UNIVERSITY OF CALIFORNIA
RIVERSIDE

The Role of Naming in Information-Centric Networks

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

Doctor of Philosophy

in

Computer Science

by

Mohammad H. Jahanian

September 2021

Dissertation Committee:

Prof. K.K. Ramakrishnan, Chairperson
Prof. Jiasi Chen
Prof. Evangelos Christidis
Prof. Rajiv Gupta

The Dissertation of Mohammad H. Jahanian is approved:

Committee Chairperson

University of California, Riverside

Acknowledgments

I would like to take this opportunity to thank everyone who helped me throughout this journey and complete this dissertation.

First, I would like to express my deepest gratitude to my advisor, Prof. K. K. Ramakrishnan for his excellence guidance and support during the course of my PhD. The completion of this dissertation would not have been possible without his mentorship and invaluable insights. I would like to thank all the members of my dissertation committee: Prof. Rajiv Gupta, Prof. Jiasi Chen, Prof. Evangelos Christidis, for their insightful comments. I would also like to thank Prof. Srikanth Krishnamurthy and Prof. Nanpeng Yu for their helpful directions as members of my PhD candidacy committee.

I would like to thank all the researchers that I was fortunate enough to collaborate with in various projects during my PhD: Dr. Jiachen Chen (WINLAB, Rutgers University), Prof. Hulya Seferoglu and Dr. Yuxuan Xing (University of Illinois at Chicago), Prof. Murat Yuksel (University of Central Florida), Prof. Toru Hasegawa and Prof. Yuki Koizumi (Osaka University), Prof. Amr Magdy and Viyom Mittal (UCR), Prof. Yoshinobu Kawabe (Aichi Institute of Technology), Prof. Masakatsu Nishigaki and Prof. Tetsushi Ohki (Shizuoka University).

I would like to thank all former and current members of our lab who helped me and made my PhD studies at UCR a more enjoyable one. I especially thank Ali Mohammakhan, Aditya Dhakal, Shahryar Afzal, Sourav Panda, and Sameer G. Kulkarni.

I would like thank my family. Without their unwavering support, this journey would not have been possible. I am deeply grateful to my entire family and friends.

This dissertation includes content published in the following proceedings and journals (those marked with '**' are described thoroughly while those marked with '*' are mentioned briefly in this dissertation):

1. Viyom Mittal, Mohammad Jahanian, K. K. Ramakrishnan, "FLARE: Federated Active Learning Assisted by Naming for Responding to Emergencies", *The 8th ACM conference on Information Centric Networking (ICN), September 2021.* *

2. Mohammad Jahanian, Jiachen Chen, K. K. Ramakrishnan, "Graph-based Namespaces and Load Sharing for Efficient Information Dissemination", *The IEEE/ACM Transactions on Networking, 2021.* **

3. Mohammad Jahanian, K. K. Ramakrishnan, "Name Space Analysis: Verification of Named Data Network Data Planes", *The IEEE/ACM Transactions on Networking, Volume 29, Issue 2, April 2021.* **

4. Viyom Mittal, Mohammad Jahanian, K. K. Ramakrishnan, "Online Delivery of Social Media Posts to Appropriate First Responders for Disaster Response", *The 3rd International Workshop on Emergency Response Technologies and Services (EmeRTeS @ ICDCN'21), January 2021.* *

5. Mohammad Jahanian, K. K. Ramakrishnan, "CoNICE: Consensus in Intermittently-Connected Environments by Exploiting Naming with Application to Emergency Response", *The 28th IEEE International Conference on Network Protocols (ICNP), October 2020.* **

6. Mohammad Jahanian, Jiachen Chen, K. K. Ramakrishnan, "Managing the Evolution to Future Internet Architectures and Seamless Interoperation", *The 29th International Conference on Computer Communications and Networks (ICCCN), August 2020.* **

7. Mohammad Jahanian, Jiachen Chen, K. K. Ramakrishnan, "Formal Verification of Interoperability Between Future Network Architectures Using Alloy", *The 7th International Conference on Rigorous State-Based Methods (ABZ), May 2020.* **

8. Mohammad Jahanian, Toru Hasegawa, Yoshinobu Kawabe, Yuki Koizumi, Amr Magdy, Masakatsu Nishigaki, Tetsushi Ohki, K. K. Ramakrishnan, "DiReCT: Disaster Response Coordination with Trusted Volunteers", *The 6th International Conference on Information and Communication Technologies for Disaster Management (ICT-DM), December 2019.* **

9. Mohammad Jahanian, Jiachen Chen, K. K. Ramakrishnan, "Graph-based Namespaces and Load Sharing for Efficient Information Dissemination in Disasters", *The 27th IEEE International Conference on Network Protocols (ICNP), October 2019.* **

10. Mohammad Jahanian, K. K. Ramakrishnan, "Name Space Analysis: Verification of Named Data Network Data Planes", *The 6th ACM conference on Information Centric Networking (ICN), September 2019.* **

11. Mohammad Jahanian, Yuxuan Xing, Jiachen Chen, K. K. Ramakrishnan, Hulya Seferoglu, Murat Yuksel, "The Evolving Nature of Disaster Management in the Internet and Social Media Era", *The 24th IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN), June 2018.* *

12. Jiachen Chen, Mohammad Jahanian, K. K. Ramakrishnan, "Black Ice! Using Information Centric Networks for Timely Vehicular Safety Information Dissemination", *The 23rd IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN), June 2017. **

To my father and the loving memory of my mother.

ABSTRACT OF THE DISSERTATION

The Role of Naming in Information-Centric Networks

by

Mohammad H. Jahanian

Doctor of Philosophy, Graduate Program in Computer Science
University of California, Riverside, September 2021
Prof. K.K. Ramakrishnan, Chairperson

The predominantly content-oriented use of today's networks has led to the development of information-Centric Networking (ICN) paradigms and network architectural designs. At the core of such designs is naming, which organizes and guides the delivery and dissemination of information in the network. This dissertation studies the correctness of such name-based networking, improving it for better scalability, and enhancing its functionality in real-world applications.

As a first topic, we propose Name Space Analysis (NSA), a network verification framework to model and analyze name-based data planes of Named Data Networks. NSA supports primitives fundamentally different from those of traditional host-centric IP network verification, such as checking host-to-content reachability in addition to the traditional host-to-host reachability considerations. We also design and formally analyze name-based delivery for inter-operation of ICNs with existing IP networks.

State-of-the-art ICN designs rely on strictly-hierarchical naming frameworks and pull-based request/response models, which can be inadequate when it comes to complex information structures and many-senders-to-many-receivers delivery. We propose POISE, a name-based and recipient-based publish/subscribe architecture for efficient information dissemination that supports complex graph-based namespaces with a workload-driven namespace graph partitioning for multicast core migration. We demonstrate that POISE achieves better efficiency in timely delivery and elimination of traffic concentration than existing alternatives.

Next, we consider infrastructure-less intermittently-connected network environments, where consistency of replicated databases is especially challenging. We propose CoNICE, a framework to ensure consistent dissemination of updates in such environments. Our proposed name-based coordinator-less consensus method improves completeness and the convergence latency in ordering updates, all with lower communication overhead. We study its correctness and scalability and also demonstrate its performance benefits.

We then look at how names can be learned from the content itself. We propose a framework that integrates Natural Language Processing techniques with Information-Centric dissemination, considering the role of social media-based incident reporting in disaster response. The framework introduces a social media engine to intelligently map social media posts to the right names, to steer posts (content) to the most relevant first responders in a timely manner. We further show how these social media engines can be enhanced, using active and federated learning, for better accuracy.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Users primarily seek information over the network without necessarily wanting to focus on its location or the underlying mechanisms used to retrieve that information. However, the current way of using "location-based" access in IP results in a less convenient and less efficient means for information dissemination and retrieval. Information-Centric Networks (ICNs) [92] separate content identity from location. ICN enables access of content based on its name, from wherever it resides, supporting mobility as well as accessing the named content from the "best" source. Typically, ICN architectures have network layers that operate based on names, in contrast to the current IP network that operates solely on IP addresses (and content-related operations are moved to application layer at end hosts). This ICN design allows for ubiquitous network wide caching to reduce access latency. There are a number of ICN architectures – Named Data Networking (NDN) [203] and MobilityFirst (MF) [160], XIA [145], PURSUIT [75], to mention a few. ICN enables content-oriented services such as request/response (*e.g.*, Interest/Data in NDN [203]) and publish/subscribe (for many-to-many multicast dissemination) [49].

Information can be organized, identified and accessed in complex ways. Naming frameworks and namespaces that identify resources, objects, services, and users, and represent their relations, are used in a variety of contexts in networks and distributed systems, ranging from tightly controlled Distributed File Systems [59,172] to large-scale NDNs [203]. In ICNs, names guide the paths packets take, separating the notions of names and addresses. These names get mapped to location addresses through Name Resolution Services. NRS can be implicit (such as in NDN) or explicit (such as in MF). At the core of a name-based delivery/dissemination framework (such as in pub/sub), is a namespace [49], that is essential to direct requests/subscriptions and deliver publications/content. The traditional hierarchical namespace (or taxonomy) is a common way of organizing information, and is followed in traditional ICN design: in NDN, content items and their relationships are identified through hierarchical namespaces [203], *i.e.*, a graph that follows a strict hierarchic and is represented using prefix trees (tries).

This dissertation focuses on the role of naming in ICNs, which organizes content and guides its delivery in the network, and aims at answering important questions such as the following: How can we ensure the correctness of name-based networks? How can we interoperate among various networks with different naming structures? How can we leverage and enhance naming for efficient information organization and dissemination? How can we generate and assign names to content in an accurate manner? How can we use naming to ensure consistency of data delivery in disconnected environments? In order to answer these questions, we use various techniques to provide solutions that are both sound and practical. While some of the contributions focus on specific ICN architectures such as NDN and MobilityFirst, for the most part we consider ICN as a general paradigm that considers content items as first class citizens, thus not limited to a specific protocol stack or architecture. We investigate the notion of naming and ICN in Future Internet, and also particular applications such as disaster management. In particular, this dissertation makes the following major contributions:

Contribution 1: Verification of Name-based Networks. Network Verification aims at analyzing the correctness of a network, e.g., regarding forwarding rules [31, 107,110]. Existing network verification tools are suited for traditional host-centric networks (e.g., IP) and not ICNs: ICNs have radically different design from IP (e.g., name-based forwarding vs. address-based forwarding) and have different intents (e.g., host-to-content reachability vs. host-to-host reachability).

We propose Name Space Analysis (NSA) [97], the first data plane verification framework for NDN. NSA builds on the theory of Header Space Analysis [107] and further includes essential NDN-specific verification applications of content reachability test (to detect name space conflicts, content censorship-freedom, etc.), name-based loop detection, and name leakage detection. Applied to the NDN testbed [148], we found a number of data plane errors in it through NSA's automatized verification. Our evaluation results on various test cases show the effectiveness, efficiency, and scalability of NSA. Furthermore, we propose methods so that NSA can help in error resolution as well [100]. In particular, we add a method for resolving name space conflicts, a challenge in name-based networking of NDN, by proposing a Name Registry component and protocol. This can help uncover name space conflicts and resolve them.

Contribution 2: Interoperability of Name-based Networks. New network architectures, such as ICNs, are continually being proposed to address specific and specific requirements. We may end up with many "bridges" of different architectures [24, 133]. Content may be in domains different from where the consumer is. Reachability across domains with different architectures is important and challenging. We believe a pragmatic approach to manage network evolution would be to design an interoperability framework between these different domains, or in other words, "bridging the many islands". There have been many attempts over several decades addressing interoperability across network architectures in the form of multi-protocol routers and gateways. We believe that the concept needs to be revisited today, since compared to previous efforts, the architectures we seek to support for interoperation, namely information-centric networks, are much more different in nature than in the past.

We propose a Content-oriented Interoperability framework (COIN) [95], for consumers to access content across domains in a secure manner, through translation gateways operating at content name level. As important features, COIN induces no architectural change, preserves architecture features (such as naming schemas, mobility recovery methods and security mechanisms), spans more than two domains interoperating, and currently supports the three architectures of NDN, MobilityFirst, and IP. We then formally analyze the information-centric interoperability [94], where we present an Alloy [91]-based model and showed how model finding can be used to analyze reachability and returnability properties across domains of NDN, MobilityFirst, and IP, with gateways translating at name level. We also enhance Alloy's model finding with a model counting approach to analyze failure and mobility scenarios, which we utilized to prove the negative impact of certain routing policies (particularly, reverse path forwarding), and the helpfulness of certain mobility-handling mechanisms (particularly, late binding), providing necessary confidence and guidelines for Future Internet interoperability.

Contribution 3: Graph-based Namespaces and Load Sharing for Pub/-Sub.

Name-based publish/subscribe (pub/sub) [49] is an information-centric design for information dissemination and multicast, according to recipients' (subscribers') interest in subsets of the namespace, such as in a system based on it for vehicular safety message propagation [50]. There are two types of namespace design for name-based pub/sub: topic-based and recipient-based. In topic-based pub/sub [49], a subscriber of a named topic, *e.g.*, "/CaliforniaWildFires", is interested in receiving all the content published at a finer granularity, *i.e.*, what is under a particular topic category, *e.g.*, "/CaliforniaWild-Fires/WoolseyFire'. In recipient-based pub/sub [48], a subscriber of a named role, *e.g.*, "/fireDepartment/fireTeam1", must receive all messages published to a coarser granularity, *i.e.*, sent to everything above that role in a command chain, *e.g.*, to "/fireDepartment". However, there is a shortcoming with traditional ICN namespace schema (such as in NDN): it is strictly hierarchical [203]. This hierarchical structure falls short in efficiently modeling complex, multi-dimensional information organizations, such as the Wikipedia knowledge base, and information flow chains in disaster response (as a name node may have multiple parent nodes).

We propose POISE [93], a framework than decouples how information is organized (information layer), and how it is forwarded via name-based forwarding (service layer). This way, we propose and support graph-based naming in the information layer in the network. We load-shared namespace maintenance and multicast among multiple Rendezvous Points (RPs) [72]. Given that the workload-per-RP distribution is non-uniform and difficult to predict, especially in disaster situations, we also proposed an automatic load splitting and core migration protocol among RPs. At the core of this splitting procedure, we present a graph partitioning algorithm, which involves a “complex” objective (the ”cut” itself affects the node/edge weights) [157]. As a result, off-the-shelf graph partitioners such as METIS [103], fall short. To overcome this, we proposed a hybrid splitting procedure consisting of a heuristic (METIS) and meta-heuristic guided search refinement (using Tabu Search [162]). With implementation and simulation experiments, we showed how our naming design, dissemination protocol, and partitioning are scalable and outperform state-of-the-art methods.

Contribution 4: Consistent Information Dissemination in Name-based DTNs. All the works mentioned above, and most works in the ICN space, consider a network infrastructure with fixed router and reliable links. However, in many real-world scenarios, information must be disseminated over intermittently-connected environments, when network infrastructure becomes damaged. An example of such scenarios is disasters in which first responders need to exchange updates about their critical tasks. If such updates pertain to many users and a single shared data set (e.g., adding/editing/removing pins on a map), their consistent dissemination is important and challenging. Many algorithms and techniques to ensure consistency have been proposed. Causal consistency ensures updates get processed at users in accordance with their causal relations [58]. Causal ordering provides a 'moderate' degree of consistency better than It is stronger than best-effort out-of-order delivery, as it orders "orderable" updates. It is weaker than agreement-based total order delivery, as it is ambiguous when it comes to ordering "un-orderable" updates. Consensus methods, on the other hand, ensure agreement and strong consistency. Most popular consensus methods, most notably Paxos [117] and Raft [151], are most suited for connected environments with reliable links. These techniques achieve agreement among a number of networked nodes, for purposes such as total ordering, by election of a leader, majority voting, and deciding on a value, through a number of rounds. In other words, they are coordinated consensus methods.

We propose CoNICE [98], a framework to ensure consistent dissemination of updates among users in intermittently-connected, infrastructure-less environments. CoNICE provides three levels of consistency to users' views, namely replication (epidemic propagation), causality (causal ordering) and agreement (consensus). Our consensus is based on the two-thirds majority rule [32]; however, it is coordinator-less and tolerates loss. To further enhance CoNICE's efficiency, we design and leverage namespaces, allowing users to indicate and filter based on their task-related interests. With city-scale simulation experiments, we show that using this namespace component, not only we increase the relevancy of content delivery, we also limit the number of consensus participants to only the relevant users. The latter effect of our naming leads to considerably higher degree of agreement completeness as well as faster convergence, which is a novel benefit of information-centric naming.

Contribution 5: Intelligent Information Name Assignment from Free-form Text. While name-based pub/sub allows named content to reach the right recipients, assigning the right name for a piece of content is still an un-addressed challenge. This can be a necessary need especially in situations where many frequent messages with free-form content are created by users who do not necessarily have access to the namespace. One prominent example is the extensive use of social media posting during disasters, mainly to ask for help and report issues [54]. We studied two major and fairly recent disasters in the US [101], namely Hurricanes Harvey and Irma in 2017, and after performing text mining on millions of collected tweets in the affected areas, we observed the temporal/spatial correlation of tweets collected with the real-world progression of events, and how the content of tweets can be used for a meaningful, systematic dissemination of information.

Motivated by this study, we propose a framework that integrates social media with Information-Centric dissemination in disaster response [96, 136]. The framework includes a social media engine to intelligently map social media posts to the right names, using NLP and ML techniques [35, 154], to steer social media posts towards appropriate and relevant first responders in a name-based network. Using data from recent California wildfires [164], We show that with little trained data and active learning, we can achieve online classification of tweets with high accuracy with near- real-time latency. This is also a very effective way of dissemination, compared to traditional methods of 911 operation (which gets intensely overloaded with limited availability during major disasters) and ad-hoc social media diffusion (such as re-tweets which have too much redundancy and low accuracy).

The organization of this dissertation is as follows: We provide background material on naming in networks in chapter 2, and go over related work in chapter 3. Chapter 4 describes Verification of Name-based Networks. In chapter 5, we present Interoperability of Name-based Networks. We propose Graph-based Namespaces and Load Sharing for Pub/Sub in chapter 6. Chapter 7 introduces Consistent Information Dissemination in Name-based DTNs. In chapter 8, we discuss Intelligent Information Name Assignment from Free-form Text. Finally, chapter 9 concludes this dissertation and describes future work.

Chapter 2

Overview: Naming and Addressing in Current and Future Internet Architectures

In this chapter, we provide a brief overview of the fundamentals of naming and addressing. We particularly focus on the evolution of naming and addressing and the relationship between them. We explore three networking models (Fig. 2.1): host-centric with location-dependence (§2.1), host-centric with location-independence (§2.2), and information-centric which is location-independent (§2.3).

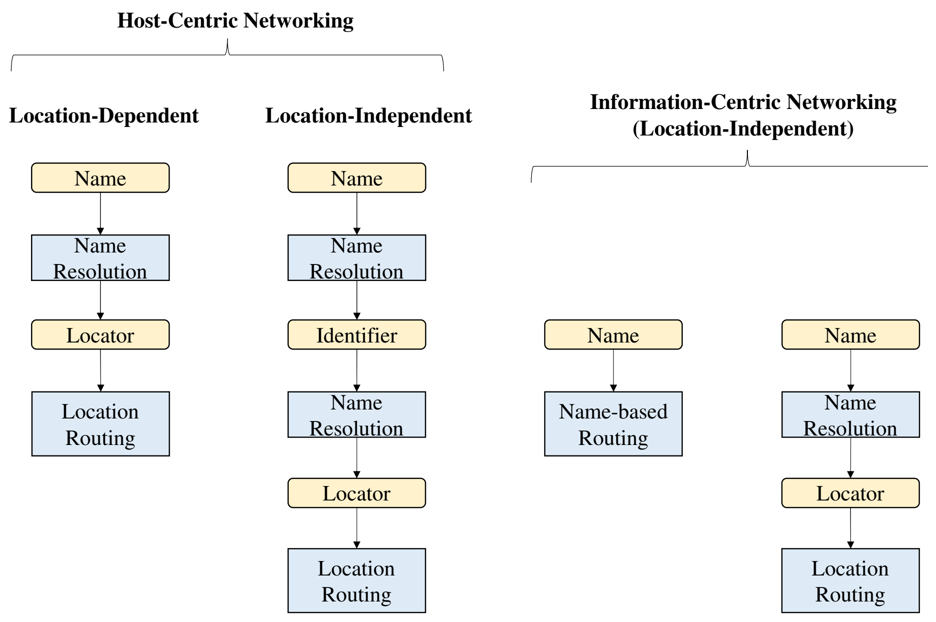


Figure 2.1: Different networking models: naming and addressing

2.1 Host-centric and Location-Dependent Addresses: Today's Internet

The current TCP/IP architecture of Internet today has been around for decades and is ubiquitous. In this architecture, IP addresses are used for forwarding, and is centered around a node's location; thus IP addresses are also locators. Host names are identified at the application layer (*e.g.*, in HTTP requests as URLs), only understandable by end hosts, and not routers. These host names get mapped to their corresponding location (i.e., their IP addresses) through a lookup-based name resolution services, namely the Domain Name Service (DNS). This design creates problems and challenges even for today's usage of networks. There have been several criticisms and challenges for this design, some of which we mention next.

2.1.1 Mobility

Mobility describes a situation in which a node moves from one point to another in the network, thus changing its point of attachment, *i.e.*, location. Since in the Internet the address of a node has everything to do with its geographical location (*i.e.*, which network/subnet it is attached to), the locator of the node has to change. In the host-centric design of the Internet, all routing and forwarding depends on locator. Therefore, as a result of mobility, the new location of a node will not be discovered and all packets to/from it will be lost/blackholed (not delivered or mis-delivered), until the node obtains the new IP address and the network forwarding state re-converges based on it. This causes excessive re-transmission and failures [84].

2.1.2 Multihoming

Most user devices today have multiple network interfaces and can potentially support multiple access technologies. End-host multihoming means an end-host has multiple addresses (locators) and thus multiple paths to/from it. With location-dependence, all those locators are seen as different end-hosts by the rest of the network (routers and other end-hosts). This also causes challenges such as failures when switching from one path to another, and not being able to efficiently utilize the multiple available paths. In addition to end-host multihoming, there can also be site multihoming. Site multihoming is a case where an end-site, such as an enterprise network, obtains IP connectivity from multiple different ISPs [165]. Such multihoming will cause excessively large forwarding tables in the routers, especially at the core of the network [112].

2.1.3 Site Renumbering

There can be situations where the IP prefix of a site changes, *e.g.*, a change in an ISP. Due to such renumbering, the site's address needs to be updated in various parts of the network, such as routing tables, DNS, and firewalls. Some of those changes may have to rely on manual human work and may be prone to errors. Nonetheless, updating them and invalidating the older sddresses are a challenge in location-dependant architectures [112].

2.1.4 Scalable Content Distribution

Many works have pointed out that the majority of the constantly growing Internet traffic consists of content (especially video) [92]. Mass distribution of content and replication in today's architecture is quite challenging: different requests cannot be aggregated and they cause excessive load on content servers and network resources, as every single request leads to a separate end-to-end channel (and all of its overhead such as TCP establishment, *etc.*). This problem stems from the fact that in TCP/IP, end-to-end paths for request/response are identified by content locations [21]. Some IP overlay solutions have aimed to alleviate this issue, namely Content Distribution Networks (CDNs) that enable geographically-distributed content caches. However, mapping requests to nearest CDN server, complexity of control plane management of CDNs, and the cost of deploying CDN servers are challenges of CDNs [79]. There is also considerable dependence and load on the DNS.

2.1.5 Security

In today's Internet, protocols such as TLS are used to secure end-end channels over which content is exchanged. In this model, to trust the content is to trust the end node that is hosting it. Therefore, the security of content is determined through the path to the location (identified by the address) of the end node hosting it. If the content gets altered or tampered with prior to entering the trusted channel, and loses its cryptographic protection right after leaving the channel, then trusting the host does not directly guarantee the content integrity [206]. Additionally, when many users request for the same content, the establishing of the many secure channels per request poses major scalability and manageability issues.

2.1.6 Challenged Networks/ Disruption-Tolerance

In many environments, the network is sparse and/or suffers from frequent disconnectivity. In such a network, *i.e.*, challenged networks, end-to-end paths are not always present. This poses major challenges for communication and content exchange in today's Internet as it relies on an end-to-end path and transport sessions that stay alive for the duration of the session over multiple hops [159]. While the challenges of such networks are unique to host-centric networks, location-dependence makes it especially challenging to support content exchange and dissemination in networks with high disruption.

2.2 Host-centric and Location-independent: Locator/ID Separation

To address some of the challenges, location-independent solutions over host-centric IP architectures have been proposed. The solutions mostly focus on addressing the first three challenges. In this paradigm, there is a decoupling/separation of host identity (ID) and IP addresses (locator). There have been a number of solutions proposed and standardized in this approach over the past two decades [112]; we describe some of the more notable ones here.

Site Multihoming by IPv6 Intermediation (SHIM6) [150] provides a sub-(shim-) layer above the network layer, called the upper-layer protocol (ULP). Assuming a host has several IPv6 addresses (locators), only one will be selected by ULP to communicate with the other host in a TCP session. This selected IPv6 address is called the upper-layer identifier (ULID), which identifies the transport flows between the hosts. The lower layer, i.e., IP routing sublayer, may use different IPv6 addresses for routing. This helps with multihoming, and provides seamlessness and resiliency of the transport-layer session in case of network failures [30].

Host Identity Protocol (HIP) [142] introduces Host Identiy (HI) and a HI (sub-) layer that sits between transport and network layer. In HIP sockets, the HI gets dynamically bound to a relevant IP address. HI is represented as an 128-bit public key, also called Host Identiy Tage (HIT). HIP uses new DNS Resource Record (RR) types to provide mapping from HI to routable address. Also, rendezvous servers maintain mapping to support mobility, which resolve HIs to addresses.

Locator/ID Separation Protocol (LISP) [67] splits the Internet's namespace into identifiers and locators (addresses). This particularly helps with the challenges regarding multihoming and renumbering. LISP introduces an Endpoint Identifier (EID) and Routing Locator (RLOC). EIDs are IP addresses (that do not depend on network topology) that get assigned to end hosts, playing the role of identifiers. RLOCs are IP addresses that are location-dependant (as regular IP addresses are used today), and help with routing. In other words, EIDs are non-routable and RLOCs are routable IP addresses in LISP. Edge routers near end hosts, called Ingress Tunnel Routers (ITR) and Egress Tunnel Routers (ETR), provide the means for tunneling between source and destination points. They perform encapsulation of packets into LISP tunnels to forward packets based on EIDs. Similar to HIP, LISP relies on DNS: it uses it to resolve domain names to EIDs. The mapping from EIDs to RLOCs (typically for the destination's ETR) is achieved through LISP-specific mapping mechanisms, such as LISP Delegated Database Tree.

Mobile IP [155, 156] is an architectural enhancement to regular IP to support mobility. It introduces a Home Address (HA) which is the identifier and a Care-of Address (CoA) which is the locator. Packets sent to the HA of a mobile node go to its home network, and the home agent forwards them to CoA through tunneling (called triangular routing). Upon further mobility, i.e. visiting additional new networks, the home agent will be informed (through signaling protocols for registration/binding) and update the CoA it maintains. While Mobile IP supports mobility, it introduces challenges such as latency overheads caused by its triangular routing.

2.3 Information-centric and Location-independent: Future Internet Architectures

While locator/identity split solutions solve some of the challenges regarding mobility, multihoming and renumbering, they still do not adequately address the other three challenges posed in §2.1. To address them, information-centric networks (ICNs) have been introduced [92].

In addition to separate identity from location, ICNs treat content as first-class entity, and enable in-network caching of content. Out of various network architectures proposed under the framework of ICN [182], two have gained major attentions: Named Data Networking (NDN) and MobilityFirst (MF).

NDN [203] is a network architecture made up of human-readable, hierarchical names for content, special packets Interest/Data that carry request/response for named content, and routers that are capable of forwarding and caching content. NDN routers store the name of each requested data in a Pending Interest Table (PIT) together with its incoming interface, and forward interests based on longest-prefix match in Forwarding Interest Table (FIB). On its path back, the content (response) gets cached at every router along the way in the router's Content Store (CS). As there is no FIB-based forwarding for the response and it gets downstream only based on PIT, NDN uses a Reverse Path Forwarding (RPF) policy. For every new request that arrives, a router only forwards it onto the proper interface based on FIB entries (longest prefix match) only if neither the content only exists in its CS (exact match) nor an interest for the same content is pending in PIT (exact match). For static content, i.e. content that does not change frequently or is independent of time of request, this is a major help and largely reduces total response time.

MobilityFirst (MF) [160] uses flat Globally Unique Identifiers (GUIDs) to identify every content, user, device, etc. A key component of MF, is a distributed Global Name Resolution Service (GNRS) that can be contacted by routers for name-to-address resolution. GNRS keeps a mapping between GUIDs and Network Addresses (NAs) and in that sense, is similar to Domain Name System (DNS); but with the difference that unlike DNS, GNRS is a network-layer service. MF helps a great deal when the majority of nodes are mobile, i.e. fixed GUIDs with frequently varying NAs. It is flexible in terms of when the name-to-address procedure takes place. It enables late binding, in which the packet is forwarded only based on GUID for most of the path, and at some point close to the destination, the router at that point asks the GNRS and the GNRS sends the most recently updated NA associated with that GUID to that router, and from then on, packet gets forwarded based on location until it reaches the destination.

ICN provides significant benefits for efficient scalable content distribution, especially when it comes to popular content. This is realized through the use of content name-aware network layer (as opposed to IP address-based network layer in IP), in-network name-indexed content caching at (potentially) every router (as opposed to no content caching in vanilla IP and limited application-layer caching in CDN), and named content request aggregation and multicast (as opposed to separate per-request end-to-end channel in IP). ICN provides content-oriented security, in which named content itself gets secured, regardless of which node is hosting/storing it and on which path it gets delivered. This de-coupling of consumer and producer, as well as name-based content caching (and store-and-forward content forwarding), helps provide a reliable and resilient content dissemination and delivery in delay/disruption-tolerant or challenged networks. In this thesis, we explore various aspects of naming and content dissemination to analyze and improve some of the said benefits.

Chapter 3

Related Work

3.1 Information-Centric Networking

Information-Centric Networking (ICN) enables access to named objects, independent of their locations. There have been a number of different ICN proposals in the past decade, *e.g.*, NDN [203], MobilityFirst (MF) [160], DONA [113], XIA [145], NetInf [57], and PURSUIT [75]. In this paper, we mainly focus on two notable ICNs, namely NDN and MF. There are differences between IP and ICNs [92], and also between different ICNs [15]. ICN has several key features and aspects, which we wish to support and preserve while enabling interoperation:

1. *Naming:* In ICN, the network layer is aware of names, while in IP it is only aware of addresses. Different ICNs have different naming schemas: NDN uses human-readable hierarchical names [203], while MF uses 20-byte flat names called GUIDs (Globally Unique Identifiers) [160]. An important service is Name Resolution Service (NRS),

which maps names to locations, either implicitly (FIBs in NDN [203]). or explicitly (DNS in IP or GNRS in MF [160]).

2. *Name-based forwarding and routing:* The ICN network layer makes forwarding and routing decisions based on names, which provides benefits such as location-independence and inherent support for mobility [92,160]. MF forwards both requests and responses based on the source/destination network address (like IP) after late-binding of the name to address, while NDN uses reverse path forwarding (RPF) policy for delivering the response back to the consumer, through Pending Interest Tables (PIT) [203].

3. *Connectionless transport:* While there has been some work on TCP-like additions to NDN [137] and MF [176], ICN primarily enables content request and retrieval without establishing an end-to-end channel, in contrast to today's HTTP/TCP/IP-based connection-oriented communication channel-based content retrieval [92].

4. *Content-oriented security:* ICN secures the data itself, as opposed to IP's channel-based and host-based security [182]. NDN uses a trust schema [198] while MF uses self-certifying objects [160] to ensure provenance and integrity.

5. *In-network content caching:* ICNs typically cache content, indexed by names, at every router [202]. This extends the selective and limited CDN-like caching done in today's IP. Many studies have shown ICN caching to be very beneficial for reducing the response time for content delivery as well as availability, especially at the edge [68,122].

3.2 Network Verification and NDN Diagnostics

Network verification aims at analyzing large, complicated networks in order to find corner case errors and investigate essential properties. There have been efforts to build models to describe and verify networks. For the purpose of building verification frameworks, some works focus on analyzing control plane (to analyze all data planes caused by configurations) and some on data plane (to analyze the current state of the network). Computational feasibility and full verification coverage are challenges of control plane verification [31, 152]. We focus on data plane verification in this paper. Some of the more notable data plane verification tools are Anteater [128], HSA [107], VeriFlow [110], and NetPlumber [106]. These methods typically consist of snapshot-based static checking. Anteater [128] models the data plane as a set of boolean expressions and runs a SAT solver to verify invariants. HSA [107] uses a geometric view of packet headers, not making any presupposition about what each packet header element represents, thus making it a flexible model for integration for new network architectures. Some verification tools additionally support real-time checking of network policies of Software-Defined Networks (SDN) such as VeriFlow [110] and NetPlumber [106]. These methods leverage and rely on control update messages issued by the centralized SDN controller for fast, incremental checking of network data planes. Thus, they can react to changes before those changes are applied to every one of the associated routers. Our proposed model is a generic one, with no assumption on how the network is managed. However, if we have NDN integrated with SDN, real-time verification using control update messages may be leveraged. Work in [31, 63] propose data plane equivalence checks. While they focus on equivalence pertaining to host-centric properties, NSA can check information-centric equivalence, *e.g.*, checking if the same subset of the content namespace of a particular content provider is reachable in two (or multiple) data plane snapshots. This is important since we may need to have multiple snapshots each with the desired differences, and compare them against each other, *i.e.*, cross checks, where the goal is not to conform to an external property, but rather to compare against a complete, separate snapshot in time [63].

An important prerequisite of data plane verification is collecting the current state of the data plane in the form of a snapshot. Based on how the network is managed, different methods can be used for this collection procedure. With traditional non-SDN networks, methods such as SNMP [132], NETCONF [89] or node-specific terminals [201], can be used to collect FIBs and topology information. NDNconf [16] presents an NDN-ized version of NETCONF, to collect NDN-specific FIBs. In addition to the capability of querying, NDNconf allows for a push-based notification of changes in the network state to management servers, which helps with real-time collection of up-to-date snapshots. SDN-controlled networks can support a more efficient snapshot collection by monitoring the forwarding rule updates (insert, modification or deletion) on the southbound interface [124]. Snapshot collection and verification are two logically independent procedures. NSA focuses on the verification component, while leveraging the complementary support of these snapshot collection methods.

Our work presents an NDN-specific verification framework. Diagnostic tools such as Ping [129] and Traceroute [26, 109, 130, 171] have been proposed and developed for NDN. While these tools are very helpful for performance measurements and small-scale connectivity checks, they are often limited in high-coverage checks across the network in a scalable way, and also use network resources. Thus, a formal approach gives us a higher level of flexibility and coverage for property checking [107].

3.3 ICN Interoperability

There have been different recent approaches for interoperability involving ICNs (surveyed in [53]):

Tunneling (Overlay/Underlay)

Some approaches use ICN-over-IP tunneling. For example, the basic design of [168] is an NDN-overlay: NDN packets are encapsulated into UDP, TCP or native IP packets traversing IP routers. This enables incremental deployment of ICN over IP and has been used as the starting point for development of software packages of most ICN architectures [46, 57, 75, 92]. Work in [133] introduces a layer 3.5 as overlay, and this layer allows new architectures such as NDN to run. In this approach, each new architecture would have its own layer 3.5 protocol, having to go through the overhead of mapping from/to the underlying layers. Similarly, each overlay has its own naming schema. However, the work does not go into details on how the right name to use is chosen or obtained by a requesting client. IP-over-ICN solutions, such as [137,170,183], allow legacy (HTTP/TCP) applications function across an ICN infrastructure, where IP packets are encapsulated in NDN headers, which get decapsulated when leaving the NDN domain. These solutions typically assume a single ICN architecture universally deployed (*e.g.*, NDN [203]) and build IP capabilities on top of it. Also, they deal with added IP-to-ICN (and back) mapping latencies at certain routers on the path [53].

Hybrid Approach

An approach to enable evolution to new architectures and interoperation is to add the semantics of a new architecture into an existing one. This results in a new hybrid network layer that is backward-compatible with the native version of the original architecture. CLIP [85] uses an IPv6 subnet prefix for content to enable ICN in IP. Work in [158] proposes the combination of HTTP and ICN, arguing that they both follow a content-centric pattern. Most recently, hICN [41] proposes to encode NDN-specific components into IPv6, and allows the coexistence of IP and ICN dual stacks at hICN-enabled routers (capable of processing both legacy and ICN-enhanced IP packets), while also making use of regular IP routers (capable of processing legacy IP packets). Consumers and data providers, however, still need to have the same semantic understanding, *e.g.*, in terms of naming and how "network" names get mapped to "application" names.

Translation

Solutions in [39,126,189] perform direct translation between HTTP and NDN/MF traffic. Translation-based interoperability solutions bring great advantages, such as not having to change domain-specific mechanisms. Work in [189] further optimizes the ability to cache in the network by adding heuristic rules. Moiseenko *et al.* [138] modify NDN packets to better support HTTP-like communication (*e.g.*, uploading large data using POST). These solutions either support only 2 domains (IP plus either NDN or MF), do not support some of the key domain capabilities, and/or require heavy changes to end nodes and routers in existing domains. Our approach overcomes these shortcomings using translation-based stateful gateways for interoperating multiple domains with different architectures, while preserving their key features.

3.4 Publish/Subscribe

Publish/Subscribe (pub/sub) systems has become a widely used, popular service over the Internet, in form of RSS feeds, online social networks, *etc.* Most popular pub/sub solutions today are server-based; where subscribers either poll a logically centralized server (via HTTP), or a long-lived connection for timely delivery is maintained [167]. These approaches can be limited in scalability. Broker-based solutions (*e.g.*, ONYX [60], TERA [29]) use an overlay network with distributed brokers, and avoid traffic concentration. However, the dependency of these solutions on XML data and assertions to decide forwarding paths, couples the information structure with the network layer, making the forwarding function complex. Having pub/sub in the network can help with scalability, and has been proposed for ICNs [49,75,144]. ICN with push-based publish/subscribe service models [49] have been proposed. COPSS [49] enhances the query/response model of NDN by allowing consumers to issue a long-standing request, *i.e.*, subscription, for all content related to (subsets of) a name, whenever they are published. It can outperform IP multicast, poll-based methods and flooding-based broadcast [49, 50], in terms of aggregate network load and latency. CNS [48] extends COPSS by introducing recipient-based pub/sub, that can help with information dissemination. Our work moves a step further by relieving the strict hierarchy restriction to enable complex free-form graph-based namespaces.

3.5 Graph-based Information Organization and Partitioning

Graph-based information organization has been gaining attention and shown to be important because of its richness and efficiency compared to the more traditional hierarchical structures across multiple application domains. Wikipedia is a very popular and notable example of an information organization system designed as a graph structure: each article can belong to a number of categories, *i.e.*, dimensions [13]. Graph structures for information have been proposed and used in many other contexts as well, *e.g.*, databases [25, 74], cloud computing [12], and file systems [59]. These works have primarily focused on information organization for storage and indexing. Our work focuses on in-network information organization for name-based information dissemination, extending the current hierarchical structure of NDN [203] to a graph-based one.

