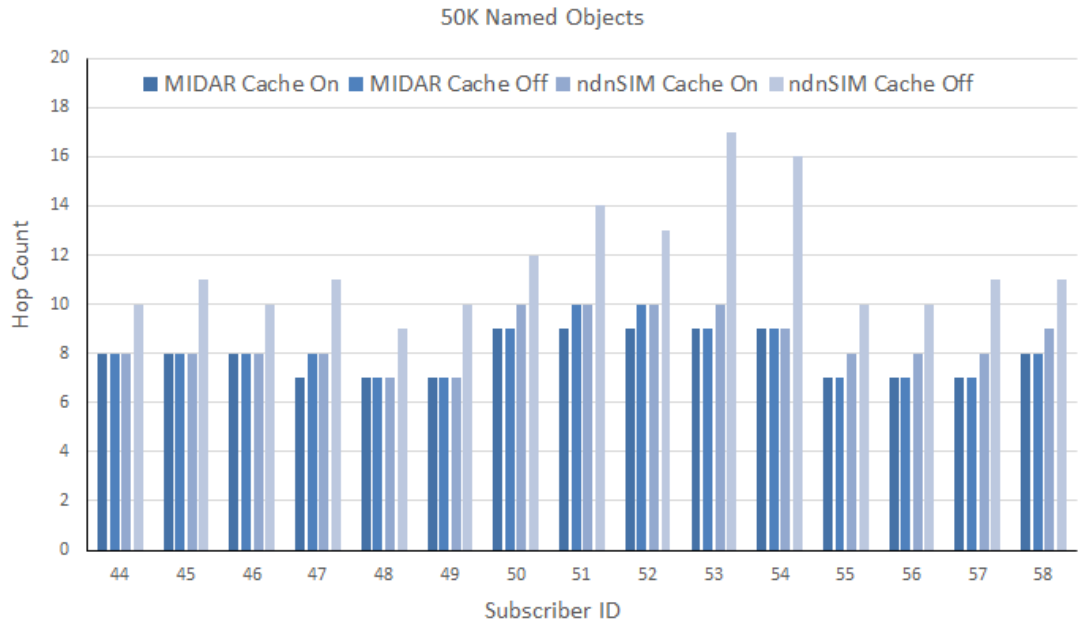

Figure 4.16: MIDAR simulation: Hop count with 50K Names

comparing its own known Anchor Directory and Landmark Directories with each of the known destination Anchor Directory and Landmark Directories as returned

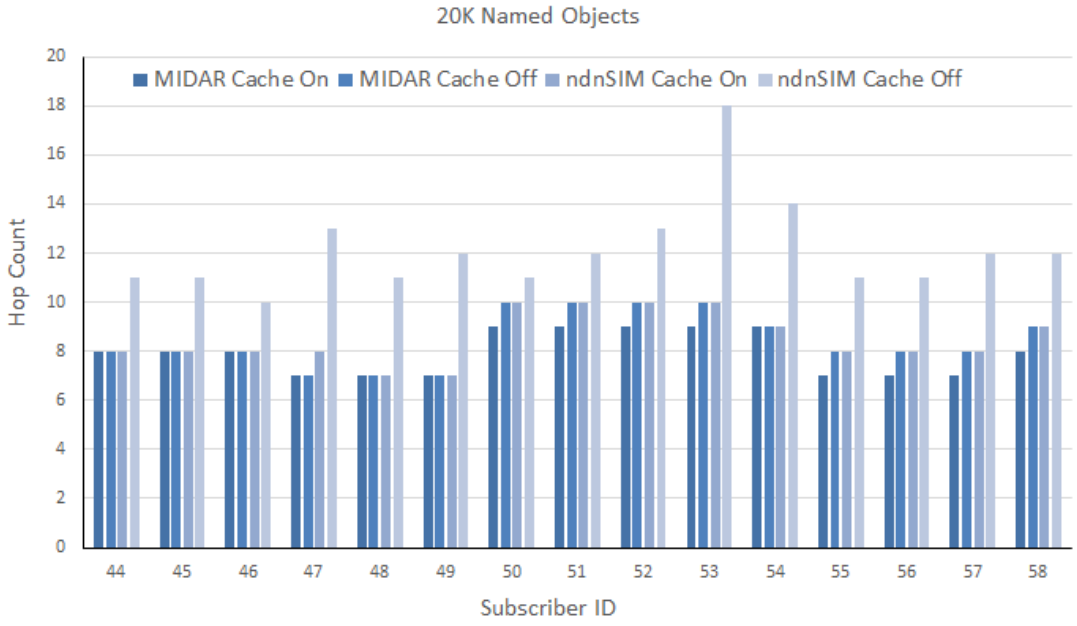

Figure 4.17: MIDAR simulation: Hop count with 20K Names

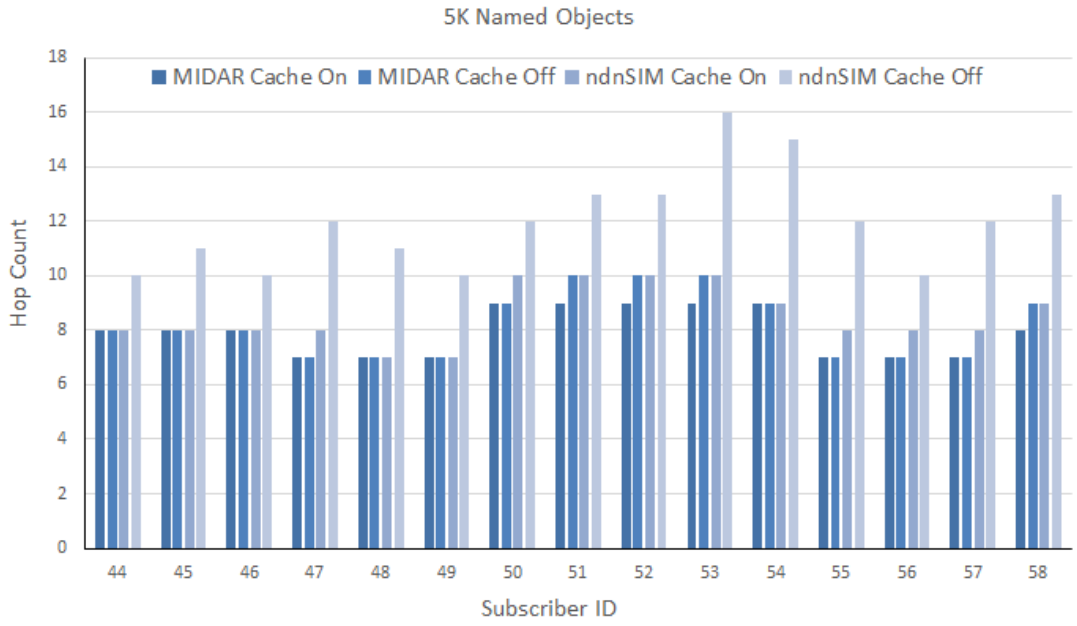

Figure 4.18: MIDAR simulation: Hop count with 5K Names

by `opus-lookup-reply`. Note that a subscriber derives a direction, and it is not an assured shortest path. The request packet carries with it prefix labels of publisher