Graph partitioning is an important graph operation, and has been an area of research for decades. Optimal graph partitioning is considered to be NP-hard [69], so solutions based on heuristics and approximations exist. A very well-known partitioning method is multi-level partitioning [104] (and its tool METIS [103]), which using heuristics, coarsens the graph, does an initial partitioning, and then uncoarsens it. METIS has been widely used for load splitting and balancing in various contexts [34,207]. The parallel version of METIS, ParMETIS [105], achieves speedups using message passing interface (MPI)-based parallel processing. ParMETIS also includes additional routines for adaptive re-partitioning, to enable fast incremental updates to an already-existing partitioning, in case of weight changes in the graph, rather than complete re-partitioning from scratch. Some methods use the streaming graph partitioning approach which is used to process partitioning a piece of data on the fly, *e.g.*, [175]. These algorithms are very fast but their solution quality is lower. This approach is most suitable for extremely large graphs (in the order of trillion vertices). There are approaches using iterative improvement methods for graph partitioning. These methods typically provide high quality solutions. A bad choice of the iterative method and its parameters can make the procedure slow. Work in [162] proposes a graph partitioning algorithm using Tabu search, and shows that it outperforms another popular meta-heuristic method, Simulated Annealing [102], regarding both solution quality and timeliness. Sometimes the objective of partitioning is more than a simple sum of weights, and can be a complex function of the cut itself. This is characterized as the "chicken and egg problem" in [157], as the objective function needed for partitioning decision must be calculated after the partitioning is done; ours belongs to this class. Approaches to solve this class of problems have been proposed in works such as [36, 143, 184] for specific cases. The work in [157] justifies the use of standard partitioning as a good starting point, and then perturb it during the iterative refinement procedures.

3.6 Causal Consistency and Consensus

Causal consistency is a popular consistency model which ensures ordering of events (*e.g.*, network messages) based on their causal relationship. Works such as [111] propose the use of physical clocks for ordering. However, physical clocks may have skews. The protocol for clock synchronization may involve significant overhead, especially in a disconnected environment. Scalar logical clock [115] defines the "happened before" relation. A message is said to be *causally delivered* at a recipient user, if all the causal prerequisites of that message have been delivered at the user too [40]. The Vector clock method [73, 131] ensures causal ordering using vectors carried as history in each message, that represent the sender's current state relative to every other users' progress. Work in [173] proposes differential clocks as an optimization to vector clocks, only sending vector differences. [28] proposes that explicitly specifying causality, by sender, helps with scalability. Work in [58] proposes a method for group causal ordering, and enabling causal delivery to multiple groups of interested users. We use the notion of vector clock but extend it to enable selectiveness through hierarchically-structured naming and a reactive mode for faster causal delivery, and capture both implicit and explicit causality.

There has been a great deal of work on consensus, the most prominent of which is Paxos [117]. Paxos achieves agreement among a number of networked nodes, by election of a leader, majority voting, and deciding on a value, through a number of rounds. Raft [151] implements Paxos, designed for strong consistency in log replication among servers in a cluster. While such solutions work well in connected networks and (partially) synchronous systems (*i.e.*, known upper bound on message latency), it has been shown in [44] that they are not suitable for disconnected environments. Their fault recovery, through Failure Detectors [76], is typically limited to node failures rather than link failures. The Heard-Of model [44] proposes a benign fault model, and proves that the consensus algorithms Paxos/LastVoting (P/LV) [37] and One-Third Rule (OTR) [32] can tolerate loss and be suitable for intermittently-connected and mobile environments. The model demonstrates that rather than assuming eventual synchrony, it is more realistic to assume “good periods” in asynchronous systems, *i.e.*, an epoch in which nodes can hear of each other (receive their messages). Work in [38] proves the one-third rule to reach correct consensus and possibly finish in one round, as long as no more than one third of the consensus participants crash. Another benefit of OTR over P/LV is that it is coordinator-less, and thus does not need to have the overhead and complexity of leader election. We build on OTR, enhancing it with an integration of naming and adding support for cases where decisions need to be invalidated *e.g.*, due to long-term network fragmentation.

3.7 Propagation in Intermittently-Connected Environments

There have been a number of works on information propagation in intermittently-connected networks [22]. Generally, these solutions rely on nodes to store, carry, and forward messages [66]. Most solutions rely on nodes taking advantage of opportunistic encounters to exchange messages (*i.e.*, gossiping), typically with high message delivery latency due to disconnections [83, 174, 185]. Methods such as Bubble Rap [88], dLife [140], SCORP [141], and EpSoc [120] use social data regarding human interactions as the basis of such routing predictions. We use Epidemic Routing [185] in this paper because of its simplicity for DTNs and the fact that it requires minimum assumptions about network (no path/geography/social-connection based decisions) which suits our scenarios, has a high delivery ratio, achieves lower delays (relatively), and is especially suitable for broadcast-oriented messaging [22] (although we can replace it with the some of the other methods mentioned if additional assumptions are reasonable and can be accommodated). Epidemic routing uses message buffers and performs store-and-forward [185]. Apart from its benefits, it is observed that epidemic routing has high overhead [22]. We enhance it with the use of naming, to reduce load.

The use of network naming for systematic organization of information for better dissemination efficiency has been introduced as the integral part [97] of the Information-Centric paradigms, such as in Named Data Networks (NDN) [203]. Work in [187] provides name-based DTN-like dissemination frameworks. However, extra steps are needed for ensuring consistent dissemination. Some works [32] propose the use of interest profiling for selective gossiping. We extend their ideas to implement the content-oriented graph-based naming for profiling as well as proposing a flexible multi-level profiling for various consistency levels. Methods such as NDN Sync [123] and Secure Scuttlebutt [178] propose log replication consistency in name-based intermittently-connected environments. However, they only guarantee causal consistency, but do not provide strong consistency or total ordering, which are important when dealing with multi-user updates on a single, shared database. Naxos [188] proposes a name-based version of Paxos for NDN. However, Naxos only supports request/response pull-based communication pattern, and assumes connected environments with centralized orchestration. We integrate name-based publish/subscribe push-based dissemination patterns, and aim at ensuring strong total order consistency by supporting consensus in dynamic intermittently-connected environments.

3.8 Use of Social Media for Information Dissemination in Disasters

There have been many studies characterizing and designing network and communication frameworks for disaster management (surveyed in [197], [64]). Disasters can have major network-related impacts such as infrastructure damage and excessive congestion [64]. Different communication and network technologies have been proposed for disaster management, including Cellular, P2P and Satellite [197]. In addition to technological challenges, there are known social and organizational challenges when designing an effective and efficient network architecture for disasters [197]: a common language between organizations and citizens, as well as a structured, non-ad-hoc organization of disaster response is needed and desired [197].

Social media has been increasingly used for information dissemination in incident response, which can be very beneficial, especially when traditional means of communications, e.g., 911, are down or overloaded [54, 101, 164]. Social media server-based extensions and plugins have been developed for help during disasters, providing users with useful information, e.g., updates, warnings, offers, maps, etc [101]. As recent examples, social media has shown to be very useful to communicate and exchange critical disaster-related information to request and offer help, in California Wildfires in 2018 [164]. Our work integrates social media with name-based communication and content dissemination, to intelligently guide social media posts to the right recipients, as opposed to the currently unstructured ways for dissemination, e.g., in form of retweets.

Chapter 4

Name Space Analysis: Verification of Named Data Network Data Planes

4.1 Introduction

In this chapter, we present Name Space Analysis (NSA), a framework for modeling and verification of NDN data planes. NSA is based on the theory of Header Space Analysis (HSA) [107]. HSA uses a geometric view of packet headers, where each packet header is generally modeled as a point in a space and network functions transform that point to another one within that space. Additionally, the ability to analyze a "space" rather than a "single point", makes this an efficient analysis approach. This flexibility and efficiency make it a good formalism for integration in the analysis of NDN. We add another geometric space in NSA, namely the *name space*, and a new function, *name space function*, that transforms a point in the header space domain to a (collection of) point(s) in the name space domain. We extend HSA by enabling flexible atoms and variable-size wildcards to model headers (to support NDN-specific packet formats [146]), and adding name spaces as an essential part of the analysis. Analyzing name spaces in NDN is necessary and very useful as they are key to accessing content. We propose NDN-specific properties that can be checked by NSA; *e.g.*, in NDN we are interested in verifying host-to-content reachability, rather than the host-to-host reachability requirement expected of traditional host-centric IP networks. NSA has a number of verification applications (to prove key properties), namely content reachability test, name-based loop detection, and name leakage detection. We additionally support verification applications that go across multiple snapshots to analyze changes between multiple states (*e.g.*, consistent producer mobility check) and report on their equivalence. The complex structure of names in NDN may cause issues, *e.g.*, interfering prefix announcements by two non-coordinated data producers, which can potentially lead to blackholes. We call this *name space conflicts* and show how NSA can identify them. The importance of having a name management method in NDN has been identified in [52, 179]. Using the concepts of NSA, we propose a name registration method that can catch and resolve such conflicts in the data plane. We implemented NSA, including all the essential components of our design: name atoms, set operations, transfer functions, state space generation, and verification applications. We also identified a number of optimizations, and evaluating our implementation on synthetic snapshots and real-world NDN testbed snapshots shows that NSA is effective, efficient and scalable.

Overall, the contributions of this work are: 1) a framework for verification of NDN data planes, focusing on the nature of NDN, rather than the previous tools for host-centric architectures; 2) modeling name spaces and name space functions, as they are the main assets required to access content in NDN; 3) specifying essential NDN-specific properties and approaches to analyze them (content reachability test, name-based loop detection, name (space) leakage detection, and cross-snapshot equivalence checks); 4) studying the practical issue of name space conflicts in NDN and guidelines for a conflict detection and resolution engine; 5) an implementation of NSA [99] with its optimizations; and 6) demonstrating NSA's applicability to the real-world NDN testbed [148].

4.2 NSA Design

In this section, we describe the formal foundations and building blocks of NSA, focusing on the components that we are adding or are different from the original HSA and demonstrate them by examples from NDN.

4.2.1 Modeling NDN Header Space

Atoms and Header Representation

The atoms of analysis in HSA are bits, since some fields can be encoded as single bits in IP. An NDN packet, on the other hand, is a set of nested Type-Length-Value (TLV) codes represented as octets [146]. Thus, the smallest possible atom in NSA is octets (bytes). With byte-based atoms, NSA header representations follow NDN's TLV octet-based encoding. Other atoms could be picked as well: *e.g.*, if checking the correctness of TLV encoding is not important in a particular analysis, atoms can be NDN fields. With field atoms, NSA header representation will be an XML-like structure. If only the name field needs to be checked, atoms can be names. With name atoms, NSA headers are represented as a combination of name components, similar to NDN regular expressions [147]. Unlike HSA's strict use of bit atoms, NSA provides the flexibility of using byte, field and name atoms for header representation. The correct atom depends on the scope of verification and the desired level of abstraction.

Unlike IP packet headers, NDN does not have a fixed header with fixed fields at fixed positions. Interest and Data packets have different types. Normally, an NDN Interest has only headers; thus, we use the terms "packet" and "header" for NDN interchangeably, throughout this chapter.

NSA can model headers of any length; however, for the sake of checking finiteness, an upper bound L (maximum header length) has to be set. Still, headers of different lengths can be processed together; variable-length wildcard atoms provide the necessary padding to facilitate this.

Wildcard Expressions

In order to efficiently model and process a header space rather than a single point, *i.e.*, a single header, we use special wildcard elements to represent atoms that can take any possible value. Wildcard expressions are supported by the set operation as we explain below.

Single-atom wildcard. Similar to the original HSA, albeit using flexible atoms rather than only bits, we sometimes use a wildcard of size one, denoted as "`[?]`", and defined as $\texttt{[?]} = a_1 \cup a_2 \cup \cdots \cup a_n$, where a_i is a possible value for an atom and n is the number of possible values for an atom; *e.g.*, with byte atoms, we have $n = 256$.

Variable-length wildcard. Unlike IP headers, the NDN header has a flexible format and there is no rule on how much information should exist between two particular fields. To efficiently incorporate this feature into NSA, we add a new wildcard type: variable-length wildcard, denoted by "$\texttt{[}*\texttt{]}$", which can be a wildcard of any size (zero or more atoms) up to the size allowed for the maximum header length. Formally,

$\texttt{[}*\texttt{]} = \varnothing \cup \texttt{[?]} \cup \texttt{[?][?]} \cup \ldots$ until length allowable by L.

Note that the "$\texttt{[}*\texttt{]}$" wildcard is not currently part of the NDN architecture; we use it as part of NSA headers for the model's representation and verification efficiency, to be used in a *symbolic execution* fashion, which we explain in §4.3.

Set Operations

Set operations are important for manipulating header spaces in order to model packet processing through transfer functions. We use a similar algebra as HSA, with the difference being that we use variable-length wildcards and flexible atoms.

Union. This is the basic operation. For header spaces h_1 and h_2, header space $h = h_1 \cup h_2$ contains all headers in h_1 and h_2. Result of union may or may not be simplifiable.

Intersection. For two headers to have a non-empty intersection, they should be of equal length and have the same values (or wildcard element) at the same position. To convert length, "$\texttt{[}*\texttt{]}$" should be converted by an appropriate number of "$\texttt{[?]}$'s", as explained above. At the atom-level, we have $a \cap a = a$, $a \cap \texttt{[?]} = a$. For two unequal atom values, $a_1 \cap a_2 = \texttt{[z]}$. Special atom "$\texttt{[z]}$" denotes an atom that has zero possible values, *i.e.*, null (empty). A header space h that has even one "`[z]`" is regarded as empty. Also, intersection of any atom-string with an all-wildcard "`[*]`" header will be the atom-string itself.

Complementation. Complement of non-wildcard atom a, denoted as $\overline{a}$, can take any values other than that of a.

Difference. Difference of two headers is defined as $h = h_1 - h_2 = h_1 \cap \overline{h_2}$. For example, with byte atoms, using these set operators, we will have:

$\texttt{ab?} - \texttt{abc} = \texttt{ab?} \cap (\overline{\texttt{abc}}) = \texttt{ab?} \cap (\overline{\texttt{a}}\texttt{bc} \cup \texttt{a}\overline{\texttt{b}}\texttt{c} \cup \texttt{ab}\overline{\texttt{c}} \cup \overline{\texttt{a}}\overline{\texttt{b}}\texttt{c} \cup \texttt{a}\overline{\texttt{b}}\overline{\texttt{c}} \cup \overline{\texttt{a}}\texttt{b}\overline{\texttt{c}} \cup \overline{\texttt{a}}\overline{\texttt{b}}\overline{\texttt{c}}) =$
$\varnothing \cup \varnothing \cup \texttt{ab}\overline{\texttt{c}} \cup \varnothing \cup \varnothing \cup \varnothing \cup \varnothing = \texttt{ab}\overline{\texttt{c}}$

This basically means any three-byte string starting with "`/a/b`" but not (*i.e.*, minus) "`/a/b/c`".

4.2.2 Modeling NDN Nodes

Packet processing in an NDN node is modeled using Network Transfer Functions, as

$$T(h, f) : T(h_0, f_0) \to \{(h_1, f_1), (h_2, f_2), \dots\}$$

where a function T maps header h_0 coming to face f_0, to all headers h_1, h_2, *etc.*, going out of faces f_1, f_2, *etc.* of the node respectively. NSA's transfer functions are at the level of a face, rather than being port-level as in HSA. While this does not change the algebraic operations on the transfer functions, in practice it does enable one to write such functions using faces, regardless of the underlying strategies associated with those faces. Domain and range of NSA transfer functions are of the same type (both Interest or both Data headers).

Transitioning from Interest to Data is not a part of NSA verification as it requires changing the state of the data plane. Depending on the functionality being modeled, function T may or may not change h_0, and may or may not depend on the incoming face f_0. Any NDN packet processing, including an NDN forwarding behavior, can be modeled using (a set of) transfer functions.

For example, the transfer function for forwarding an Interest as a result of the Longest Prefix Matching (LPM) on the FIB, assuming there are two entries with indexes (prefixes) n_1 and n_2 in the FIB, can be written as:

$$T_{I.fwd}(h,f) = \begin{cases} \bigcup(h, f_i^{n_1}), & \text{if } FIBM(name(h), n_1), \\ & \forall f_i^{n_1} \in SF(n_1) \\ \bigcup(h, f_i^{n_2}), & \text{if } FIBM(name(h), n_2), \\ & \forall f_i^{n_2} \in SF(n_2) \\ \varnothing, & \text{otherwise} \end{cases}$$

where the FIB is a collection of *(prefix, set of faces)* pairs; assuming the use of LPM, the FIB match function $FIBM()$ returns true for at most one FIB entry; and depending on forwarding strategy, *i.e.*, best route, multicast, *etc.*, the function $SF()$ (selected faces) will return the appropriate corresponding outgoing faces.

In general, a typical Interest processing transfer function can be modeled as $T_I(.) = T_{I.fwd}(T_{I.CS}(T_{I.PIT}(.)))$. What elements we put into a transfer function depends on our architecture and the purpose of the analysis. For example, if we have the assumption of the CS and PIT being empty upon the arrival of an Interest, then we can simply have

$T_I(.) = T_{I.fwd}(.)$. Additional functions can be added to the pipeline as well, including those that modify the incoming header space, *e.g.*, function $T_{HopLimit}$ that decrements the *HopLimit* field in the Interest [146] if it is above zero and passes it to the subsequent transfer function in the pipeline, and drops it otherwise:

$$T_{HopLimit}(h,f) = \begin{cases} (h',f), & \text{if } HopLimit(h) > 0, \\ & HopLimit(h') = HopLimit(h) - 1 \\ \varnothing, & \text{otherwise} \end{cases}$$

Using similar patterns, we can model Data forwarding or any additional Interest forwarding transfer functions such as the full forwarding pipelines in the NFD specification [19] or link object processing [20], complicated forwarding strategies and Nonce checks, *etc.* A packet processing pipeline can be modeled as a cascade of functions, *i.e.*, $T_n(T_{n-1}(\ldots T_1(.)))$ where each T_i is a specific function (step) in the pipeline. It can also be a named function, performing an operation if the Interest name has a particular prefix; this operation can involve changing the name in the header. *E.g.*, an arbitrary Interest anonymizer, that encrypts the name with key K and encryption algorithm *Enc*, and triggered by the prefix "`/Anon`", will have a transfer function in its Interest process pipeline, as follows:

$$T_{anon}(h,f) = \begin{cases} (Enc(h,K),f), & \text{if } prefix(h) = \texttt{"/Anon"} \\ (h,f), & \text{otherwise} \end{cases}$$

Generally, a condition on a header is modeled as a header space (which may or may not have wildcard expressions) and the result depends on the output of a logic operation on the incoming header and the condition. This depends on the process and the condition and may in some cases be tricky. *E.g.*, for LPM checking, for a header to be forwarded out of a face, the FIB entry index corresponding to that face has to be a prefix of the header's name (non-empty intersection) in the Interest, and a longer FIB index must not be a prefix of that header (empty intersection). For example, consider an NDN node with FIB consisting of two rules "/a $\rightarrow$ f1" and "/a/b $\rightarrow$ f2". Given an all-wildcard input header, Interest headers coming out of face f1 are those whose names start with "`/a/`" (*i.e.*, "`/a/`$*$") and not with "`/a/b`" (*i.e.*, "`/a/b/`$*$").

NSA does the conversion of the NDN FIB table to NSA transfer function. For a FIB table with e entries, the worst case complexity of this procedure would be $O(e^2D2^d)$: for every entry e_i, we need to check all other entries to find descendants, *i.e.*, at finer granularity of e_i. For each descendant of e_i, *i.e.*, e_i^j, $2^{d_{ij}}$ corresponding NSA rules need to be generated, where d_{ij} is the granularity distance between e_i and e_{ij}. *E.g.*, granularity distance of prefixes $e_1 =$"`/a`" and $e_2 =$"`/a/b/c`" is 2, as e_2 is a descendant of e_1 and has two additional name components. As a result, corresponding to e_1, NSA would create rules for "`/a/`$\overline{\texttt{b}}$`/c/`$*$", "`/a/b/`$\overline{\texttt{c}}$`/`$*$", and "`/a/`$\overline{\texttt{b}}$`/`$\overline{\texttt{c}}$`/`$*$" for the network transfer functions (so the outcome would be determined by the intersection of incoming header to every rule). D and d denote the maximum number of descendants and granularity distance in the given FIB table.

4.2.3 Modeling Name Spaces

We add the notion of name spaces as a key component of our analysis approach. Name spaces show relations between content names, in a content repository and across the network. They are an important part of NDN, and NSA factors them carefully in its analysis. As far as NSA is concerned, a name space is any structure representable by a graph. We assume a special case of that, namely NDN-style hierarchically structured tries (prefix trees), in this chapter.

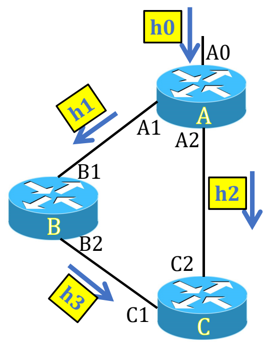
Formally, a name space in NSA represents names and their relations, and is a domain separate from the header space domain. A name space function, transforms a point in the header space domain to a name space domain, *i.e.*, its corresponding name(s). Name space function $\Omega()$ is introduced in NSA. It transforms a (set of) header space(s) (after parsing it to the individual name parts) to a name space. In particular, $\Omega()$ performs the following two steps on an input header space h: 1) extracts the (prefix) names associated with h, 2) provides the reverse construction of the prefix tree from the list of prefixes derived in step 1. This resulting prefix tree is the name space, used in NSA verification applications.

4.3 Verification Applications in NSA

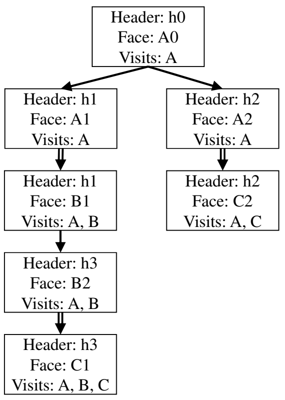
This section explains a number of verification applications that NSA can check. Specifically, we look at the important applications of testing content reachability, loop detection and name leakage detection. NSA provides significant benefits both in terms of the verification results and its efficiency compared to simulation-based tests.

An important part of NSA's formal verification approach that facilitates automated checking is the generation and analysis of the state space, or *propagation graph*. The graph represents all possible *paths* any packet can take, rather than a single trace that a simulation-based approach would support. This provides the desired coverage we need for verification. An example is shown in Fig. 4.1. Each node in this propagation graph is a *state*, mainly consisting of a header and a face, denoting the arrival/departure of the header to/from the face. Depending on the specific application, there can be additional state information, such as a visited nodes list, *e.g.*, for loop detection. We record as much information within a state (*e.g.*, list of visited nodes) as needed, so that checks could be done by looking at the state only, so that extra processing on the graph would not be necessary (*e.g.*, checking all ancestors of a state by traversing paths). The initial states, *i.e.*, parent-less nodes in the graph, represent injections to the network. For example, in Fig. 4.1, the propagation graph (Fig. 4.1(b)) implies that header $h0$ is injected to face $A0$ of node A (as shown in Fig. 4.1(a) as well). State transitions in the propagation graph can be through network transfer functions (*i.e.*, processing packets within a node) represented by single arrows in Fig. 4.1(b), or topology transfer functions (*i.e.*, moving over physical links) shown by double arrows.

While NSA can be used for both Interest and Data packets, we focus on Interests in the remainder of this chapter. As pointed out in [19], Interest processing is more complicated than Data processing: it has longer, more complicated pipelines, has additional procedures such as forwarding strategy selection, and its forwarding decisions are made through the



(a) Topology and header spaces (yellow boxes: header spaces are created and transferred on links)



(b) Propagation graph ($\longrightarrow$: network transfer function transitions; $\Longrightarrow$: topology transfer function transitions)

Figure 4.1: Propagation graph example

result (*i.e.*, FIB) of complicated distributed algorithms (*i.e.*, routing protocols). All these motivate more careful attention.

4.3.1 Content Reachability Test

Reachability analysis in HSA, and other host-based verification solutions, focuses on host (content provider) reachability. We extend this to content (name space) reachability in NSA, since this is a main concern in NDN. This analysis generates name spaces that can reach content repositories, *i.e.*, at producers or content stores. To this end, we apply a name space function on the header space received at a content repository:

$$CR_{A\to B}(h,f) = \bigcup_{A\to B \text{ paths}}\{\Omega(T_n(\Gamma(T_{n-1}(\ldots\Gamma(T_1(h,f))))))\}$$

where CR denotes the *content reachability function*, its range being all the content names, in form of name spaces, received at content repository B, having injected h at face f of A, and functions T_i and Γ_i being switch network and topology transfer functions on the path, respectively. Function Ω is the name space function that transforms header spaces to name spaces.

A big part of name space reachability analysis is comparing the received name space request, *i.e.*, $NS_B^{rcv} = \Omega(h_B)$ with the hosted (actual) name space NS_B^{hos} at node B, where B is a content provider (or a router equipped with a content store). Ideally, we desire both name spaces, NS_B^{rcv} and NS_B^{hos} to be equal. Generally, there can be three cases possible when comparing NS_B^{rcv} and NS_B^{hos}:

1. If part of NS_B^{rcv} is not in NS_B^{hos} (Case 1: *unsolicited names*), it means B would receive Interests for names the node does not have, *i.e.*, those packets get blackholed.

2. If part of NS_B^{hos} is not in NS_B^{rcv} (Case 2: *unreachable names*), it means part of B's name space is untouched, *i.e.*, requests for them would never be received. Cases 1 and 2 need not be disjoint.

3. If neither cases occur, verification is successful, *i.e.*, $NS_B^{hos} = NS_B^{rcv}$ (Case 3).

The process is exemplified in Fig. 4.2, where header space h_A injected at host A traverses nodes (*e.g.*, routers) with transfer functions T_C and T_B where the header space h_B gets transformed and compared with the content name space at B. Node B can generally be any node in the network that has the capability of storing and serving content, be it a content publisher or an ICN-capable router with content store.

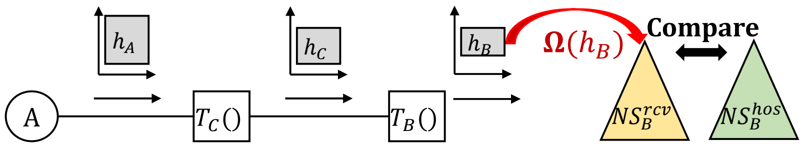


Figure 4.2: Content reachability test

Algorithm 1 specifies the application for the name space reachability test in a network, denoted by its *Network Space* N, which is the collection of all name spaces, and transfer and transform functions. Starting from an initial header space, typically of all-wildcard elements, this application generates output headers of each node, step-by-step, by walking through the header propagation graph. It can start from one (as shown in Algorithm 1), or any arbitrary number of consumers. The name space functions and comparisons are applied and performed at all the nodes in the network that are considered content providers. The application finds all case 1 and case 2 errors for each content provider and also returns the overall verification result, as either True (verification success, no bugs found) or False (verification failure, bugs exists), for the whole network. In the case of verification failure, NSA can provide the counterexamples, *i.e.*, "unsolicited" or "unreachable" names at each content repository.

The time complexity of an NSA content reachability test for injecting a header to a consumer that leads to a single content provider is $O(dLR^2s)$, where d, L, R, and s are maximum network diameter (number of hops), maximum header length, maximum number of node rules, and maximum number of paths in a trie-based content provider name space. This analysis is based on the *linear fragmentation* assumption in [107], which says that

Algorithm 1 Content Reachability Test

```
1: procedure ConReach($C, h_0, N$)    ▷ Injecting $h_0$, at $C$, network space $N$
2:     Start with $h_0$ at $C$    ▷ Typically all wildcard, i.e., [*]
3:     Calculate all $h_{P_i}$'s    ▷ Headers reached at provider $P_i$
4:     for all $P_i$ do
5:         $NS_{P_i}^{rcv} \leftarrow \Omega(h_{P_i})$
6:         $NS_{P_i}^{UW} \leftarrow NS_{P_i}^{rcv} - NS_{P_i}^{hos}$    ▷ 'Unsolicited' names
7:         $NS_{P_i}^{UR} \leftarrow NS_{P_i}^{hos} - NS_{P_i}^{rcv}$    ▷ 'Unreachable' names
8:         if $NS_{P_i}^{UR} \cup NS_{P_i}^{UW} = \varnothing$ then
9:             $Result_{P_i} \leftarrow$ True    ▷ Success at $P_i$
10:        else
11:            $Result_{P_i} \leftarrow$ False    ▷ Failure at $P_i$
12:        end if
13:    end for
14:    return $\bigwedge_{\text{all} P_i\text{'s}} Result_{P_i}$    ▷ Overall verification result
15: end procedure
```

typically very few rules in a node match an incoming packet. Unlike NSA, the complexity of a simulation-based test would be $O(da^L Rs)$, where a is the maximum number of values an atom can take; *e.g.*, with byte-based atoms, a would be 256. This shows the huge benefit of NSA over purely simulation-based approaches, for a content reachability analysis with high coverage.

NSA's content reachability application can be used to reason about various issues, both in current NDN as well as in a more general research context, as explained in the following examples:

- **Route computation outcome correctness.** We can use NSA's content reachability analysis to see if a particular content request reaches the nearest (or all/any) content, in case a content resides at two repositories with the same names. This is very useful to analyze the correctness of the computation outcome (*i.e.*, resulting state in the FIB, and *not* the routing protocol itself) of traditional routing protocols such as NLSR [86] (only focusing on content providers) or nearest replica routing protocols [27] (focusing on both content providers and ICN router content stores).

- **Security infrastructure soundness.** In NDN's content-oriented security design, keys that are used to perform security-related operations (authentication, *etc.*), are just like any other content: they have names, their names/prefixes populate FIBs, and they can/should be retrieved using Interests [198]. The reachability of the correct keys is important for NDN security mechanisms to be sound. As an important case, NSA can check if all public keys (*e.g.*, data with "`/KEY`" prefix) can be reached at appropriately, requested from all appropriate end points in the network.

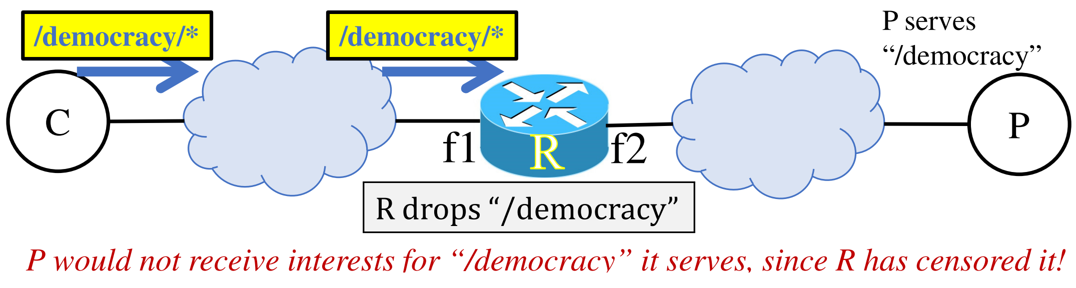

/democracy/*
/democracy/*
P serves "/democracy"
C
f1
R
f2
P
R drops "/democracy"
P would not receive interests for "/democracy" it serves, since R has censored it!

Figure 4.3: Content censorship example

- **Content censorship-freedom.** Censorship leads to content reachability errors; in the example in Fig. 4.3, censoring node R may drop all interests for "`/democracy`" [114]. This would result in (all or part of) content provider P's name space to be unreachable, injecting headers from C. This is an undesired effect that can easily be detected by NSA. While NSA cannot definitively deduce that such a problem is caused by content censorship, the lack of existence of such errors would imply content censorship-freedom. Furthermore, the effectiveness of a censorship countermeasure mechanisms can be checked using NSA.

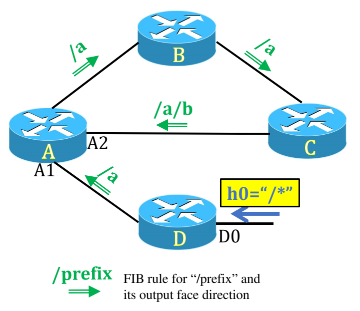
Content neutrality. We define *Content Neutrality* as not favoring a content provider over another (by not discriminating), with regards to same prefixes that they serve. With multicast forwarding strategy at every router for every prefix, NSA can check whether all content providers receive Interests matching their entire name space, for every 'all-wildcard' injection. While NSA cannot detect if a reachability error is caused by discriminatory neutrality violation or benign configuration mistakes, an error-free data plane could be used to show if content neutrality holds.

4.3.2 Loop Detection

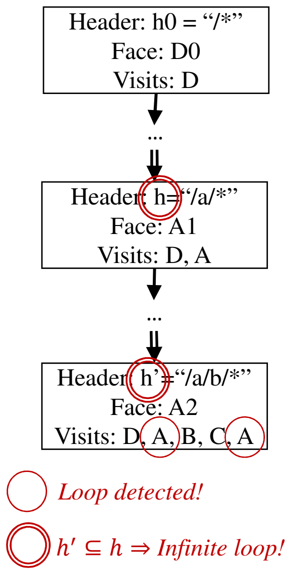
Loop freedom is an important property in networks. For NDN in particular, looping Interests is a widely known issue, which led to the addition of extra processes in the forwarding pipelines, such as a Dead Nonce List [19]. While such reactive measures detect looped Interests after they occur, looped Interest would not be prevented and could potentially waste a large amount of network resources. Also, it is very likely that an Interest is looping because it is not satisfied; *i.e.*, did not reach its intended content provider(s) due to errors in the forwarding state of the network. As a result, making a local decision at an NDN router to discard or drop a looping Interest does not solve the problem of unsatisfiability of certain Interests. Thus, it would be highly desirable to detect all potential loops in a data plane, before they occur, with a holistic view of the network data plane.

NSA helps in identifying all Interests that might potentially loop. NSA typically does this by injecting all-wildcard headers and looking for possible loops. Thus, we can track every possible Interest and find all potential loops by following FIB rules established in a given data plane. We therefore achieve a purely *name-based* loop detection, rather than a *nonce-based* detection. NSA models the transition of all packets within a single data plane snapshot, thus enabling a robust loop detection algorithm (as does HSA [107]). As all FIB rules causing the loops are contained in one single snapshot and it is possible to analyze them with transitioning packets (headers), NSA can catch all potential loops.

The loops detected can be potentially infinite or finite. Suppose node A appears twice in a single path in the propagation graph, visiting two header spaces h and h' (in that order); if $h' \subseteq h$, then this would be a potential infinite loop. An example is shown



(a) Topology, injected header, and FIB rules



(b) Propagation graph (partial)

Figure 4.4: Loop detection example

in Fig. 4.4, where NSA first detects a loop (as node A appears twice in one particular path), and second, it determines the loop to be infinite, checking the header spaces h and h' associated with the visits, where headers with name "`/a/b/*`" return back to node A. Having $h' \cap h = \varnothing$ implies a certainly finite, thus non-hazardous, loop which NSA ignores. By adding the history of each state to NSA, *i.e.*, the sequence of headers and faces, NSA can easily detect infinite loops by checking whether a particular header space (subset) has been visited by a node twice or not.

4.3.3 Name (Space) Leakage Detection

What if a consumer issuing an Interest for a particular name, wishes (parts of) the name, *e.g.*, his ID or a particular content name, to not be visible in the network except for certain authorized nodes, *e.g.*, those in his home network? This can be a desirable property for a variety of reasons. Works such as [181] have identified the need for Interest name privacy.

In NSA, inspired by HSA's slice isolation check, we can check whether or not any confidential name leaves a particular set of nodes authorized for read-access. Let us call this set of nodes as a *zone*. A zone can be a particular router, a local network, a service provider network, *etc.*

Let us consider the example in Fig. 4.5: Consumer C issues Interests with header h_0, which results in headers h_1, h_2 and h_3 leaving the authorized zone of routers, denoted as $Z1$. We define all the headers going out of $Z1$ as $h_{out} = h_1 \cup h_2 \cup h_3$. NSA allows us to define and apply access control rules on names in a number of ways, and check name constraints on h_{out} accordingly, *e.g.*, the following examples:

- Headers of particular form, *e.g.*, containing a particular name component or prefix, should not appear in any packets leaving zone Z1. Then we should have $h_{out} \cap h_{prohibited} = \varnothing$, where the left-hand side of the equation denotes the intersection of all headers leaving $Z1$ with all prohibited headers. Prohibited headers can be built using NSA's atoms and algebra, as described in §4.2. The "$\varnothing$" on the right-hand side means that we do not want any header in the result of the intersection to leave $Z1$.

- Packets associated with name space NS_0 should not leave $Z1$; then we should have $\Omega(h_{out}) \cap NS_0 = \varnothing$. This way of defining a rule is more efficient for constraints of a larger set of prefix-suffix name relations representing a portion of a name space graph:

instead of checking many prefixes one by one, we can check once against name space NS_0 comprising all those prefixes.

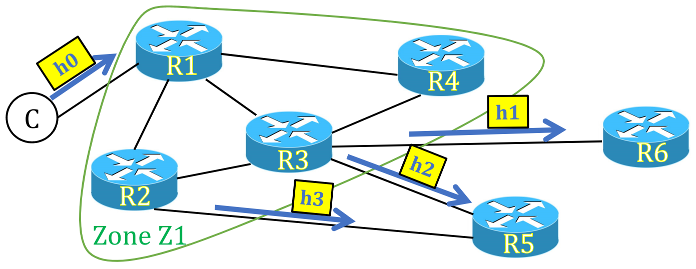


Figure 4.5: Name leakage detection example

4.3.4 Cross-Snapshot Equivalence Check

The applications above focused on checking properties *within* a single snapshot of the data plane, *i.e.*, a single state. However, in many cases we may wish to check properties across multiple snapshots. An important class of multi-snapshot checks is to do a comparison between two (or more) separate snapshots of the network. NSA enables a *Cross-snapshot Equivalence Check*. A pair of snapshots may be fed as inputs and we can check the equivalence between the two with a custom notion of equivalence. The two snapshots can represent two versions of a data plane, or different states of the same data planes at two different points in time (collected at certain intervals or triggered by certain events). In particular, our goal is to check how the point of attachment of a producer affects its content reachability. Ideally, we want it to have no effect. While single-snapshot analysis checks a snapshot against an external property as a reference (*e.g.*, content reachability), cross-snapshot analysis checks a snapshot against another snapshot, *i.e.*, *the reference snapshot*, and makes sure the two are equivalent. This can be defined as $s1 \equiv_{EP} s2$, where $s1$ and $s2$ are the two comparable snapshots and EP is the case-specific *equivalence property*, *i.e.*, the notion of equivalence we want to check, by comparing the snapshots provided. We explain this by way of an example.

Example use case: Producer Mobility Correctness. Mobility is a major feature of NDN. However, especially when it comes to producer mobility, handling it in a correct way (*i.e.*, making sure the producer's content reachability properties stay the same after the mobility and network re-convergence) can be quite challenging [204]. We can use NSA's Cross-snapshot Equivalence Check to check this correctness property.

Let us consider two snapshots $s1$ and $s2$ of an NDN network, where $s1$ and $s2$ are identical in every way except that the network point of attachment of producer P is different in the two snapshots, as depicted in Fig. 4.6. In other words, state $s1$ is collected before P's move and $s2$ is collected after P has moved. The state in the network (i.e., FIBs in the routers) has been re-populated and routing convergence, according to the protocol, has been partially or completely achieved.

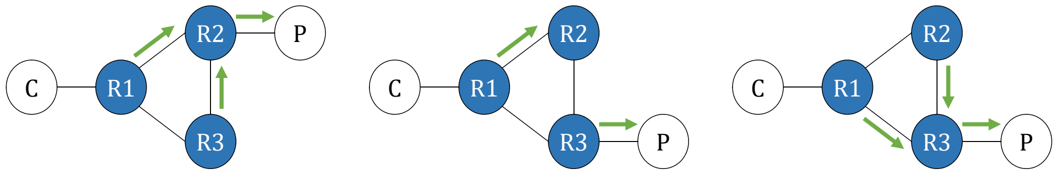
Now Let us assume that we want to make sure that P's name space reachability in $s2$ is exactly equal to that in $s1$. That would be our desired equivalence property. To check this, we use the Content Reachability function as described in §4.3.1. We produce $CR^{s1}_{x \to P}$ and $CR^{s2}_{x \to P}$, which provide all names reached at P (from any starting node x) in $s1$ and $s2$ respectively. If the ranges of $CR^{s1}_{x \to P}$ and $CR^{s2}_{x \to P}$ are equal, then we say $s1$ and $s2$ are "equivalent in regard to reachability of P's name space". Thus, the mobility of producer P

is handled correctly with respect to this property. In other words,

$$EP : Range(CR^{s1}_{x \to P}) = Range(CR^{s2}_{x \to P})$$

where EP defines the equivalence property for this case. This would mean that $s1$ and $s2$ are in the same *equivalence class* with respect to property EP (it is trivial to see that the specified EP is an equivalence relation). Differences between the ranges of $CR^{s1}_{x \to P}$ and $CR^{s2}_{x \to P}$ indicate incorrectness and will be reported as errors. Examining the non-overlapping parts of $CR^{s1}_{x \to P}$ and $CR^{s2}_{x \to P}$, the network manager can infer as to which forwarding rules are causing the error. Having said that, deducing the root cause of what aspect of the mobility handling protocol is causing the error may be difficult in more complex scenarios (*i.e.*, those which involve many mobility events), since NSA does not explicitly determine the root cause.

Fig. 4.6 shows a simple mobility example, where producer P, which serves name prefix "`/a`" moves from its initial point of attachment $R2$ (initial snapshot, $s1$, Fig. 4.6(a)) to $R3$ (finals snapshot, $s2$, Fig. 4.6(c)). For simplicity, we consider a naive routing-based mobility handling solution that re-populates all FIBs with an updated announcement after the new attachment. An intermediate state ($s1.5$, Fig. 4.6(b)) shows the state after P's move but before full re-convergence of the network ($R3$ has been notified of the update, but $R1$ and $R2$ have not yet been notified). Using the mobility equivalence property EP defined above, we will have $s1 \equiv_{EP} s2$, but $s1 \not\equiv_{EP} s1.5$ since the range of $CR^{s1}_{C \to P}$ and $CR^{s1.5}_{C \to P}$ do not match; in other words, interests for "`/a`" from C that reach P in $s1$, do not do so in $s1.5$. Similarly, $s1.5 \not\equiv_{EP} s2$. This example shows that during the transition, the network is temporarily incorrect. The property needs to ultimately hold for the initial and



(a) Initial state (s1) (b) Intermediate (s1.5) (c) Final state (s2)

Figure 4.6: Data plane state changes due to producer mobility (P serves "/a", green arrows: FIB entries for "/a")

final snapshots. Also, a fast mobility solution, creates erroneous intermediate snapshots that are fewer and last for shorter durations.

Furthermore, checking two different intermediate snapshots of two different mobility solutions can be helpful. *E.g.*, suppose $R2$ in Fig. 4.6(b) has received an 'invalidation' signal from P once it moves. NSA gives us the full header space leaving $R2$ in the two cases: 'without invalidation message' *vs.* 'with invalidation message' (as two intermediate snapshots). The smaller size of the latter shows that fewer interests will be blackholed by using the invalidation message.

While the toy example above only deals with one mobility event and only 3 snapshots, it can easily be generalized to more complex checks. Since the goal is putting different snapshots in their respective equivalence classes, many possible snapshots may be checked through NSA's equivalence checks. Also, multiple producer mobility events can be supported, *e.g.*, if multiple producers re-attach in $s2$. This is possible in NSA's snapshot-based data plane verification approach, since the impact of all the mobility events can be captured in a single snapshot.

Consumer mobility can be checked for correctness in a similar manner. Cross-snapshot equivalence check application can be very useful to check the *outcome* (not the protocol itself, per se) of mobility handling protocols on the data plane, especially with regard to how they re-populate the network FIB or re-direct requests. This may be complimentary to other protocol verification methods used for specific mobility handling protocols [204].

4.4 Complexity Analysis

First, we analyze the complexity of transfer function generation, which is an important step where NSA converts the NDN FIB table to NSA transfer functions that capture those rules (as described in §4.2.2). We now look at its time complexity. For a FIB table with e entries, the worst-case complexity of this conversion would be $O(e^2 D 2^d)$: for every entry e_i, we need to check all other entries to find its descendants, *i.e.*, at finer granularity of e_i. For each descendant of e_i, which we represent as e_i^j, there would be $2^{d_{ij}}$ corresponding NSA rules that need to be generated, where d_{ij} is the *granularity distance* between e_i and e_i^j. For example, granularity distance of prefixes $e_1 =$"`/a`" and $e_2 =$"`/a/b/c`" is 2, as e_2 is a descendant of e_1 and has two additional name components. As a result, corresponding to e_1, NSA would create rules for "/a/$\overline{\texttt{b}}$/c/$*$", "/a/b/$\overline{\texttt{c}}$/$*$", and "/a/$\overline{\texttt{b}}$/$\overline{\texttt{c}}$/$*$" for the network transfer functions (so the outcome would be determined by the intersection of incoming header to every rule). Also in the complexity formula, D and d denote the maximum number of descendants, and granularity distance in the given FIB table, respectively.

Next, we analyze the time complexity of the execution of the verification procedures on prepared network space, starting with content reachability (§4.3.1). Let us assume we have an injection of a header at *one* consumer, that leads to *one* content provider through a single path. Let us also define d, L, R, and s as maximum network diameter (number of hops), maximum header length, maximum number of node rules, and maximum number of paths in a trie-based content provider name space, respectively. The time complexity of the NSA content reachability test will then be $O(dLR^2s)$. This analysis is based on the *linear fragmentation* assumption in [107], which says that typically very few rules in a node match an incoming packet. On the other hand, the complexity of a simulation-based test (as well as ping or traceroute) would be $O(da^LRs)$, where a is the maximum number of values an atom can take; *e.g.*, with byte-based atoms, a would be 256. NDN headers have no specific upper bound; however, it is recommended that a reasonable MTU (which can be thousands of bytes) be conformed to by NDN applications [17]. This will make a^L very large. This way, the simulation approach will reach a very large and exponentially growing complexity. This shows the huge benefit of NSA for a content reachability analysis with high coverage.

Using the same similar method and the linear fragmentation assumption, we can analyze other NSA applications as well. NSA completes loop detection (§4.3.2), in particular to check if a header injected at a node A returns to A, in $O(max(c, d) \times LR^2)$, where c is the length of the longest cycle (loop, in terms of number of hops) in the network. Loop detection only checks the forwarding rules, and not the content available at nodes (hence the removal of s from the complexity expression). NSA's name leakage detection (§4.3.3), in particular to check if an injected header at node A in zone $Z1$ will cause a name leakage at node B in zone $Z2$, has the complexity of $O(dLR^2 \times P)$, where P is the maximum number of prohibited names per zone. For multi-snapshot checks (§4.3.4) with n snapshots,

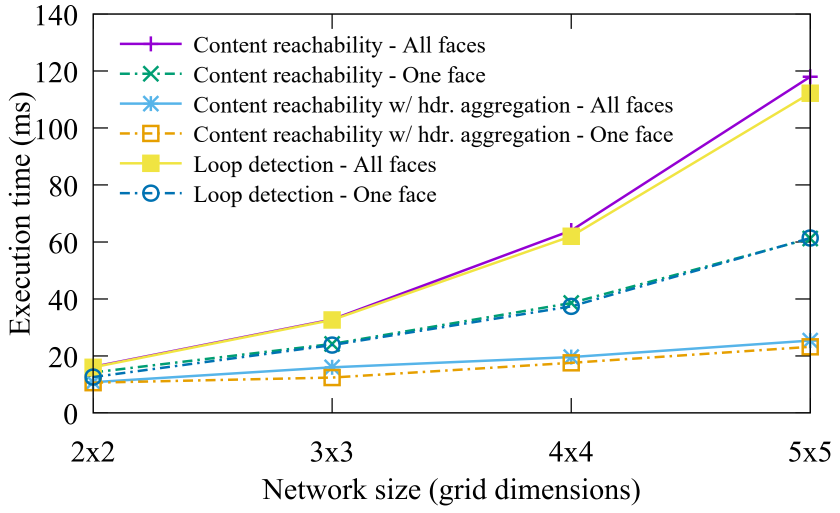


Figure 4.7: Results for small grid snapshots

if the check's complexity within a snapshot is $O(f)$, then the total worst-case complexity is $O(n^2f)$, as every snapshot will have to be compared and put in the right equivalence class.

4.5 Evaluation

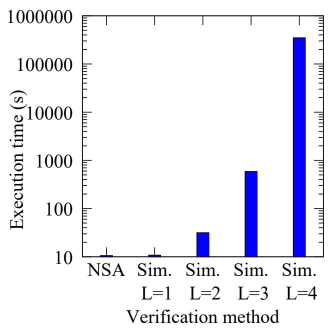


Figure 4.8: NSA & simulation-based check

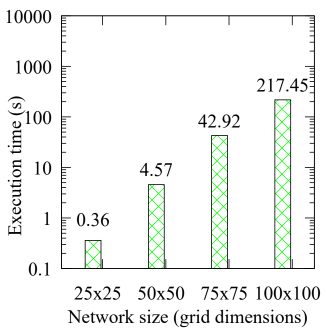


Figure 4.9: Large grids, vary network size

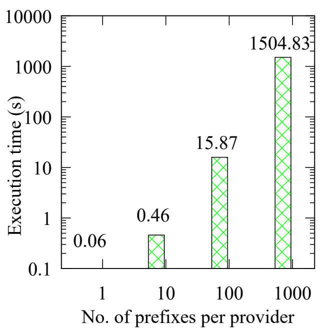
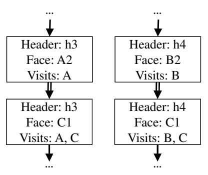
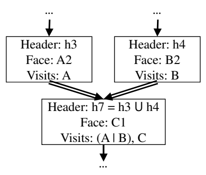


Figure 4.10: Large grids, vary prefix count



(a) Original propagation graph



(b) Aggregated propagation graph

Figure 4.11: Propagation graph aggregation example

We have implemented NSA, including its main components and modules, in Java; the source code is available at [99]. We start by evaluating the performance of NSA using synthetic grid and ring topologies, and then apply it to the NDN testbed topology for evaluating a network that is actively used [148]. All evaluations have been done on a machine with Ubuntu 14.04.6 LTS using Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz dual-socket with 14 cores each with hyper-threading enabled, and 252GB RAM. We do not utilize the whole RAM capacity though; we set the maximum memory heap size of our Java Virtual Machine (JVM) to 10GB only. For each verification application, all wildcard headers, *i.e.*, "/*" is injected to all faces or nodes. While reporting our evaluation results, we identify and present optimizations that further improves NSA's performance.

4.5.1 Synthetic Networks

Content Reachability Analysis and Loop Detection

To evaluate NSA's content reachability analysis and loop detection we use customized $n \times n$ grid topologies (to allow many branches in the propagation graph), with n pub-

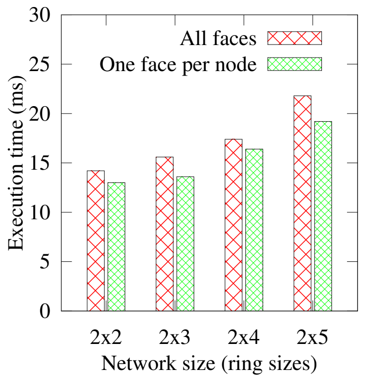


Figure 4.12: Name leakage

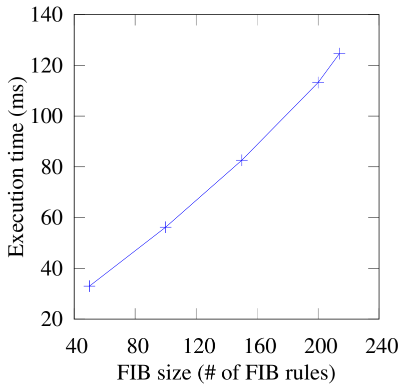


Figure 4.13: Rule generation ('UCI' node)

Table 4.1: Execution Time (ms) for NDN Testbed Verification

Application	Best-route	Multicast
Content Reachability Analysis	196	2,481
Content Reachability Analysis (w/ header aggregation)	75	342
Loop Detection	190	2,416

lishers in each case, each serving one distinct prefix; these prefixes are advertised and populated in every node's FIB in the snapshot being verified. Verification performance results for these grid networks are presented in Fig. 4.7, in terms of execution times, in milliseconds.

First, Fig. 4.7 shows the execution time of content reachability on the grid networks. This verification, as explained in §4.3.1, checks both unreachable and unsolicited names. Typically, NSA injects all-wildcard headers into all faces, since some node rules may depend on the incoming faces ('All faces' in Fig. 4.7. As seen in the Fig., the growth of execution time for 'All faces' injection mode is linear with respect to the input network size growth (note the input growth on x-axis is n^2). Since we are only dealing with FIB rules that do not depend on the incoming face, we can limit our injection to 'One face per node' injection only. This would not change the outcome of the verification results. Fig. 4.7 shows that this optimization significantly improves the performance of NSA, which is due to the fact that its fewer number of injections leads to smaller propagation graph.

For the full reachability check, we need to go through a separate propagation graph fragment, built and checked for each injection, to check both unsolicited and unreachable names. If our goal is to only check unsolicited names (and not unreachable names), we can make all injections at once into a single propagation graph fragment, aggregating the headers (Fig. 4.11). This way, we preserve all reached header spaces, but not their exact paths from origin in the visited list. Fig. 4.7 also shows the significant performance enhancement of this optimization, compared to full reachability analysis, if our goal is only to detect unsolicited names.

The use of wildcards is an important benefit of NSA (and HSA), compared to simulation-based methods (which have to generate all possible packets within a range). We show the empirical results for the use of wildcards in Fig. 4.8. Each simulation scenario ('Sim') is a typical simulation-based content reachability analysis (using the aggregation optimization with the sole purpose of detecting misdirected packets) that injects Interests with L name components, each being a single alphabetical letter. Using diagnostic tests through ICN/NDN ping and traceroute tools has the same theoretical complexity as the simulation-based approach, with the additional disadvantage of using too much network resources (as every test packet injected will have to actually traverse the network). Fig. 4.8 shows the large benefit, in terms of performance and scalability, of NSA compared to these simulation-based verifications.

To demonstrate NSA's performance and scalability, we examine its utility with larger test cases, with $n \times n$ grids. The results are shown in Figs. 4.9 and 4.10. Each node's FIB is populated with entries (rules) for all prefixes, with random outgoing faces. We only show the results for the reachability test with aggregation optimizations, with all-wildcard injections at all nodes, one face per node. Fig. 4.9 shows the execution times (shown in seconds in log scale) when increasing the grid dimensions. The largest case (100×100) has a total of $100\times100\times100\times1$=1 million rules in the network, which is similar with the largest test case considered in HSA's performance evaluation [107]. Thus, NSA's performance is in the same order of that of HSA, even though NSA adds the significant feature of checking name-based (information-centric) reachability. Further, note that our grid topology yields much higher number of paths compared to HSA's simpler backbone topology [107]. The growth rates in Fig. 4.9 is reasonable: note that along the x-axis, each scenario exponentially increases the number of paths, where an $n \times n$ grid has $O(n!)$ paths between two nodes, leading to much higher length and number of paths in the propagation graph. Next, we pick a 10×10 grid (with 10 providers), and gradually increase the per-provider number of prefixes. The largest case (*e.g.*, "1000") leads to a total of $10\times10\times10\times1000$=1 million rules in the whole network. Again, the performance and scalability of NSA with these large test cases is reasonable (both compared to the absolute values for HSA, and comparing the relative growth in execution time). This is especially compelling, considering the high complexity of these test cases and how they affect the runtime.

We also evaluated the performance of NSA's loop detection on the same grid networks, injecting all-wildcard headers. Fig. 4.7 shows the results for both cases of 'All faces' and 'One face per node' injection. The complexities, growth rates and optimization benefit of face selection in loop detection are similar to those of full content reachability analysis. Also, compared to NDN's built-in loop detection mechanisms, NSA can prevent all possible loops caused by forwarding rules, without using network resources, and allowing for hints on how to resolve the loop errors.

Name Leakage Detection

To verify name leakage-freedom, we use two-ring topologies; where two rings of size n are connected by one node, *i.e.*, is a gateway between the rings. Each ring is considered its own zone, and has one publisher serving (and advertising) two prefixes, one prefix visible to everyone, and one prefix visible only to the nodes within the local zone. Thus, each NDN node has rules for the three prefixes (that are visible to it): two prefixes of its own zone, and one prefix that is public from the other zone. In each of its rounds, NSA's name leakage detection application injects all-wildcard headers to the faces/nodes of one zone, generates headers that reach the other zones, and checks whether or not they violate each zone's name privacy requirements. The performance of name leakage on the two-ring topologies are shown in Fig. 4.12, indicating its scalability (showing a linear growth) with the increase in network size.

4.5.2 NDN Testbed

To evaluate NSA's performance on an operational, practical NDN, we considered the NDN testbed [148]. This is the largest real-world NDN with publicly available forwarding state, with relatively large forwarding tables (of the order of hundreds of entries per node). We captured a snapshot of the testbed on 2019/03/09 14:43:16 CST. We use the globally available topology. Each node in the testbed provides its (near-)real-time local status (FIBs, *etc.*) through a separate webpage. We collected the full network status by crawling these individual local status pages. Some nodes were offline or unresponsive and we removed them from our analysis.

An important pre-processing step for NSA verification is generation of the transfer functions. We have implemented these components in NSA. The topology transfer function generation is trivial. For network transfer function generation for LPM-based forwarding rules in NDN nodes, additional processing needs to be done: for each FIB entry, all other rules (*i.e.*, FIB entries) have to be visited. To show this empirically, we picked one particular node from the testbed, the 'UCI' node. It has 214 FIB entries in our selected snapshot. We randomly pick 50, 100, 150 and 200 FIB rules from it and perform the network transfer function generation. The execution time (including the case with all 214 entries) is shown in Fig. 4.13. These results show that NSA's transfer function generation for this particular real-world case is reasonably efficient and scales well with number of FIB rules.

We performed content reachability (both full and aggregated) and loop detection on the snapshot (we did not perform name leakage detection on it since the name leakage-freedom is not one of the properties of the NDN testbed) using two forwarding strategy modes (for all), namely the best-route and multicast, and found several errors. In the best-route mode, we found 450 content reachability errors, either caused by forwarding state errors or physically unavailable/offline nodes. For example, the name "`/kr/re/kisti`" is reachable only in 31% of injections. Also, 704 loop-freedom violations were found; note that this is not the number of loops (cycles) per se, but rather the total number of looped Interests detected as a result of injections. For example, for the prefix "`/kr/re/kisti`", a loop was found between the two nodes 'TNO' and 'GOETTINGEN'. In the multicast mode, we found hundreds of errors too. The performance results of our verifications (execution times in milliseconds) are shown in Table 4.1, showing its latency is reasonable.

From a practical standpoint, our experiments and results show that it is feasible to have NSA integrated into the NDN testbed (in one of its nodes), and periodically check for data plane errors, and checking various states of the data plane. Given that these checks only take seconds in total, including transfer function generation and the analysis, it would be quite reasonable to have new NDN snapshots (which can be generated every few minutes or seconds) be verified. The network administrator can run NSA on one of the nodes (a controller or any router) on the testbed, periodically collect snapshots at that node (using methods such as NDNconf [16]) and provide continuous verification results of the network. This would be very helpful for the users of the NDN testbed, and for their research experiments.

4.6 Case Study: Name Space Conflict Detection/Resolution

As an important case study, in this section, we explain name space conflicts, and how NSA can detect and help resolve it.

NSA can be used to investigate and catch a wide variety of corner cases that may result in errors. In this section we explore as an example, property violations caused by *name space conflicts* across different content providers. In NDN, different content providers,

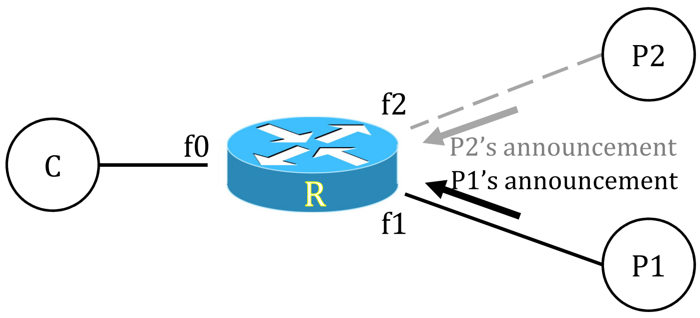


Figure 4.14: Name space conflict example (case I: only P1 is present; case II: both P1 and P2 are present)

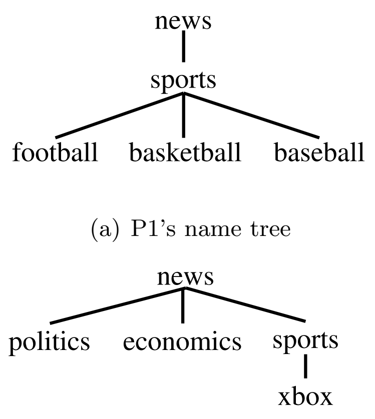
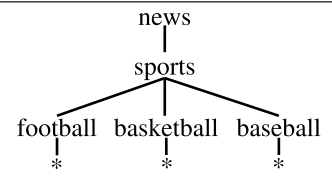
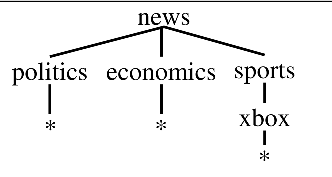


Figure 4.15: Name trees of the producers in Fig. 4.14

Table 4.2: Name Space Conflict Scenarios for Case II

Scenario	P1 announcement	P2 announcement	FIB at R	Result	Comments
II-1	/news/sports	/news	/news/sports → f1 /news → f2	fail	P2 requests reach P1 (blackhole)
II-2	/news/sports	/news/politics /news/economics /news/sports/xbox	/news/sports → f1 /news/politics → f2 /news/economics → f2 /news/sports/xbox → f2	success	relatively large FIB (4 entries)
II-3	/news/sports/football /news/sports/basketball /news/sports/baseball	/news	/news/sports/football → f1 /news/sports/basketball → f1 /news/sports/baseball → f1 /news → f2	success	relatively large FIB (4 entries)
II-4	/news/sports/football /news/sports/basketball /news/sports/baseball	/news/politics /news/economics /news/sports/xbox	/news/sports/football → f1 /news/sports/basketball → f1 /news/sports/baseball → f1 /news/politics → f2 /news/economics → f2 /news/sports/xbox → f2	success	relatively huge FIB (6 entries)

Prod. ID	Name Space
P1	 news sports football basketball baseball * * *
P2	 news politics economics sports * * xbox *

(a) Global name space

Prod. ID	Routable Prefixes
P1	"/news/sports"
P2	"/news"

(b) Global routable prefixes

Figure 4.16: Databases at name registry

potentially with different subsets of content in the name hierarchy, can use and announce the same prefixes. This can be true especially when the names are topic-based. No content provider has sole ownership or authority to announce a certain prefix. While this allows for the democratization of content and better efficiency, it can cause conflicts that can lead to blackholed interests. We illustrate this using an example. In addition to checking with NSA, we also ran these scenarios in ndnSIM [149] and observed that the blackhole effect in question indeed does occur.

Fig. 4.14 shows a simple network topology. Suppose in the beginning, there exists only one producer $P1$ (case (I)), with the name tree depicted in the Fig. 4.15(a) and creates a name announcement for "`/news/sports`". The announcement is used to populate the router R's FIB in accordance with NDN policies. Announcing "`/news/sports`" implies that $P1$ claims that it has 'everything' under "`/news/sports`", which is correct from the network layer's perspective. In reality, a producer may not have 'everything' under a particular name prefix it announces, *i.e.*, there may be a possible suffix not covered in this announced 'everything' set. However, since there is no other producer to 'challenge' $P1$'s claim, $P1$'s announcement stating that requests for anything under "`/news/sports`" will be available at $P1$ does not cause any conflict.

Now let us assume the same network but with two producers $P1$ and $P2$, as shown in Fig. 4.14 (case (II)), with $P2$'s name tree shown in Fig. 4.15(b). The subset of $P2$'s name tree that is interesting for this discussion is "`/news/sports/xbox`". Note that there is no malicious intent on $P2$'s part; evidently, $P1$ does not recognize video games as 'sport'; however $P2$ does (with the sport being 'xbox'). $P2$, unaware of this conflict, announces his prefix also at the coarsest granularity, announcing "`/news`" (Scenario II-1 in Table 4.2). Using NSA's content reachability test, we can show that this scenario is erroneous: requests for "`/news/sports/xbox/1`" reach $P1$ instead of $P2$. $P2$ is this Interest's (most) relevant provider. Formally, header space of form "`/news/sports/xbox/`$*$" reaching R at $f0$, has a non-empty intersection with R's FIB rule at $f1$ ("`/news/sports/`$*$") rather than that of $f2$ ("`/news/`$*$"), assuming a best-route forwarding strategy at R. Thus, "`/news/sports/xbox/`" will be a name reaching $P1$ and not $P2$:

$$\text{“}\texttt{/news/sports/xbox/}\text{”} \in \mathit{Range}(CR_{C\to P1})$$

$$\text{“}\texttt{/news/sports/xbox/}\text{”} \notin \mathit{Range}(CR_{C\to P2})$$

This undesirable effect can be remedied by changes in the prefix announcement. In particular, we can change the granularity of the prefix announced by $P1$ or $P2$, or both. While NSA does not 'directly' resolve errors, the counterexamples it provides can give us guidance on how certain bugs can be resolved. For scenario II-1, comparing unsolicited names at $P1$ with unreachable names at $P2$, *e.g.*, observing that the name "`/news/sports/xbox`" is part of N_{P1}^{UW} and also N_{P2}^{UR} (as given by Algorithm 1) suggests that with more fine-grained announcements, *i.e.*, announcing names at lower levels in the name tree, interests have a better chance of reaching their most relevant producers. Table 4.2 shows three examples of these alternative announcements (which we call scenarios II-2, II-3 and II-4), making the NSA verification of each example successful. However, we see their costs, in terms of scalability and FIB size are different An important takeaway from the results in Table 4.2 is the (possibly) inverse relation between correctness (absence of name space conflicts) and FIB size. We can say that finer granularity of prefix announcements, leads to less conflict but larger FIB sizes. In scenario II-2, $P1$ announces one prefix at a coarse granularity of "`/news/sports`", while $P2$ announces three prefixes at different granularities, namely "`/news/politics`", "`/news/economics`" and "`/news/sports/xbox`". This will populate R with 4 prefixes, as shown in Table 4.2. Using NSA, we can see that an Interest of form "`/news/sports/xbox/`$*$" reaching R at fact $f0$ will leave R on face $f2$ (rather than $f1$), thus reaching its intended producer, $P2$. Therefore, this makes the verification successful. In scenario II-3, $P1$ announces three fine-grained prefixes, namely "`/news/sports/football`", "`/news/sports/basketball`" and "`/news/sports/baseball`" and $P2$ announces one coarse grained prefix, namely "`/news`". NSA in this scenario shows that "`/news/sports/xbox/`$*$" goes out of face $f2$ since it has an empty intersection with all of the forwarding rules leading to $f1$ (towards $P1$). In scenario II-4, both $P1$ and $P2$ announce prefixes at fine granularity. While R's FIB size in scenarios II-2 and II-3 are 4, that for scenario II-4 is 6. All these three are correct, *i.e.*, absent of name space conflicts.

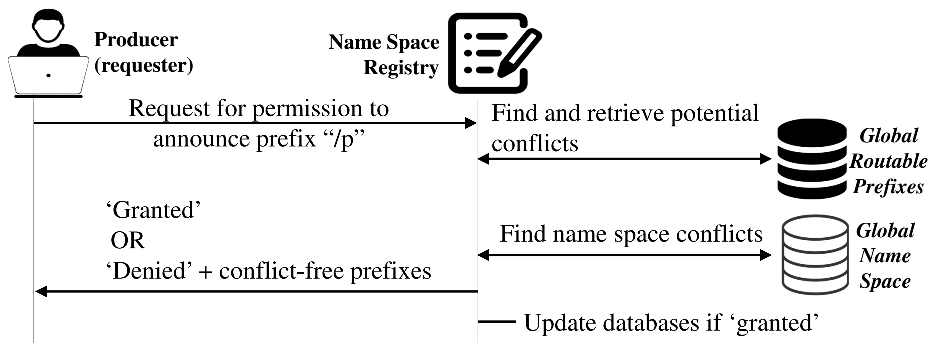

Producer (requester)
Name Space Registry
Request for permission to announce prefix “/p”
Find and retrieve potential conflicts
Global Routable Prefixes
‘Granted’ OR ‘Denied’ + conflict-free prefixes
Find name space conflicts
Global Name Space
Update databases if ‘granted’

Figure 4.17: Name registration procedure

This case study shows that achieving correctness has a cost. It may be important to find the most efficient refinement to the name space announcement so as to keep the FIB size manageable. There is a need for an approach to detect and perhaps resolve conflicts, before they happen. We provide some guidelines for the design of such an approach next.

4.6.1 Name Space Registry Guidelines

The name space conflict observed in the case study in §4.6 may be quite common. While NSA is useful in finding conflicts, an automatic approach, or protocol, for conflict detection/resolution can be very beneficial in NDN. To prevent content provider name space conflicts, a *Name Registration* protocol may be such a solution. The idea of name registration in NDN has been suggested in previous works, such as in [52], mainly to prevent prefix hijacking. Our case study however shows it is important for non-malicious Scenarios as well.

A name space registry can be implemented as a distributed or centralized engine to be contacted by producers whenever they want to announce a prefix. The producer sends his requested announcement prefix and (a pointer to) his content name tree. The response data from name registry would be a signed packet containing the prefix announcement permission result, *i.e.*, ‘granted’ or ‘denied’, plus possibly new prefix announcements suggested by the name registry that are conflict-free. The name registry’s decision is based on an analysis of name spaces across different providers, which are in its database, called a *global name space* (Fig. 4.16(a)). The name registry may have another database of *global routable prefixes* (Fig. 4.16(b)). Both databases are indexed by *Producer ID* which can be a real ID or a locally generated (but unique) ID. If permission is granted, the new prefix announced will be added to the global routable prefix list, and the producer’s name space will be added to the global name space database. If the request is denied, the requester needs to pick another high-level name for the prefix announcement (just as in today’s IP network, domain names have to be tried one after another, until one is available) or use suggestions provided by the name registry. The suggestions for the prefix announcement can be provided at different granularity levels to help manage the growth of the FIBs. While we describe it as a logically separate entity, the name registry can be integrated with possible existing name resolvers, such as NDNS [18]. An overview of the name registration procedure is depicted in Fig. 4.17. Upon receiving a request for permission to announce a prefix “`/p`” from a producer, the name registry performs the following steps:

1) Examine the global routable prefixes list to find *potential conflicts*. Two announcements can cause potential conflict if one is the “prefix” of the other (*i.e.*, their intersection is non-empty). An example for potential conflicts between names is $P2$’s an-

nouncement "`/news`" that is a prefix of $P1$'s announcement "`/news/sports`" in Scenario II-1 in Table 4.2. The name registry returns the indices, which are producer IDs.

2) Retrieve every individual name space associated with the previously found producer IDs. These are *potentially conflicting name spaces.* 3) The name registry compares the requester's name space against any other potentially conflicting name space. If a conflict is found, it will follow the steps outlined in step (4) onwards; otherwise follow step (6) onwards. A conflict is found if starting from the root on any of the two name spaces, the announcement prefix exists on any of the other name spaces, but at least one descendant does not. For example, in Scenario II-1 in Table 4.2, "`/news/sports`" on $P2$'s name space exists in $P1$'s as well, but it leads to the name "`xbox`" that does not exist on $P1$'s name space. The condition for announcement prefixes $pr1$ and $pr2$ (associated with producers $P1$ and $P2$ respectively) to be conflict-free, can be specified by the following assertion:

$$(pr1 \text{ is a prefix of } pr2) \bigwedge (pr2 \subseteq NS_{P1})$$

$$\Longrightarrow \forall \text{ non-wildcard sequence of name components } X :$$

$$``/pr2/X'' \subseteq NS_{P1} \iff ``/pr2/X'' \subseteq NS_{P2}$$

4) Generate conflict-free announcement prefixes for the requester. This can be done by checking finer granularities on the name space; *e.g.*, Scenario II-2 in Table 4.2. A more fine-grained prefix includes more detail about a category of content items, and has a lower chance of conflicting with other prefixes. Conflict-freedom of alternatives that are found can be checked using the assertion in (3). Note that a conflict-free alternative may not exist; in that case the requester has to pick another, different, high-level name. Otherwise, some of previous producer announcements need to change, which can lead to much more complexity.

5) Respond to the requester with a signed data stating ‘denied’; possibly together with prefix suggestions. Stop.

6) Send a signed response to the requester stating ‘granted’.

7) Add the requester’s announced prefix and name space to the associated local databases at the name registry.

Our case study shows, that having a conflict detection and resolution engine (name registry) in NDN is necessary to prevent the occurrence of name space conflicts, and subsequent blackholing of Interests. NSA can be used to analyze the correctness of the outcome of such a mechanism. While we outlined design guidelines and principles of such an engine, space limitations preclude a detailed specification.

Chapter 5

Information-centric Interoperability for Future Network Architectures

5.1 Introduction

In this chapter, we propose COIN, a framework for interoperability between legacy and future Internet architectures, focusing on the important class of ICNs, which are significantly different from today's IP architecture. COIN does not require any change in existing individual domain architectures and preserves their key features and mechanisms. Additionally, COIN does not require content to be moved or replicated to allow access to it from users and end-systems that are in a different network domain. It also does not require the identity/name of a content item to be replicated for each existing domain and end host's understandable semantics. For content-oriented interoperability, it is key that naming is harmonized across different domains. An integral part of this harmonization is that the native naming schema of each domain type (*e.g.*, hierarchical structure of NDN [203]) is retained while enabling access and retrieval of content across domains with different naming schema. This goal would be very difficult to achieve with an overlay approach, especially for traversals across multiple domains of different architectures [133]. On the other hand, an efficient translation of content requests and responses between domains can support this interoperability. To this end, rather than creating a new universal layer or overlay, COIN provides translation-based interoperation across multiple domains, with the use of gateways that process requests/responses and retain state information. COIN supports both static and dynamic content requests. Using encryption/decryption as well as iterative signatures performed as we cross from one domain to another domain, COIN ensures confidentiality, integrity and provenance. To obtain content names in a foreign domain, COIN incorporates an Object Resolution Service (ORS) [14]. ORS is an important capability that enables cross-domain name retrieval and usage through a systematic, application-layer procedure. Our ORS design, importantly, relieves the interoperability framework and content providers from re-naming content for each domain, and consumers from having to understand new name formats foreign to them. Focusing on the candidate cases of IP, Named Data Network (NDN) [203] and MobilityFirst (MF) [160], we implement and experiment with our design for interoperation. We focus on NDN and MF for two reasons: 1) The Future Internet Architecture (FIA) community treats these as two prominent ICN projects [9], with continuing research efforts and community involvement. 2) The two architectures have significant differences and represent two different “sub-classes” of ICN architectures: NDN supports hierarchical naming with implicit name resolution in the network, while MF supports flat names with explicit name resolution. Taking these differences into account, we evaluate COIN and show that it is effective and efficient.

The contributions of this part are: 1) a generic interoperability framework among ICN and IP domains for secure static and dynamic content retrieval that preserves key features of domains, thus managing evolution in a flexible manner; 2) an implementation of the framework [6] for interoperability among IP/HTTP, NDN and MF; and 3) measurements based on the implementation of the framework across different domains to demonstrate its utility from a performance perspective.

We present an Alloy [91]-based formalization of Information-Cetric Interoperability (ICI), to analyze interoperability correctness. We cover both pull-based (request/response) and push-based (publish/subscribe) [49] content retrieval services, and their most essential properties such as content reachability and returnability. To analyze content-oriented services, we distinguish between static and dynamic content, justifying their differences, and specifying no-conflict properties, especially for dynamic content retrieval. For verification of these properties, we use Alloy Analyzer’s built-in SAT solver-based model finding engine [4]. We also consider failure and mobility; to analyze them, mere model finding is not sufficient, as failure and mobility, when severe, can cause any network protocol to become “incorrect” (and raise counterexamples). Thus, for such analysis, we resort to model counting (to count and compare the number of satisfying instances and counterexamples) to assess “how well” a particular domain or architecture is doing under failure and mobility.

The major contributions of this part are: 1) a model finding method to analyze basic properties (mainly reachability and returnability) of information-centric interoperability (ICI); 2) a formally-verified ICI framework; and 3) a model counting method to analyze gateway failure and mobility.

5.2 Motivation and Overview

5.2.1 Design Goals and Rationale

COIN provides interoperability between legacy (current) and future architectures guided by these requirements:

- It should support host-centric (*e.g.*, IP) *and* information-centric (*e.g.*, NDN and MF) networks.
- It should add no architecture/protocol change to the existing individual domains (*i.e.*, no new layer or protocol change).
- Introduce minimal change to end host logic, so clients in one domain use native mechanisms to seamlessly exchange information with another domain.
- Support request for both static (*e.g.*, find a movie) and dynamic content (*e.g.*, query for current weather information), potentially across multiple ($\geq$2) domains.
- Preserve ICN's domain-specific features; *i.e.*, interoperate between different naming schema, between connection-oriented and connectionless transport, between stateless and stateful forwarding, between channel-based security and content-oriented security, and support in-network caching.

- Each domain's content namespace should be limited only to include the objects in that domains.
- Inter-domain message exchanges must be secure (*i.e.*, provenance, confidentiality, and integrity ensured).

To satisfy the above requirements, we use a translation-based approach primarily to retain each domain independently (and preserve their key features), without changes to existing architectures (thus allowing for easier deployability and evolution). This approach overcomes some of the shortcomings and challenges of alternative approaches:

- Tunneling (overlay/underlay) and hybrid approaches require both the consumer and content provider to have the same semantics and formats, including components such as the naming schema [41, 133]. Translation can achieve the goal of every end host only having to "speak its own language".
- Overlays cannot take advantage of all the capabilities in the underlying domain since the underlay usually does not understand the semantics of the overlay; *e.g.*, in ICN-over-IP [168], IP does not provide many of the advantages that would be obtained from ICN, such as content caching or stateful forwarding. Hybrid approaches are also limited in terms of satisfying all key ICN features through their integrated protocols [41, 53]. With translation, we can retain domain-specific features as well as essential ICN features across domains.
- Overlay approaches introduce considerable overhead and complexity at the overlay-enabled routers, having to perform the mapping between the different decoupled lay-

ers; this may be encountered at (potentially) many routers on the path [53]. With translation gateways, routers retain their native domain-specific designs and implementations.