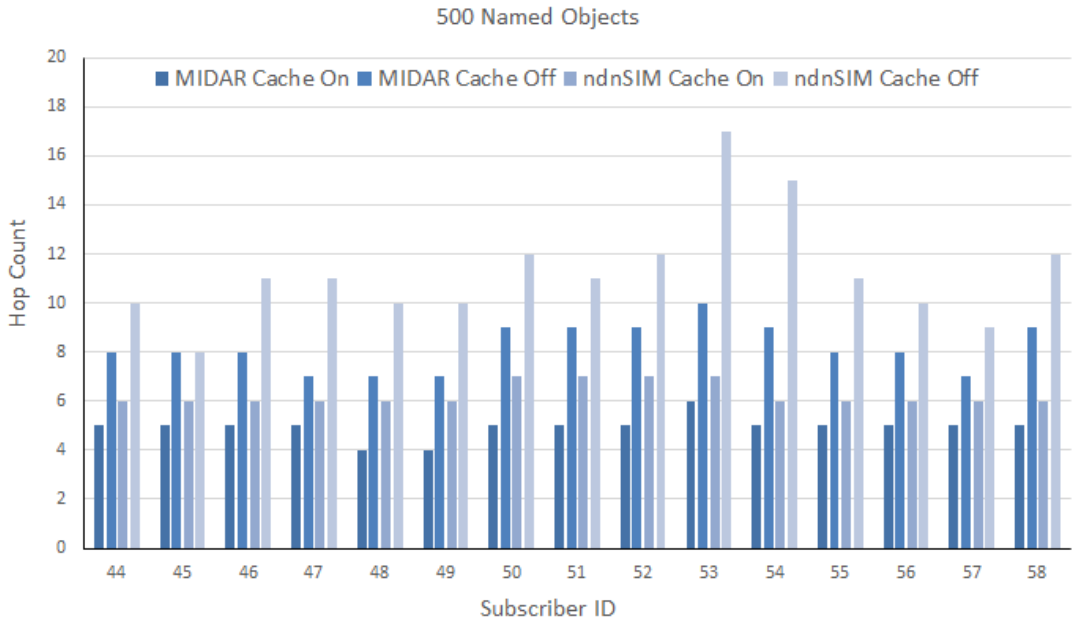

Figure 4.19: MIDAR simulation: Hop count with 500 Names

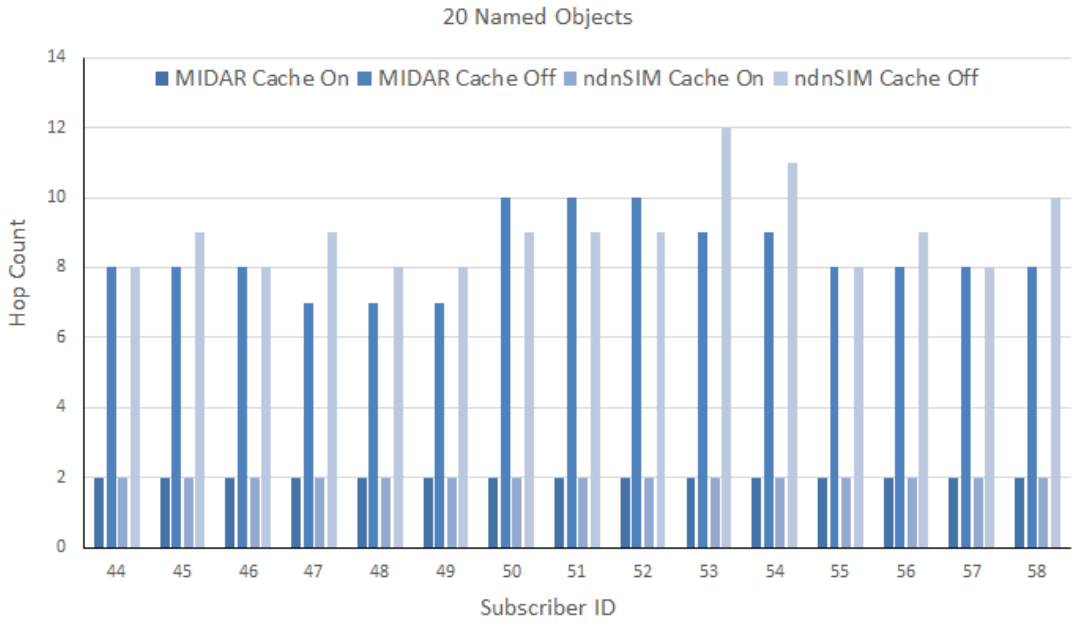

Figure 4.20: MIDAR simulation: Hop count with 20 Names

anchor directory and landmark directories such that at each hop, there is always an opportunity to optimize the routing until the request reaches the publisher.

The correctness of this design in MIDAR is proved by the simulation results in Figures 4.11-4.20. Figures 4.16,4.17,4.18,4.19,4.20 shows that both MIDAR and NDN for cache enabled have almost similar round trip hop count measures for data set sizes over 5K names. However, the corresponding delay measurements show that MIDAR incurs far less delay than NDN. In NDN, a smaller round trip hop count does not necessarily mean smaller delay because of PIT's pending interests. On the other hand, in MIDAR, the decoupling of name resolution and interval routing for stateless lookup, request, and response allow for faster routing with minimal overhead. Another key observation from the simulation results is that MIDAR scales far better in terms of delay and throughput than larger drops seen in NDN as the number of addressable information names increases in the network.

4.5 Summary

We introduced MIDAR for multi-instantiated information name resolution and routing in ICN. By decoupling the name resolution from the routing plane, MIDAR achieves greater scalability in deriving the initial the direction for routing by comparing its known anchor directory and landmark directories with each of the known destination anchor directory and landmark directories it has learned via name resolution. By way of extensive simulation, we have shown that MIDAR scales better than NDN when the number of addressable information increases. MIDAR incurs an order of magnitude far less control overhead than NDN, which relies on mechanisms like broadcast to populate FIB. The increase in delay, as seen in NDN, could be explained by the architecture of NDN [19] where routers manage traffic load by managing the Interest forwarding rate on a hop-by-hop basis; where it throttles or stops sending Interest packets to a neighbor. In MIDAR,

forwarding is instantaneous in order to maintain stateless functioning. Simulation comparison between MIDAR and NDN on a real-world topology validates the efficiency and scalability of MIDAR as the number of addressable contents and the number of instances of those contents increases in an ICN.

Chapter 5

ACED: Adaptive Cache Enabled Directory Assisted Routing

In this chapter, we introduce Adaptive Cache Enabled Directory Assisted Routing of Content in ICN (ACED). The preamble of ICN[77] states the independence of data from location, application, storage, and the means of transportation. By way of uniquely addressing data, in-network caching and replication becomes an essential design area for network performance improvement. Whether host-centric or information-centric, caching of in-need data close to the subscriber has always been the de-facto design to increase network performance. As discussed in chapter 1, the exponential growth of user-generated content proved that fetching content from the source for every request is never optimal. Web performance significantly increases when contents are cached between the subscriber and publisher[78][79][80]. The several key advantages of any caching are [81]:

1. Network traffic and congestion decreases with web caching thereby increasing bandwidth

2. Publisher load is greatly reduced as it does not need to serve all requests

3. For documents not found in the cache, the traffic to reach publishers flows much faster due to the reduced number of overall requests to publishers

Various caching mechanisms have been proposed for ICN to improve network efficiency. Apart from the basic caching policies such as First In First Out (FIFO), Least Recently Used (LRU) and Least Frequently Used (LFU) for ICN, caching schemes based on path and data information[82], caching schemes based on ICN network parameters such as distance from the source, reachability of the router, frequency of content access[83] and caching schemes that assigns a transportation cost to every node in the network which is derived as a function of cache hit probability of the content at a particular node[84] have been proposed. Authors of [85] have used cache hit/miss ratio to characterize the performance of cache and propose a multi-path routing strategy and later concluded that while simple randomized policies may perform almost as well as more complex ones, a multi-path routing may play against a content-centric network efficiency. Drawing upon the observation of these prior work, we focus exclusively on "adaptive" load sharing by the cache anywhere between the publisher and subscriber. That way, a cache can opportunistically take on the role of a multi-instantiated publisher as it reaches full utilization. Understanding that it is ever impossible to design a big enough cache to contain all possible contents available in a network, cache in ACED is more of a congestion sink such that for traffics not being served by the cache experiences much less congestion reaching publishers. Besides, ACED is an extension of MIDAR, it inherently supports multi-path strategy by using Directory assisted derived direction for routing. By way of extensive simulation, we show that ACED performs and scales much better than NDN[66].

5.1 Basic Operation

Fig. 5.1 illustrates the basic operation of ACED. We extend the basic operation of DARCI 3.1 and MIDAR 4.1 by modifying the cache at each router to transform from a transient cache to temporarily an authoritative publisher of content. All routers send sequence numbered HELLO messages to its 1-hop neighbors. Routers and directories store their neighborhood information and sequence numbers in neighbor table NT. All routers maintain a sequence-numbered sn distance to directories within r hops. Only a directory can update the sequence number in its following updates. Such a route is loop-free, proved in CORD[44] and ODVR[45]. Routers and directories store sequence numbers in their persistent storage, sequencing increments with every update. On a reset or routing state initialization, $sn = 0$ and distance to directory is ∞. Routers and directories store the shortest distance to neighbor directories in directory table DT. HELLO messages periodically send all updates to a router's neighbor table to neighbors.

Every entry in the cache $[CANOP]^e$ at router e has a state γ which is either *fluid* or *crystallized*. In the example, router e has been forwarding contents NO_-1 and NO_2 and, hence, caching it in $[CANOP]$. The γ for each content in the cache is set to *fluid* upon entry. Contents NO_1 and NO_2 in e's cache is in *fluid* state for as long as $(t - t^\star) \leq \lambda$, where t is the current timestamp, $t^\star$ is the entry time of the content in the cache and λ is a pre-determined and configurable interval. Cache entries in *fluid* are subject to eviction policy based on cache utilization. For any content NO_1 or NO_2 in *fluid* state and $(t - t^\star) > \lambda$ transitions to *crystallized* mode. In a *crystallized* state, the router e is the authoritative publisher of contents NO_1 and NO_2. Router e follows the publish protocol as described in DARCI 3.1 and MIDAR 4.1 by sending PUBLISH message to its closest anchor directory - in the example, this being directory a. Fig.