- The addition of new architectures requires significant changes to the overlay at tunnel end-points, both in terms of standardization and deployment. The same is true with hybrid approaches, which requires the embedding of domain-specific components of the new architecture into the integrated network layer protocol. With translation, we can add any number of new network types and attach them to existing domains via gateways supporting them.
- The above challenges become even more severe when dealing with a multitude of interoperating domain types (more than two). We alleviate this in COIN.

It is sometimes noted that translation-based interoperation is counter to the end-to-end principle of the Internet, as argued in [133]. However, we believe that new architectures (mainly ICNs) already challenge the pure end-to-end principle; *e.g.*, in NDN, the procedure for requesting and receiving content is asynchronous, with routers managing transport on a hop-by-hop basis, without necessarily having a complete end-to-end communication [203]. Also, in today's Internet, middleboxes such as NATs add additional indirection in the network [55]. We believe a translation-based approach is suitable and pragmatic for interoperability between current and future networks.

With a focus on content-oriented services, our translation is performed at the "content name level", *i.e.*, in the layer that identifies content names, be it the application layer in legacy IP domains (*e.g.*, URLs to identify content in HTTP) or network layer in NDN domains (in form of hierarchical human-readable names). This provides a significantly higher abstraction than the address-based design of legacy interoperation and is important since names are “first-class” entities in information-centric paradigms. In such environments, it is also important for consumers to pick “the right name” for a content request, and receive that content. Recent works such as [41, 133] allude to the importance and challenges of such mechanisms, although they do not provide a solution for it. We propose a protocol for Object Resolution, to enable the retrieval of the necessary names.

5.2.2 Overview of COIN

COIN provides interoperability among any number of domains, each having a distinct network design and architecture, including legacy (IP) or future (ICN) Internet architectures. COIN gateways provide this interoperability through translation and state maintenance. A client in one domain can request for content (static or dynamic) multiple domains away, and receive the corresponding content in the response. COIN makes no change to existing domain-specific architectures, and preserves key domain features, including domain-specific security models and mechanisms. Most notably, COIN preserves namespace size and structure of each domain, and does not create a new naming schema. Content can be universally identified using its domain-specific (*i.e.*, native) name, plus its domain ID. A client requesting content from another domain, uses the content’s native name and its domain ID. To acquire that information, COIN uses an *Object Resolution Service (ORS)*, which is an application-layer search engine-like service providing names as response to keyword queries. The foreign name provided to consumers are not distinguishable from native ones, thus making the consumer’s request for content seamless.

5.2.3 Addressing Challenges for Gateway-based Interoperability

While our solution helps achieve our design goals, there are additional concerns to be addressed. Most of these challenges exist for other translation-based approaches as well. We explain how COIN overcomes them.

Evolution flexibility; Too many pair-wise translators? Typically, in a translation-based interoperability solution, for n different network architectural designs, one might end up needing n^2 translators [133]. Not only would it be too complex to design so many translators, it also can make it very inflexible for adding a new domain architecture: n additional translators would need to be implemented. COIN overcomes this by having an *internal canonical form* at the gateways, and *adapters* that convert domain-specific packets to/from this canonical form. This way, for n different network designs, we will only have n adapters at the gateway (rather than n^2 individual translators), and one canonical form that is consistent across all gateways. Note that this canonical form is not 'yet another network layer'; it is only an internal design component inside the gateway.

Too many requests going through gateways? Only the requests going across a domain to another domain need to go through a gateway. However, this may still end up resulting in an excessive number of requests that a gateway has to process. This can make the gateway un-responsive and be a single point of failure. This is a general problem of gateway-based interoperation. To overcome this, COIN leverages in-network caching, a key domain-specific feature that COIN preserves, because of its use of the native naming schema of the domain. Content coming from another domain through a gateway can still be cached in the consumer domain. Many works have shown that in-network caching is very beneficial since the content demand in the Internet follows a Zipfian distribution [68]. Work in [122] has shown that with a proper caching scheme, in-network caching for a typical web workload can achieve up to 70% hit rate even with cache capacities as little as 2% to 7% percent of the whole content space. With caching enabled, and assuming Zipfian workload, the majority of the requests in COIN would be satisfied in the consumer domain, thus not having to necessarily go to gateway to be processed.

Storage overhead at gateways? Gateways have to keep a small amount of information as state for every incoming request. While in-network caching can dramatically reduce the number of requests the gateway has to process, storing state associated with each of them may still be a challenge. To overcome this, gateways in COIN leverage *request aggregation* for requests for the same content (typically static content). Only the first request is forwarded, while the subsequent identical ones get aggregated (similar to NDN PIT [203]). In addition, COIN gateways can cache content themselves, thus enabling them to respond without having to issue a new request (and thus store associated state) in the next domain. These methods, combined, greatly reduce the amount of memory consumption at the gateways as well as the number of requests going out of them.

5.3 Architecture and Design

5.3.1 Preliminaries

COIN's network environment may be made up of a number of different domains (*e.g.*, IP, NDN, and MF) with gateways connecting pairs of domains, in addition to clients,

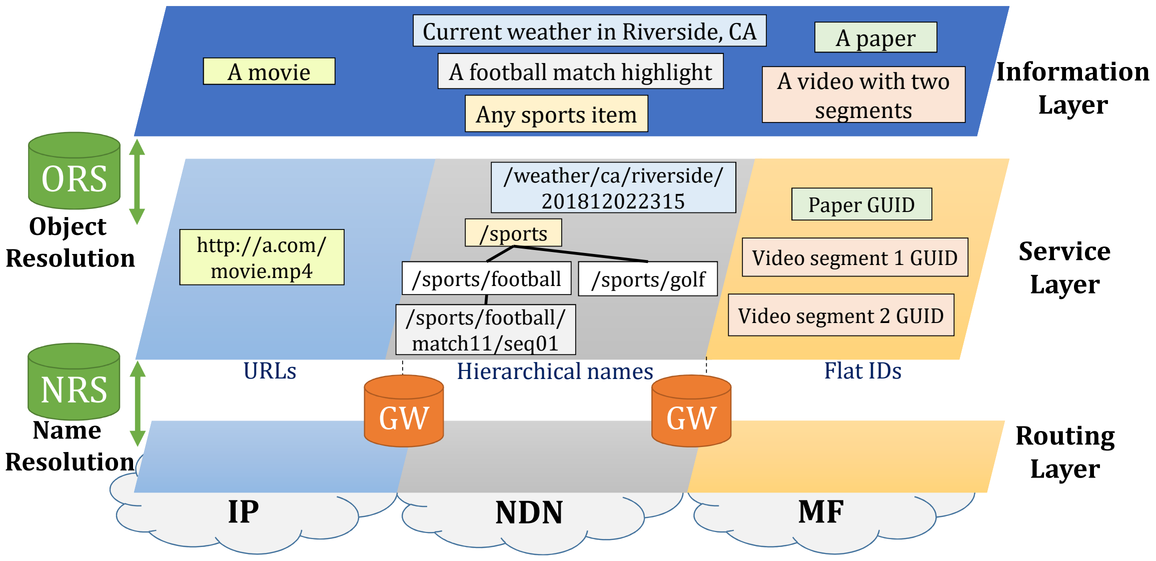


Figure 5.1: Layered architecture overview

servers, publishers, and content repositories that can reside in any of the domains. A high-level, layered architecture view is shown in Fig. 5.1. We identify three layers (similar to [93]), each characterizing an important aspect of COIN's content-centric view. The *Information layer* captures accessible objects and content items in various applications. A *Service layer* shows in what format (hierarchical format, *etc.*) each object in an Information layer is named and identified. The *Routing layer* takes care of transmitting packets to an appropriate (one or more) recipient(s) using routing/forwarding protocols within that domain. It is important to note that we are not adding any new layers; rather, we are recognizing the logical layers that represent functionality that is common in the domain-specific architectures. For example, the service layer is part of the network layer in NDN and MF, and part of the application layer in IP. The Object Resolution Service (ORS) generates names understandable by the corresponding domain's service layer and the domain-specific Name Resolution

Service (*e.g.*, DNS in IP and GNRS in MF). It helps names to be mapped to location information in each domain for routing. The figure also shows that each domain (and producers and consumers in the domain) only needs to understand its own naming structures (whether hierarchical or flat). Gateways facilitate appropriate translations and bridging between different network architectures, preserving their key features and internal mechanisms.

5.3.2 Service Interface

The primary services supported by COIN are static and dynamic content retrieval. Both follow a query/response model, a very popular model in today's Internet as well as in ICNs [92]. This distinction between static and dynamic content requests is important, since they need to be treated differently: The response for a dynamic content request might depend both on consumer's input and the current state of the server (such as a keyword-based search that may depend on time, location, or server policies). Thus, the response cannot be from a cache, since the current server-generated response is desired. Static content requests, on the other hand, have no such restriction. Cached content, in routers or Content Delivery Networks (CDNs), can be returned to consumers, as long as it has the right version. Note that other possible services, *e.g.*, publish/subscribe [49] are not the focus of COIN's current design. However, with additional modules for processing those other services, *e.g.*, push-based multicast or repetitive poll-based request generation component implementation for publish/subscribe with the capability to translate to/from an internal canonical format, COIN can support them as well.

5.3.3 Common Information Elements

These are common elements on which the translation between different formats have to be performed and are primary parts of a COIN gateway's canonical form. COIN supports all protocols that have these common elements (*e.g.*, HTTP/IP, NDN, MF, XIA, NetInf, and even FTP):

- *Request type* (to distinguish between dynamic and static content requests; request type can be a pre-existing field in the packet header or body based on implementation choice);
- *Destination domain and content name* (generally as "DstDomain/ContentName" to identify content and target domain);
- *Content version* (as content may have different versions);
- *Exclude for static content request* (to allow consumers to get the latest version of a content item);
- *Input for dynamic data request* (to allow a consumer to pass parameters to a dynamic data provider); and,
- *Demultiplexing key* (to identify a corresponding request when the response data comes back to a gateway).

5.3.4 Naming

Naming is key to enabling content-oriented interoperability. COIN primarily performs translations at name level. This principle brings important benefits: 1) Each domain keeps its naming schema (*e.g.*, NDN's hierarchical naming [203] or MF's flat IDs [160]), which helps with evolvability. While consumers have to use a globally unique expression for each content as "ContentDomain+ContentName", they do not have to understand such a syntax. It will look seamless to users, as if they are using a native name. This is facilitated by the ORS, as described in §5.3.7. For example, a consumer in an IP domain requesting for content named "/ICCCN/papers/COIN.pdf" in an NDN repository, will send an HTTP request for "http://NDN/ICCCN/papers/COIN.pdf", which is just like any other HTTP request. After going through gateway processing, the NDN repository receives the request as an NDN Interest with name "/ICCCN/papers/COIN.pdf", just like any other NDN Interest. 2) Each domain keeps its namespace size, which helps with scalability. No domain has to keep track of, or maintain, another domain's content namespace (just needs the IDs of other domains). 3) The name-to-location mapping in each domain, utilized in order to deliver to/from gateway, consumer, or producer, is handled by the domain's already existing name resolution service (*e.g.*, GNRS in MF).

5.3.5 Transport, Routing and Forwarding

Motivated by its design principle, COIN allows the composition of connection-oriented and connectionless transport across domains. For example, if a consumer using HTTP/TCP/IP is requesting content from a producer in NDN, the gateway acts as the second end of the TCP connection (*i.e.*, similar to a proxy server listening on an HTTP port) to the consumer (establishing sessions, *etc.*), while acting as a typical NDN client (sending a content Interest in a connectionless manner) towards the NDN side. When the data comes back, the gateway sends the data to the consumer, using the stored IP address and source port information of the consumer.

Similar to today's Internet, COIN decouples intra- and inter-domain routing [133]. Domain-specific routing mechanisms can be leveraged, and be stitched at gateways. In the case of multiple gateways between two domains, inter-domain routing can be used to connect one gateway to the nearest other gateway. Existing architectures need not know or implement these inter-domain algorithms. As for state, gateways connecting the same domains on either side can exchange and share state information, so any gateway can process the response.

COIN gateways process requests and responses differently, with an important distinction being that response forwarding is stateful (§5.3.6). Importantly, gateway forwarding conforms to domain-specific forwarding policies; *e.g.*, NDN has a reverse path forwarding (RPF) policy where every node traversed by the request has to be in the responses path [203]. Other architectures, such as MF, may not have such a restriction, thus allowing secondary gateways to route the response back towards the consumer. Conforming to such forwarding policies, COIN can provide a seamless interoperation across domains, without having to change existing infrastructures.

5.3.6 COIN Gateways

A COIN gateway translates requests for information received from one domain to a request meaningful in the adjacent domain (and similarly for responses). We design and implement the interface to each distinct domain as a "pluggable adapter" on the gateway in

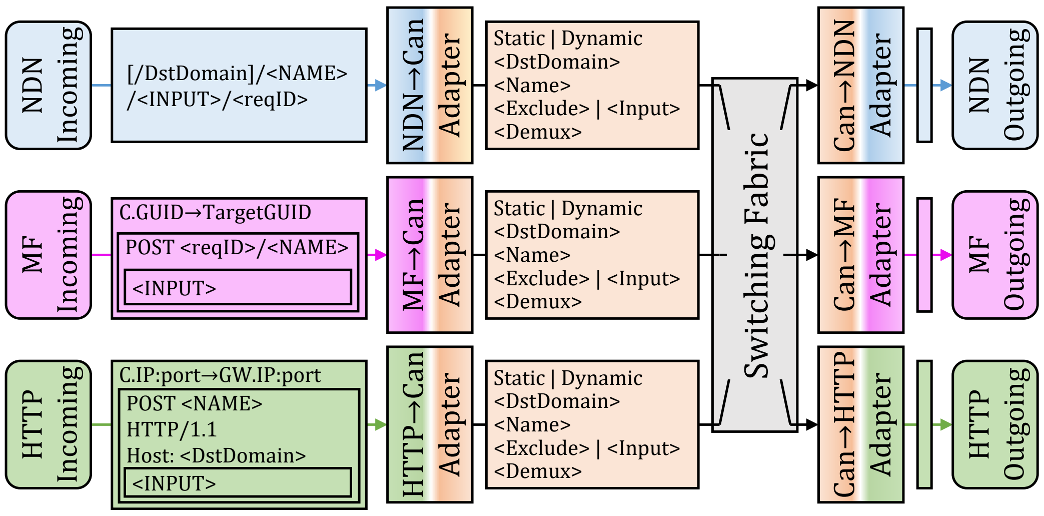


Figure 5.2: COIN gateway design: processing requests

each direction. We choose to translate the incoming request or the headers of the response to an “internal” canonical form (5.3.3).

Incoming request processing involves recognizing whether the request is for static or dynamic content. For NDN, a request with a specific version is seen as a request for dynamic content while a request with just a prefix (and exclude) is for static content. For HTTP, we use POST and GET methods as dynamic and static content respectively. It also determines the destination domain based on the “Host” field in case of HTTP, the destination GUID in MF, and the domain prefix in NDN. The opaque string (from the originating domain’s perspective) that is the name on the destination domain will be extracted from the request (marked as the field “$<Name>$” in Fig.5.2. For dynamic requests, the incoming request processing recognizes the body of the POST in MF and HTTP, and the penultimate component of NDN name, as the request input. The demultiplexing entity ("$<Demux>$") depends on the different cases. For static content requests, we use the tuple $<domainName, contentName, exclude>$. For dynamic content requests, we use client $<IPaddress, port>$ (socket) for HTTP case, client $<GUID, reqID>$ in MF case and $<clientID, reqID>$ in NDN case.

The incoming request processing results in an internal canonical request (orange boxes in Fig. 5.2). The gateway can respond to requests for static content from the local cache, aggregate requests for the same static content (with same exclude) or consume them. The remaining requests (in canonical form) are sent to the "switching fabric", where inter-domain routing determines the forwarding to the proper outgoing request processor. The outgoing request processing forms a domain-specific outgoing request.

When the response returns, the gateway matches it based on the state information and forwards the content to all the pending requests waiting on this key (similar to matching a PIT entry in NDN). This enables native multicast, similar to NDN. We use the "Last-modified" field in HTTP and MF and the version field in the NDN name as the version of the response. The gateway sends the version using the domain-specific format.

5.3.7 Object Resolution

In COIN, an important step in requesting a piece of content residing in another domain is to acquire the content's name and the ID of the domain it is in. This is achieved using an Object Resolution Service (ORS). ORS is an application-layer search engine that: 1) returns names for keyword queries, and 2) leverages a combination of crawling, registration and indexing methods to gain and store knowledge of content names. This way, ORS plays the same role that today's popular search engines, *e.g.*, Google or Bing do. More

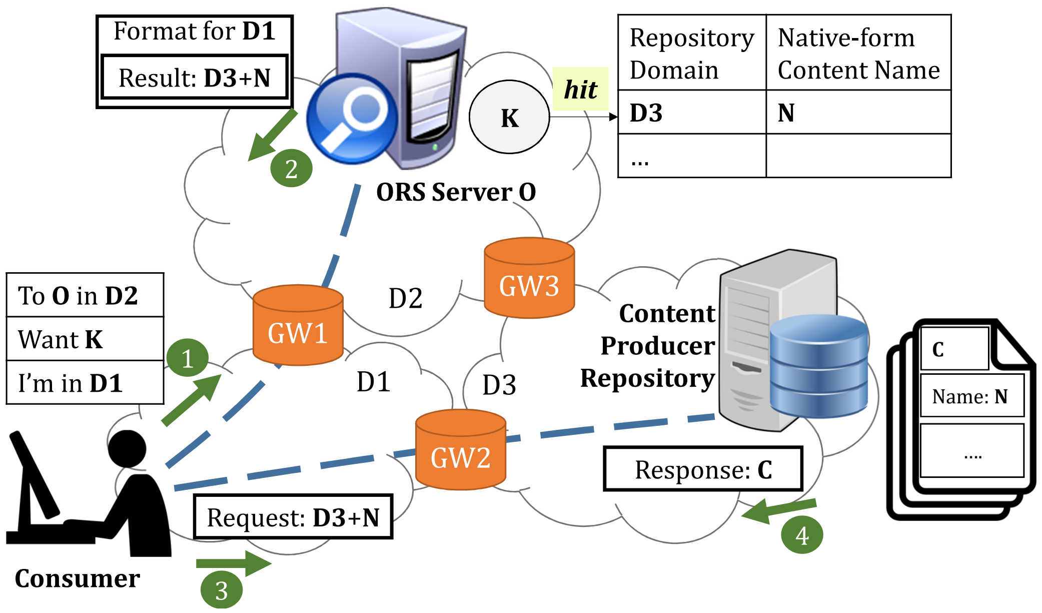


Figure 5.3: A schematic view of object resolution and content retrieval in COIN

specifically, today's single-domain search engines could be considered as a special case of ORS. ORS has to additionally take both the content's and consumer's domain(s) into account: it has to provide the content domain ID in its query result, presented in a format understandable in the consumer domain (and thereby by the consumer). This is an important design choice, as having ORS servers that understand multiple domain languages avoids having all data providers/servers in the world learn the other domains' languages. There has been prior work on ORS in ORICE [14], which we use and extend in COIN.

Fig. 5.3 shows a high-level schematic view of the object resolution procedure. There are three domains with different network architectures $D1$, $D2$, and $D3$, with three COIN gateways stitching them together. The consumer, object resolution server O, and content repository for content C reside in $D1$, $D2$, and $D3$ respectively. We put the consumer and the ORS server O in two different domains to show a more complicated scenario; normally, $D1$ could have an ORS server too which the consumer can ask without having to go across domains. The consumer generates a query for keyword (phrase) K, asking ORS O in $D2$. Identifying the ORS's name and its domain are important to make sure that the query goes to the right gateway. Although the figure only shows the information at a high level, the specific formats depend on what the domain is. For example, if the consumer is in an IP domain, he will perform a DNS lookup on "D2", obtaining the IP address of $GW1$. In NDN, the consumer will send a packet with prefix "/D2/O/", which will be directed to $GW1$ ($GW1$ has already announced and registered itself as "D2" in $D1$). The consumer also specifies that he is in $D1$; so O generates the result in a format understandable to a user in $D1$. With the help of $GW1$'s translation and state-maintenance, the query can reach O and its result sent back to the consumer. Some packet-level details, such as demultiplexing keys are omitted in the figure. More on protocol exchange details are in §5.3.8 (ORS query is an example of a dynamic content retrieval).

Upon receiving the consumer's query, O searches for K in its database of indexed content names and their domains (including content in $D3$). The way this database is managed is similar to today's search engines' crawling and registration methods; more details are provided in [14]. Assume K hits one entry with a content named N in domain $D3$ (for presentation simplicity, we assume only one item in the result, while in practice there can be many more). O generates a result combining $D3$ and N ("D3+N" in Fig. 5.3), formatted for $D1$. What N looks like depends on the naming structure of $D3$; *but* the formatting of the result depends on the semantics of $D1$. For example, if both $D1$ and

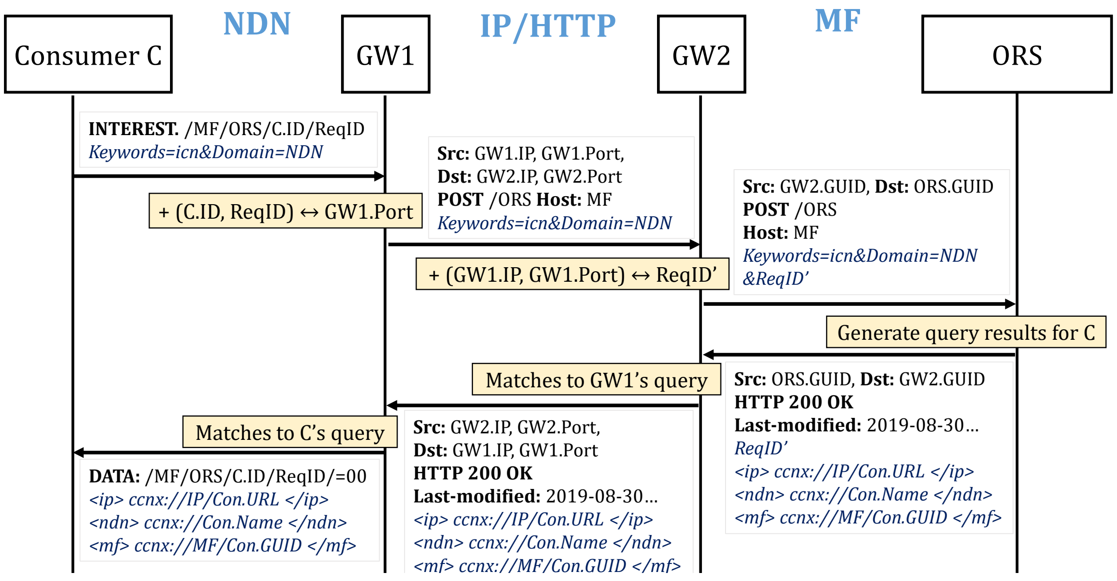
$D3$ are in an HTTP/IP domain (as in today's Google search), then N would be a URL (*e.g.*, "abc.com/def"), formatted and presented to the consumer as "http://abc.com/def" (no indication of $D3$ is needed for same-domain pairs). As another example, if $D3$ is MF, then N would be a content GUID (*e.g.*, "1234"). If $D1$ is NDN, then the result would be the enriched name "/MF/1234".

After gathering the result, the consumer generates a request for the content itself, using the acquired name N combined with $D3$, getting routed to $GW2$. Note that for this purpose, the consumer's own domain ID is not needed, since the repository returns content C (named N), not knowing (and not needing to know) where the consumer resides.

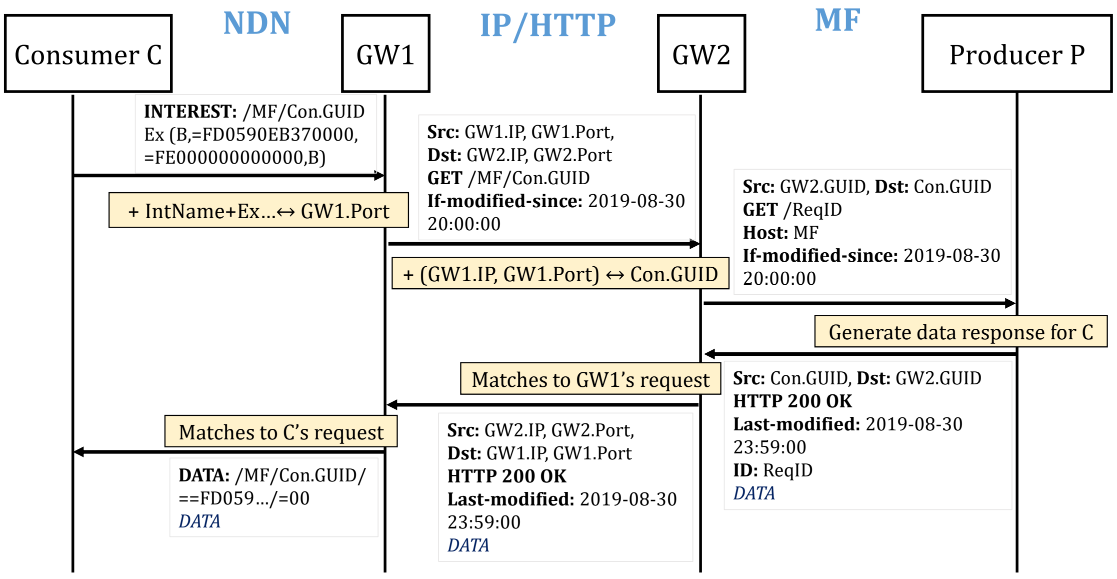
While at first glance, it may seem a burdensome task to acquire names through the ORS for content requests, it follows the pattern that users use on the Internet today in practice [14]. For example, most often, retrieving a webpage for the music video of the 2017 song "Despacito", is proceeded by a (Google) search such as for "Despacito music video". ORS in COIN plays the same essential role as the search engine. It is also worth mentioning that ORS is a service which can be provided by many entities (as we have Google, Bing, *etc.*) and each can have many physical servers. We believe ORS is an important, convenient service to deploy by ISPs or third-party entities, and provides benefits for interoperability.

5.3.8 Protocol Exchange

To illustrate the translation-based exchange for interoperating across multiple domains, we use a 3-domain setting where a consumer residing in an NDN domain wishes to receive content from a server/producer residing in an MF domain, with an IP domain in


NDN
IP/HTTP
MF
Consumer C
GW1
GW2
ORS
INTEREST. /MF/ORS/C.ID/ReqID
Keywords=icn&Domain=NDN
+ (C.ID, ReqID) ↔ GW1.Port
Src: GW1.IP, GW1.Port,
Dst: GW2.IP, GW2.Port
POST /ORS Host: MF
Keywords=icn&Domain=NDN
+ (GW1.IP, GW1.Port) ↔ ReqID'
Src: GW2.GUID, Dst: ORS.GUID
POST /ORS
Host: MF
Keywords=icn&Domain=NDN
&ReqID'
Generate query results for C
Matches to GW1's query
Src: ORS.GUID, Dst: GW2.GUID
HTTP 200 OK
Last-modified: 2019-08-30...
ReqID'
<ip> ccnx://IP/Con.URL </ip>
<ndn> ccnx://Con.Name </ndn>
<mf> ccnx://MF/Con.GUID </mf>
Matches to C's query
Src: GW2.IP, GW2.Port,
Dst: GW1.IP, GW1.Port
HTTP 200 OK
Last-modified: 2019-08-30...
<ip> ccnx://IP/Con.URL </ip>
<ndn> ccnx://Con.Name </ndn>
<mf> ccnx://MF/Con.GUID </mf>
DATA: /MF/ORS/C.ID/ReqID/=00
<ip> ccnx://IP/Con.URL </ip>
<ndn> ccnx://Con.Name </ndn>
<mf> ccnx://MF/Con.GUID </mf>

(a) Dynamic content retrieval (object resolution example)



(b) Static content retrieval

Figure 5.4: Protocol exchange across 3 domains

the middle (Fig. 5.4). We examine two cases: dynamic content retrieval (DCR) using the example of ORS (Fig. 5.4(a)); and static content retrieval (SCR, Fig. 5.4(b)).

The three different architectures take care of content naming at different domain-specific layers: the HTTP application layer, in IP; network layer in NDN; and either HTTP or network layer in MF. When a client generates an NDN Interest, to enable correct translation, we use the destination domain ID in the name. To distinguish between DCR and SCR, we use POST and GET methods in HTTP respectively; we check the existence of Exclude or Input in NDN Interests. For DCR (Fig. 5.4(a)), the retrieved response should not be from a cache, since the current server-generated response is desired. This requires individual requests to be distinguishable (globally unique), to have the correct response-to-request mapping at the servers and gateways, including even those made by the same client. In TCP/IP, client IP and port numbers provide this demultiplexing capability. For NDN and MF, we introduce a unique Request ID (ReqID) generated by the consumer or gateway. The ReqID can be a component of the DCR Interest name in NDN, and part of the request payload in MF. Gateways create state (marked as '+' in the Fig.) associated with each outgoing request, and maintain state for demultiplexing. For example, in Fig. 5.4(a), the mapping on an NDN-to-IP gateway is a 3-tuple of <Client ID, Request ID, GW1 port number>. When the response data is returned, the gateway can find the corresponding request based on its port number (which is the source port number that was previously used to connect to GW2) in the response. As can be seen in Fig .5.4, using domain-specific naming, in-network caching can be supported and provide benefits, in COIN.

5.3.9 Security

Securely bridging communication across different network architectures that have different security models and mechanisms seamlessly, without significant changes to the individual architectures is challenging. We aim to unify different security models across the architectures. They may be classified as being either channel-based (for host-centric networking *e.g.*, IP), or content-oriented (for ICNs). The fundamental distinction between the two security models lies in the relationship between the “name” layer (content retrieval functionality) and security layer (ensuring confidentiality, provenance, integrity, *etc.*) functionality in the service layer. With connection-oriented security the security layer (TLS/SSL) operates below the name layer (HTTP). Interoperability gateways do not have access to information such as keys, as they are encrypted. For interoperability, the gateways have to decrypt the information exchanged to get the name and other features required for content retrieval. In contrast, content-oriented security may just encrypt the data (payload) and leave the content-retrieval headers (*e.g.*, NDN/MF headers, including content names) in the clear. Thus, gateways can reformat the headers without modifying or having to access the payload. COIN supports a number of mechanisms to unify access to information across these two security models. We focus on two important security use cases of COIN: Encryption (to ensure *confidentiality*); and Signatures (to ensure *provenance* and *integrity*). The mechanisms presented here are security-enhancements to protocol exchange presented in §5.3.8.

Encryption

Encryption prevents unauthorized network nodes (including eavesdroppers) from accessing confidential content. The common approach to achieve this is to encrypt the data (*e.g.*, RSA and ECC [134]) or encrypt the channel between the data consumer and producer (*e.g.*, HTTPS). The producer and a (set of) predefined (authorized) consumer(s) have to agree on a common encryption mechanism. We focus on content retrieval across compositions of content-oriented (ICN) and channel-based (IP) security models, with endpoints having the same security model, and when they are different.

Case 1: Both endpoints with content-oriented security model, and intervening domains with channel-based security. We consider a scenario with a consumer and producer in two separate NDN domains using content-oriented security, and an IP domain in between using channel-based security. Fig. 5.5 shows COIN's encryption-enhanced protocol exchange for this case. With both the consumer and producer using content-oriented encryption, the authorization information ($auth_C$) and $Data$ would be encrypted when traversing the gateways. The authorization information can be the consumer's public key (pub_C), following a priori consumer-producer consensus on the authorization mechanism. The gateways simply translate between NDN names and HTTP/HTTPS URLs, without needing to decrypt and/or re-encrypt $auth_C$ or $Data$. Thus, COIN ensures end-to-end confidentiality.

Case 2: Either endpoint with channel-based security When at least one of the two endpoints (consumer and/or producer) uses channel-based security (*e.g.*, HTTPS) and the other(s) use content-based security, gateways would then need to re-encrypt the

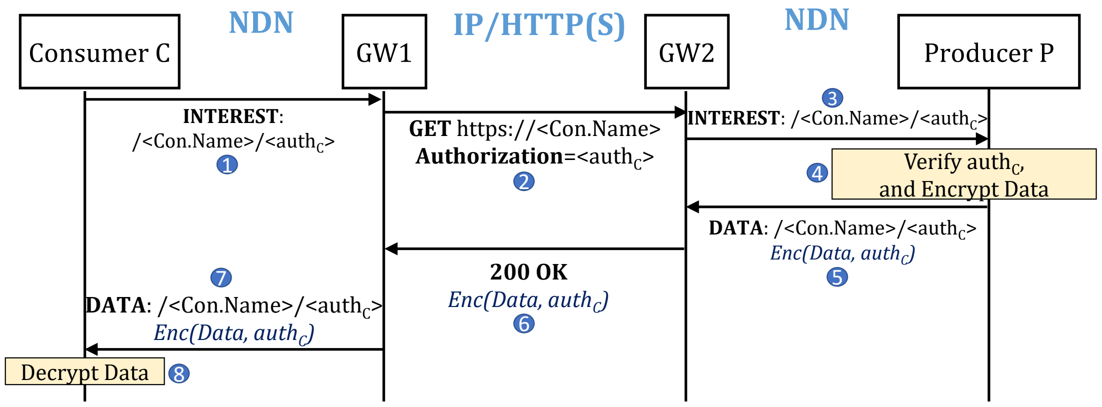


Figure 5.5: NDN/IP/NDN encryption

data retrieved in one domain to provide confidentiality while delivering the content to the other domain. COIN's gateways borrow ideas from the popular state-of-the-art solution of HTTPS proxies. The gateway has to decrypt the HTTP header inside a TLS connection, in order to discover the content name (URL). The gateway acts as a proxy, trusted by both consumer and producer. We believe this is acceptable, as it is a well-established practice to trust HTTPS proxies.

Unlike Case 1, COIN's mechanism in Case 2 decouples encryption and authorization, to allow composition of two different security models of the end points: An HTTP/IP end point achieves this by using the HTTP header "Authorization" field, or Web-based authorization (the consumer provides the username and password, which are carried in the HTTP request body). The producer can then verify the authorization.

Fig. 5.6 shows an example for this case with a gateway between HTTPS/IP and NDN. The consumer in the IP domain requests the content as if the gateway is an HTTPS proxy, by establishing a secure connection to the gateway using the public key of its own

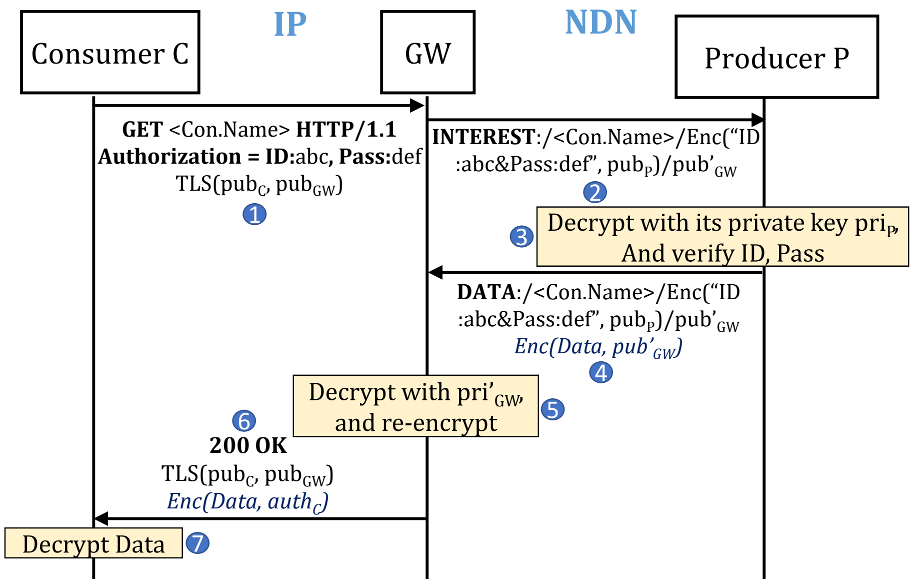

IP
NDN
Consumer C
GW
Producer P
GET <Con.Name> HTTP/1.1
Authorization = ID:abc, Pass:def
TLS(pubC, pubGW)
1
INTEREST:/<Con.Name>/Enc("ID:abc&Pass:def", pubP)/pub'GW
2
3
Decrypt with its private key priP, And verify ID, Pass
DATA:/<Con.Name>/Enc("ID:abc&Pass:def", pubP)/pub'GW
Enc(Data, pub'GW)
4
Decrypt with pri'GW, and re-encrypt
5
6
200 OK
TLS(pubC, pubGW)
Enc(Data, authC)
Decrypt Data
7

Figure 5.6: IP/NDN encryption

(pub_C) and the gateway (pub_{GW}) (Diffie-Hellman key exchange in TLS). In the HTTP header (or body) over the TLS connection, the consumer sends the content name and the authorization information (username and password in step 1 in the Fig., or alternately the public key of the consumer). The gateway creates an Interest in the NDN domain. We make minor modifications to the name in NDN (to provide the authorization), using the format "$/name/auth_C/encrypt$", to provide the producer with the needed information for authorization and encryption. In the Fig., the gateway encrypts the authorization information with the public key of the producer (pub_P) and uses its own public key (pub'_{GW}). The gateway could use different key pairs (*e.g.*, pub'_{GW} and pub_{GW}) for the two different domains (step 2). For MF, the request packet format would include a new field for the authorization information. When the field is not set, authorization information is used as encryption information, as in Case 1. On receiving the request, the producer will decrypt it using its own private key and verify the authorization information (step 3). Upon verification, the producer sends the NDN Data packet (to the gateway) whose payload is $Data$, encrypted by pub'_{GW} (step 4). The gateway decrypts the data with its own private key pri'_{GW} (step 5) and sends the data over the TLS connection to the consumer (step 6). The consumer can then decrypt and access the data (step 7). The reverse, ICN-IP scenario, would follow a similar pattern.

Signatures

In the scenario where the producer allows the content to be shared with *anyone* in the network. The consumers need to verify that: 1) the data is coming from a trusted producer (*provenance*) and 2) no one on the path has tampered with the content (*integrity*). To ensure the integrity of the content, a cryptographic hash function (*e.g.*, MD5, SHA-1, SHA-256) can be applied to the data and announced to the consumer. Provenance is verified by a digital signature: the hash encrypted by the private key of the producer (*e.g.*, RSA signing, ECDSA, EdDSA [186]). The consumer can decrypt the signature with the public key of the producer, and compare the result with the hash of the content, possibly followed by some trust schema [198]. For interoperability across different domains, it is highly likely that the consumer may not understand the producer's signature algorithm or the trust schema (or both). To overcome this, COIN takes advantage of *transitive trust* [161] with domain-by-domain signatures: the gateway on the producer side verifies the provenance and integrity of the data on behalf of the consumer and re-signs data with its own private key for the next domain. The consumer verifies (and trusts) the last hop gateway.