5.1 shows the code allocation table $[CAT]^a$ with entries for contents published by router e. Subscriber S1 querying for NO_1 sends LOOKUP message to its anchor directory a. Directory a replies with LOOKUP-REPLY and all known mappings for content NO_1. Subscriber S1 derives the shortest path to publisher by comparing information received in LOOKUP-REPLY and its neighbor table and directory table. For S1, router e is the closes publisher for content NO_1. S1 sends DATA-REQUEST to its neighbor router f towards router e. Router e replies to S1 with the requested content in DATA-REPLY. ACED, similar to DARCI and MIDAR, thus relies on the interval routing for forwarding name resolution packets and on prefix label routing using longest prefix match for forwarding replies and data requests.

Crystallized cache entries maintains its state for as long as $(t - t^\star) \leq \rho$, where t is the current timestamp, $t^\star$ is the last known transition or reset timestamp of the content in the cache from *fluid* to *crystallized*, and ρ is a pre-determined and configurable interval. Further cache hits for content name NO_1 while in *crystallized* state resets $t^\star = t$. Cache in router e does not apply eviction policy to entries in *crystallized* state. NO_1 transitions from *crystallized* to *fluid* for condition $(t - t^\star) > \rho$, resetting $t^\star = t$. Router e stops updating the mapping to the directory trie and falls back to functioning as a cache. MIDAR, unlike other cache proposals in ICN thus reduces the churn caused due to focus on eviction policy. By focusing on best-effort cache retention, router e reduces the number of requests that reach upstream nodes towards directories or publishers.

5.2 ACED

ACED extends MIDAR implementation of multi-instantiated Directory assisted routing for contents in ICN. Unlike prior work where complex algorithms

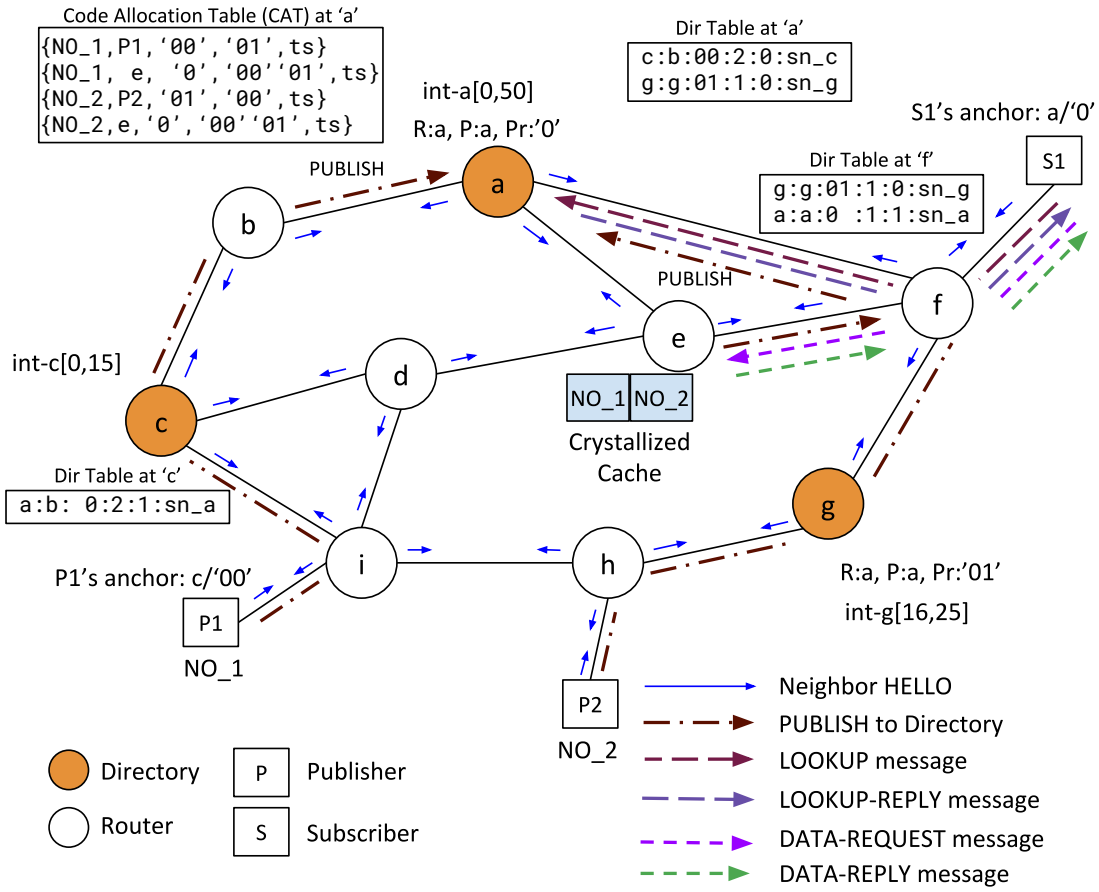

Figure 5.1: Basic operation of ACED

make up the cache replacement policy, ACED uses simple affinity by tracking when an entry stays inside the cache. Any new entry into cache is considered *fluid*. For a pre-determined and configurable time λ, any entry that has been resident of cache for more than λ while not evicted during the period λ is considered *crystallized*. The cache tracks *crystallized* entries using a pre-determined and configurable time ρ. While an entry is in *crystallized* state, any cache hit for that entry resets the ρ thus extending the *crystallized* state. The entry returns to *fluid* state when ρ expires without being reset by cache hits for that entry.

5.2.1 Information Stored & Exchanged

Our implementation of $[CANOP]^i$ as cache for router i in ACED is an extension of queue. The $[CANOP]^i$ queue keeps track of a named object mapping to its corresponding pointer to storage blob. A $[CANOP]^i$ entry consists of named object prefix $NO^\star$, a pointer to data store blob PTR, a state γ which could be either *fluid* or *crystallized* and a time-stamp t^i. $[CANOP]^i$ maintains knowledge of its usable size by using Λ to store the ordinal in the queue beyond which any *crystallized* entries are not evicted unless they transition to *fluid* state. A router with *crystallized* entries in cache publish towards the Directory trie each of the entries as if it were the authoritative publisher.

5.2.2 ACED Example

Figure 5.2 show the state transition of cache entry from *crystallized* to *fluid* and vice-versa. Figure 5.2 (i) at $t = t_1$ is a snapshot of cache where there are two entries a and d. Let ${t^\star}_a$ and ${t^\star}_d$ be the time a and d entered the cache. γ holds the state for each entry. a and d is in *fluid* state because of the satisfying condition:

$$\begin{aligned} t - {t^\star}_a < \lambda; \gamma = fluid \\ t - {t^\star}_d < \lambda; \gamma = fluid \end{aligned} \tag{5.1}$$

At $t = t_7$ in Figure 5.2 (ii), the cache has 6 entries. Contents a and d however transitions from *fluid* to *crystallized* state because of the following satisfying condition:

$$\begin{aligned} t - {t^\star}_a \geq \lambda; \gamma = crystallized \\ t - {t^\star}_d \geq \lambda; \gamma = crystallized \end{aligned} \tag{5.2}$$

At every *crystallization* event, the queue packs the *crystals* to front of the queue and resets the queue marker Λ to point to the top of the queue right after the