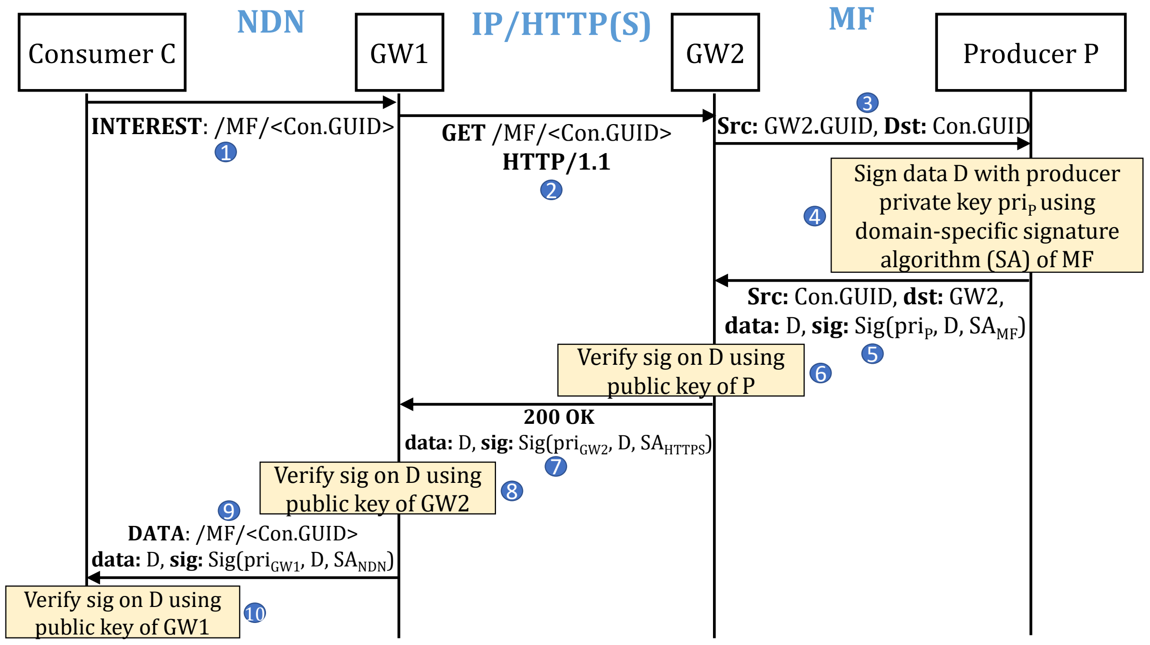

NDN
IP/HTTP(S)
MF
Consumer C
GW1
GW2
Producer P
INTEREST: /MF/<Con.GUID>
1
GET /MF/<Con.GUID> HTTP/1.1
2
3
Src: GW2.GUID, Dst: Con.GUID
4
Sign data D with producer private key pri_P using domain-specific signature algorithm (SA) of MF
Src: Con.GUID, dst: GW2, data: D, sig: Sig(pri_P, D, SA_MF)
5
Verify sig on D using public key of P
6
200 OK data: D, sig: Sig(pri_GW2, D, SA_HTTPS)
7
Verify sig on D using public key of GW2
8
9
DATA: /MF/<Con.GUID> data: D, sig: Sig(pri_GW1, D, SA_NDN)
Verify sig on D using public key of GW1
10

Figure 5.7: NDN/IP/MF signatures

Fig. 5.7 shows an example of our solution spanning 3 domains. After receiving the request, the producer will sign the data (D) with its own private key (pri_P) based on the signature algorithm in the domain (SA_{MF}). On receiving the content, $GW2$ will verify the provenance and integrity using the public key of the producer (pub_P) on behalf of the consumer, since it understands the signature algorithm and can also utilize local certificate authorities (CAs) to check its trustworthiness. Once $GW2$ confirms that the content is trustworthy, it will re-sign the data with its own private key (pri_{GW2}) using the signature algorithm in the IP domain. $GW1$ will thus trust the producer, since it trusts $GW2$ (due to transitive trust). Once the signature is verified using $GW2$'s public key, $GW1$ will forward the data to the NDN domain and sign the content using its own private key. Since the consumer trusts $GW1$, it concludes that the content is trustworthy.

Denial of Service (DoS) Attacks

While request aggregation and content caching alleviate COIN from some negative impacts of excessive request and response processing load (that is *benign*) on gateways, *malicious* DoS (and DDoS) attacks can impact gateways' availability. IP and ICN domains, with different security models, can be the source of different DoS attacks, such as bandwidth depletion and reflection attacks in legacy IP domains [62]. With ICN's content-oriented security models, DoS attacks manifest themselves mainly as Interest flooding (too many malicious requests for non-existing content) and content/cache poisoning (responding with fake or corrupted content) [77]. In COIN, each domain retains its own security model and mechanisms, to allow development of countermeasures for attacks meaningful in its own domain. Thus, in COIN, DoS attacks in a domain are contained within that domain. For example, with Interest flooding, excessive requests will be dropped at or before the ICN-border gateway (through mechanisms such as statistic-based rate limiting [51, 77]). Similarly for content poisoning, with the proper use of content-oriented signature validations, fake or corrupted content will be detected and discarded at or before the first gateways it encounters.

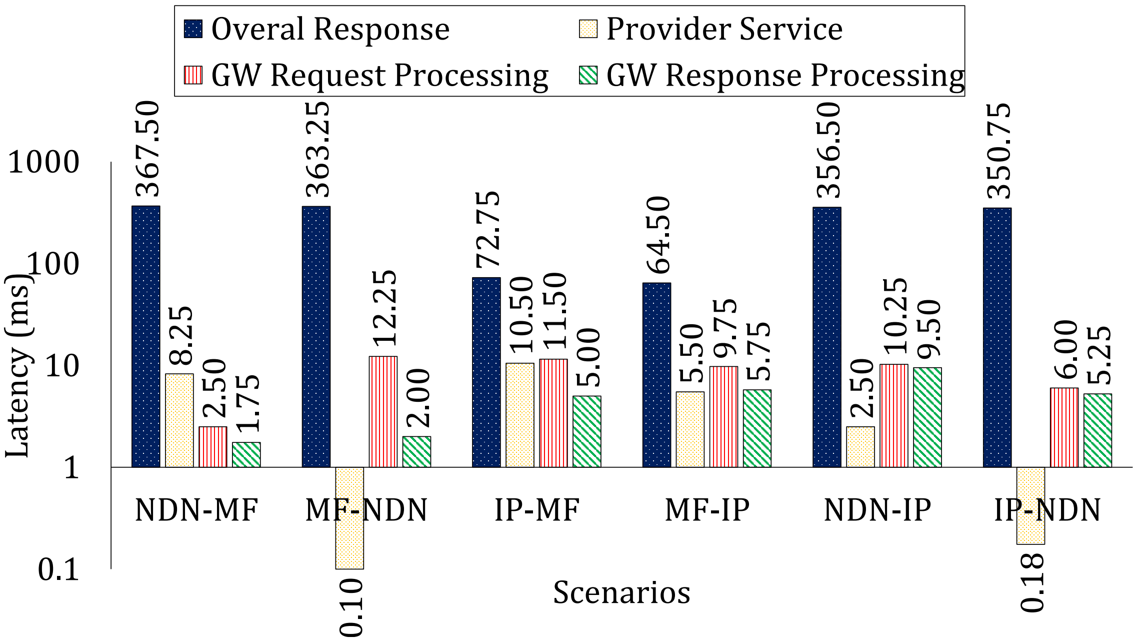
5.4 Experimental Results

For evaluation, we use a representative implementation for each domain: CCNx v0.8.0 (it contains all the essential components of NDN needed for our framework); the latest version of MobilityFirst project [8] for MF domain; and a basic Linux implementation of IP forwarding. The implementation of COIN, including all its essential components such as gateway and adapter modules are available in [6] as proof of concept. We experimented on various combinations and settings, and observed COIN's ability to satisfy its design goals, especially its ability to preserve each domain's key features. While there are a number of aspects to consider including correctness, user convenience, deployment flexibility etc., we focus on the performance of COIN here (we have proved the correctness of the proposed translation procedures in [94]).

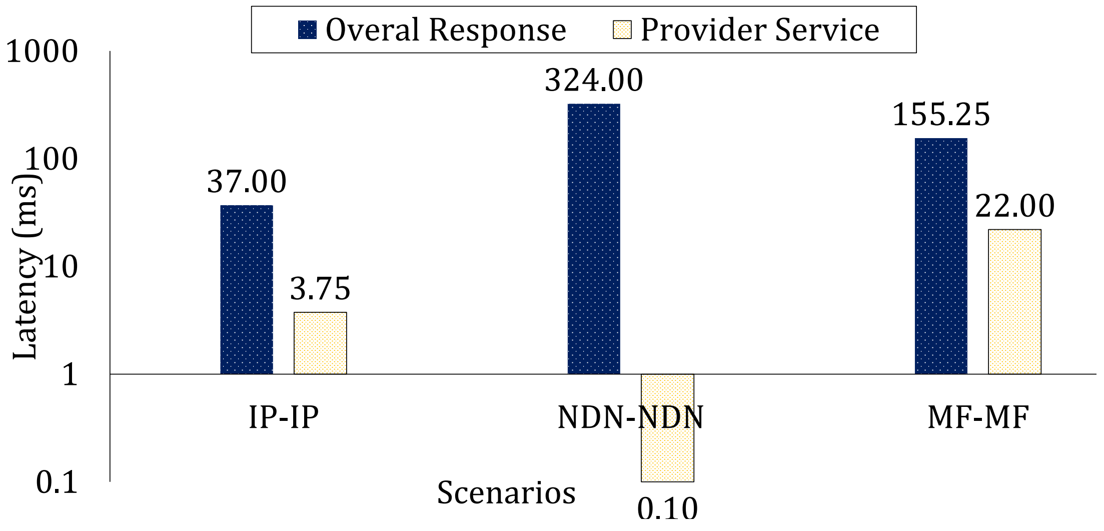
5.4.1 Forwarding Efficiency

To evaluate the forwarding efficiency of the implementation, we set up a testbed with five VMs with the topology as "$C \leftrightarrow R_1 \leftrightarrow GW \leftrightarrow R_2 \leftrightarrow P$", in which client C and router R_1 are in domain D_1, and content provider P and router R_2 are in domain D_2. Node GW, an implementation of COIN gateway, is in between the two domains and performs translation. Each VM has 1GB memory and runs Ubuntu 14.04. With cases of domains D_1 and D_2 being both distinct (*i.e.*, *interoperation* scenarios) and same (*i.e.*, *native* scenarios), we evaluated all 9 combinations (each being IP, NDN, MF). In native scenarios, GW is replaced by a regular router, with the same configuration as R_1 and R_2. We tested functionality with a client asking for content residing in a remote domain of a potentially different architecture, and getting the content back.

Fig. 5.8 shows the latencies measured for requests for static content. The overall content retrieval time (response time) at the consumer, the provider's service time and gateway processing time for request and response (averaged over several runs, discarding outliers), are shown. Note the difference in the y-axes of the different bars in the figure.



(a) Interoperation scenarios (different domain types for D_1 and D_2)



(b) Native scenarios (same domain types for D_1 and D_2)

Figure 5.8: Latencies (static content retrieval): total response, content provider, gateway request and response processing (logarithmic axes). Note that there are no gateways (and thus gateway latencies) in native scenarios (b).

As shown in Fig. 5.8(a), the processing time at the gateway in interoperation scenarios, including reformatting and maintaining state between the two domains, while not negligible, is reasonable for an initial software implementation. The gateway contributes between 4-19 ms of processing delay, compared to the total response time of 60-360 ms, in this small-scale topology. The gateway contributes a relatively small portion of the overall response time, especially in the ICN cases. It should be noted that the higher response time observed whenever one side is NDN, is not due to the interoperability gateway, but rather the NDN logic itself: the client waits for sending a second query to ensure it has received the latest piece of content. In fact, we observed a response time of $\sim$300 ms for NDN$\rightarrow$NDN on our testbed. This is seen in Fig. 5.8(b) as well, which shows the latencies for native scenarios, *i.e.*, those with same domain types throughout the topology, with no COIN gateways. Our results show that the overhead of COIN's forwarding is reasonably small.

5.4.2 Scalability

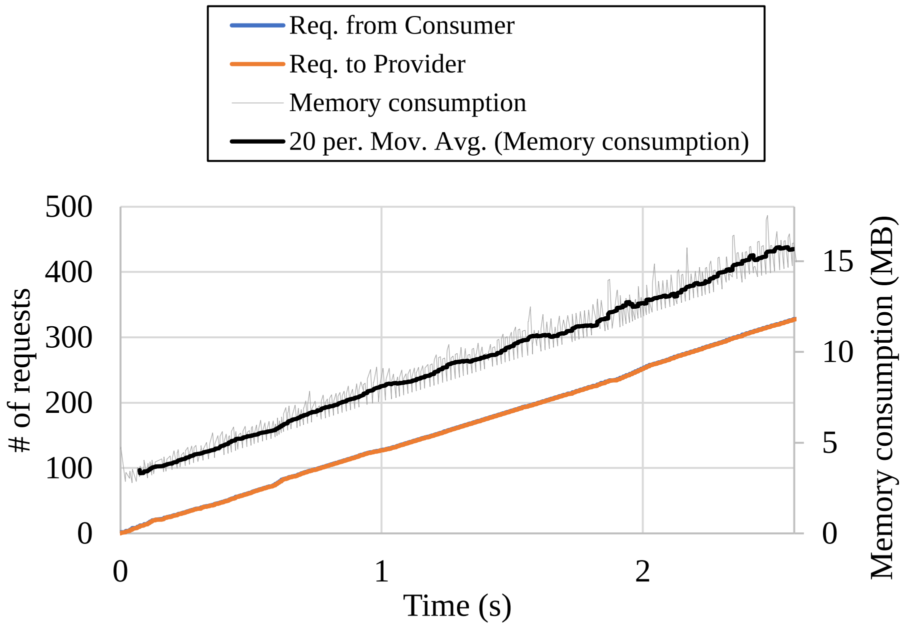
We use the ORBIT testbed [10] to evaluate the scalability of COIN. ORBIT is a network of 400 nodes in a grid topology. Each machine has 4GB memory and runs Ubuntu 14.04. We run each router (forwarding engine), provider, consumer and gateway on separate physical nodes. In our topology, we included 50 consumers (in one domain), 1 provider (in another domain) and 1 gateway. The consumers are connected to the gateway via a pre-configured access network. Our server stalls the response to each request for 3 seconds to batch more requests on the gateway (especially for NDN, and 3 sec. because the request timeout time in NDN is $\sim$4 sec.)

We measure the amount of state stored in the gateway (memory consumption) *vs.* different numbers of requests from consumers. The implementation is in Java, which has automated memory management. We call garbage collection very frequently during run-time to get a better estimate of memory consumption. This would make our gateway slightly slower compared to production use. We evaluate the requests to static and dynamic content separately. Only the results of the experiments for IP→NDN is shown in Fig. 5.9.

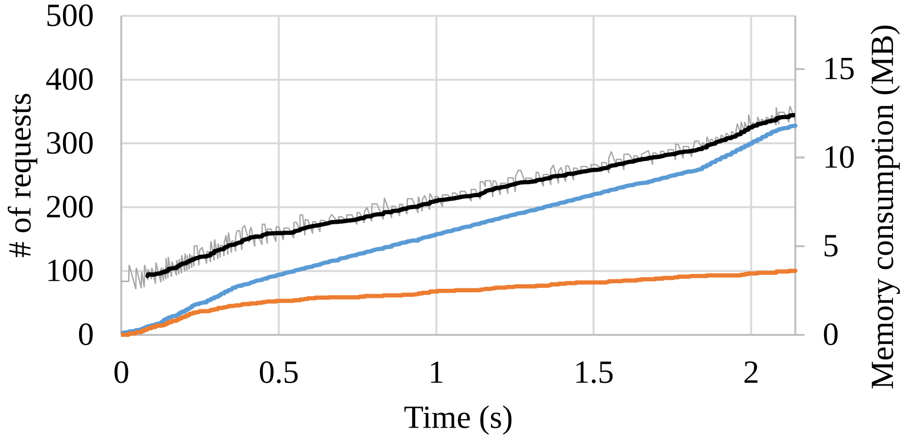
Evaluation for dynamic content: We have 50 clients sending 328 dynamic content requests simultaneously. Fig. 5.9(a) shows the instantaneous memory consumption (and moving average over 20 values) *vs.* the number of incoming and outgoing requests for this scenario.

Since consumers request dynamic content, we do not see any aggregation at the gateway – each request from the client results in a distinct request to the provider. Therefore, the incoming and outgoing request values are very close to each other in the Fig. We observe that the memory consumption grows linearly with the number of incoming requests since we keep the states for each request.

Evaluation for static content: Clients make 328 requests spread across 100 static content items simultaneously. The popularity of the content follows a Zipf distribution with α=0.81. [121] shows that this is a realistic content demand model. The server still waits 3 seconds before sending the response to allow request accumulation. Fig. 5.9(b) shows the results. Since we keep the state on the gateway, we can aggregate multiple requests for the same content (name). Therefore, the number of requests arriving at the provider is smaller than the number of requests generated by the consumers. The memory consump-



(a) Dynamic content retrieval



(b) Static content retrieval

Figure 5.9: Scalability: memory consumption and number of requests

tion grows sub-linearly relative to the incoming requests. The memory consumption is also lower, compared to that of dynamic content for the same number of requests.

We ran the same experiments in other domain combinations (NDN→IP, MF→NDN, *etc.*) and saw similar results. Although keeping per-session state puts additional burden on the gateways (state grows with number of flows), it is analogous (and no worse than) maintaining PIT state at an NDN router. Since COIN allows request aggregation and content caching at the gateways, this interoperability framework scales better.

5.5 Formal Modeling of Information-Centric Interoperability

We now describe the basics of our formal model[1]. First and foremost, let us define information-centric interoperability (ICI):

Definition 1 *A sequence of interconnected domains in a network are information-centrically interoperable if and only if any client in any of the domains can access information-centric services provided in any other domain.*

Throughout this paper, we use the term "network" to mean "a composition of multiple network domains", each domain being a different type of standalone architecture (*e.g.*, IP, NDN, or MF). An interoperability framework (such as [41]) is a set of protocols and architectural components that allow interconnected networks of different types to interoperate. Information-centric services are broadly sub-categorized as: 1) requesting for and retrieving content (pull-based), and 2) subscribing to and receiving content (push-based). Both of these may be based on namespaces defined by content producers. An example 3-

[1]Full source files are available in [1].

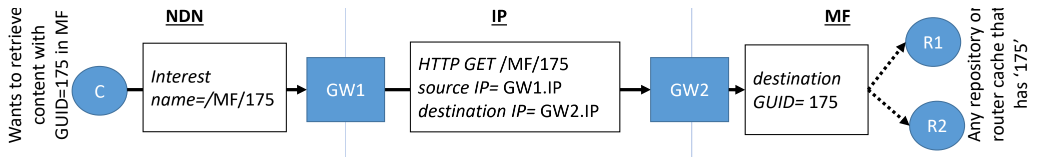


Figure 5.10: Information-centric interoperability (ICI): request for content

domain ICI scenario is depicted in Fig. 5.10. As shown, ICI accesses content by name, rather than an address. Also, requests can be satisfied at any cache node, not just the original server. As for formal analysis, in ICI, the main property we care about is *node-to-content reachability* [100], while in traditional host-centric interoperability (HCI) analysis [199], the focus is on *node-to-node reachability*.

We model our networked environment using Alloy's relational and logical atoms. We have *Domains* (as abstract signatures), each of which can be an IP, NDN, or MF type (extended signatures) (Listing 5.1). A *Node* is at least in one *Domain* and has at least one *NodeID*. A *Node* can be either a *Client*, *Repos* (repository/server), or *GW* (gateway). A gateway is associated with exactly two *Domains* (constrained using facts), that it is stitching together (Listing 5.2.)

Listing 5.1: Domains

```
abstract sig Domain{}
sig IPdomain extends Domain{}
sig NDNdomain extends Domain{}
sig MFdomain extends Domain{}
```

Listing 5.2: Nodes

```
abstract sig Node{domains: some Domain, id: some NodeID}
sig Client extends Node{...}{...}
sig Repos extends Node{...}{...}
sig GW extends Node{...}{#domains=2 && ...}
```

Our declarations specify a network *meta-model* [91], which maps to a number of instances (models) each being a network configuration (*i.e.*, with their own topology, content, namespace, *etc.*). An example 2-domain instance is depicted in Fig. 5.11, as a high-level schematic, showing objects and their inter-relations. The *Client* here wishes

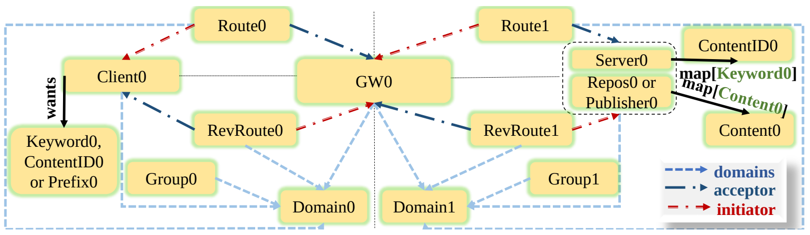


Figure 5.11: Example (partial) instance for ICI Alloy model (objects and relations)

to retrieve some *Content* using its *ContentID* or a (set of) *Keyword*(s). Objects of type *Route* and *RevRoute* (reverse route) couple the notion of "a series of links" and "packets carried over them", the packet carrying content request and response, respectively. A *Route* has attributes such as *initiator*, *acceptor*, and a request for *ContentID*. We also extend signatures to add more fine-grained, domain-specific characteristics. One of *Route*'s extended object types, namely *IPRoute*, inherits its attributes and constraints, and also has additional attributes such as *srcIPaddress* and *destIPaddress*, and constraints saying that source and destination IP addresses must correctly correspond to initiator and acceptor nodes. Gateways perform translation for forwarding requests (over a composition of *Route*s), and retain state information which they use to forward the content back to the client (over composition of *RevRoute*s). We also add a number of additional facts, such as uniqueness of node ID, absence of self-looping routes, and the existence of one-to-one mapping between NDN's forward and reverse routes (to reflect NDN's RPF policy [203]).

We define a global-state relation C that captures routes to/from gateways. To model connectivity, we use the transitive closure of the route-connections relation C where $(r1, r2) \in C$ if and only if there exists a gateway between two domains that connects routes $r1$ and $r2$. *E.g.*, if we have $C=\{(r1, r2), (r2, r3)\}$, then its transitive closure

C^{+}=$\{(r1, r2), (r2, r3), (r1, r3)\}$ will represent existing paths of any length (*i.e.*, number of routes). We define object type *Connections* (as a singleton) to capture these connections (*i.e.*, relation C); it has attributes being relations themselves, primarily *connected* and *revconnected*, to capture connection relations of *Route*s and *RevRoute*s respectively. Relation *revconnected* has an additional constraint, which says that for two reverse routes $rr1$ and $rr2$ connected at gateway gw, corresponding state information (associated with the *ContentID* or other multiplexing/demultiplexing values in $rr1$ and $rr2$) must be stored on gw, so that the content can be carried over this cascade of reverse routes towards the consumer (Listing 5.3). Additionally, we define a fact (*path_exists*, Listing 5.4) that ensures any two nodes are connected (through one or multiple *Route*s or *RevRoute*s), to reduce our instance space to only the ones with strongly connected topology.

Listing 5.3: Connections

```
one sig Connections{connected: Route->Route, revconnected: RevRoute->RevRoute, chain: Group->Group,
      revchain: Group->Group}
fact connectivity{ -- conditions for two (reverse) routes being ‘‘connected’’
    all r1,r2:sRoute, c:Connections |
        (r1->r2) in c.connected <=>
           r1.acceptor = r2.initiator && r1.contentID = r2.contentID && r1.reposdomain = r2.reposdomain
                -- requests for same content
    -- similar condition for RevRoute paths (with extra criteria: gateway state information should
          match for the two connecting reverse routes) ...
}
```

Listing 5.4: All paths existence constraint

```
fact path_exists{
    all co:Connections, disj n1,n2:Node, cid:ContentID, rd: Domain |
        (some r:Repos | rd in r.domains =>
            (some r1,r2:Route | (r1->r2) in ^(co.connected) && r1.initiator = n1 && r2.acceptor = n2 &&
                 r1.contentID = cid && r2.contentID = cid && r1.reposdomain = rd && r2.reposdomain = rd))
     -- similar condition for RevRoute paths ...
}
```

While *Route*s represent unicast exchange paths, we define *Group*s to denote multicast groups (one-to-many communication), enabling push-based notification models. Following the principles of ICN, each group is associated with a content name *Prefix* [49] and can be used for publish/subscribe exchanges regarding that prefix. Each group belongs to one domain. To model a connection of groups across multiple domains, we add relation attributes *chain* and *revchain* to *Connections* (Listing 5.3), to capture connectivity of groups (as a chain) for subscription and publication respectively. To ensure strong connectivity, we add a fact that says any two groups serving the same prefix are chained (Listing 5.5).

Listing 5.5: Group rules constraints

```
fact GroupRules{
   all disj g1,g2:Group, co:Connections | -- group chain conditions
       (g1->g2) in co.chain <=>
           (g1.prefix = g2.prefix &&
           (some gw:GW| g1.domain in gw.domains && g2.domain in gw.domains))
   all disj d1,d2:Domain, co:Connections, p:Prefix | -- chains for each prefix
       some disj g1,g2:Group |
            g1.domain = d1 && g2.domain = d2 && (g1->g2) in ^(co.chain) &&
            g1.prefix = p && g2.prefix = p
   -- similiar conditions for revchain ...
}
```

Content naming is integral in ICI. We define names, *i.e.*, *ContentID* objects for each *Content*. Based on domain type, *ContentID* can be either *URL* (in IP), *NDNName* (in NDN) or *ContentGUID* (in MF) (Listing 5.6). Each *ContentID* is a leaf node under a *Prefix* in the prefix tree (*PTree*). An example prefix tree is shown in Fig. 5.12, which represents the network's content namespace. *PTree* may contain a number of fragmented sub-trees (*i.e.*, as a forest), each sub-tree representing the namespace of a different (set of) content provider(s) in different domains. To represent the structure of hierarchical prefixes, we use binary relations to model the immediate parent-child relationship between prefixes in *PTree*. In Fig. 5.12, the relation $P = \{(P1, P2), (P1, P3), (P2, P4), (P2, P5)\}$ represents such relationships, and is captured in the prefix-to-prefix relation *map* in *PTree* (Listing 5.6). We also use its transitive closure to model the ancestor-descendant relationships. We add additional facts to ensure basic constraints on the tree, such as the non-existence of loops.

Listing 5.6: Content IDs and Prefix Tree

```
abstract sig ContentID{prefix: Prefix}
sig URL extends ContentID{} -- if in IP
sig NDNName extends ContentID{} -- if in NDN
sig ContentGUID extends ContentID{} --if in MF
sig Prefix{parent: lone Prefix, domains: some Domain} -- each Prefix has exactly one parent and is at
      least in one domain
one sig PTree {map: Prefix set -> set Prefix}
```

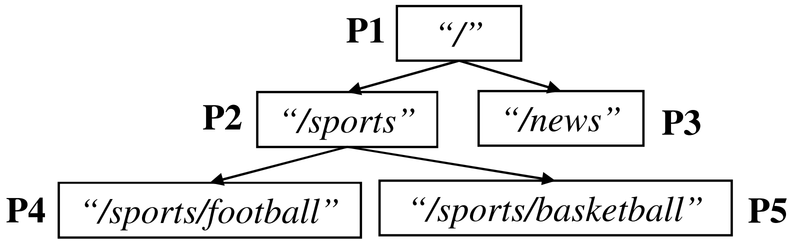


Figure 5.12: Prefix tree example

5.6 Satisfying Information-Centric Service Properties

There are a number of important properties that are required from the framework, to ensure interoperability as defined in Definition 1. We consider properties of two classes of information-centric services here: pull-based (for unicast request/response), and push-based (for multicast publish/subscribe) content retrieval. We further divide the pull-based services into two categories: static content retrieval (SCR) and dynamic content retrieval (DCR). This distinction is important as the nature, protocol for retrieval, and thus formal properties of the two are different: static content is one that does not change in a long time (*e.g.*, a movie) and can be retrieved from its original producer as well as a cache, while dynamic content is created once on demand (*e.g.*, result of a Google search), and must be retrieved from its original server (not from a cache). Additionally, we assume content requests are assumed to be genuine and correct, *i.e.*, false and bogus content requests are not our focus here.

We study essential invariant properties, guaranteed to hold at all times. These properties are primarily associated with content-oriented reachability and returnability. We formally specify these properties, using Alloy predicates and assertions. For verification, Alloy's built-in model finding engine is used to find satisfying instances and counterexamples. Any counterexample found indicates interoperability violations: *e.g.*, a client cannot generate a request native to its domain, or the gateway does not know what to do with a returned response.

5.6.1 Pull-based Retrieval: Request/Response

Static Content Retrieval.

In the static content retrieval (SCR) service, the request packets carry content IDs which the client requests, and the response packets produced by repositories (can be content producers or router caches) carry the data associated with that content ID. We describe two of SCR's essential content-oriented properties using Alloy (Listings 5.7 and 5.8).

Property 1.1. SCR Reachability: For every client that wants to retrieve content associated with a content ID and has a direct route to a gateway, there is a repository with content having that ID reachable from that gateway.

Property 1.2. SCR Returnability: For every client that reaches a repository with a request, there is a path back to the client for the response with the content.

Listing 5.7: SCR reachability property

```
pred reach[c:Client, cid:ContentID, re:Repos, gw:GW]{ -- reachability predicate
    all co: Connections | cid in c.want => -- if requested
       (some r:Route, con:Content | r.initiator = c && r.acceptor = gw &&
       r.contentID = cid && (cid->con) in re.map =>
           some r1,r2:Route | (r1->r2) in ^(co.connected) && r1.initiator = gw &&
           r2.acceptor = re && r1.contentID = cid && r2.contentID = cid &&
           r1.reposdomain in re.domains && r2.reposdomain in re.domains)}
assert reach{ -- reachability assertion
    all c:Client, cid:ContentID| some re:Repos, gw:GW | reach[c,cid,re,gw]}
```

Listing 5.8: SCR returnability property

```
pred return[c:Client, cid:ContentID, re:Repos, gw:GW]{ -- returnability predicate
    all co:Connections | some gw1:GW | reach[c,cid,re,gw1] => -- if reachable
      (some r,r1,r2:RevRoute | (r1->r2) in ^(co.revconnected) &&
      r1.initiator = re && r2.acceptor = gw &&
      r1.content = re.map[cid] && r2.content = re.map[cid] &&
      r.initiator = gw && r.acceptor = c && r.content = re.map[cid])}
assert return{ -- returnability assertion
    all c:Client, cid:ContentID, re:Repos | some gw:GW | return[c,cid,re,gw]}
```

Dynamic Content Retrieval.

In DCR, every request has to be mapped to a unique response, as opposed to SCR. To facilitate this, having a *demux* value (for multiplexing/demultiplexing) is essential for DCR, to provide the correct mapping of responses to requests; since every generated response is specific to not just the request's name, but also its input parameters. To access dynamic content from a server, a client generates a query for which the gateway keeps state as <nodeID, demux> of the requesting side and <demux> for the serving side. Reachability and returnability are still important in DCR (Properties 2.1–2.2). However, if the same SCR protocol is used for DCR, there can be *conflicts* between multiple requests, *e.g.*, a cached content may get sent back to multiple distinct clients. Therefore, we define no-conflict properties for DCR (Property 2.3).

Property 2.1-2.2. DCR Reachability and Returnability: These two properties are similar to those of SCR; with the difference being additional constraints regarding elements of DCR requests, *i.e.*, including generation and verification of the correct *demux* values at gateways (*i.e.*, in addition to contentID, *etc.*).

Property 2.3. No-conflict between distinct requests/clients: For every client that searches for two distinct content items (*no-conflict-A*, Listing 5.9), or a dynamic content requested by two different clients (*no-conflict-B*, Listing 5.10), two distinct, appropriately associated responses, should be received back. In *no-conflict-A*, the focus is on the distinction between two *return*-ed contents, associated with two distinct requests made by a given *Client* for distinct *Keywords* $k1$ and $k2$. On the other hand, *no-conflict-B* focuses on the distinction between two *return*-ed contents, associated with requests for a particular *Keyword* initiated by two distinct *Clients* $c1$ and $c2$.

This property shows the importance of having two separate *demux* values in packets, namely both the request ID (required for Property 2.3.a) and client ID (required for Property 2.3.b), to make each dynamic request globally unique, for correct multiplexing/demultiplexing. If we remove either of those two elements, this property will be violated and counterexamples will arise; *i.e.*, the gateway would not know how to demultiplex incoming response data to serve the correct, corresponding requesting client.

Listing 5.9: DCR - No conflict between 2 distinct requests from the same client

```
assert no-conflict-A{ -- Property 2.3.a
    all c:Client, disj k1,k2:Keyword | some s1,s2:Server, gw1,gw2:GW |
        return[c,k1,s1,gw1] && return[c,k2,s2,gw2] => some n1,n2: NodeID,
        d1,d2,d3,d4:Demux |
            (n1->d1->d2) in gw1.state && (n2->d3->d4) in gw2.state &&
            n1 in c.id && d1 in c.demux && d2 in gw1.demux &&
            n2 in c.id && d3 in c.demux && d4 in gw2.demux &&
            !(n1 = n2 && d1 = d3 && d2 = d4) && (some disj r1,r2:RevRoute |
                r1.initiator = gw1 && r1.acceptor = c && r1.contentID = s1.map[k1]
                && r1.demux = d1 && r2.initiator = gw2 && r2.acceptor = c
                && r2.contentID = s2.map[k2] && r2.demux = d3)}
```

Listing 5.10: DCR - No conflict between 2 identical requests from two distinct clients

```
assert no-conflict-B{ -- Property 2.3.b
    all c1,c2:Client, k:Keyword | some s1,s2:Server, gw1,gw2:GW |
        return[c1,k,s1,gw1] && return[c2,k,s2,gw2] => some n1,n2: NodeID,
        d1,d2,d3,d4:Demux |
            (n1->d1->d2) in gw1.state && (n2->d3->d4) in gw2.state &&
            n1 in c1.id && d1 in c1.demux && d2 in gw1.demux &&
            n2 in c2.id && d3 in c2.demux && d4 in gw2.demux &&
            !(n1 = n2 && d1 = d3 && d2 = d4) && (some disj r1,r2:RevRoute |
                r1.initiator = gw1 && r1.acceptor = c1 && r1.contentID = s1.map[k]
                && r1.demux = d1 && r2.initiator = gw2 && r2.acceptor = c2
                && r2.contentID = s2.map[k] && r2.demux = d3)}
```

5.6.2 Push-based Retrieval: Publish/Subscribe

In pub/sub, we have domain-specific multicast groups that are associated with prefixes [49]. We want a client to be able to subscribe to and receive all relevant publications in accordance with the prefix tree of the namespace over "chain" of groups across domains. Groups $G1$ and $G2$ form a chain if and only if the publisher of $G1$ can be a subscriber of $G2$, and is then able to relay data received from $G2$ to his subscribers in $G1$.

Property 3.1. Ability to subscribe to any prefix. For every client that wants to retrieve future publications under/associated with an existing prefix and has a direct route to a gateway, if there is some publisher that will publish content under that prefix, then that publisher is accessible through a chain of groups.

Property 3.2. Ability to receive any content published directly associated with the subscribed prefix. For every client who is subscribed to a prefix and can reach the associated publisher, there is a path back to the client to carry any content with a content ID belonging to that prefix. For example, a subscriber of $P2$ in Fig. 5.12 should receive publications pertaining to $P2$ across domains.

Property 3.3. Ability to receive all content published that is associated with prefixes under the subscribed prefix. This property says that for every client that has subscribed to a prefix and has reached the associated publisher, there is a path back to the client to carry any content with content ID either directly belonging to that prefix or under it in the hierarchy on the prefix tree. For example, a subscriber of $P2$ in Fig. 5.12 should receive publications pertaining to $P2$ *and also* $P4$ across domains. The assertion *rcvall* in Listing 5.11 depends on how relationships among groups and also between content IDs and prefixes are represented by *Connections* and *PTree*. For a domain with a namespace that does not capture relationships between prefixes, *i.e.*, does not map a prefix to a set of multiple relevant prefixes according to a graph, then *rcvall* would be equivalent to receiving a single content element (Property 3.2). Properties 3.1-3 collectively model and verify properties of a service offering hierarchical pub/sub.

Listing 5.11: Pub/Sub - receiving all relevant publications

```
assert rcvall{ -- all relevant publications in accordance with the prefix tree
    all pub:Publisher, con:Content, cid:ContentID |
        all co:Connections, pt:PTree | (cid->con) in pub.map =>
            ((some c:Client, p:Prefix | (p in c.want || (all p1:Prefix |
            (p1->p) in ^(pt.map) && p1 in c.want)) && cid.prefix = p =>
                (some r1,r2:Route | r1.initiator = pub && r2.acceptor = c &&
                (r1>r2) in ^(co.connected) && some g1,g2:Group |
                    g1.domain = pub.domain && g2.domain = c.domain &&
                    g1.prefix = p && g2.prefix = p && (g1->g2) in ^(co.revchain)))}
```

5.7 Reasoning about Failure and Mobility

In addition to the basic invariants, there are other important aspects of formal analysis of networks that warrant a more quantitative analysis; among them are failure and mobility analysis. Failures and mobility of nodes can occur in a network, causing disruption and lack of content availability. To better compare how different network architectural components, *e.g.*, routing, impact the number of success and violation scenarios, we perform model counting [80]. While we can consider the probability for all instances as being equal, we can also calculate each instance's probability by additionally factoring in the real-world probability of individual elements causing failures and mobility, provided as external information (*e.g.*, the probability of a gateway failing when processing a content request, a route disconnecting while carrying a packet, *etc.*). Thus, we can provide a more realistic probabilistic analysis for the effect of failures and mobility using weighted model counting methods [45].

While the Alloy Analyzer (v4.20) [4] allows for a limited, graphical iteration over instances, it does not enable an explicit counting of instances in an efficient manner. To perform model counting, we wrote an application [1] that counts all SAT solutions, using the SAT4J solver [118] (SAT4J can be replaced by any off-the-shelf SAT solver). We feed the Alloy model and properties, in Kodkod format [180], to our application. Predicates and assertions are used for counting instances that satisfy or violate (counterexamples) respectively. Through this counting, we can also look into the details (relations and values) within each instance, and gain insight such as possible cause of violations (in case of counterexamples) and calculate the probability of occurrence of each instance in real-world scenarios.

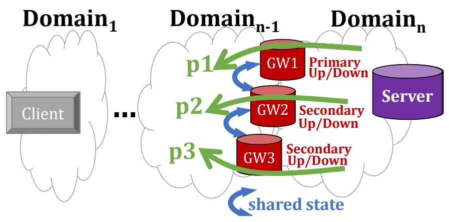


Figure 5.13: Gateway failure scenario

Table 5.1: Model finding

Domain n constraints	Returnability
Const. 1	X
Const. 2	X

Table 5.2: Model counting

Domain$_n$ constraints	Returnability		
	I	C	R
Const. 1	x_1	y_1	$x_1/(x_1+y_1)$
Const. 2	x_2	y_2	$x_2/(x_2+y_2)$

While we do not focus on the performance aspects of model counting in this paper, optimizations of this procedure can be leveraged for enhancing the scalability of our approach in case of very large problem sizes. At a minimum, our approach can provide a rough estimate of failure probabilities. Even if the model counting provided by the SAT solver is through "approximate" model counting (*e.g.*, using repetitive halving procedures) [43] rather than an "exact" one, it still gives us a good enough assessment of the degree of success and violation of properties.