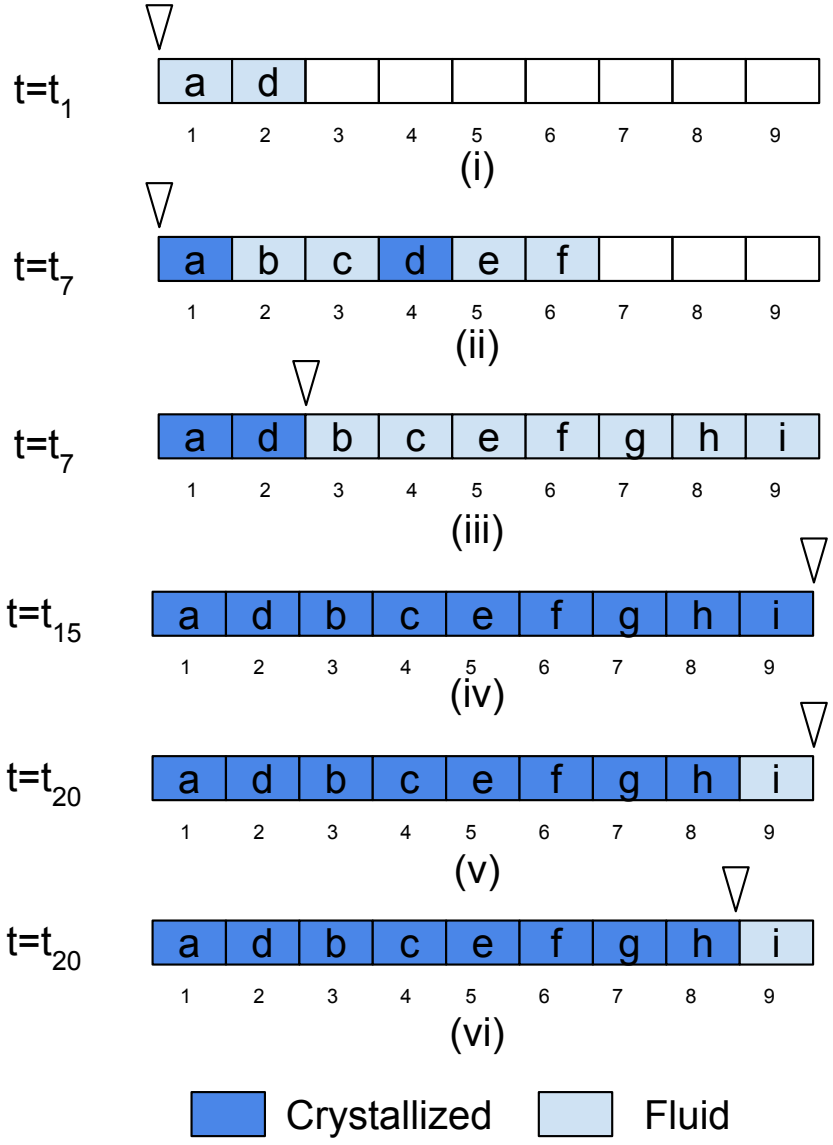

Figure 5.2: ACED Cache Progression

last packed *crystallized* entry. This is shown in Figure 5.2 (iii) where a and d are packed to the front of the queue. The queue marker Λ now points to 3. ${t^\star}_a$ and ${t^\star}_d$ now holds the time at which a and d transitioned state. At $t = t_7$, the following are the conditions of entries:

$$
\begin{aligned}
t - {t^\star}_a \leq \rho; \gamma = crystallized \\
t - {t^\star}_d \leq \rho; \gamma = crystallized \\
t - {t^\star}_b < \lambda; \gamma = fluid \\
t - {t^\star}_c < \lambda; \gamma = fluid \\
t - {t^\star}_e < \lambda; \gamma = fluid \\
t - {t^\star}_f < \lambda; \gamma = fluid \\
t - {t^\star}_g < \lambda; \gamma = fluid \\
t - {t^\star}_h < \lambda; \gamma = fluid \\
t - {t^\star}_i < \lambda; \gamma = fluid
\end{aligned}
\tag{5.3}
$$

Figure 5.2 (iv) shows the state of cache when it is fully *crystallized*. The queue marker Λ is at the end of the queue. In a fully utilized and *crystallized* mode, the cache does not perform evictions. ACED thus avoids unnecessary churns in cache due to frequent evictions. We do understand that frequent evictions may be indicative of highly popular contents more than what the cache can hold, however, reducing the churn and redirecting those requests to caches upstream to the publisher or to the publisher results in better performance as we see in our simulation results. At $t = t_{15}$, the following is the state of the packed entries:

$$
\begin{aligned}
t - {t^\star}_a &\leq \rho; \gamma = crystallized \\
t - {t^\star}_d &\leq \rho; \gamma = crystallized \\
t - {t^\star}_b &\leq \rho; \gamma = crystallized \\
t - {t^\star}_c &\leq \rho; \gamma = crystallized \\
t - {t^\star}_e &\leq \rho; \gamma = crystallized \\
t - {t^\star}_f &\leq \rho; \gamma = crystallized \\
t - {t^\star}_g &\leq \rho; \gamma = crystallized \\
t - {t^\star}_h &\leq \rho; \gamma = crystallized \\
t - {t^\star}_i &\leq \rho; \gamma = crystallized
\end{aligned}
\tag{5.4}
$$

Figure 5.2 (v) at $t = t_{20}$ the entry at location 9 expired on the timer while in *crystallized* state. This entry transitions back to *fluid* state. ${t^\star}_i$ now stores the time at which the transition from *crystallized* to *fluid* happened. This transition event triggers packing of the queue. Figure 5.2 (vi) shows the queue marker Λ after packing to point to 9. The satisfying conditions for Figure 5.2 (vi) are:

$$
\begin{aligned}
t - {t^\star}_a \leq \rho; \gamma = crystallized \\
t - {t^\star}_d \leq \rho; \gamma = crystallized \\
t - {t^\star}_b \leq \rho; \gamma = crystallized \\
t - {t^\star}_c \leq \rho; \gamma = crystallized \\
t - {t^\star}_e \leq \rho; \gamma = crystallized \\
t - {t^\star}_f \leq \rho; \gamma = crystallized \\
t - {t^\star}_g \leq \rho; \gamma = crystallized \\
t - {t^\star}_h \leq \rho; \gamma = crystallized \\
t - {t^\star}_i < \lambda; \gamma = fluid
\end{aligned}
\tag{5.5}
$$

Algorithm 13 ACED: Process `DATA-REQUEST` message at router i

```
1: input: NT^i, DT^i, NORI^i, CANOP^i, message;
2: procedure ProcessDataRequestAtRouter
3:     a_p ← GetPubAnchorDirectoryFrom(message);
4:     l_p[] ← GetPubLandmarkDirectoriesFrom(message);
5:     a_s ← GetSubAnchorDirectoryFrom(message);
6:     l_s[] ← GetSubLandmarkDirectoriesFrom(message);
7:     c ← GetContentNameFrom(message);
8:     if c ∈ CANOP^i then                                  ▷ In cache?
9:         entry ← GetFromCANOP(c);
10:        if entry.γ = crystallized then
11:            entry.t* = CurrentTimestamp();               ▷ Reset timestamp
12:        end if
13:        sendDataReply(entry.data, a_s, l_s[]);
14:    else
15:        if ← RouteToNextHopTowardsAnchor(a_p, l_p[]);    ▷ Get interface
16:        forwardToAnchor(if);
17:    end if
18: end procedure
```

Algorithms 13, 14 and 15 describe the implementation of ACED. The procedures

Algorithm 14 ACED: Process `DATA-REPLY` message at i

```
1: input: NT^i, DT^i, NORI^i, CANOP^i, message;
2: procedure ProcessDataReply
3:     subID ← GetSubscriberIDFromMessage(message);
4:     c ← GetContentNameFrom(message);
5:     data ← GetDataFrom(message);
6:     if subID = MyID() then
7:         sendToApplication(c, data);
8:         return;
9:     end if
10:    if c ∈ CANOP^i then
11:        entry ← GetFromCANOP(c);
12:        if entry.γ = crystallized then
13:            entry.t* = CurrentTimestamp();          ▷ Reset timestamp
14:        end if
15:    else
16:        UpdateCANOP(c, data);                       ▷ Add or update cache
17:    end if
18:    a_s ← GetSubAnchorDirectoryFrom(message);
19:    l_s[] ← GetSubLandmarkDirectoriesFrom(message);
20:    if ← RouteToNextHopTowardsAnchor(a_s, l_s[]);   ▷ Get interface
21:    forwardOpusDataReply(if);
22: end procedure
```

`ProcessDataRequestAtRouter` and `ProcessDataReply` resets the timestamp for any *crystallized* entries if it is referenced in response to data request. The procedure `CANOP-Pack-Unpack` is periodically called on the queue to update the states of each entry. The outcome is an updated position of Λ that separates *crystallized* entries from *fluid* entries.

Router node with fully crystallized cache sends PUBLISH message to its closes anchor directory. Anchor directory computes the code for each content name and forwards PUBLISH message using interval routing on directory trie. ACED's publish, subscribe and routing follows similar operation as described in MIDAR publish 4.2.2, subscribe 4.2.2 and routing 4.2.2.

Algorithm 15 ACED: CANOP pack-unpack at i

```
1: input: NT^i, DT^i, NORI^i, CANOP^i;
2: procedure CANOP-Pack-Unpack
3:     Λ ← GetQueueMarker(CANOP^i);
4:     Q ← GetQueue(CANOP^i);
5:     for i ← Λ to SizeOfQueue(CANOP^i) do                    ▷ Pack queue
6:         ts_now ← GetCurrentTimestamp();
7:         if (ts_now − Q[i].t*) ≥ λ AND Q[i].γ = fluid then
8:             temp_entry ← Q[i];
9:             Q[i] ← Q[Λ];
10:            Q[Λ] ← temp_entry;
11:            Q[Λ].γ ← crystallized;
12:            Λ + +;
13:        end if
14:    end for
15:    for j ← 1 to Λ − 1 do                                   ▷ Unpack queue
16:        ts_now ← GetCurrentTimestamp();
17:        if (ts_now − Q[j].t*) ≥ ρ AND Q[j].γ = crystallized then
18:            temp_entry ← Q[j];
19:            Q[j] ← Q[Λ − 1];
20:            Q[Λ − 1] ← temp_entry;
21:            Q[Λ − 1].γ ← fluid;
22:            Λ − −;
23:        end if
24:    end for
25: end procedure
```

5.3 Simulation

We implemented ACED in ns-3 version 3.27[65] and used ns-3 implementation of NDN[66] for comparing performance metrics of average throughput, average delay, and average round trip hop count as seen by subscriber nodes for an increasing number of content names. We derived a 59 node fixed topology as shown in Figure 3.14 from "BT North America" dataset[67]. To stress the adaptive cache design, we used one of the largest website workload data sets captured during the 1998 World Cup[86]. During this time, the site received 1.35 billion requests. This

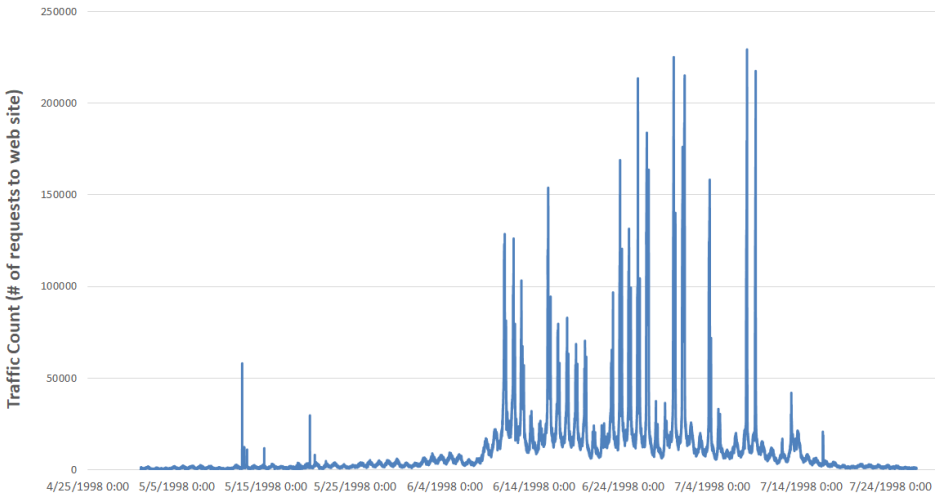

Figure 5.3: ACED simulation: 1998 World Cup Total Traffic (April 30-July 26)

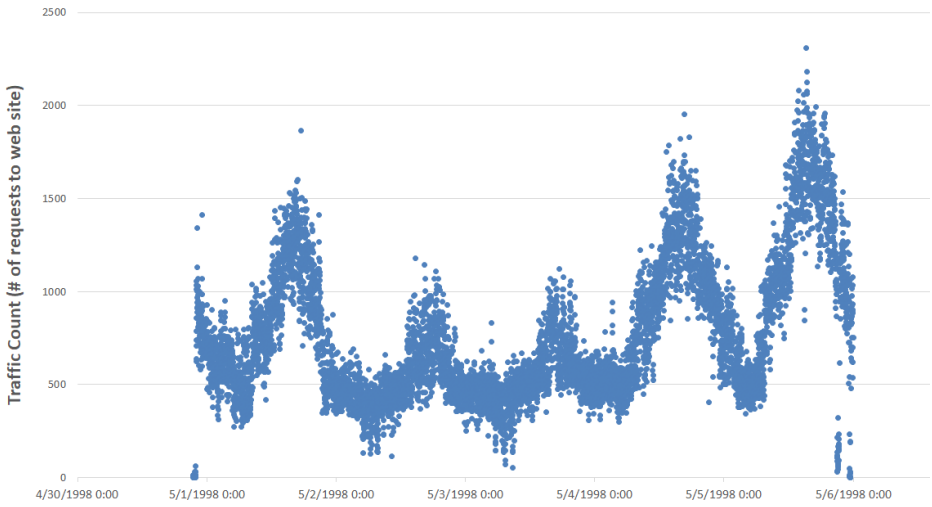

Figure 5.4: ACED simulation: 1998 World Cup flash-crowd (April 30-May 5)

data set allows for simulating flash-crowd content requests. The characteristic of requests for this period is as shown in Figure 5.3. For our simulation, we take a subset of this data from 4/30/1998 21:30 to 5/5/1998 23:59; which totals to $5,597,207.00$ requests5.4. The data set captures the timestamp of the request, a unique identifier for the client that issued the request, a unique identifier to track

the requested URL, the number of bytes sent back as response and the HTTP method used for the request (e.g., POST, GET). For the subset of the data set, we divided the number of requested URLs among the 8 publishers. All subscribers were made aware of the global URL list. At each minute, the total client requests per minute were equally divided among the subscribers. The distribution of content to publishers and the distribution of request initiation to subscribers were programmed equally for ACED, DARCI, and NDN. The simulation was ran for 6 days by generating traffic pattern from the data set 4/30/1998 21:30 to 5/5/1998 23:59. We evaluate the three protocols in a wired network with fixed topology. In DARCI and ACED, we use a maximum distance of 2 hops for a publisher or a subscriber to select an anchor directory. Nodes in ACED, DARCI and NDN are connected using point-to-point CSMA links with a data rate of 1Gbps, delay of 6.5 ms, and MTU size of 1400. For NDN, the size of the content store is 100 for on-path caching and set to a size of 1 for no caching scenarios. The caching policy for NDN is LRU. For DARCI and ACED, the size of CANOP is 100 for on-path caching and is 0 for no caching scenarios. The caching policy in DARCI is LRU, and the routing strategy is "/localhost/nfd/strategy/multicast". The subscribers and publishers are nodes with no caching enabled. For ACED, the value of λ is to 3 minutes, and the value of ρ is 10 minutes.

5.4 Performance Comparison

Figures 5.5, 5.6 and 5.7 compares the results of ACED with NDN and DARCI. ACED performs far better than DARCI and NDN in all 3 metrics. The increase in throughput in ACED is not entirely due to higher cache hits rather a co-ordinated effort between the chain of crystallized caches upstream towards the publisher. Our analysis on the performance of ACED implementation revealed

further opportunities for performance improvement in the algorithm used to pack crystallized entries in the queue. Given the flash-crowd traffic pattern, we see from Figure 5.7 that ACED requests, on average, travels less hop count than NDN. Since the cache size of 100 is relatively smaller than the total number of addressable contents in the network, NDN during flash-crowd sees higher eviction rates in its LRU implementation of cache policy. On the contrary, when a cache is crystallized, the churn due to eviction is deterred for that cache, whereas the next popular content takes advantage of being crystallized in the cache next hop towards the publisher.

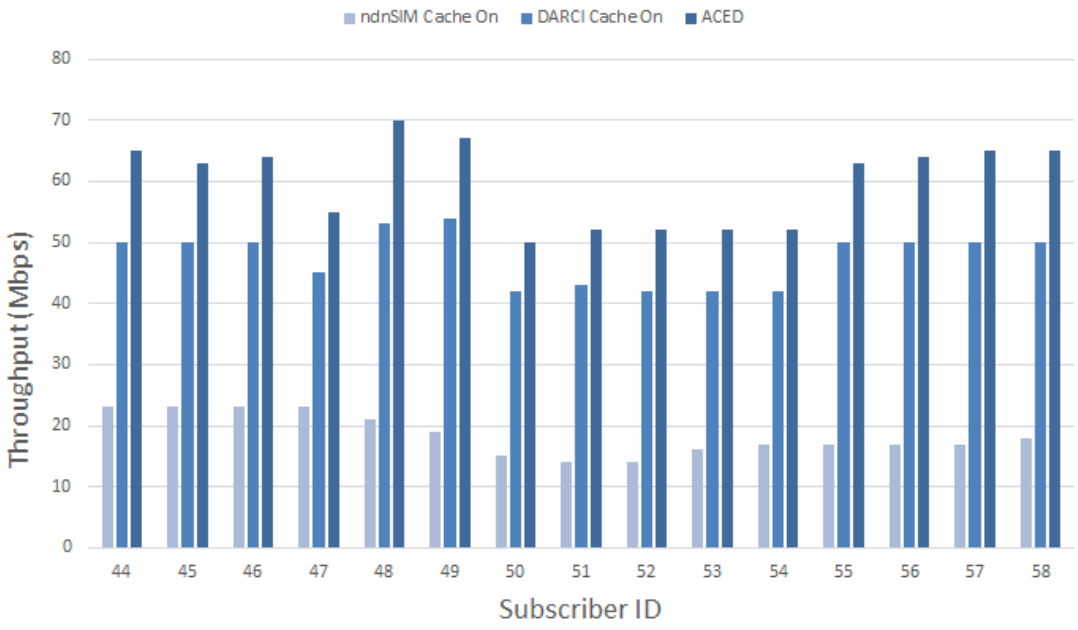

Figure 5.5: ACED simulation: Throughput (Mbps)

5.5 Summary

In this chapter, we introduced Adaptive Cache Enabled Directory Assisted Routing of Content in ICN (ACED). Caching in any form is a design de-facto