5.7.1 Failure

Our interoperability framework depends on gateways that retain state information. What would happen to a response packet if that state is lost at the gateway for any reason? For reliability, we consider state sharing between redundant gateways that have the same domains on either side. Fig. 5.13 depicts an example for this. Consider the gateway that received the request and created the state as the *primary* gateway for the request ($GW1$ in the Fig.), and the replicas that have the shared state as the *secondary* gateways ($GW2$ and $GW3$). Formally, we add an extra condition to our reachability and returnability properties such that, for two routes to connect, the gateway attaching them must be up and running at the time the packet is received. Additionally, for returnability, the state information must be present at the gateway. If any gateway goes down, the corresponding potential path going through it ($p1$–3) back for the content cannot be leveraged. If the gateway is neighboring an NDN domain (*e.g.*, in $Domain_n$ or $Domain_{n-1}$), then the gateway has to be the primary only, for correct operation with the NDN reverse-path-forwarding (RPF) policy [203]. For other domain types, a secondary gateway that is active and has the shared state information is adequate to forward the response data back. We model the conditions representing this in Alloy as shown in Listing 5.12.

Listing 5.12: Failure scenario constraints

```
all r1,r2: Route, c:Connections | -- forward routes (request) condition
    (r1->r2) in c.connected <=> r1.acceptor = r2.initiator &&
    r1.initiator.status1 in Up && r2.initiator.status1 in Up
all r1,r2: RevRoute, c:Connections | -- reverse routes (response) condition
    (r1->r2) in c.connectedR <=> r1.acceptor = r2.initiator &&
    r1.initiator.status1 in Up && r2.initiator.status1 in Up &&
    ((r1.domain in NDNdomain || r2.domain in NDNdomain) =>
        r1.acceptor.type in Primary) -- NDNdomain enforces RPF policy
```

Gateways can go down due to various reasons such as completely failing or just losing state information due to a software failure. Our method can be used to reason about various scenarios and measure failure probability given an input configuration space, *i.e.*, a set of Alloy facts that set constraints on some objects or variables while relaxing others. As Table 5.1 shows, a simple model finding analysis does not provide a helpful comparison between different such constraints: it will say that both cases lead to counterexamples raised (*e.g.*, for the case that all gateways go down). To gain a better assessment of which constraint does better, we resort to model counting (Table 5.2). Using model counting, we can count (satisfying) instances (I) and counterexamples (C), and calculate (even if approximately [80]) the probability of reliability ($R = I/(I + C)$). This reliability indicates to what degree interoperability is impacted in presence of failure, given certain conditions (*i.e.*, choice of domain policies, *etc.*).

5.7.2 Mobility

To model and analyze mobility (Fig. 4.6), we add the notion of "time" to our model. In particular, we associate timeout values to state entries at gateways and $birthTime$ and $deathTime$ to routes (and similarly for reverse routes). We assume gateways are stationary, but other nodes can move, causing the "death" of their route ($route1$) to/from their closest gateway. A new route to the gateway is "born" ($route2$) after some time, assuming the existence of a domain-specific method to handle mobility. Temporal conditions must be incorporated into reachability/returnability properties. The most critical case is when a mobility event occurs while the packet is in-flight [205]. At high-level, the sum total latency formulated as $firstDeliveryAttempt + recovery + secondDeliveryAttempt$, must be below a certain $expiration$ threshold (at every gateway and consumer). $firstDeliveryAttempt$ is the incomplete partial delivery latency via $route1$ and $secondDeliveryAttempt$ is the delivery via $route2$ (continuation in MF, and complete retransmission in IP and NDN). The $recovery$ delay is the time it takes for the packet to be transmitted back on the new path again; it includes re-registration (MF and IP), FIB re-population (IP and NDN in case of provider mobility) and/or PIT re-population (for NDN in the case of consumer mobility) delays [160, 203, 205]. Using this formal method, we check properties in the presence of mobility, find appropriate values for a timeout threshold on gateways and investigate the effect of domain-specific mobility handling methods on interoperability. Listing 5.13 generally specifies how the reachability property (to deliver a named request) depends on the condition of mobility (stationary or mobile) and the domain policy on handling mobility (early binding or late binding). Returnability is similarly specified (for content). Predicates *stationary*, *mobileEarlyBinding*, and *mobileLateBinding* specify timing conditions for successful delivery assuming their corresponding conditions (details of the three properties are omitted here due to space but are in [1]). As shown in Fig. 4.6, we only consider intra-domain mobility here, *i.e.*, the mobile node changes its location and point of attachment, but stays within its domain.

Listing 5.13: Reachability in presence of mobility

```
pred reach[c:Client, p:Producer, cid:ContentID]{ -- a client and content producer
   (stationary[c,p,cid] && p.mobility in Stationary) -- producer p stationary
  || (mobileLateBinding[c,p,cid] && p.mobility in Mobility
      && Domain.binding in LateBinding) -- p mobile, domain does late binding
  || (mobileEarlyBinding[c,p,cid] && p.mobility in Mobility
      && Domain.binding in EarlyBinding)} -- p mobile, domain does early binding
```

5.8 Formal Analysis Results

We implemented the ICI framework discussed in our model, with gateways for interoperation among IP, NDN, and MF (Fig. 5.10 as an example) in a software testbed (implementation details in §5.3). This section provides the description and results of our analysis of the ICI framework (our Alloy source code is approximately 800 lines of code in total [1]).

To check for correctness, we performed verification (supported by Alloy Analyzer's model finding engine) of our ICI framework model, against the information-centric services properties. In order to reach convincing proofs (as advised in [200]), we pick the scopes for verification in Alloy that are large enough to contain all necessary cases (*i.e.*, minimum number of actors and objects for each service), and small enough so that we do not en-counter model explosion. The scopes, *i.e.*, upper bounds on the number of key objects, are provided in Table 5.3. For most properties, we consider 1 *Client*, 1 *Server*, 1 *Content*, and 1 *ContentID*. That is, different <client, request> pairs are considered independent of each other. However, for Properties 2.3.a/b, such a dependency matters, and we want to show lack of conflicts. For Property 2.3.a, we set 1 *Client* and 2 *Content*s (to generate scenarios where *one* client makes *two* separate request for *two* different contents), and for Property 2.3.b, we set 2 *Client*s and 1 *Content* (to look for conflicts between request for *one* content but by *two* clients). We use 3 *Domain*s for most properties, as it contains all cases with 1, 2, or 3 domains of any type, *i.e.*, IP, NDN, or MF. Also, with upper bound n on the total number of *Node*s, *i.e.*, sum of *Client*s, *Server*s, and *GW*s, we specify the upper bound on the number of *Route*s (as well as *RevRoute*s) to be $n(n-1)$, enabling the existence of any possible (uni-directional) route. For pub/sub services (*i.e.*, Properties 3.1–3), we set 3 *Prefix*es, *ContentID*s, and *Content*s, to capture inter-relationship of content IDs in a large enough namespace. Additionally, with the upper bound on *Domain*s and *ContentID*s both set at 3, we set the upper bound on total number of *Group*s (and *GroupID*s) to be 3×3=9, so as to contain cases with one group per content ID per domain. The blank cells in Table 5.3 indicate either "N/A" or "no particular upper bound set", in which case Alloy picks a default value. Within this scope, our verification passes successfully for each property, showing that the stated properties are *invariants* of our ICI framework. In other words, the framework design ensures that *any sequence of interconnected IP, NDN, and MF domains are information-centrically interoperable.*

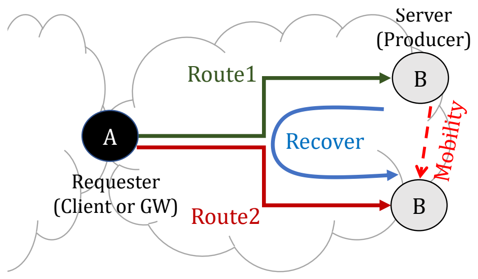


Figure 5.14: Mobility scenario example: Route 2 established after B moves and changes its point of attachment

We use our proposed model counting approach to analyze scenarios with the failure of one or multiple gateways. The most important factor affecting returnability in scenarios with the possibility of failure, is domain-specific routing policies, in particular, whether or not it allows for a secondary (backup) gateway to relay the returning response content. Different domains have different policies; MF and IP decouple the forward (request) and return (response) paths, and they can be delivered through different gateways, while NDN strictly requires the two paths to be the same, due to RPF policy. To investigate the impact of that policy, we considered a scenario of two domains, with two gateways between them (one primary and one secondary), sharing state. Both gateways are *Up* (working) when the request is forwarded, and either *may* go *Down* (failing) when the response is one its way back. Table 5.4 shows different scenarios for reachability and returnability, with different domain constraints (with different routing policies). In particular, the two domain constraints we consider are the following: 1) no constraint on what any of the domains are; and 2) one domain is definitely NDN. The table shows the values of I (instances), C (counterexample), and R (reliability) for each scenario. Our results for R in Table 5.4 prove that having an NDN domain on one side dramatically reduces the returnability

Table 5.3: Verif. scopes for ICI services properties

Property	Client	GW	Repos/ Server/ Publisher	Domain	ContentID/ Keyword	Prefix	PTree	Content	Connections	Route	RevRoute	NodeID	Port	NDNreqID	MFreqID	Group	GroupID	**Verif. Result**
1.1	1	2	1	3	1			1	1	12	12	6						✓
1.2	1	2	1	3	1			1	1	12	12	6						✓
2.1	1	2	1	3	1			1	1	12	12	6	6	3	3			✓
2.2	1	2	1	3	1			1	1	12	12	6	6	3	3			✓
2.3.a	1	1	1	2	2			2	1	8	8	4	4	4	4			✓
2.3.b	2	1	1	2	1			1	1	8	8	5	4	4	4			✓
3.1	1	2	1	3	3	3	1	3	1	12	12					9	9	✓
3.2	1	2	1	3	3	3	1	3	1	12	12					9	9	✓
3.3	1	2	1	3	3	3	1	3	1	12	12					9	9	✓

reliability ratio, since basic NDN forwarding strictly forbids data coming back on a different path than the original path taken by the request.

When a content producer (server) moves while a content request is in-flight (Fig. 5.14), the domain's handling of mobility recovery determines the reachability probability. NDN and IP use early binding with retransmissions, while MF supports late binding with rerouting. We compare the impact of these mechanisms and techniques using our model counting method, with results shown in Table 5.5. Our modeled scenario consists of two nodes in a domain, one requester (client or gateway) and one server (producer) with a route established among them. The 'Stationary' columns in the table show reachability results in the stationary server case. With 'Mobile', the route dies due to a server mobility event (at time t=10), leading to the birth of the second route. We set the re-registration and

Table 5.4: Failure analysis results

Cases	Reachability			Returnability		
	I	C	R	I	C	R
No domain constraints	290	0	1.00	56	210	0.21
One NDN domain	176	0	1.00	8	168	0.04

Table 5.5: Mobility analysis results

Cases	**Stationary**			**Mobile**					
				Late Binding			**Early Binding**		
DL range	I	C	R	I	C	R	I	C	R
[0,20]	100	8	0.92	72	24	0.75	92	64	0.58
[0,18]	96	0	1.00	72	8	0.90	92	48	0.65
[0,15]	84	0	1.00	64	0	1.00	92	24	0.79
[0,10]	64	0	1.00	44	0	1.00	84	0	1.00

re-population delays to 1 each. Also, a retransmission is initiated 1 time unit after the mobility event. Different binding techniques for mobility, *i.e.*, late and early binding, are also shown in Table 5.5. We compare cases with different ranges for *Delivery Latency* (DL), which is time approximately needed for a packet to travel from requester to server. For a delivery latency range of $[0, 20]$, we see a higher R for stationary *vs.* mobility cases. The reason is that when the server does not move, the original route stays active, thus providing a higher chance for requests to reach the server. Comparing the two binding techniques, late binding leads to higher chance of reachability compared to early binding, as it allows for packets to be re-routed on the newly-born route, rather than retransmitting from the original requester. These results serve as proof that under similar scenarios, late binding outperforms early binding in ICI. Also, changing the delivery latency ranges, we can find out at what points, reachability is an invariant (if ever) under mobility conditions. As the table shows, with ranges within $[0, 18]$, $[0, 15]$, and $[0, 10]$ (rows in Table 5.5 labeled in first column accordingly), reachability becomes an invariant in cases of Stationary, Late Binding, and Early Binding, respectively; as zero counterexamples are raised. With a small enough delivery latency ranges, namely $[0, 10]$, reachability becomes an invariant, no matter the mobility conditions or binding techniques. Our approach can be used to find such points of invariance, comparing different techniques, and prove them.

Chapter 6

POISE: Graph-based Namespaces and Load Sharing for Efficient Information Dissemination

6.1 Introduction

This chapter presents POISE, designing dynamic graph-based namespaces for in-network name-based pub/sub, with the introduction of an information layer to manage it. Although POISE supports both topic-based and recipient-based pub/sub, our particular focus here is on recipient-based pub/sub. Additionally, we propose a rendezvous point (RP)-based pub/sub protocol, with the dissemination logic following the graph-based namespaces, to deliver all relevant information to their intended/required recipients (mainly first responders) in a timely manner using push-based multicast. POISE handles possible cycles in the graph through preventive DFS-based cycle detection in the graph, as well as data plane nonce-based loop detection [19]. We share the workload among multiple RPs, where each RP is responsible for managing a subset of the namespace graph and functions as the core of the subscription trees associated with those names. Often in many real-world scenarios, the workload-per-RP distribution is non-uniform and difficult to predict. In an RP-based pub/sub, this could cause an excessive load on one RP managing names corresponding to those more intense workloads, thus making it a "hot spot"; an example of this may be in a multi-player online gaming environment, as in [47]. While it is practically infeasible to optimally balance the load across the whole network (due to the amount of frequent periodic communication needed which is especially difficult with large and/or bandwidth-limited networks), we eliminate the traffic concentration by automatically splitting a congested RP's (*i.e.*, hot spot's) workload: the RP partitions its namespace (sub-)graph with the objective of finding two balanced segments while minimizing inter-RP communication, decides which names to relinquish, and triggers the migration of subscription tree cores related to those names to a new RP or an existing under-loaded RP. Our graph partitioning problem involves calculation of weights for vertices and edges. However, due to the nature of our partitioning formulation, these weights depend on the cut itself; thus making our objective function a "complex" one [157]. As a result, off-the-shelf graph partitioners such as the popular tool METIS [103] (as well as its parallelized version, ParMETIS [105]) fall short. To overcome this, we propose a hybrid splitting procedure consisting of a heuristic (METIS) and meta-heuristic guided search refinements (using Tabu Search [194]). Our results show the effectiveness of our design. While we consider a number of example use cases of POISE in different environments, we primarily focus on the application of POISE to information dissemination among first responders in disaster scenarios, an application that requires timely information delivery.

The key contributions of POISE are the following:

1. A recipient-based pub/sub framework with automatic load splitting for efficient information dissemination.

2. Support for free-form, *i.e.*, not limited to a particular structure such as hierarchy, graph-based namespaces. Prior to describing POISE, we provide a brief example of name-based pub/sub with hierarchical namespaces in §6.2. Additionally, POISE proposes an information layer to capture rich information organization structures; our simulation results show that this is more efficient than using state-of-the-art hierarchical namespaces.

3. An automatic name-based and workload-driven, novel hybrid graph partitioning procedure and load splitting along with a seamless and lossless core migration mechanism; our results show the effectiveness and correctness of our core migration, and improved quality and resulting network efficiency of our partitioning procedure, compared to popular off-the-shelf graph partitioning tools. We further demonstrate the benefit of using our Tabu search-based refinement compared to an iterative implementation of METIS, as well as ParMETIS with Adaptive Repartitioning.

4. An implementation of a POISE RP including its graph-related operations on a DPDK-based platform; our micro-benchmarking shows that the overhead of graph-based operations justifies using our RP-based solution rather than name-expansion at every hop.

6.2 Using ICN for Timely Vehicular Safety Information Dissemination

In this chapter, we briefly overview and example of an information-centric pub/sub framework using hierarchical namespaces. Vehicles are increasingly equipped with special purpose sensors and Global Positioning System (GPS) for use in safety applications. Beyond using these sensors, sharing information among vehicles can substantially improve the safety of the overall transportation environment. Enabling each vehicle to get the "right information at the right time", to avoid potentially dangerous situations can be valuable. Information-Centric Networks (ICN) that uses the notion of "named-object" enable information retrieval and delivery regardless of its location, publisher or requester. Using ICN, especially supporting publish/subscribe can provide timely delivery of relevant vehicular safety information.

Our V-ICE architecture (Fig. 6.1) utilizes Roadside Units (RSUs) to act as infrastructure-based information aggregators to communicate with vehicles that generate notifications of safety-related information. RSUs also disseminate this information to the right vehicles who subscribe to the information relevant to the path they are traveling on.

A use case that we used to evaluate the benefit of V-ICE, is its use in propagating "black ice" warnings to vehicles that will likely be affected by the black ice event on their routes. The critical need is for the information to be delivered in a timely manner, compared to a server-based infrastructure. This provides other cars sufficient time to react.

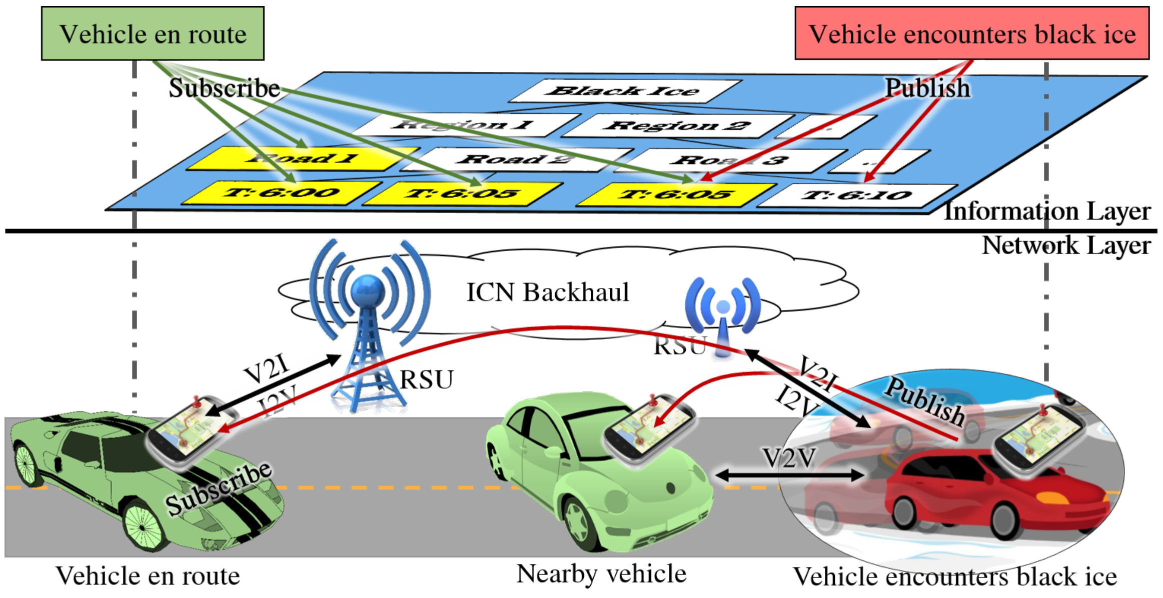


Figure 6.1: Overall architecture of V-ICE

We build V-ICE's namespace (Fig. 6.2) and architecture based on a representative city environment, using the roadways of Luxembourg as an example, and evaluate our approach using a 4-hour traffic trace generated by the SUMO synthetic traffic generator. The work shows that V-ICE scales and performs better than a server-based approach or V2V broadcast, in terms of timeliness, relevance, and network traffic.

A key component of V-ICE's architecture is the name space that enables identification of the geographical location of the black ice events, along with a time interval for the event. The name space is organized as a hierarchy, to allow for aggregation of events that occur in a wider region. Subscriptions from vehicles (possibly an application such as a route planner with a GPS) will be for a number of names, corresponding to the road segments the vehicle will travel, along with the approximate time period it expects to be on that segment. The application can update or generate new subscriptions as the vehicle

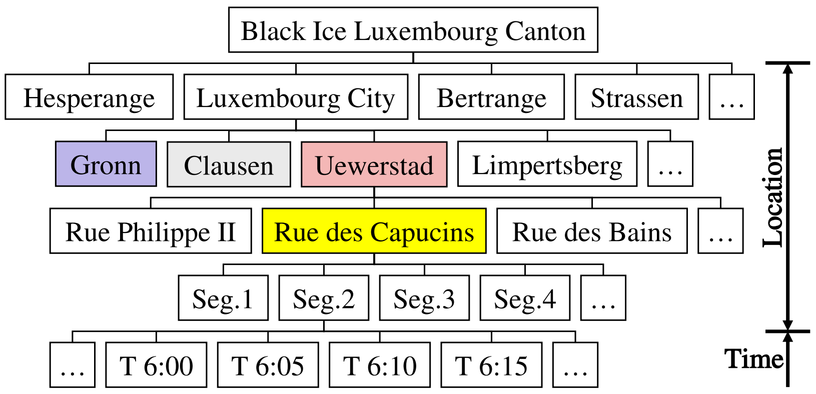


Figure 6.2: V-ICE namespace example

travels, based on the route, current position and speed. With the rich naming framework that supports hierarchies, subscriptions can use an aggregated name. Thus, a black ice event on any road segment within a region will result in that notification being delivered to a vehicle subscribed to events in the region. This will assist in re-routing the vehicle without selecting any road segments with black ice in that region. More details on the design and experiments on V-ICE are provided in [].

6.3 Use Cases of POISE

POISE is applicable in a variety of different contexts, where a pub/sub communication model is needed. Its biggest benefit is in cases where the dissemination is done according to a complex, multi-dimensional namespace, and timely delivery of relevant information to all intended recipients is required. In this section, we illustrate how POISE can specifically help with managing sample namespaces for several use cases.

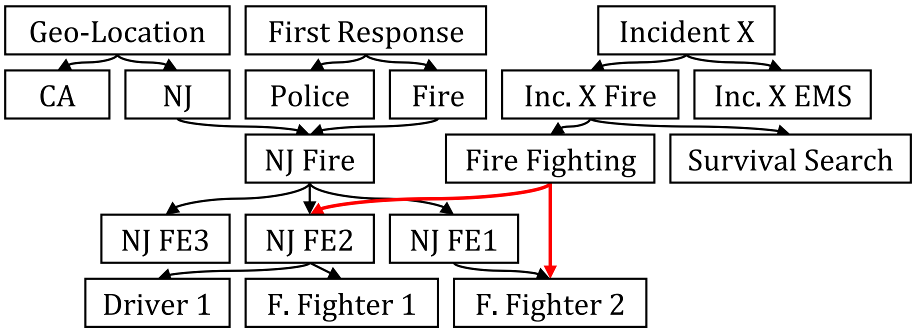


Figure 6.3: Graph-based namespace: incident command chain example

6.3.1 Disaster Management

During disasters, a large amount of information is disseminated, and it involves distributing it to many participants such as incident commanders, first responders, volunteers and civilians. ICN and name-based pub/sub have been shown to be very suitable and beneficial for disaster scenarios [48, 177]. Furthermore, due to the complex nature of today's command chains of first responders, a graph-based naming framework (an example disaster management namespace is shown in Fig. 6.3, is needed to represent the multiple reporting hierarchies. Furthermore, in an RP-based framework for name-based pub/sub for disaster management, overload and hot spots are highly likely to occur. In a given disaster, particular roles (*i.e.*, names) may receive much higher demands than other roles (*e.g.*, 'firefighting' roles in a wildfire incident). This motivates our RP splitting and graph partitioning to eliminate traffic concentration. Furthermore, a major problem often caused by disasters (especially natural disasters), is that the network and servers can experience excessive load and congestion, making many services unavailable [48]. Our work addresses this by creating an efficient information organization and load sharing framework that dramatically reduces network load during disasters. Works such as [90,139] have been proposed

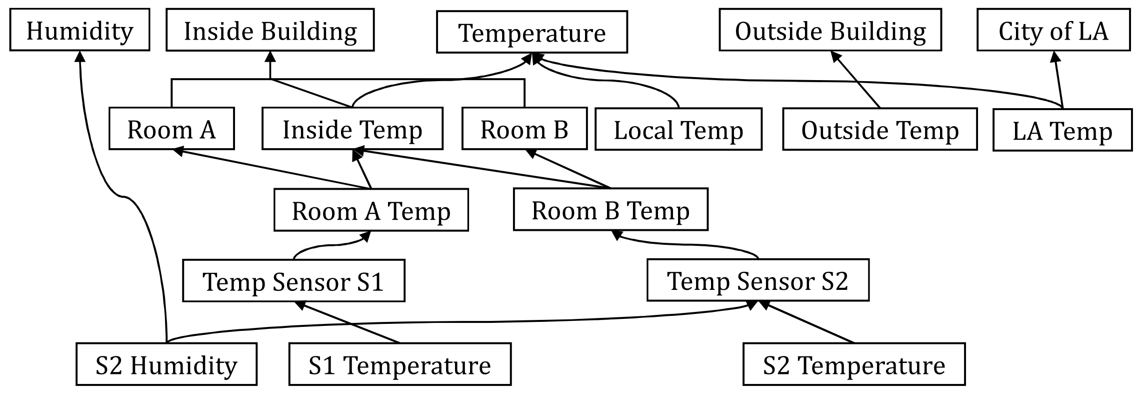


Figure 6.4: Example namespace for IoT HVAC system

to leverage delay/disruption-tolerant routing with ICN in disasters in case of complete or partial infrastructure failures. These works, while orthogonal to ours, can be leveraged by POISE. POISE's information layer can run on top of such delay-tolerant protocols in disconnected environments.

6.3.2 HVAC System: A Smart Building with IoT

The use of Internet of Things (IoT) for building smart cities, buildings and homes is becoming increasingly more popular. The interactions between different elements in an IoT environment, *i.e.*, sensors, actuators, *etc.*, can be complex. Fig. 6.4 shows a small (partial) namespace for a HVAC (Heating, Ventilation, and Air Conditioning) system in a building as an example use case for POISE. Information can have multiple dimensions: type of information (*e.g.*, temperature or humidity), location of the information (*e.g.*, inside the building or room A), *etc.* The naming schema used in current name-based architectures for IoT (such as NDN RIOT [82]) use strictly hierarchical structures, which falls short in efficiently capturing such complex multi-dimensional structures. Using POISE's graph-based

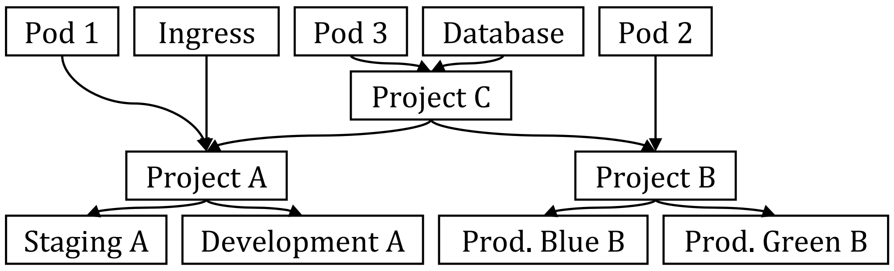


Figure 6.5: Example namespace for containerized platforms

namespace (such as in Fig. 6.4), we can support a publish/subscribe (topic-based) capability to support many-to-many delivery from sources (sensors) to destinations (actuators). Sensors publish to names; a new publication can be generated periodically, or if there is a substantial update to report. All subscribers of that name and all the names reachable from it will receive it. Different actuators (*e.g.*, heating or cooling) can choose different granularity levels based on their settings, subscribing to "`Temperature`" (to get every temperature reading from any source: its own sensor, inside building, outside building, and externally from city of LA news report), subscribing to "`Room B`" (to get any information in Room B: temperature, humidity, *etc.*). By subscribing to "`Inside Temperature`", an actuator will receive all temperature readings recorded in Room A or B, whenever they are published.

6.3.3 Resource Management in Clouds

The use of virtualized container environments for cloud services have become popular in the past decade. In large-scale systems, they can involve the interaction across many users and resources, warranting a scalable data structure and communication system of managing their orchestration. Kubernetes [12], as a prominent example, defines namespaces for resource orchestration. With POISE, we can have a generalized graph-based

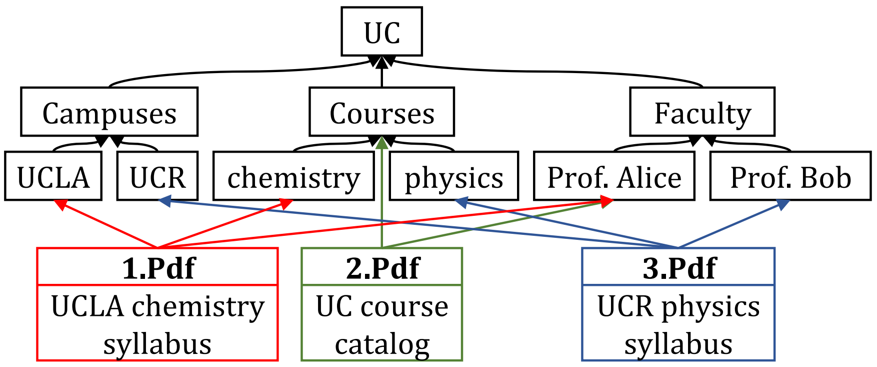
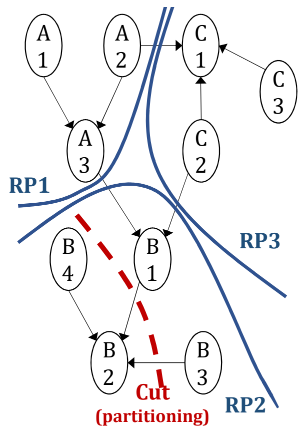


Figure 6.6: Example namespace for distributed replica management

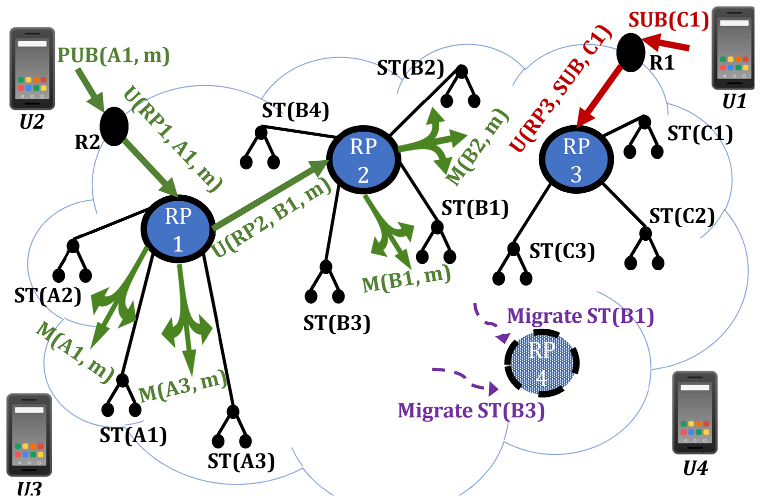
namespace (with nesting) that captures all the different resources and components such as clusters, projects, and libraries, while allowing a systematic push-based pub/sub notification of changes to relevant users. Fig. 6.5 shows a (partial) POISE namespace as an example for such environment. The edge directions denote the flow of information. For example, upon any change in "`Database`", an immediate notification will be sent to all subscribers of "`Development A`" (as well as other ancestors of "`Database`"), which are members of a team working on the development of "`Project A`", and using the "`Database`" cluster. This is especially helpful at an enterprise-level with a large complex namespace. The authorization and access can be captured with the directed edges. Any publication made to a name can be generated by any user that has the authorization to make changes (*e.g.*, modification of data or code) to the resource(s) associated with that name. The POISE namespace captures a flexible integration of isolation and sharing; *e.g.*, in Fig. 6.5, "`Pod 1`" is hidden from anyone not associated with "`Project A`", while "`Pod 3`" is visible to members of both "`Project A`" and "`Project B`".

6.3.4 Data Replica Management in Distributed File Systems

Distributed file systems are widely used in cloud systems, where each server in the datacenter hosts files belonging to parts of the overall directory. Work in [208] shows the benefit of using ICN for distributed file systems in a datacenter. A graph-based namespace for the directory system can be more efficient than just a hierarchical tree structure. Fig. 6.6 shows a small example of such a namespace for a university system's distributed file system. A strictly hierarchical version of the namespace graph would lead to a large number of duplicate nodes; for example the file "`1.pdf`" would have at least three name hierarchies, one for "`/UC/Campuses/UCLA/1.pdf/`", one for "`/UC/Courses/chemistry/1.pdf`", and one for "`/UC/Faculty/Prof.Alice/1.pdf`'. For scalability and reliability, distributed file systems typically require replicas of files in different data servers. For example, the Hadoop Distributed File System (HDFS) [172] requires exactly 3 replicas of a given file chunk (in default mode). As files can be modified, consistency of data among replicas becomes an issue. POISE can make the management of data replicas quite convenient and efficient. For example, server D1 hosting all the files under the courses category, will subscribe to "`Courses`". Server D2 hosting all the physics files, will subscribe to "`physics`" (which is under "`Courses`"). Any modifications to a file belonging to physics, *e.g.*, file "`3.pdf`" will be published to the name "`physics`", thus both D1 and D2 will receive the notification, because the published update will be propagated along the name paths in the namespace, and will apply the changes to their stored replica files (or caches). This will provide flexibility for replica management, enabling multiple ways for different replicas' sub-directories to overlap.



(a) Namespace structure



(b) Information dissemination architecture

Figure 6.7: A schematic overview of the architecture of POISE

6.4 Overview

An overview of POISE's architecture is shown schematically in Fig. 6.7. POISE's namespace supports free-form graph structures, as shown in Fig. 6.7(a), rather than being restricted to the state-of-the-art hierarchical namespaces [100, 169]. This enhancement is possible through decoupling of *information layer* (which manages names and their relations in their natural form, supporting complex graphs) and the *service layer* (which manages the names used for name-based forwarding at every ICN router), which are coupled together in current Named Data Networks [203]. Each vertex in the graph in Fig. 6.7(a) is a name, *i.e.*, a role or attribute, and the edges show relations among them. POISE's information dissemination framework is a name-based pub/sub [48] with the support of name-oriented core-based multicast, with *rendezvous points* (RPs) being the cores for groups; each name also identifies a multicast group. In addition to being the core for the multicast tree (similar to traditional PIM-SM [72], NDN COPSS [49], or MF multicast [144]), POISE's RPs operate at the information layer; in other words, they are information-layer-enhanced RPs. As shown in Fig. 6.7(a) and Fig. 6.7(b), the namespace is shared among three RPs (*i.e.*, RP1, RP2, and RP3), each RP managing the sub-graph it is responsible for, and maintaining the subscription trees associated with each of the names (groups) it is hosting; *e.g.*, RP1 is the core for the subscription tree (ST) for $A1$, $A2$, and $A3$. A *name-to-RP* mapping resolution service resolves a name in the namespace to the RP it is hosting; *e.g.*, the name $C1$ would be mapped to RP3.

The subscription path is shown in Fig. 6.7(b) (in red); user $U1$ wants to subscribe to $C1$ (which implicitly means subscribing to all ancestors of $C1$ in the namespace as well). $U1$ sends this request as $SUB(C1)$, without the need to know which is the associated RP or where it resides. $U1$'s first hop router R1 performs the resolution and relays the request as a unicast (U) message to the correct RP, *i.e.*, RP3; thus, $U1$ *joins* $ST(C1)$. The publication path is also shown (in green); $U2$ wants to publish message m to all subscribers of name $A1$ (which implicitly means publishing to subscribers of all descendants of $A1$ as well). $U2$'s first hop router relays this publication as a unicast message to its corresponding RP, namely RP1. Expanding $A1$ to its descendants (*i.e.*, name expansion), it sends m as a multicast (M) downstream to $ST(A1)$ as well as $ST(A2)$. Additionally, RP1 recognizes that there is an edge leading from $A1$ towards a name outside RP1. Thus, RP1 sends a unicast message for this name, $B1$ to its RP, RP2. Note that RP1 only has visibility of namespace up until $B1$ and not further. Performing a similar name expansion, RP2 processes the received request by going through its namespace, which leads to multicasting m downstream along

$ST(B1)$ and $ST(B2)$. Thus, users for both subscription and publication scenarios need only send *one packet*, destined to only *one name*; the network takes care of expanding the packet to additional names, if needed.

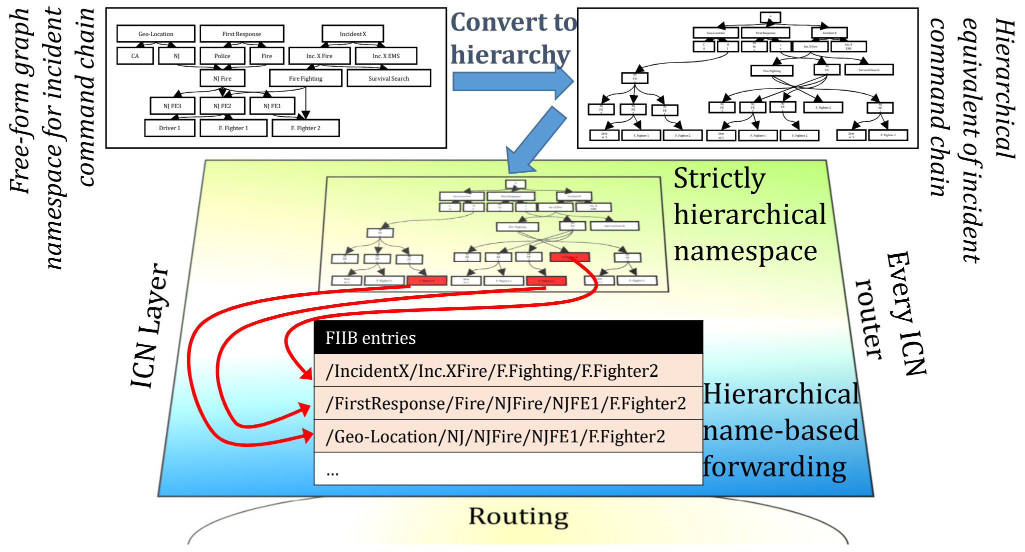
Each RP's workload has a correlation with the part of the namespace it is managing. However, additionally, the load-per-name distribution is likely to be non-uniform, and hard to predict. To address this, another important feature of POISE, *automatic load splitting* is performed to eliminate traffic concentration. Consider the case when RP2 encounters a large amount of workload exceeding its threshold, thus making it a *hot spot*. Triggered by this, RP2 will perform a *partitioning* procedure on its own sub-graph, to provide two balanced *segments*, shown as *Cut* in Fig. 6.7(a). It thus decides to keep $B2$ and $B4$, and relinquish $B1$ and $B3$ to another RP (*e.g.*, RP4), which can be a regular ICN router configured to be a new RP for this environment. As a result, the subscription trees for $B1$ and $B3$ will be migrated to RP4, via a *core migration* procedure. The name-to-RP mapping will then be updated accordingly.

6.5 Architecture and Design

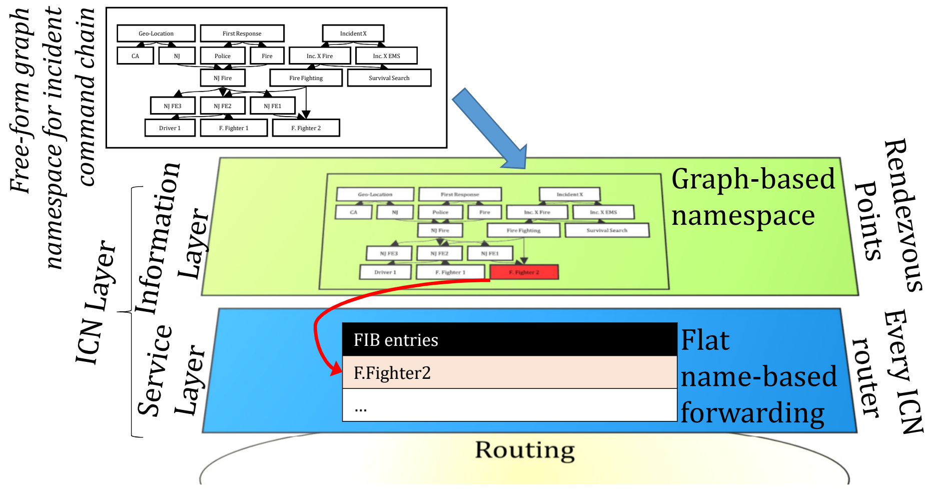
6.5.1 Information Layer and Graph Namespace

POISE supports free-form graph namespaces with their natural structure for in-network information-centric dissemination, without the need to restrict them to any particular data structure, such as a hierarchy or prefix tree as NDN [203], or NDN-based solutions such as CNS [48] do. Using the disaster management use case as example, let us consider Fig. 6.3, a simple namespace of an incident management command structure. As we can see, it does not follow a strict hierarchy. The incident management structure is a directed graph, with each node in the graph being a name that denotes a role. The higher levels denote coarser granularity (*e.g.*, "`Fire`" is a broad organizational role for everyone having something to do with fire-related issues), while the lower levels denote finer granularity (*e.g.*, "`NJ FE1`" denotes a specific fire engine dealing with a fire-related task in New Jersey). Edges in the graph represent name relationships and the direction of the edges show the flow of control in the command chain; *e.g.*, "`NJ FE1`" (NJ fire engine 1) is a higher-level authority than "`F.Fighter2`" (fire fighter 2). This representation of the namespace captures the different dimensions in *one* graph, *i.e.*, time, region, department, *etc.* Modification of the namespace is achieved through addition, modification and/or deletion of nodes and/or edges in the graph. In the namespace graph shown in Fig. 6.3, two sub-namespaces, one organizational, on the left-hand side, and one incident-specific, on the right-hand side, are connected through the two edges shown in red. These two red edges represent the fact that the incident commander has dispatched "`NJ FE2`" and "`F.Fighter2`" roles to take care of Incident X's "`FireFighting`" tasks, shortly after it occurred.

Support for graph namespaces in information dissemination is made possible in POISE through a decoupling in the ICN layer and the introduction of information layer. Fig. 6.8 shows the two designs with regard to the ICN layering. In current name-based network architectures, in particular NDN, the information layer and service layer functionalities are coupled together in the ICN layer, supported at every ICN router, as shown in Fig. 6.8(a). This restricts the in-network namespace to one particular structure. This means every command structure of the organizations (*e.g.*, namespace in Fig. 6.3) needs to


Free-form graph namespace for incident command chain
Convert to hierarchy
Hierarchical equivalent of incident command chain
Strictly hierarchical namespace
ICN Layer
Every ICN router
FIIB entries
/IncidentX/Inc.XFire/F.Fighting/F.Fighter2
/FirstResponse/Fire/NJFire/NJFE1/F.Fighter2
/Geo-Location/NJ/NJFire/NJFE1/F.Fighter2
...
Hierarchical name-based forwarding
Routing

(a) State-of-the-art NDN



(b) POISE's design

Figure 6.8: Design choices for ICN layer

be converted to a strict hierarchy. As can be seen, this conversion makes the namespace unnecessarily larger and more complex to manage and use; *i.e.*, the "`F.Fighter2`" node in Fig. 6.3 will appear 3 times in the hierarchical equivalent on each reachable path, and will populate 3 entries in the FIB tables, with a separate entry for each (in this case, 3) different path leading from any root to node "`F.Fighter2`". The FIB table provides the name-based forwarding functionality leveraging the lower layer routing mechanism used for navigating the packet to its relevant locations. In contrast, in POISE (Fig. 6.8(b)), we decouple information and service layer functionalities of the ICN layer. Only RPs (designated ICN routers that perform name expansion) need to understand and maintain name relationships in the graph. As can be seen by comparing Fig. 6.8(a) and Fig. 6.8(b), our design choice in POISE (Fig. 6.8(b)) brings a number of significant benefits: it makes the ICN-layer namespace simpler and smaller, leads to smaller FIB tables, and eliminates redundant messages used for subscription and publication. This design choice in POISE (Fig. 6.8(b)) brings a number of great benefits compared to the state-of-the-art solution (Fig. 6.8(a)):

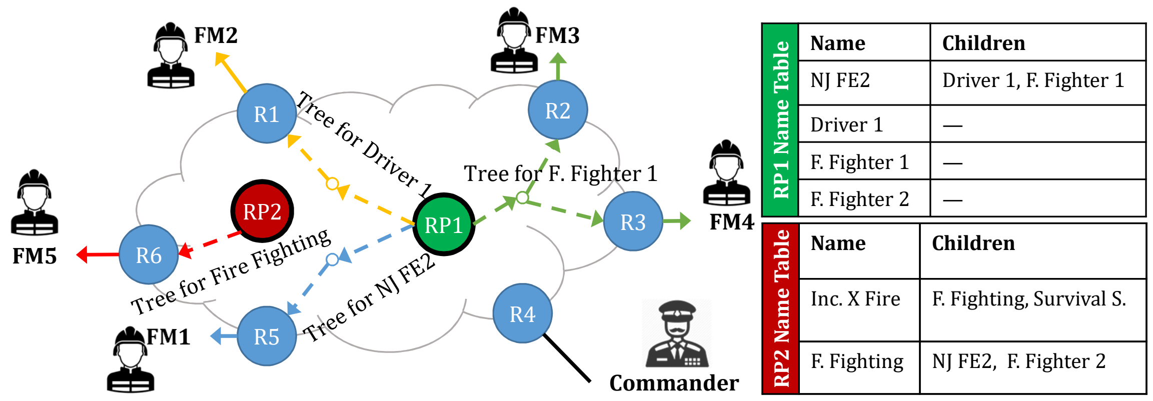
1) The ICN-layer namespace in Fig. 6.8(b) is more natural, and thus simpler and smaller than its converted equivalent in Fig. 6.8(a). Each name appears once rather than multiple times. This makes managing the namespace more efficient and scalable, as well as its modification less costly, which is important in the highly dynamic disaster scenario.

2) Complicated role relationships, which is prevalent in disaster management, leads to larger FIB tables in Fig. 6.8(a) compared to Fig. 6.8(b). As seen in the Fig., "`F.Fighter2`" appears 3 times in Fig. 6.8(a)'s FIB tables *vs.* once in Fig. 6.8(b)'s. Thus, POISE consumes less ICN router memory.

3) The above difference also leads to higher number of subscription and publication messages in Fig. 6.8(a), when it comes to recipient-based pub/sub, which we use here. Subscribing (publishing) to role "`F.Fighter2`" would result in 3 subscription (publication) messages in NDN's hierarchical naming framework as seen in Fig. 6.8(a), while it leads to only one message in Fig. 6.8(b). This makes POISE more efficient in a disaster scenario for pub/sub as it reduces the number of messages, thus reducing the network/user load.

6.5.2 Recipient-based Pub/Sub

POISE enables recipient-based pub/sub [48], enhanced with an information layer supporting graph-based namespaces. In this name-based pub/sub, subscribing to a name means implicitly also subscribing to a set of names related to that specific name, in accordance with the namespace. POISE's pub/sub logic follows the command chain graph-based namespace. In the case of disaster management, first responders and volunteers subscribe to (listen to) names, and civilians and incident commanders publish to names. Given the namespace in Fig. 6.3, subscribing to "`F.Fighter2`", implicitly means also subscribing to all of its ancestors, *i.e.*, "`NJ FE1`", "`FireFighting`", "`NJ Fire`", *etc.* Conversely, publishing to "`NJ Fire`", implicitly means also publishing to all of its descendants, *i.e.*, "`NJ FE1`", "`NJ FE2`", "`F.Fighter2`", *etc.* Expanding a name to all of its descendants on the publication path according to the namespace graph, is performed by the RPs, in a load-shared way. This design is beneficial where dynamically-formed interacting groups and individuals involved need to be notified with messages relevant to their tasks in a timely manner, whenever they are published or available, making sure maximum coverage and accuracy is achieved.



RP1 Name Table

Name	Children
NJ FE2	Driver 1, F. Fighter 1
Driver 1	—
F. Fighter 1	—
F. Fighter 2	—

RP2 Name Table

Name	Children
Inc. X Fire	F. Fighting, Survival S.
F. Fighting	NJ FE2, F. Fighter 2

Figure 6.9: Example network topology: 5 Firemen subscribe to different roles in the namespace in Fig. 6.3 and 2 RPs share the workload

To demonstrate the protocol exchange for information dissemination in POISE, we use a small example: consider the namespace graph in Fig. 6.3, and the topology in Fig. 6.9, where we have 5 firemen (FM1—FM5), and one commander, all connected to the network via routers R1-–R6. These firemen subscribe to different names: FM1→“`NJ FE2`”, FM2→“`Driver1`”, FM3 and FM4→“`F.Fighter1`”, and FM5→“`FireFighting`”. There are two RPs, with each of their name tables shown (partially) in Fig. 6.9. The commander wishes to send a message to the name “`FireFighting`”, for it to be received by all relevant firemen. When a publication is sent to “`FireFighting`”, the message will be sent to the RP serving it, namely RP2. RP2 searches its name table to find out the reachable subgraph visible to it under “`FireFighting`”, using BFS/DFS traversal. It sees that it needs to forward the message to “`FireFighting`”, “`NJ FE2`” and “`F.Fighter2`”. Although the name should be further expanded under “`NJ FE2`”, we do not require RP2 to do this. RP2 would forward the message as a multicast to “`FireFighting`” based on the subscription, since “`FireFighting`” is served by itself. It then sends 2 publications to “`NJ FE2`” and

“F.Fighter2” since it cannot find the entries for these names in the name table. These messages will reach RP1 based on the underlying network performing the lookup (*e.g.*, NDNS lookup in NDN [18] or GNRS lookup in MF [160]) to find that RP1 is responsible for these names. It will then get expanded at RP1.

An important difference between graphs and (hierarchical) trees is the potential existence of cycles in graphs, which needs to be addressed with graph-based namespaces. While semantically it is a poor design to have cycles in the namespace graph, it is possible that frequent changes in the namespace result in (possibly transient) cycles. These loops in the publication dissemination path can have significant performance impact. ICN routers, as in NDN, have inherent support for detecting and discarding looped packets with the use of Nonces [19]. POISE uses a similar data plane approach for resiliency against namespace graph cycles: fresh Nonces are used for each new end-user-generated publication and is carried in all its subsequently expanded packets. RPs maintain a list of <name, Nonce> pair for (publication) packets they have seen; if an RP encounters a packet with name and Nonce matching any entry in the list, it will discard it, thus breaking the loop.

6.6 Automatic Load Splitting

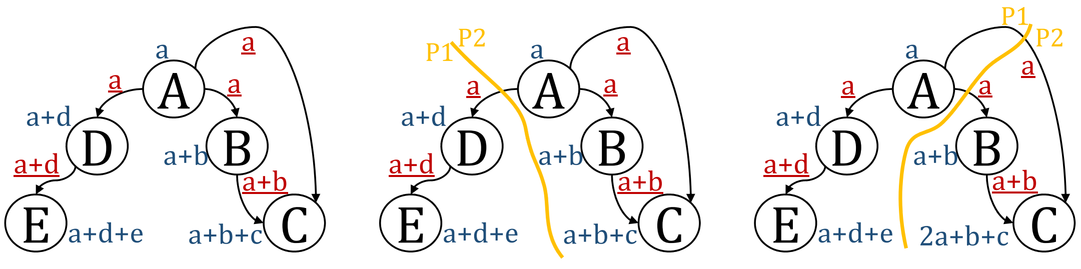
6.6.1 Partitioning Namespace Graphs

POISE’s namespace graph partitioning aims at distributing the load among RPs if traffic concentration overloads an RP. Partitioning is performed locally on the congested RP, only on the (sub-)namespace that it is hosting, dynamically. We mainly use the monitoring of the recent queue size at RPs to measure its load, and use the recent multicast and unicast workloads (explained below) to label the graph for partitioning.

Problem Description and Solution Overview

We leverage graph partitioning algorithms to determine which part of the namespace should reside at which RP for load splitting. We treat the namespace as a directed graph with weights (labels) on vertices (*i.e.*, names) and edges. The initial (input) vertex weights represent messages sent to each name explicitly from publishers (we call it *incoming unicast load*). To determine the number of messages multicast from a node (called *multicast workload*), we need to consider the incoming unicast load from all of its ancestors. The weight of the edges going out of a name are set to be the multicast workload of that name. The total weight of the edges going out of an RP to other RPs represents the total amount of outgoing inter-RP communication (which we call *outgoing unicast load*). We try to balance the sum of multicast workload and outgoing unicast load, in the two partitioned segments and seek to minimize the cut cost. A complexity here is that the decision of the partitioning can alter the weights, *i.e.*, "the chicken and egg problem" [157], thus making the off-the-shelf graph partitioners inadequate; we explain this with an example.

Fig. 6.10 shows a simple namespace graph at different stages. Let us denote the incoming unicast load of each name (node) as a, b, c, *etc.* Assuming no partitioning (*i.e.*, whole namespace in same RP), the multicast workload of each name is shown in Fig. 6.10(a) (blue labels next to each name). *E.g.*, name `C` has to send out publications related to itself, `A`, and `B` to its subscribers; thus making its multicast workload a+b+c. Edge weights in



(a) Before partitioning (b) Partitioning 1 (c) Partitioning 2

Figure 6.10: Partitioning impacts multicast workload weight of names

Fig. 6.10, denoting the RP-to-RP communication on that link, in case it gets cut, is shown in red and is underlined.

Considering the graph shown in Fig. 6.10(a), there may be multiple ways to partition a graph. Two examples are depicted in Fig. 6.10(b) and Fig. 6.10(c). As seen in the figures, the result of multicast workload of name `C` (and thus the total cost of the resulting graph) differs in the two figures; it is $a+b+c$ in Fig. 6.10(b) and $2a+b+c$ in Fig. 6.10(c). The one extra message `C` receives from `A` in Fig. 6.10(c) is due to the fact that in this scenario, `B` relays what it has received from `A` to `C`, not knowing that `A` is also a parent of `C`; while in Fig. 6.10(b) the graph cut is in a way that it does not cause such duplication. This shows that the graph's multicast workload weight values are a function of partitioning itself; thus, a standard partitioning tool with fixed weights is not sufficient to solve our partitioning problem. While such static methods can provide a fast, scalable partitioning solution, they do not achieve a sufficiently high-quality and come close to optimality, as they do not take into account the weight changes due to the cut. To address this, we propose the use of a hybrid approach of heuristics (classic static graph partitioning) followed by a refinement period. This refinement is an iterative procedure that dynamically adapts to weight changes as it progresses. One possible approach for it is to devise an iterative approach with successive runs of METIS, where at each iteration, the new weights based on the previous cut is fed back to METIS. Thus, at each iteration the best partitioning so far is returned as the solution of that iteration. A more efficient variation of this approach is to use ParMETIS's adaptive re-partitioning routine [105] at each iteration, rather than running METIS from scratch each time. While both these approaches may improve upon the solution quality of METIS, they are prone to getting stuck at local optima at an early stage, and also in not exploring the best possible search paths. To address this issue, we use Tabu search for the refinement period in POISE, as it provides a more thorough, but guided search as a meta-heuristic, to iteratively improve on the initial result provided by METIS.

Our algorithm supports bi-partitioning as well as k-way partitioning (with $k > 2$). Typically, bi-partitioning is preferred over k-way partitioning in POISE, since: 1) it is computationally less complex, and more importantly, 2) starting from one RP in the beginning, it results in having fewer RPs (which means less inter-RP communication). Additionally, POISE does a *local* partitioning, using only the information at the triggered RP. However, additional information from neighboring RP's can be added and considered for the partitioning decision if needed. An extreme case of that, however, meaning a global partitioning and placement using all the information in the network, while theoretically possible in our graph partitioning approach, is practically not feasible as it requires too much communication to exchange data, which is not desirable in our network environment. Thus, we focus on local, bi-partitioning.

As described earlier, we pay special attention to the quality of the partitioning solution, as a high-quality partition significantly reduces the resulting network traffic overhead and latency in POISE. Given that:

1. our partitioning is only performed occasionally and only upon RP overload (instead of all the time),
2. is run locally (rather than coordinating across multiple RPs),
3. concerns itself with balance among the two graph segments within an RP (rather than across the whole network), and,
4. deals with graph sizes of moderate sizes (in the order of hundreds or thousands of vertices, rather than millions, for each separate graph connected component, that is input to a partitioning pipeline),

we believe it is reasonable to favor partitioning quality more over scalability. That said, we refrain from using brute-force approaches and instead use the parameters of our proposed algorithm in a way that produces high-quality results with the least amount of processing overhead. Also, in cases where the graph namespace consists of a number of separate connected components, each connected component sub-graph can be processed for partitioning independently and concurrently, thus enabling faster and more scalable computation in POISE.

Theoretical Foundations

In this sub-section, we formally define and mathematically represent the weighted namespace graph for partitioning, and how weights are calculated. Assuming we are dealing with graph $G(V, E)$, some notations we use are defined in Table 6.1.

Multicast Workload (MW). An important part of our weight calculation, is calculating $MW(v_j)$ for each vertex v_j. Its value contains the aggregation of the vertex's ancestors' incoming unicast loads, reaching vertex v_j over all possible paths. When traversing within an RP for propagation, *e.g.*, using DFS traversal, only one path needs to be counted for multicast. On the contrary, if traversing across graph cuts (inter-RP), every path with a unique input-output pair of vertices needs to be counted. The value of $MW(v_j)$ depends upon the amount of load that v_j experiences, caused by every other vertex. Thus, we need to solve "how many times the incoming unicast load from v_i is received at v_j", which we call the *duplication factor*. We describe these in further detail.

Let us define a *simple path* as a path between two vertices that *does not* include a cycle fully contained in one segment. We then define P_{ij} as all simple paths from v_i to v_j:

$$P_{ij} = \{p_{ij}^1, p_{ij}^2, \dots\} \tag{6.1}$$

where p_{ij}^k is the k-th simple path from v_i to v_j, and is an ordered tuple of vertices, formed as $p_{ij}^k = (v_i, \dots, v_j)$.

In a partitioned, *i.e.*, cut graph, we define a *Cut* C as the multi-set of edges cut by partitioning:

$$C = \{(v_i, v_j) \mid part(v_i) \neq part(v_j)\} \tag{6.2}$$

where $part(v_i)$ denotes which graph part (segment) the vertex v_i belongs to, after partitioning.

In order to avoid over-counting, we need to find how many of p_{ij}^k's need to be counted. We define *border* portions of p_{ij}^k as the ordered tuple of (v_{m1}, v_{m2}) pairs, a subset of p_{ij}^k, where $(v_{m1}, v_{m2}) \in C$. We define $Borders$ as:

$$\begin{aligned} Borders(p_{ij}^k) &= ((v_{m1}, v_{m2}), (v_{m3}, v_{m4}), \dots) \\ &= \{(v_m, v_n) \mid (v_m, v_n) \subseteq p_{ij}^k \wedge (v_m, v_n) \in C\} \end{aligned} \tag{6.3}$$

Each border element (v_m, v_n) consists of an *exit* point (v_m) and an *entry* point (v_n). We define entry points (EP) as:

$$\begin{aligned} EP(Borders(p_{ij}^k)) &= ((v_{m2}), (v_{m4}), \dots) \\ &= \{(v_n) \mid \exists\ v_m\ s.t.(v_m, v_n) \subseteq p_{ij}^k \wedge (v_m, v_n) \in C\} \end{aligned} \tag{6.4}$$

The main part of a border tuple impacting the result, is the entry point to the next sub-graph. We define two paths are related by R if and only if they have border nodes with same entry points (with same order):

$$p_{ij}^{k1}\, R\, p_{ij}^{k2} \iff EP(Borders(p_{ij}^{k1})) {=} EP(Borders(p_{ij}^{k2})) \tag{6.5}$$

Relation R is an *equivalence* relation: it is *reflexive* $(p_1\ R\ p_1)$, *symmetric* $(p_1\ R\ p_2 \implies p_2\ R\ p_1)$ and *transitive* $(p_1\ R\ p_2 \wedge p_2\ R\ p_3 \implies p_1\ R\ p_3)$. Thus, R divides P_{ij} into disjoint sets. All paths inside the same equivalence class are *similar paths* and together, they carry the load from v_i to v_j only once. Thus, the duplication factor n_{ij} is defined as the number of *dissimilar paths* from v_i to v_j, which is the cardinality of the equivalence class:

$$n_{ij} = |P_{ij}/R| \tag{6.6}$$

Table 6.1: Notations for graph

Notation	Definition
$IW(v)$	Incoming unicast load
$MW(v)$	Multicast workload
$UW(v)$	Outgoing unicast load

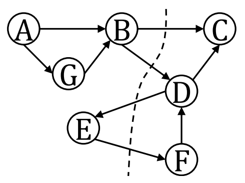


Figure 6.11: Weight calculation example

Using the definition of duplication factor in Eq. 6.6, the multicast workload of a vertex u will be:

$$MW(u) = \sum_{v \in V} n_{ij}.IW(v) \tag{6.7}$$

Using the example in Fig. 6.11 (cut denoted by the dashed line) and following the aforementioned equations, the duplication factor of 'A' to 'C', *i.e.*, the number of times every message to 'A' needs to be eventually multicasted at 'C' as well, is 3. Using this and Eq. 6.7 we will have: $MW^A(C) = 3 \times IW(A)$ showing the number of outgoing multicast messages out of 'C' caused by explicit incoming load at 'A'.

Outgoing Unicast Load ($\boldsymbol{UW}$). Outgoing Unicast Load (or unicast workload) of a name denotes the number of messages that leave one RP, and enter the other RP, because of that name. If a node v_k has no edge towards the other RP, its UW would be 0. Otherwise, it would be related to the number of its outgoing *effective edges* for every $IW(v_i)$ it receives:

$$UW^{v_i}(v_k) = n_{ik} \times ee_k^{v_i} \times IW(v_i) \tag{6.8}$$

where $UW^{v_i}(v_k)$ is the additional unicast load of v_k that is caused by incoming unicast load at v_i. The number of effective edges out of v_k because of v_i is the number of edges that carry data from v_k to the other RP, *e.g.*, to a node v_j in the other RP. If v_j has more than one edge coming to it from the other RP, only one of them will be used, due to DFS's single-visit traversal. The total outgoing unicast load of v_k would be:

$$UW(v_k) = \sum_{\forall v_i \in V, v_i \neq v_k} UW^{v_i}(v_k) \tag{6.9}$$

Algorithm

While we can design an algorithm that strictly follows the formulas mentioned previously for weight calculation, we can design more efficient algorithms that consume less memory. The mathematical representation presented previously is used to prove and cross-validate the algorithm, as an alternative way of arriving at the final result. The calculation of these weights are done through an iterative diffusion algorithm which follows our propagation logic described. Algorithm 2 calculates the two weight values of each vertex, namely MW and UW (stored in maps 'multicastLoad' and 'unicastLoad' respectively) using propagation from each source vertex (that has incoming unicast load 'iLoad') to any reachable ancestor vertex, be it in the same or different sub-graph (part). Starting from the vertex 'nodeName', the algorithm traverses the graph and finds all the traversed vertices, using a modified DFS algorithm (Algorithm 3). Final weights of any traversed vertex within the same sub-graph as 'nodeName' will be incremented accordingly (by 'iLoad'). For any traversed vertex 'u' that is not in the same sub-graph as 'nodeName', *i.e.*, pointing to another RP, the procedure is recursively called, starting propagation from 'u', with 'iLoad' in the other sub-graph. Any vertex 'pu' having a link to another sub-graph will have its UW updated accordingly.

Algorithm 3 describes the modified DFS for traversal of two connected subgraphs $G1$ and $G2$ according to our protocol, used for propagation and calculation of graph weights.

The ‘modifiedDFS’ function and its helper function, ‘modifiedDFS_util’ carry the source vertex ‘name’, as well as the original source vertex name ‘origName’. The ‘originality’ argument can be either 1 (the call has been made by the original source) or 0 (otherwise). If ‘name’ and ‘origName’ are equal, and also ‘originality’ is 0, then the procedure terminates. This distinction is important in order to catch loops and prevent a message from indefinitely circling in case of cycle.

Graph Partitioning Procedure

To prepare the graph for partitioning, the RP labels its local namespace sub-graph, which mainly consists of calculating and assigning appropriate weights explained earlier. The weights are calculated for each solution instance, including an initial solution provided by METIS [103]. We use Tabu search for iterative refinement of our graph partitioning solutions [162,194], as described in Algorithm 4. Each solution (candidate) of the procedure provides a cut, which partitions the RP’s namespace sub-graph into two *segments* (assuming bi-partitioning).

Initial solution: Tabu search typically starts with a random initial solution and improves it. To get a better initial partition [157], we try to use the result from the problem closest to ours – the (static) multi-criteria graph partitioning where the weights will not change according to the partition decisions. We use METIS for this stage as it is a highly popular tool that has been shown to be fast, while providing high quality solutions.

Objective Function: As mentioned, to reduce the search space, we adopt a bi-partitioning approach, where the heavily loaded RP’s namespace is partitioned to be split between two RPs, *i.e.*, the current RP and the new RP. The objective (fitness) function we use to evaluate our partitioning solution, takes into account the cost of both segments (sub-namespaces managed by the two RPs) and provides a combined measurement of 'minimizing the imbalance between the two RPs', 'minimizing the maximum single segment load', and 'minimizing the inter-RP communication' (G_1 and G_2 are the two segments, associated with the two RPs):

$$\begin{aligned} F(G_1, G_2) = \alpha \cdot |TC(G_1) - TC(G_2)| + \\ \beta \cdot max(TC(G_1), TC(G_2)) + \gamma \cdot (UC(G_1) + UC(G_2)) \end{aligned} \tag{6.10}$$

where α, β, and γ are optimization coefficients. Setting higher coefficients for some of the terms would result in the final solution being impacted more by those terms. However, the coefficients can be adjusted. We set all of them to 1 in our test cases, since these values appeared to provide reasonably good benefit, in our experiments. The aim is to minimize F. Function $UC(G_i)$ (segment total unicast cost) is the sum of cut edge weights initiated in G_i, and $MC(G_i)$ (segment total multicast cost) is the sum of all vertex weights in G_i. Furthermore, total load cost of a segment would be:

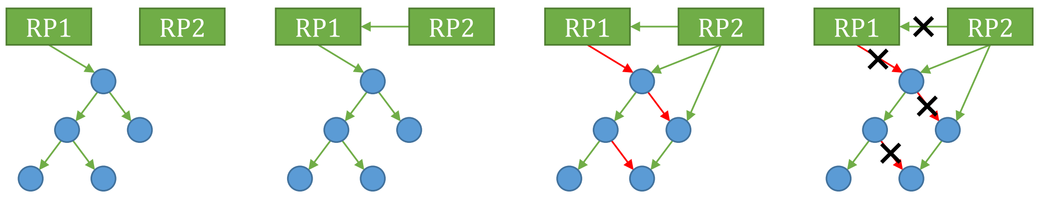
$$TC(G_i) = UC(G_i) + MC(G_i) \tag{6.11}$$

Stopping criterion: We allow both fixed and adaptive stop criteria. If fixed, a parameter *Max Iterations* i is pre-defined, and Tabu search stops when i is reached. Our adaptive stopping criterion, on the other hand, starts with an *Iteration Base* b, and any time the 'so far found best solution' is changed, b gets added to the current iteration number and makes up the new final iteration number. This ensures that our Tabu search procedure stops only after running with b iterations of no improvement. To prevent the Tabu procedure to keep iterating indefinitely, with this adaptive stopping criterion, an upper bound on the number of iterations is also specified.

6.6.2 Migrating Cores

Once the graph partitioning is done, the names in one segment need to be migrated to another core. The RP selection function is similar to that in IP multicast [125]. It may be performed by a network manager or calculated by a Network Coordinate function such as [56]. Once the RP is selected, the process essentially migrates the names in the partitioned subspace to the other RP. However, this has to be done carefully because if a router discards the original subscriptions before it receives all the publications that are in-flight (before the original 'pipe' is drained), these publications will be lost.

To address this issue, we propose a 3-stage (make-before-break) solution to ensure reliable delivery during migration, as shown in Fig. 6.12. Before migration, we assume there is a multicast tree rooted at RP1 (Fig. 6.12(a)). When RP1 decides to move a name to RP2, in stage 1 (Fig. 6.12(b)), it notifies RP2 *and* also subscribes to RP2 (creating new green line). Meanwhile, it notifies the network that RP2 is now serving that name (routing update in IP, FIB propagation in NDN, or a GNRS update in MobilityFirst). RP2 now becomes the RP for the name, and routers with the new RP information will send publications to RP2. However, reusing the original multicast tree, we continue to make sure that the publications are delivered during the transient phase. Routers may have not yet updated the name-to-RP mapping, and there can be publications in-flight during the mapping update. Thus, some publications to those names will still reach RP1. We adopt the late-binding concept of MobilityFirst: when an RP receives a publication that is not



(a) Before (b) Init (c) After M1 (d) After M2

Figure 6.12: Reliable RP splitting: RP1 relinquishing a name to RP2

served by itself. It hands the publication back to the network to then be forwarded to the correct RP accordingly.

Then, at stage 2 (the 'make' stage), RP1 sends out a special marker packet (we call it M1) to all the nodes in the subscription tree. M1 is treated just as a normal multicast packet. To make sure that all the subscribers in the tree receive the M1 marker packet, RP1 has to send that packet after it is sure that the new mapping has propagated into the network and the subscriptions based on the old mapping have joined the tree. On receiving M1, routers subscribe towards the new RP and mark the original ones as 'stale' if the original entry in the subscription table is different from the new entry. Fig. 6.12(c) shows the subscription after M1 is propagated to the network. The green arrows are the new subscriptions and red arrows are the 'stale' ones. Note that while we mark the subscriptions as stale, we do not delete them. When RP2 sends publications, it sends them along all the subscription links, to ensure delivery. A nonce can be used in the packets to eliminate redundant traffic during this transient phase.

After all the nodes subscribe to the new RP, RP1 can send a second marker packet (we call it M2) to start the third and final stage (the 'break' stage). On receiving this marker packet, the intermediate nodes clean up the 'stale' subscriptions (as is shown in Fig. 6.12(d)). When a node has no downstream subscriptions (*e.g.*, RP1 in the Fig.), it will unsubscribe from the upstream naturally. Since all the nodes have subscribed to the new RP, the M2 marker packet acts as the last packet in the pipe. Thus, we will not lose packets if we close the 'pipe' (unsubscribe) after we receive M2.

It is also important to provide resiliency to RP failures. POISE's RP splitting mechanism can be used for recovery. The namespace managed by an RP would have to be backed up and replicated (possibly pro-actively) at a backup router. On detecting an RP failure, the backup router can become active and using the above protocol, the subscription trees for that part of the namespace would be shifted to the backup RP. This would be transparent to publishers and the protocol minimizes loss of in-transit packets.

6.7 Evaluation

To evaluate POISE, we compare it to a number of existing and theoretical alternatives. In terms of overall architectures, we compare POISE to NDN/CNS [48], a recipient-based push-based pub/sub architecture for notification systems, which is the closest architecture to ours. For namespaces, we compare POISE's graph-based naming with the most advanced state-of-the-art ICN naming, which is NDN's hierarchical naming (as in CNS as well). For load-splitting, we compare POISE to the most popular graph partitioning tool METIS [103]. We use the same design principles of the simulator in [48], while adding the functionality of our information layer graph namespace design and splitting procedures. Our simulator is open-sourced and available in [11]. For the partitioning component, we use the current implementation of METIS (and ParMETIS [105]), plus our refinement and

Table 6.2: Solution quality of alternatives and global optimum

Vertices	Edges	Optimum	METIS	POISE	Solution itr	Final itr
10	14	1,916	2,093	1,916	12	22
10	20	2,434	2,736	2,434	14	24
15	19	1,400	1,763	1,400	6	21
15	21	4,744	5,876	4,744	5	20
15	26	6,753	10,460	6,753	28	43
15	65	6,119	16,271	6,119	6	21
20	29	2,594	3,162	2,856	20	40
20	42	9,342	18,905	9,342	10	30
20	89	7,689	15,587	10,480	10	30
25	44	5,966	7,230	5,966	27	52

weight/objective calculation procedures. We also describe our POISE RP implementation on DPDK, and provide micro-benchmarking results to justify POISE.

6.7.1 Evaluating the Graph Partitioning Algorithm

In this section, we evaluate the quality of POISE's graph partitioning, *i.e.*, the hybrid "METIS+Tabu" algorithm. To compare the quality of solutions provided by METIS, and METIS+Tabu (starting from METIS and then doing a Tabu search) with the global optimum (using exhaustive search), we use 10 relatively small graphs, randomly picked and labelled with weights (taken from [78]), as described in Table 6.2. METIS+Tabu uses v iterations with Tabu tenure of $\sqrt{v}$ for a graph with v vertices. For finding the global optimum (*i.e.*, *the* optimal solution), we generated all possible solutions using a brute-force (exhaustive search) approach. For the METIS+Tabu (POISE) case, Table 6.2 also shows the Tabu iteration at which the best solution was found ('Solution itr') as well as

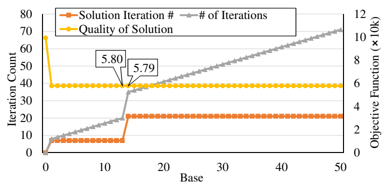


Figure 6.14: Impact of base in adaptive stop criteria

Table 6.3: Quality of METIS *vs.* POISE

Vertices	Edges	METIS	POISE
50	75	4,639	4,042
50	84	99,292	57,897
100	191	23,030	18,980
489	731	47,904	40,745

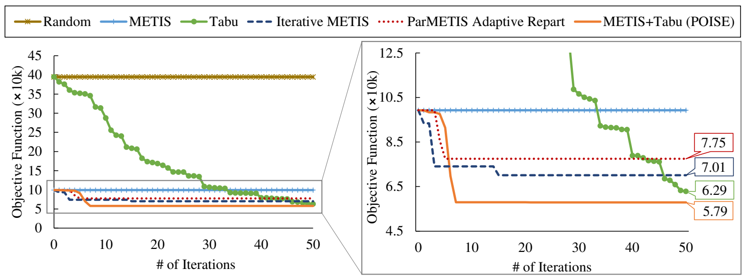


Figure 6.13: Effectiveness of different graph partitioning approaches

the number of the last iteration ('Final itr'). For the cases in Table 6.2, METIS+Tabu (the approach in POISE) finds the optimal solution most of the time (*e.g.*, in 8 out of the 10 cases considered). It also reaches the global optimum within a reasonable number of iterations. Comparing the complexity of the Tabu search and the brute-force approaches, we see a significant benefit of using our Tabu search approach. While the brute-force approach finds the global optimum by checking $2^{n-1} - 2$ solutions (assuming bi-partitioning and filtering out of duplicate permutations and no-cut solutions), Tabu search finds that solution or one reasonably close to it by checking $O(iv)$ candidate solutions, which in our case is $O(v^2)$, since we set the number of iterations i to be $O(v)$ and at each iteration, $O(v)$ neighboring solutions are visited and evaluated. Even though in Table 6.2, POISE found the exact global optimum for most of the cases, this does not necessarily have be to the case for all input graphs. In particular, for two graphs, namely G=$(20, 29)$ and G=$(20, 89)$, POISE did not reach the global optimum. For these two cases, increasing the Tabu iterations by a factor of 10 did not improve the solution either. However, as Table 6.2 shows, POISE's meta-heuristic guided search-based partitioning does find the global optimum for a majority of times, and reaches a near-optimal (at least nearer compared to METIS) solution in all cases. Additionally, as the table also shows, POISE consistently achieves better solutions, *i.e.*, closer to the optimum, compared to the state-of-the-art METIS, further showing the benefit of POISE's partitioning.

Finding the global optimum through brute-force search is not computationally feasible for large graphs, as the number of candidate solutions to visit grows rapidly exponentially. For larger graphs, we only need to do a comparative evaluation, showing that METIS+Tabu finds relatively better, and in most cases, significantly better solutions, than alternative approaches. To show this comparison, we use one of the graphs available online in the repository in [78] (from its "AT&T graphs" package). It is a directed graph with 50 vertices and 84 edges. The graph is unweighted, so we assign random values between 0 and 100 to each vertex, to denote the incoming unicast load for each name. Fig. 6.13 shows the comparison across different alternatives, in terms of the quality of solution (objective function) for this graph. The following scenarios are used: Random (average of three randomly generated solutions), METIS, Tabu (average of three runs of Tabu-only starting from random initial solution), Iterative METIS, ParMETIS with Adaptive Repartitioning (parallelized on two processors, with the 'coupling' of sub-graphs with processors for the best performance [105]), and METIS+Tabu (POISE). Note that the "Random" and "METIS" scenarios are not iterative procedures therefore achieve a fixed solution quality. We vary the number of iterations for the solutions that support refinement (*i.e.*, dynamic solutions), with a fixed stop criterion, to show how quickly the search-based approaches converge to a good quality solution.

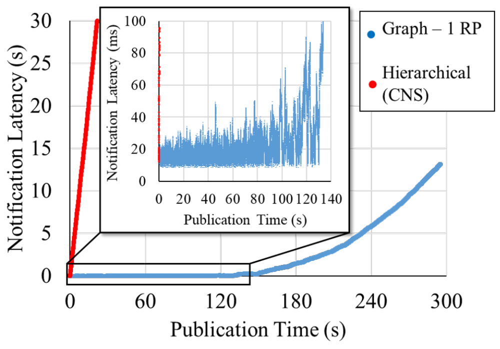
Fig. 6.13 shows that METIS+Tabu outperforms the rest. Using METIS as initial solution (METIS+Tabu) *vs.* starting from a random initial point (Tabu) is very effective as the algorithm reaches its convergence point (for the range of iterations we examined) much faster (with fewer iterations). The Tabu-only method outperforms METIS, Iterative METIS, and ParMETIS Adaptive Repart only after a relatively large (above 40) number of iterations. The METIS+Tabu approach outperforms METIS very early, after just 5 iterations. It also outperforms Iterative METIS and ParMETIS Adaptive Repartitioning early, before the 10^{th} iteration. This shows that the Tabu search is the preferred refinement approach.The random partitioning solution is much worse than the other alternatives. Fig. 6.13 also shows the importance of choosing an appropriate stop criterion. The number of iterations being too small precludes reaching a good solution, and it being excessively large results in waste of time and compute resources.