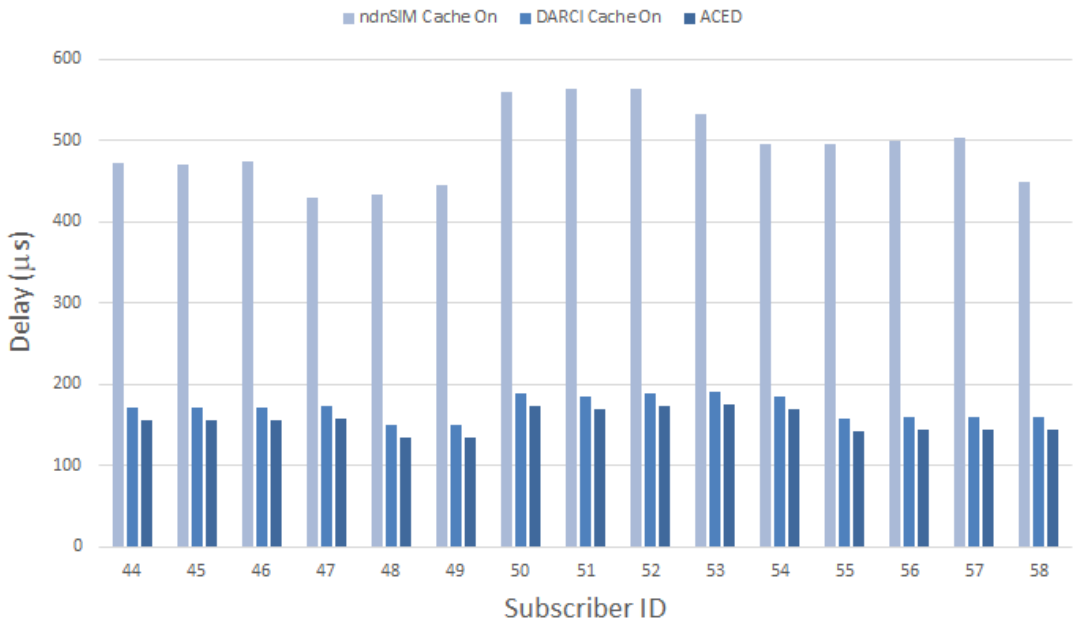

Figure 5.6: ACED simulation: Delay (μs)

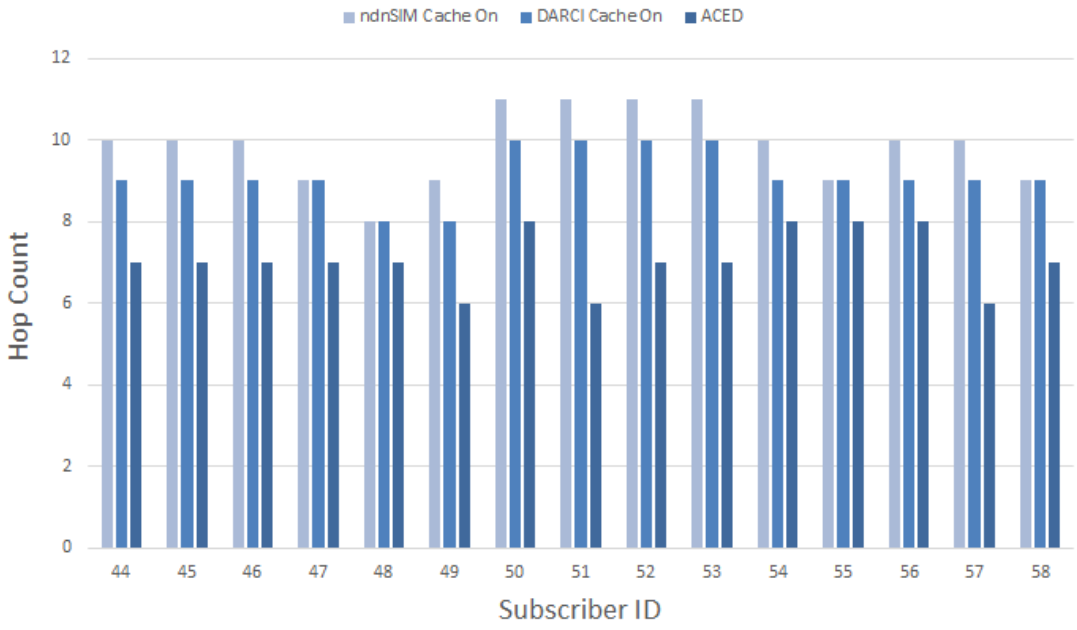

Figure 5.7: ACED simulation: Hop count

for performance improvements in a network. The preamble of ICN[77] states the independence of data from location, application, storage, and the means of trans-

portation. By way of uniquely addressing data, in-network caching and replication becomes an essential design area for network performance improvement. ACED solves quite a few problems seen with various cache implementations such as complex computations for determining eviction strategy, sub-optimal eviction policy-induced churn, and lack of support for multi-instantiated copies of the content. ACED builds upon DARCI and MIDAR, uses a derived direction from the interval and prefix labels to aide the name resolution and routing of contents in ICN. The simulation results on real-world flash-crowd data and comparing with DARCI and NDN results validate the efficiency and scalability of ACED.

Chapter 6

Conclusion

Information exchange in communication is an ever-evolving paradigm for centuries. Discovering and routing information in early centuries in the form of smoke and light reflected signals to the modern-day always connected and internetworked communication between two computers explicitly had "information-centricity" masked by "host-centricity". As inter-connections matured in modern-day host-centric Internet architecture, it ushered an era of exponential growth in user applications and user-generated information. Uses of the Internet shifted from using the Internet for accessing compute resources to accessing information across hosts. Users have moved far beyond the universal desire to be connected and reachable to an insatiable desire to be part of an "information-network" and "information-dissemination". To address this ever-growing information from discovery to routing, a new approach called Information-Centric Networking (ICN)[77] has been proposed with the core philosophy of being able to uniquely and securely identify information by its name. Various ICN architectures support information addressing independent from the location of information and means of transportation. One of the differences among most of the proposed architectures for ICN is in the name resolution and data routing. We see two approaches: coupled

and decoupled. In the coupled approach, the information provider returns information following the reverse path over which request traveled. In the decoupled approach, the name resolution function does not influence the data routing path. Another key functionality that differentiates various implementation of ICN is in the approach taken to information naming. The two approaches used are hierarchically structured naming and flat naming. Naming, name resolution, data routing, and caching mechanism contributes as key architectural differentiators for ICN implementations.

In Chapter 3 we introduced DARCI. In order to build an architecture to support ICN, we looked into components that allow scalability and extensibility as the number of addressable information increased in the network. Prior research in host-centric networking that focused on efficiency amid scarcity of resources looked into concepts of *Interval Routing*[57] and *Prefix Label Routing*[87][58]. To address the ever-growing information by uniquely naming without mandating users to follow a rigid naming convention, two-way encoding such as space-filling curve (SFC)[21] (e.g., Hilbert curve) is chosen over simple one-way hashing. By encoding content names to Hilbert curve codes of fixed-length integers while also preserving the lexicographical closeness to other similar names, DARCI achieves memory efficient and wire-speed lookup performance in the order of magnitude much higher than the non-deterministic delay in LPM lookup on variable-length names. Addressing content names using fixed length codes allows cheap routers with much less power consumption and storage. To address the first two pillars of ICN architecture, viz. naming and name resolution, DARCI arranges specialized routers called directories, that maintain the mapping between information name and authoritative publishers, in a trie. The Directory trie partitions the code space for a given SFC such that each Directory owns intervals within the

code space that maps information name to a unique code. Also, each Directory in the trie is prefix labeled. Name resolution thus uses the code generated from the requested information name and uses the code, the prefix labels of its parent and children, and the knowledge of intervals at links on the Directory to forward the messages. DARCI decouples name resolution and routing. The intervals built on SFC code ranges and the prefix labels assigned to each Directory derive a direction for information routing. Unlike a host-centric network where a message packet explicitly carries source and destination address identifiers, DARCI data packets for ICN carry the name of the content and the best-effort information regarding *Anchor Directory* $\mathcal{A}$ and *Landmark Directory* $\mathcal{L}$ of the publisher.

In Chapter 4 we introduced MIDAR. Building upon DARCI's architecture, MIDAR supports name resolving and routing for multi-instantiated information in an ICN. While users are oblivious to which copy they receive, MIDAR abstracts name resolution and routing by extending the coding function of SFC to generate a 1 dimensional SFC code from 3 dimensional input of information name, *Anchor Directory* $\mathcal{A}$ and *Landmark Directory* $\mathcal{L}$. Name resolution returns all records for a given name. Subscribers perform an informed and optimized decision to request the closest copy based on a direction derived by comparing *Anchor Directory* $\mathcal{A}$ and *Landmark Directory* $\mathcal{L}$ of all known instances with that of subscriber's *Anchor Directory* $\mathcal{A}$ and *Landmark Directory* $\mathcal{L}$.

In Chapter 5 we introduced ACED. To address the fourth pillar of ICN viz: caching, ACED modifies the cache behavior by acknowledging that no cache design can be large enough to cache all available information within a network. Most cache design proposals for ICN focuses on algorithms to better determine eviction strategy; ACED retains cache entries. To this effect, cache in ACED, when occupied by "crystallized" entries, transforms into a transient publisher that performs

publishing to the Directory trie like any other authoritative publisher. Cache in ACED works in tandem with *Authoritative Directory* $\mathcal{U}$ to reduce the number of name resolution and data request traffic that reaches publishers.

DARCI, MIDAR, and ACED are compared with NDN using real-world network topology[67] and real-world network traffic representative of web information retrieval [68] and flash-crowd[86].

An architecture to support, scale, and evolve the Future Internet is found in DARCI. The modular approach to design opens up a vast opportunity to further this architecture to build optimization components. Edge caching provides more significant gain during flash-crowd. Further research is required in designing local protocols so that Anchor directories are aware of caches at the edge with specific contents. Further research in modeling the minimum number of directories needed to balance with the NORI name cache provided by intermediate routers in the network is required. DARCI platform can further study the impact of mobility on performance, the feasibility of multiple namespaces separated code spaces to map content names, securing contents and content names, effects of various caching policies on performance, and policy-based routing of contents.

Bibliography

[1] Cisco. Cisco annual internet report (2018-2023) white paper. 2020.

[2] Cisco. Cisco global cloud index: Forecast and methodology. 2018.

[3] RFC. *Domain Name System (DNS) IANA Considerations*. 2011.

[4] National Research Council. *Signposts in Cyberspace: The Domain Name System and Internet Navigation*. The National Academies Press, Washington, DC, 2005.

[5] Jaeyeon Jung et al. Dns performance and the effectiveness of caching. *IEEE/ACM Transactions on Networking*, 10(5):589–603, 2002.

[6] V. Ramasubramanian et al. The design and implementation of a next generation name service for the internet. In *Proceedings of the 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM'04, pages 331–342, New York, NY, USA, 2004. Association for Computing Machinery.

[7] K. Park et al. Codns: Improving dns performance and reliability via cooperative lookups. In *Proceedings of the 6th Conference on Symposium on Operating Systems Design & Implementation - Volume 6*, OSDI 2004, page 14, USA, 2004. USENIX Association.

[8] Y. Chen et al. Dns noise: Measuring the pervasiveness of disposable domains in modern dns traffic. In *Proceedings of the 2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, DSN'14, pages 598–609, USA, 2014. IEEE Computer Society.

[9] Paul Vixie. What dns is not. *Queue*, 7(10):10–15, November 2009.

[10] Steve Souders. Sharding dominant domains. 2009.

[11] J.S. Otto et al. Content delivery and the natural evolution of dns: Remote dns trends, performance issues and alternative solutions. In *Proceedings of the 2012 Internet Measurement Conference*, IMC'12, pages 523–536, New York, NY, USA, 2012. Association for Computing Machinery.

[12] B. Krishnamurthy et al. On the use and performance of content distribution networks. In *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, IMW'01, pages 169–182, New York, NY, USA, 2001. Association for Computing Machinery.

[13] Steve Souders. High performance web sites: 14 rules for faster loading pages. 2009.

[14] James Hamilton. The cost of latency. 2009.

[15] M. Handley. Why the internet only just works. *BT Technology Journal*, 24(3):119–129, July 2006.

[16] G. Xylomenos et al. A survey of information-centric networking research. *IEEE Communications Surveys Tutorials*, 16(2):1024–1049, 2014.

[17] B. Ahlgren et al. A survey of information-centric networking. *IEEE Communications Magazine*, 50(7):26–36, 2012.

[18] M. F. Bari et al. A survey of naming and routing in information-centric networks. *IEEE Communications Magazine*, 50(12):44–53, 2012.

[19] L. Zhang et al. Named data networking. *SIGCOMM Comput. Commun. Rev.*, 44(3):66–73, July 2014.

[20] Named Data Networking. *http://www.named-data.net/*.

[21] Hans Sagan. *Space Filling Curves*. Springer-Verlag New York, 1994.

[22] Craig Wills and Hao Shang. The contribution of dns lookup costs to web object retrieval. 09 2000.

[23] RFC 7871. *Client Subnet in DNS Queries*. 2016.

[24] Fangfei Chen, Ramesh K. Sitaraman, and Marcelo Torres. End-user mapping: Next generation request routing for content delivery. *SIGCOMM Comput. Commun. Rev.*, 45(4):167âĂŞ181, August 2015.

[25] Teemu Koponen, Mohit Chawla, Byung-Gon Chun, Andrey Ermolinskiy, Kye Kim, Scott Shenker, and Ion Stoica. A data-oriented (and beyond) network architecture. *ACM SIGCOMM Computer Communication Review*, 37:181–192, 10 2007.

[26] FP7 PURSUIT project. *https://www.fp7-pursuit.eu/*.

[27] FP7 PSIRP project. *http://www.psirp.org/*.

[28] FP7 SAIL project. *https://sail-project.eu/*.

[29] FP7 4WARD project. *http://www.4ward-project.eu/.*

[30] FP7 COMET project. *http://www.comet-project.org/.*

[31] FP7 CONVERGENCE project. *http://www.ictconvergence.eu/.*

[32] Content Centric Networking project. *http://www.ccnx.org/.*

[33] NSF Mobility First project. *http://mobilityfirst.winlab.rutgers.edu/.*

[34] ANR Connect project. *http://anr-connect.org/.*

[35] G. Garcia, A. Beben, F. J. Ramon, A. Maeso, I. Psaras, G. Pavlou, N. Wang, J. Sliwinski, S. Spirou, S. Soursos, and E. Hadjioannou. Comet: Content mediator architecture for content-aware networks. In *2011 Future Network Mobile Summit*, pages 1–8, 2011.

[36] W. K. Chai, N. Wang, I. Psaras, G. Pavlou, C. Wang, G. Garcia de Blas, F. J. Ramon-Salguero, L. Liang, S. Spirou, A. Beben, and E. Hadjioannou. Curling: Content-ubiquitous resolution and delivery infrastructure for next-generation services. *IEEE Communications Magazine*, 49(3):112–120, 2011.

[37] COMET Project. (2011 December) COMET deliverable 3.2: Final specification of mechanisms protocols and algorithms for the content mediation system. *http://www.COMET-project.org/deliverables.html.*

[38] Andrea Detti, Nicola Melazzi, Stefano Salsano, and Matteo Pomposini. Conet: A content centric inter-networking architecture. 08 2011.

[39] C. Yi et al. Adaptive forwarding in named data networking. *SIGCOMM Comput. Commun. Rev.*, 42(3):62–67, June 2012.

[40] C. Yi et al. A case for stateful forwarding plane. *Comput. Commun.*, 36(7):779–791, April 2013.

[41] J. J. Garcia-Luna-Aceves et al. A fault-tolerant forwarding strategy for interest-based information centric networks. In *2015 IFIP Networking Conference (IFIP Networking)*, pages 1–9, 2015.

[42] J. J. Garcia-Luna-Aceves et al. Enabling correct interest forwarding and retransmissions in a content centric network. In *2015 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, pages 135–146, 2015.

[43] J. J. Garcia-Luna-Aceves. Eliminating undetected interest looping in content-centric networks. In *2015 6th International Conference on the Network of the Future (NOF)*, pages 1–6, 2015.

[44] J. J. Garcia-Luna-Aceves et al. Cord: Content oriented routing with directories. In *2015 International Conference on Computing, Networking and Communications (ICNC)*, pages 785–790, 2015.

[45] J. J. Garcia-Luna-Aceves and Ehsan Hemmati. ODVR: A unifying approach to on-demand and proactive loop-free routing in ad-hoc networks. In *28th International Conference on Computer Communication and Networks, ICCCN 2019, Valencia, Spain, July 29 - August 1, 2019*, pages 1–11. IEEE, 2019.

[46] A. Dabirmoghaddam et al. Characterizing interest aggregation in content-centric networks. In *2016 IFIP Networking Conference (IFIP Networking) and Workshops*, pages 449–457, 2016.

[47] H. Dai et al. On pending interest table in named data networking. In *2012 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, pages 211–222, 2012.

[48] Y. Wang et al. Scalable name lookup in ndn using effective name component encoding. In *2012 IEEE 32nd International Conference on Distributed Computing Systems*, pages 688–697, 2012.

[49] M. Varvello et al. On the design and implementation of a wire-speed pending interest table. In *2013 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 369–374, 2013.

[50] H. Yuan et al. Scalable pending interest table design: From principles to practice. In *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*, pages 2049–2057, 2014.

[51] M. Virgilio et al. Pit overload analysis in content centric networks. In *Proceedings of the 3rd ACM SIGCOMM Workshop on Information-Centric Networking*, ICN'13, pages 67–72, New York, NY, USA, 2013. Association for Computing Machinery.

[52] J. J. Garcia-Luna-Aceves. A more scalable approach to content centric networking. In *2015 24th International Conference on Computer Communication and Networks (ICCCN)*, pages 1–8, 2015.

[53] J. J. Garcia-Luna-Aceves et al. A light-weight forwarding plane for content-centric networks. *CoRR*, abs/1603.06044, 2016.

[54] J. J. Garcia-Luna-Aceves et al. Content-centric networking using anonymous datagrams. In *2016 IFIP Networking Conference (IFIP Networking) and Workshops*, pages 171–179, 2016.

[55] J. J. Garcia-Luna-Aceves et al. Content-centric networking at internet scale through the integration of name resolution and routing. In *Proceedings of the 3rd ACM Conference on Information-Centric Networking*, ACM-ICN'16, pages 83–92, New York, NY, USA, 2016. Association for Computing Machinery.

[56] Alexander Afanasyev, Cheng Yi, Lan Wang, Beichuan Zhang, and Lixia Zhang. Snamp: Secure namespace mapping to scale ndn forwarding. In *In Proc. of Global Internet Symposium*, 2015.

[57] V. Leeuwen et al. Interval routing. *Comput. J.*, 30(4):298–307, August 1987.

[58] Erwin M. Bakker, Jan [van Leeuwen], and Richard B. Tan. Prefix routing schemes in dynamic networks. *Computer Networks and ISDN Systems*, 26(4):403 – 421, 1993.

[59] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM âĂŹ01, page 149âĂŞ160, New York, NY, USA, 2001. Association for Computing Machinery.

[60] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content-addressable network. In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM'01, pages 161–172, New York, NY, USA, 2001. Association for Computing Machinery.

[61] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. volume 2218, pages 329–350, 01 2001.

[62] Ben Zhao, John Kubiatowicz, and Anthony Joseph. Tapestry: a fault-tolerant wide-area application infrastructure. *Computer Communication Review*, 32:81, 01 2002.

[63] Y. Tang, S. Zhou, and J. Xu. Light: A query-efficient yet low-maintenance indexing scheme over dhts. *IEEE Transactions on Knowledge and Data Engineering*, 22(1):59–75, 2010.

[64] Sriram Ramabhadran, Sylvia Ratnasamy, Joseph Hellerstein, and Scott Shenker. Prefix hash tree: An indexing data structure over distributed hash tables. 01 2004.

[65] ns 3. *Discrete-event network simulator*. v3.27.

[66] ndnSIM. *Named Data Networking simulator*. v2.7.

[67] R. Bowden et al. Cold: Pop-level network topology synthesis. In *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies*, CoNEXT'14, pages 173–184, New York, NY, USA, 2014. Association for Computing Machinery.

[68] M. Meiss et al. Modeling traffic on the web graph. volume 6516, pages 50–61, 12 2010.

[69] A. Ballardie. Rfc2201: Core based trees (cbt) multicast routing architecture, 1997.

[70] S. Deering, D. L. Estrin, D. Farinacci, V. Jacobson, Ching-Gung Liu, and Liming Wei. The pim architecture for wide-area multicast routing. *IEEE/ACM Transactions on Networking*, 4(2):153–162, 1996.

[71] Hitesh Ballani and Paul Francis. Towards a global ip anycast service. In *Proceedings of the 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM'05, pages 301–312, New York, NY, USA, 2005. Association for Computing Machinery.

[72] Dina Katabi and John Wroclawski. A framework for scalable global ip-anycast (gia). In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM âĂŹ00, page 3âĂŞ15, New York, NY, USA, 2000. Association for Computing Machinery.

[73] B. N. Levine and J. J. Garcia-Luna-Aceves. Improving internet multicast with routing labels. In *Proceedings 1997 International Conference on Network Protocols*, pages 241–250, 1997.

[74] S. Weber and Liang Cheng. A survey of anycast in ipv6 networks. *IEEE Communications Magazine*, 42(1):127–132, 2004.

[75] J. J. Garcia-Luna-Aceves. Routing to multi-instantiated destinations: Principles and applications. In *Proceedings of the 2014 IEEE 22nd International Conference on Network Protocols*, ICNP'14, pages 155–166, USA, 2014. IEEE Computer Society.

[76] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva. Directed diffusion for wireless sensor networking. *IEEE/ACM Transactions on Networking*, 11(1):2–16, 2003.

[77] https://irtf.org/icnrg. *Information-Centric Networking Research Group ICNRG*. 2020.

[78] Ramón Cáceres, Fred Douglis, Anja Feldmann, Gideon Glass, and Michael Rabinovich. Web proxy caching: The devil is in the details. *SIGMETRICS Perform. Eval. Rev.*, 26(3):11–15, December 1998.

[79] Bradley Duska, David Marwood, and Michael Feeley. The measured access characteristics of world-wide-web client proxy caches. 08 2000.

[80] Thomas Kroeger, Darrell Long, and Jeffrey Mogul. Exploring the bounds of web latency reduction from caching and prefetching. 12 1997.

[81] Jia Wang. A survey of web caching schemes for the internet. *SIGCOMM Comput. Commun. Rev.*, 29(5):36–46, October 1999.

[82] Liangzhong Yin and Guohong Cao. Supporting cooperative caching in ad hoc networks. In *IEEE INFOCOM 2004*, volume 4, pages 2537–2547 vol.4, 2004.

[83] B. Panigrahi, S. Shailendra, H. K. Rath, and A. Simha. Universal caching model and markov-based cache analysis for information centric networks. In *2014 IEEE International Conference on Advanced Networks and Telecommuncations Systems (ANTS)*, pages 1–6, 2014.

[84] Vasilis Sourlas, Paris Flegkas, and Leandros Tassiulas. A novel cache aware routing scheme for information-centric networks. *Computer Networks*, 59:44 – 61, 2014.

[85] Dario Rossi. Caching performance of content centric networks under multi-path routing (and more). 2011.

[86] Martin Arlitt and Tai Jin. Workload characterization of the 1998 world cup web site. *https://www.hpl.hp.com/techreports/1999/HPL-1999-35R1.html*, 1999.

[87] N. Santoro et al. Labelling and implicit routing in networks. *Comput. J.*, 28:5–8, 1985.

[88] N. Santoro. Sense of direction, topological awareness and communication complexity. *SIGACT News*, 16(2):50–56, July 1984.

[89] N. Santoro et al. Sense of direction: Definitions, properties, and classes. *Networks*, 32(3):165–180, 1998.

[90] M. Tang et al. Compact routing on random power law graphs. In *2009 Eighth IEEE International Conference on Dependable, Autonomic and Secure Computing*, pages 575–578, 2009.

[91] A. Afanasyev et al. Interest flooding attack and countermeasures in named data networking. In *2013 IFIP Networking Conference*, pages 1–9, 2013.

[92] M. Wählisch et al. Lessons from the past: Why data-driven states harm future information-centric networking. In *2013 IFIP Networking Conference*, pages 1–9, 2013.

[93] M. Wählisch et al. Backscatter from the data plane - threats to stability and security in information-centric network infrastructure. *Comput. Netw.*, 57(16):3192–3206, November 2013.

[94] J. K. Lawder et al. Querying multi-dimensional data indexed using the hilbert space-filling curve. *SIGMOD Rec.*, 30(1):19–24, March 2001.

[95] Hilbert D. *Uber die stetige Abbildung einer Linie auf ein Flachenstuck.* Springer, Berlin, Heidelberg, 1935.

[96] S. Shailendra, S. Sengottuvelan, H. K. Rath, B. Panigrahi, and A. Simha. Performance evaluation of caching policies in ndn - an icn architecture. In *2016 IEEE Region 10 Conference (TENCON)*, pages 1117–1121, 2016.

[97] Ioannis Psaras, Richard G. Clegg, Raul Landa, Wei Koong Chai, and George Pavlou. Modelling and evaluation of ccn-caching trees. In Jordi Domingo-Pascual, Pietro Manzoni, Sergio Palazzo, Ana Pont, and Caterina Scoglio, editors, *NETWORKING 2011*, pages 78–91, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[98] Somaya Arianfar, Pekka Nikander, and Jorg Ott. Packet-level caching for information-centric networking. 01 2010.

[99] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *SIGCOMM Comput. Commun. Rev.*, 31(4):149–160, August 2001.

[100] David Karger, Eric Lehman, Tom Leighton, Rina Panigrahy, Matthew Levine, and Daniel Lewin. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, STOC âĂŹ97, pages 654–663, New York, NY, USA, 1997. Association for Computing Machinery.