We also examine using an adaptive stop criterion. Fig. 6.14 shows the impact of the base parameter on the quality of solution found by our graph partitioning (yellow line) in terms of objective function (right-side y-axis). This shows that the base parameter in an adaptive stop criterion needs to be selected carefully as well; *i.e.*, not too small or too large. The jump between base values of 13 and 14 indicates that a new solution is found with base of 14 that would not have been found with smaller base values, showing an example of how Tabu search escapes local optima. Fig. 6.14 also shows the resulting number of iterations (gray line) and the iteration where the last improvement was found (red line), for each base value, in terms of count/number (left-side y-axis). The difference between these two last numbers shows the number of wasted iterations for that setting of base value. In this example, we see that with a base of higher than 15, increasingly more and more iterations are wasted.

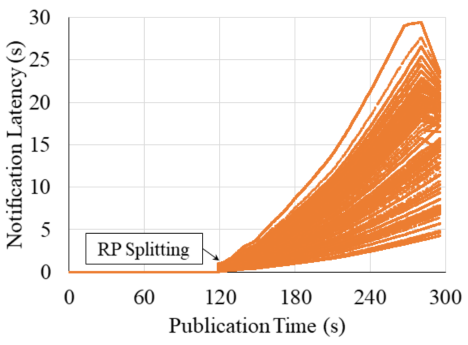
We have also tried our algorithm on a number of different graphs, including larger graphs. Table 6.3 summarizes results for several of those graphs. For these cases we use the adaptive stop criterion. For each graph with v vertices, we choose the base iteration number to be v, Tabu tenure to be $\sqrt{v}$ and upper limit on iterations to be $10v$. The first three graphs are taken from [78], with randomly-assigned initial weights between 0 and 100. The last graph, namely $G{=}(489, 731)$, is the graph-based "Disaster Management" category namespace from the Wikipedia database [191], with the number of content items under each category as vertex weights. This is a reasonable namespace size scale (*i.e.*, generally in the order of hundreds or thousands of vertices [207]), and our Tabu search-based partitioning works well on. We use this $G{=}(489, 731)$ graph as our namespace in our network simulation scenarios as well. Comparing the quality of solution shows that our approach (METIS+Tabu) achieves better solutions.

6.7.2 Overall Solution Evaluation

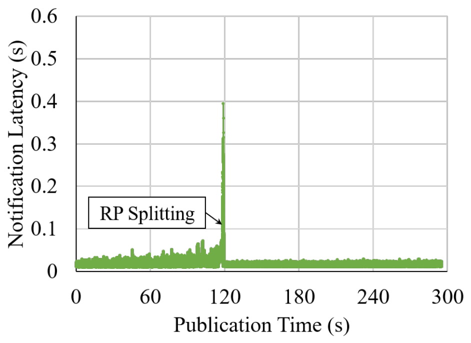
To evaluate the performance of POISE, we implemented an event-driven, packet-level simulator. The simulator supports name-based pub/sub, exploring different alternatives within that paradigm, using any type of multicast network layer underneath. We can compare name-based multicast to alternatives such as pull-based pub/sub, IP multicast-based pub/sub, and broadcast-based pub/sub such as those examined in [49]. To evaluate



(a) Hierarchical & Graph - 1RP



(b) Graph - Random Split



(c) POISE

Figure 6.15: Notification latency over time in different solutions (Note the difference in the scale of notification latency in POISE)

the behavior, we needed a realistic network environment with a number of forwarding routers and end-points that are publishers and subscribers. For this, the network topology we use to evaluate POISE and compare with various alternatives is the Rocketfuel 1221 Telstra [127] with some modification for a state-wide disaster scenario. Our topology contains 46 core routers, with additional 231 routers placed at the edge each linking to 2 core routers closest to them. We use the graph-based "Disaster Management" category namespace from the Wikipedia database as our namespace [191]. Exploring 6 levels below that category, we obtain 489 categories and 732 relationships. If we seek to extract a set of hierarchies based on the approach in [15], we obtain 1,468 hierarchical names. We use the associated pages and files from the Wikipedia database (8,577 items total, 436 per category maximum, 17.49 on average per category) as the publications. We duplicated each content 60 times (514,620 publications) and ordered their publication randomly to load the network. While the namespace is static in our experiments, the publication workloads are dynamic and vary. Publications are generated using a Poisson distribution (to model human behaviors such as calling for, or offering to, help) with a *monotonically increasing arrival rate* over time (to model the increasing nature of such publications, as the disaster unfolds and more people become aware and get involved). We experiment with two example publication workloads: 1) moderate (average arrival rate varying from 1,500 pkt/s to 2,000 pkt/s) and 2) intense (arrival rate varying from 1,500 pkt/s to 3,500 pkt/s). We create 6 subscribers for each category (2,934 in total), distributed randomly on the 231 edge routers in the network. Eventually we generate 20,022,480 delivery events.

Table 6.4: Average notification latency & aggregate network traffic

Solution	Notification Latency (s)	Network Traffic (Gb)
POISE	0.018	492.39
Graph - 1RP	2.741	483.08
Graph - Random Split	4.725	625.69
Hierarchical (CNS)	247.742	866.27

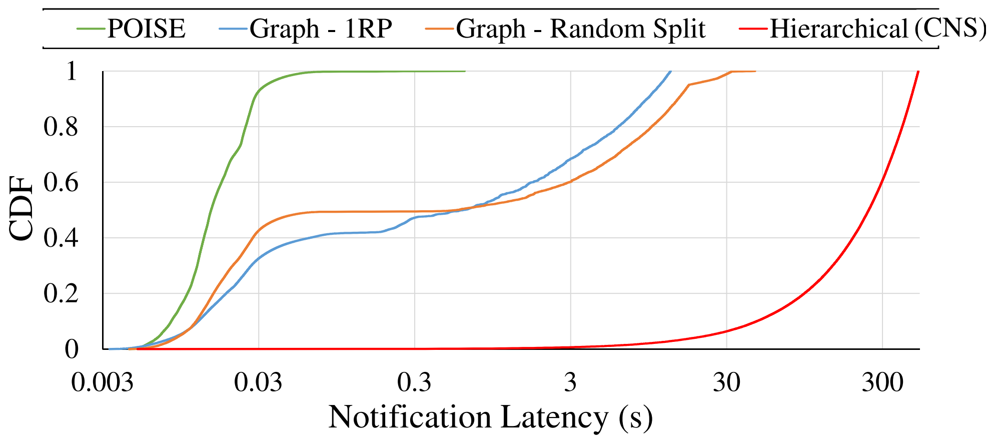


Figure 6.16: Notification latency CDF in different solutions

Experiments with moderate workload: We first consider the notification latency, to deliver a publication to all recipients. This reflects the impact of queuing in the network that arises from having to route through an RP, the selection of an appropriate number of RPs at the correct point in the network topology, adapting to the namespace and workload. We also look at the total network traffic to understand scalability.

We compare the performance of POISE with a number of alternatives. First, is the use of a strict hierarchical namespace (as in NDN/CNS). To be liberal to the hierarchical alternative, we avoid each subscriber having to subscribe to every name. Therefore, when there are multiple hierarchical names for a category, he subscribes to *any* one of the names. The publisher publishes to *all* the hierarchical names of the category. We also compare with having a single RP (no splitting), as well as a simple random splitting of the RP to one of the nodes in the network. The latter is used to demonstrate the need to use a near-optimal splitting of the RPs and load balancing in POISE.

From the CDF of the notification latency in Fig. 6.16 (and the average reported in Table 6.4), we see that due to the high workload on the RP caused by hierarchical names, the notification latency is excessive. Having only 1 RP (graph-1RP) as well as random splitting of RPs perform reasonably at lower loads (for rates <1700 pkt/s) and are even better than using hierarchical names at low loads. However, at higher workloads, random split and hierarchical names perform poorly compared to POISE as well as even having just one RP.

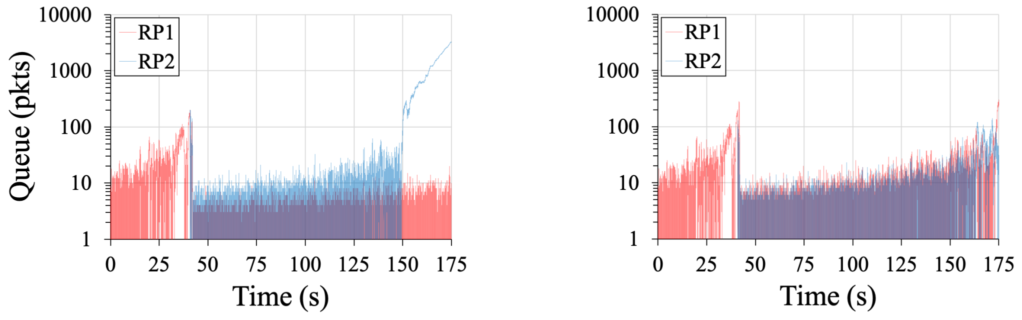
Fig. 6.15 shows the notification latency as the load gradually increases, for all the solutions. With a random split, Fig. 6.15(b)), the notification latency goes up very rapidly after the split, because the entire system is overloaded by packets sent back and forth between RPs. It is even worse than having a single RP, with no splitting (blue line in Fig. 6.15(a)). This shows the importance of a sensible partitioning algorithm. For the same workload, the latency of POISE (with METIS+Tabu, Fig. 6.15(c)) is dramatically better (by 2-orders of magnitude). As the load goes up, RP partitioning is triggered. For a short transient period, queuing causes a relatively small (compared to other alternatives) increase in latency. But congestion is immediately relieved by RP splitting and the latency drops back down. The maximum transient latency is 400 ms with POISE, compared to multiple seconds with other alternatives.

Next, we look at the total network traffic, summarized in Table 6.4. Splitting the RP in POISE results in slightly higher traffic ($\sim$1%) due to the unicast between RPs, compared to having only a single RP. Yet by doing so, we avoid the significant latency impact of congestion. Random splitting of the RP performs much worse (*and* causing 27% more traffic compared to sensible splitting). In fact, if one were to just consider the relative increase in the amount of traffic because of RP splitting (compared to having just one RP and not having any RP splitting), then random splitting with 133.3 Gb of extra traffic results in 14.3 times more than POISE (9.3 Gb) in terms of extra traffic. Compared to the hierarchical solution, the graph-based solution of POISE reduces the amount of network traffic dramatically (by 75.9%) since we do not have to deal with the extra names and publications.

To dig a little deeper into the impact of the partitioning method used, we provide more detailed metrics in Table 6.5 to compare the use of METIS and METIS+Tabu (POISE). For the moderate input workload, using METIS+Tabu leads to slightly (24μs) improved average notification latency (per delivery), while adding 0.002% total traffic. The reason for this better latency is better balance, and thus less queuing delay, even at the cost of slightly more traffic (just like a single RP having the least total traffic in Table 6.4). The load metrics (in terms of # of messages) measure the RP load from the time of the split until the end of simulation (*i.e.* during the time the system has 2 RPs; labeled with '-2RP'). The table shows the values for the three terms in Eq.6.10. For METIS+Tabu, maximum RP load and load imbalance are significantly better, while for METIS, the # inter-RP messages is lower (leading to slightly less traffic). These combined, make METIS+Tabu's solution

Table 6.5: Comparison of METIS and POISE's partitioning

Metric	METIS	POISE
Moderate workload		
Average latency (s)	0.018171	0.018147
Aggregated traffic (Gb)	491.242	492.392
Max Load - 2RP (#msgs)	1,321,220	1,230,533
Load imbalance - 2RP (#msgs)	360,693	633
Inter-RP messages - 2RP (#msgs)	136,571	314,139
Objective - 2RP	1,818,484	1,545,305
Intense workload		
First split time (s)	40.868	
Second split time (s)	150.388	174.252
Average latency in [0,170s] (s)	0.049686	0.020387



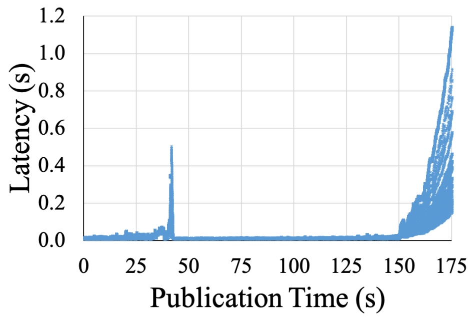
(a) METIS

(b) METIS+Tabu (POISE)

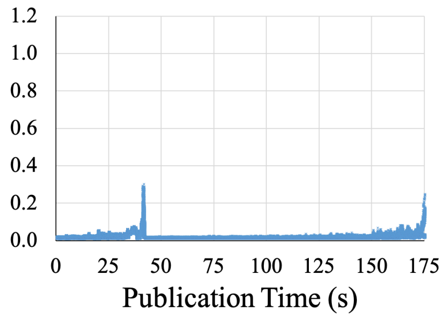
Figure 6.17: RP queue sizes for intense workload

more balanced with a lower peak, as confirmed by the 'Objective' (sum of the above three terms), validating the effectiveness of our partitioning approach.

Experiments with intense workload: The benefit of METIS+Tabu over METIS is even more significant when we generate a higher intensity workload. The same publication trace (for ∼300s) was generated over a shorter duration (∼217s) by increasing the average inter-arrival rate. We also increase the RP splitting threshold. Table 6.5 shows the time at which the first and the second RP splits occur; the first split is same for both (*i.e.*,



(a) METIS



(b) METIS+Tabu (POISE)

Figure 6.18: Notification latency for intense workload

40.868s) while the second split occurs ∼24s later with METIS+Tabu compared to METIS (174s *vs.* 150s). The better balance with METIS+Tabu helps the single RP maintain the namespace for a longer time with lower dissemination latency; the same RP is used for 21% longer than the case of METIS. This is important, since the splitting procedure introduces protocol overhead, with additional notification latency for a short period, as shown in Fig. 6.15(c). Therefore, postponing splitting and reducing its frequency during the lifetime of the overall system is beneficial. However, if the split is postponed for too long, this latency would increase significantly, as seen in Table 6.5. For the period of [0,170s], the average notification latency of METIS is more than twice the latency of METIS+Tabu. Fig. 6.17 shows the instantaneous queue size of each RP in the two cases, for the period of [0,175s]. Most importantly, it shows the better balance between the two RPs in case of POISE (METIS+Tabu, Fig. 6.17(b)) than METIS (Fig. 6.17(a)). We also see that the size of the queue in RP2 for METIS goes up above 3,000 during that period, much larger than METIS+Tabu, which only goes up to 140 for the same time. As the figure shows, RP1's queue size is a little higher in POISE than METIS. However, the queue grows much more

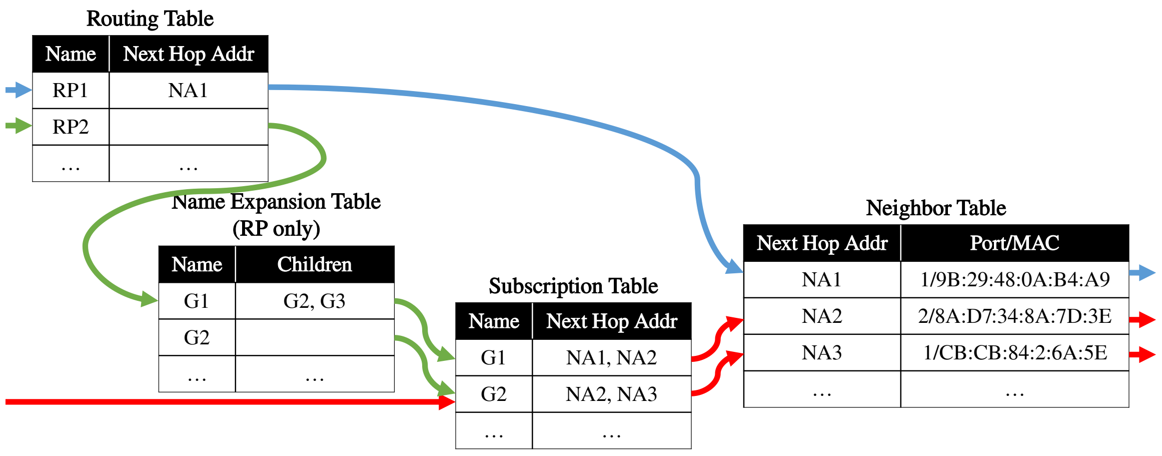


Figure 6.19: Data structures and data flows in the POISE implementation. Blue: upstream publication packet with destination=RP1; Red: downstream publication with name=G2; Green: publication with name=G1 expanded at the RP (flow after subscription table omitted)

at RP2 in METIS than with POISE. This is the tradeoff that POISE makes, producing a better balance between the two RPs, thus helping prolong the need for splitting the RPs. The latency per publication for the intense workload is also shown in Fig. 6.18, indicating a much higher increase for METIS (Fig. 6.18(a), seeing congestion after 150s) compared to METIS+Tabu (POISE, Fig. 6.18(b), which stays low throughout, until 175s).

6.7.3 Implementation

We implement POISE in C to demonstrate the feasibility and efficiency of the protocol. To eliminate the performance impact of handling kernel interrupts, we take advantage of Data Plane Development Kit (DPDK) [7] poll-mode driver so that our user-space program can receive/send data directly from/to the NIC. We implement a routing table, a subscription table and a neighbor table (similar to ARP tables) on every forwarding engine to route packets to the RP and multicast data to the subscribers (see Fig. 6.19). On the RPs, we also add logic for expansion based on the graph-based namespace. The graph is implemented using a hash table whose key is the parent name and the value is a list of descendant names. A breadth-first search (BFS) is performed for each packet reaching the RP. To maximize throughput, we use lock-free data structures [2] and use read-copy-update (RCU) [3] on the data entries. To further reduce the latency for multiple hash calculations, we adopt the technique of "chasing pointers". For example, in the routing table in Fig. 6.19, the value of name `RP1` is `NA1`. When a data packet with destination=`RP1` reaches the forwarding engine, the forwarding engine has to perform a hash lookup for `RP1` in the routing table and another hash lookup for `NA1` in the neighbor table to determine the outgoing interface and the MAC address to encapsulate the packet (the blue flow in the figure). This involves 2 hash lookups. In our implementation, instead of writing the value of `NA1` in the routing table, we have a pointer pointing to the neighbor table entry of `NA1`. When the data packet reaches the forwarding engine, we only need to perform 1 lookup (in the routing table) and follow the pointer to get the outgoing port and MAC address. We thus save the second hash calculation and table lookup. This technique reduces the overhead dramatically on the RPs where multiple lookups have to be performed for the name expansion table in BFS.

We perform micro-benchmarks on the implementation using servers in the ORBIT testbed [10]. The machines use Intel Xeon E5-2640 CPU @2.4GHz (20-cores, hyper-threading turned off) with 256GB of memory, and a Mellanox MT27710 25 Gbps NIC that supports DPDK. We schedule our program on cores 10-19 to prevent cross-NUMA node accesses. We are able to forward $\sim$15.3 million packets per second (Mpps) when dealing with upstream packets (of 64 Bytes in size). For downstream packets, we are able to achieve similar performance when there is only a single next-hop downstream. When there are additional next-hops downstream, the performance drops dramatically (down to serving $\sim$4.1Mpps incoming packets for 2 next-hops, and $\sim$3.2Mpps incoming packet for 3 next-hops). This is mainly due to the overhead of DPDK copying packets or the use of multi-segment packets depending on the replication solution we choose. In comparison, the overhead for name expansion, which we observe on the RP module is much higher. Performing the name expansion on a random graph with 128 names, the processing rate only achieves $\sim$0.8Mpps, even with only 1 next-hop. This demonstrates the necessity of limiting the namespace expansion only to the information layer (otherwise, every single forwarding engine has to behave like the RP module and suffer the performance penalty). Then, with the approach such as POISE, we can have multiple RPs in the network to share the load, as needed, dynamically.

Algorithm 2 Weight calculation in the graph

```
1: procedure CALCULATE(Node nodeName, int iLoad, Graph G, Graph G0, Graph G1, Map <Node, Boolean> done)
2:     ▷ multicastLoad<Node, int> and unicastLoad <Node, int> are maps that store $MW$ and $UW$ values for each vertex
   (node)
3:     if nodeName in G0 then
4:         thisG←G0, otherG←G1
5:     else
6:         thisG←G1, otherG←G0
7:     end if
8:     for all Node u in thisG.modifiedDFS(nodeName, nodeName) do
9:         if u in thisG then
10:            thisG.multicastLoad.put(u, thisG.multicastLoad.get(u)+iLoad)
11:            G.multicastLoad.put(u, G.multicastLoad.get(u)+iLoad)
12:        else
13:            initialize Boolean todo←TRUE, Node pu ← “”                    ▷ empty string
14:            initialize Map <Node, Boolean> done2 to all <v, FALSE>
15:            for all pu in thisG.adj.keyset do
16:                if u in thisG.adj.get(pu) AND
17:                  pu in thisG.modifiedDFS(nodeName,nodeName) then
18:                   if u NOT in thisG then
19:                       pu1 ← pu
20:                       if done2.get(pu)=TRUE then
21:                           todo←FALSE
22:                       end if
23:                   end if
24:                end if
25:            end for
26:            if todo=TRUE AND pu1 ≠ “” then
27:                done2.put(pu1, TRUE)
28:                thisG.unicastLoad.put(pu1, thisG.unicastLoad.get(pu1)+iLoad)
29:                G.unicastLoad.put(pu1, G.unicastLoad.get(pu1)+iLoad)
30:                CALCULATE(u, iLoad, G, G0, G1, done2)
31:            end if
32:        end if
33:    end for
34: end procedure
```

Algorithm 3 Modified DFS for graph traversal

```
1: procedure MODIFIEDDFS(Node name, Node origName)
2:     initialize traversedNodes←{}
3:     for all Node v in G do visited(n)←FALSE
4:     end for
5:     MODIFIEDDFS_UTIL(name, origName, 1)
6:     return traversedNodes
7: end procedure
8: procedure MODIFIEDDFS_UTIL(Node name, Node origName, Boolean originality)
9:     if name=origName and originality=FALSE then return
10:    end if
11:    visited(name) ← TRUE
12:    traversedNodes.add(name)
13:    if name not in G then return
14:    end if
15:    for all v in adjacency(name) do
16:        if visited[v]=FALSE then
17:            MODIFIEDDFS_UTIL(v, origName, FALSE)
18:        end if
19:    end for
20: end procedure
```

Algorithm 4 Graph partitioning procedure of POISE

```
1: Start with an initial solution (a partitioning solution),
2: Calculate the neighborhood solutions by picking vertices to move.
3: Calculate the objective function of all neighbors.
4: Filter out the moves in the Tabu list, unless the Tabu move satisfies the aspiration criteria, i.e., if it is better that the
   best solution so far.
5: Pick the neighbor with lowest objective function, as next move candidate.
6: Tabu the picked move for a number (Tabu tenure) of iterations.
7: Go to 2 if stop criteria has not been met.
8: Report minimal objective and the corresponding partition.
```

Chapter 7

CoNICE: Consensus in Name-based Intermittently-Connected Environments

7.1 Introduction

We propose CoNICE (Consensus in Name-based Intermittently-Connected Environments), a framework for consistent dissemination of updates of a shared database among mobile users, in an intermittently-connected environment. We assume no networking infrastructure, no geographical routing or synchronized physical clocks. CoNICE uses graph-based naming [93] to systematically divide the physical space (through region-ing) and the consensus space (through user subscriptions) into hierarchically structured subsets, optimizing the consensus participation to get higher completion rate and faster completion times. CoNICE extends existing name-based information dissemination schemes (such as [93, 203]) by ensuring consistency, and resiliency in infrastructure-less environments. CoNICE is inherently failure-resilient, where disconnection is not just a corner case scenario, but is rather a common case. Inspired by the multi-level consistency requirements provided by cloud and database systems [87, 153], CoNICE provides the coexistence and flexibility of the following three incremental consistency levels for the network: replication (weakest consistency, lowest complexity), causality, and agreement (strongest consistency, highest complexity). All these consistency levels are integrated with a topic-based hierarchical naming schema, through Name-based Interest Profiles (NBIP). The consensus protocol of CoNICE (running on top of D2D propagation protocols), provides users with a strongly-consistent view that respects both agreement and causality. CoNICE extends OTR with naming and decision invalidation handling procedures, for a total and causal ordering of updates. Its decision invalidation helps with overcoming the consensus property violations in case of long-term physical fragmentation in the network, mainly since we define the complementary notions of local and global consensus sessions, pertaining to being within and across fragments, respectively. This invalidation is an important novelty of CoNICE, as it allows the consistency to be achieved even when many fragmented users connect after a long time (such as users from disjoint and remote shelters, in the aftermath of a natural disaster). An important part of the study of consensus algorithms has been proving their correctness [117]; while existing proof methods for asynchronous consensus algorithms assume good periods throughout the whole network (such as in [44]), we take a step further and assume good periods inside fragments that are disconnected for a long time, thus expanding the scope of the asynchronous consensus problem and also making it fit our application scenario of emergency response.

A key novelty of CoNICE is its integration of consistency and dissemination through naming. In other words, the graph-based namespace works as a common interface across the modules that take care of multiple levels of consistency, as well as the protocols for content dissemination throughout the users in the network. The benefit of this use of naming is twofold: 1) it enhances the relevance of information dissemination (*i.e.*, recipients can identify relevant content with respect to their interests) in a decoupled pub/sub manner (*i.e.*, publishers and subscribers do not need to keep track of each other); 2) it enhances the degree of information consistency among relevant users (*i.e.*, optimized by related name-based groups, consensus can be reached faster, and to a higher degree). We will demonstrate these benefits through our results.

The major contributions of this work are:

1. A framework for consistent information dissemination in intermittently-connected environments, considering the important case of emergency response (our source code and data are available GitHub[1]);

2. Enabling different incremental consistency levels (replication, causality, and agreement) for information updates in intermittently-connected networks;

3. A systematic coupling of information flow organization with various consistency preservation procedures, using naming graphs;

[1]https://github.com/mjaha/CoNICE

4. Extending the OTR consensus with a protocol that leverages naming and supports recovery from invalidated decisions;

5. A rigorous proof of CoNICE's consensus protocol using the Heard-Of model formalism, extending the classic OTR proof to cases supporting long-term fragmentation and decision invalidations.

6. Simulation results that show our enhancement leads to a higher degree of agreement among users, with lower overhead;

7. Simulation experiments that demonstrate how CoNICE is resilient to long-term fragmentation scenarios through an effective procedure for decision invalidation, extending existing work on asynchronous consensus solutions.

7.2 Overview of CoNICE

Emergency Response Scenario. We outline an example use case for emergency response where first responders seek to individually update map tags on their devices and then need to arrive at a consistent, coherent view across users as they opportunistically connect with each other. The map in CoNICE is made up of a base layer and data layer, as shown in the example in Fig. 7.1. The base layer is the map background, available offline to each user. Informed by the geography pertaining to the map, it is divided into hierarchically-structured regions (*e.g.*, county, city, *etc.*). For example, region $R11$ is part of $R1$, and is made up of $R111$, $R112$, and $R113$. This hierarchical structure is captured in a namespace, as shown in Fig. 7.2. Users dynamically create updates on the map (*i.e.*, pins or other shapes with data on them), which updates the map data layer; *e.g.*, update

‘a’ as a point in $R111$, or ‘b’ as a shape spanning regions $R122$ and $R123$ in Fig. 7.1. Users create and are interested in receiving updates related to the regions they are dealing with (or to a part of the region they are interested in). As shown in Fig. 7.3, the environment we consider is one without infrastructure-based communication and users rely on D2D [61] communications, with frequent disconnections. Users are equipped with mobile devices capable of D2D wireless communication (*e.g.*, Bluetooth or WiFi-Direct), and have the CoNICE application on their device. In the example scenario (Fig. 7.3), user A (a first responder) creates a pin on region $R111$ and propagates it at time $t1$. Users C and D, who are both interested in $R111$ (through subscribed interest in regions $R11$ and $R111$ respectively), have no path to A at $t1$. However, thanks to user B moving between the two fragments and acting as a mule doing store-carry-and-forward [66], the update gets propagated to C and D, and they can add it to their view of the map. Different users can create updates (on top of their accumulated ’view’ of the data layer) and disseminate them at any time, without any coordination. Considering the example in Fig. 7.1, let us assume update ‘a’ has been (at least partially) propagated among interested recipients of region $R111$. If now two different users simultaneously create new updates that modify ‘a’, they would both consider that as ”the next update in $R111$”. This may cause discrepancy and differences in the order in which different users apply the updates. This is an important issue that we need to address for the effectiveness and correct functionality of CoNICE in critical scenarios, such as disaster management. Thus, our primary goal is to make sure all the users converge to a consistent view of updates on the map data layer, in this environment, as much as possible.

CoNICE Overview. An overview of the architecture of CoNICE is depicted in Fig. 7.4. It consists of the integration of *multi-level consistency* and *multi-level naming*. There are three incremental levels of consistency. Consistency level 0, namely *Replication*, suggests how much of the generated updates have been delivered to individual users. The *Gossiping* component in each user's device is responsible for this function, using Epidemic Routing [185]. Consistency level 1, namely *Causality*, ensures that orderable updates are applied according to their causal relationships and precedence. This is provided by the *Causal Ordering* component, which provides a moderately-consistent view (*Moderate View*) of the map to the user in the application. CoNICE uses a Vector Clock-based approach [73, 131], extended by a selective and reactive repair mode for causal ordering. Consistency level 2, namely *Agreement*, deals with achieving agreement between different users' views, even for un-orderable updates. The *Consensus* component enables this, and provides the user with a strongly-consistent view (*Strong View*). For this component, CoNICE implements a solution based on the One-Third Rule (OTR) consensus algorithm [32], extending it by supporting selective participation and decision invalidations for highly fragmented and intermittently connected scenarios. Every user is equipped with a single, unified namespace; a hierarchically structured graph pertaining to the regions in the map. This namespace drives the various consistency level components, achieved by *Name-Based Interest Profiles* (NBIP) in CoNICE. There is a NBIP for every consistency level, each pointing (as a subscription) to a particular subset of the namespace. The use of NBIPs allows the various components to achieve better efficiency and accuracy in dissemination.

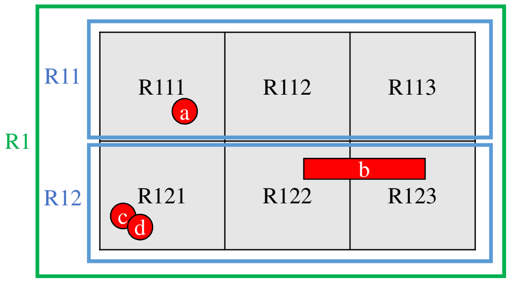


Figure 7.1: Example region-ed map with base layer (background) and data layer (pins/shapes)

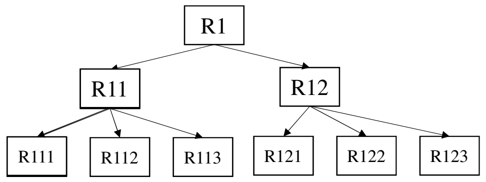


Figure 7.2: Namespace pertaining to the map in Fig. 7.1

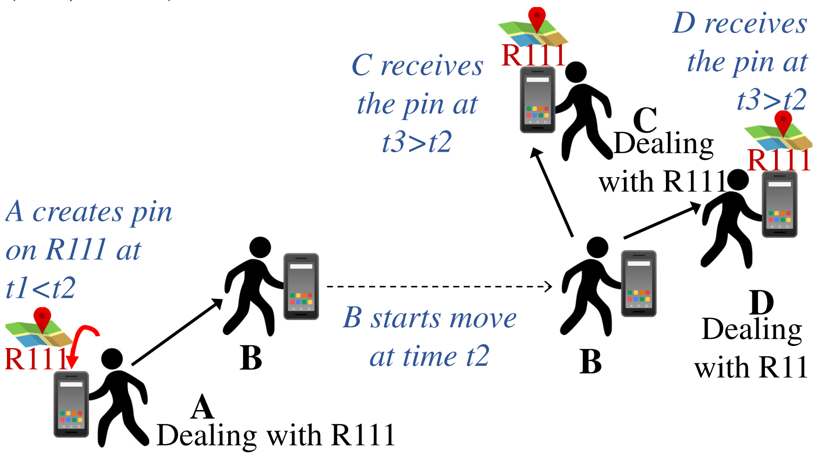


Figure 7.3: A scenario overview

While we recognize security is important to make CoNICE's design robust and usable, we have to address it in detail in a separate work complementary to this work. Here we address the basic protocol of CoNICE and its properties. To ensure authentication and integrity, we can use hash chains [116], similar to [178]; it complements CoNICE's design, since it is based on sequential updates, each update depending on (and cryptographically linked to) the previous one. Further, to ensure fine-grained access control, attribute-based encryption [33] can be leveraged, similar to [119]; it fits well with CoNICE, since it incorporates a namespace, and each user's access privileges corresponding to their role-based

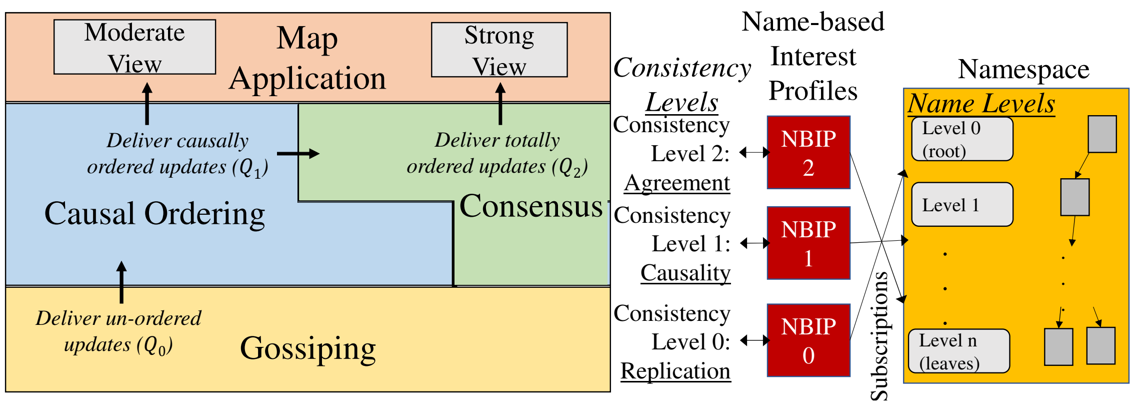

Moderate View
Map Application
Strong View
Deliver causally ordered updates (Q_1)
Deliver totally ordered updates (Q_2)
Causal Ordering
Consensus
Deliver un-ordered updates (Q_0)
Gossiping
Consistency Levels
Name-based Interest Profiles
Consistency Level 2: Agreement
Consistency Level 1: Causality
Consistency Level 0: Replication
NBIP 2
NBIP 1
NBIP 0
Subscriptions
Namespace
Name Levels
Level 0 (root)
Level 1
Level n (leaves)

Figure 7.4: Architecture overview of CoNICE

subscription. Thus, CoNICE can prevent malicious attacks such as impersonation and forgery, via information-centric security [182], which secures content itself, rather than the channel used for delivery. Additionally, Machine Learning-based solutions, such as those proposed in [196] can be used for malware detection and privacy protection.

7.3 Naming and Consistency Levels

7.3.1 Graph-Based Naming Framework

The naming schema in CoNICE is designed and represented as a graph structure (*e.g.*, Fig. 7.2), according to the hierarchical structure of map region-ing: the higher levels in the hierarchy correspond to larger regions. An example of the namespace may be something like "County→City→*etc.*" For mere representation simplicity, we assume each node in the namespace graph has a unique name, so that we do not need to identify a node by mentioning its whole prefix path; *e.g.*, we use "$R111$" instead of "$/R1/R11/R111$".

Region-bound Messages (Publishing). Creation of a message (*e.g.*, an update) that needs to reach interested recipients is a publication in which the region that the message relates to is specified. The region is a name in the namespace and helps with relevancy and selectiveness of dissemination and consistency preservation procedures. A message *belongs to* exactly one region; namely the smallest region large enough to contain that message. For example in Fig. 7.1, update 'a' belongs to $R111$ while update 'b' belongs to $R12$. To ensure higher coverage during dissemination, we define that an update *covers* region set $\{R_i\}$, containing all *leaf-level* regions that contain that update. For example, update 'a' covers $R111$ (same region it belongs to) while 'b' covers $R122$ and $R123$. The relevance follows the namespace hierarchy. A message that belongs to region R_i is relevant to subscribers of R_i *and* the ancestors of R_i in the namespace. For example, for the namespace in Fig. 7.2, update 'a' (specified to be in $R111$) should be consistently delivered to all subscribers of $R111$, $R11$ and $R1$.

Name-based Interest Profiles (Subscribing). NBIPs capture the subscriptions of a user locally on their devices. Each NBIP points to one or more nodes (names) in the namespace, and includes the sub-namespaces (as a set of names) below in the hierarchy, rooted at the pointed names. There are three NBIPs: NBIP0 specifies the scope of the user's involvement in the gossiping procedure, NBIP1 for causal ordering procedures, and NBIP2 for consensus sessions. We have $\mathit{NBIP2} \subseteq \mathit{NBIP1} \subseteq \mathit{NBIP0}$ for every user.

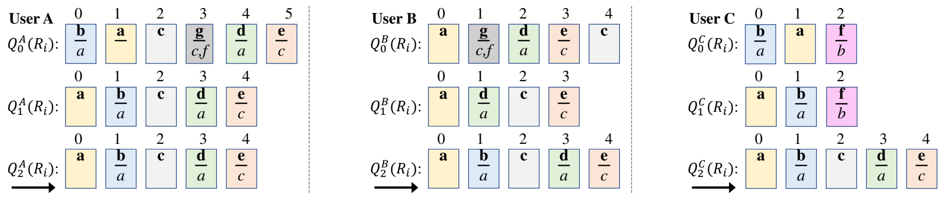


Figure 7.5: Example for per- name per- consistency level update queues across three users A, B, and C

7.3.2 Multi-level Consistency

There are three consistency levels in CoNICE. Users maintain a queue for each region (name) they are interested in for each consistency level, as shown in the example in Fig. 7.5. All three users A, B, and C in the Fig. are interested in region R_i (we just focus on R_i here, so queues related to other possible regions of the users' interest are not shown). $Q_i^U(R_i)$ denotes the level i queue at user U, and its *slot* number 0, *i.e.*, $Q_i^U(R_i, 0)$, is the first element. The consistency levels are incremental, each feeding the level above it (Fig. 7.4); elements in Q_0 serve as input for Q_1, and Q_1 serves as input for Q_2.

The level 0 (replication) queue (Q_0) shows the received updates in the order in which they have been received and entered the message buffer (cache) used by the gossiping component. The updates in Q_0 in the Fig, are marked by the update content ('a', 'b', *etc.* shown in bold) and can contain dependencies (*e.g.*, $\frac{b}{a}$ indicates that 'b' depends on 'a'). As seen in Fig. 7.5, different users can receive the updates, each in different order, causing potential loss or out-of-order delivery because of the use of flooding or other D2D propagation methods, such as [185], as their connectivity is intermittent and there may be no established path between users. In addition to epidemic buffering, this consistency is important as it is the starting point for the next level of consistency and for making sure messages are replicated sufficiently to all users in the network, despite the challenging nature of the connectivity.

The level 1 (causality) queue (Q_1) contains the causally ordered updates, respecting the order for the causally orderable updates. It leverages the dependency information, contained as "meta-data" in each update message. As seen in Fig. 7.5, updates are only added to Q_1 only after all their dependencies have also been added. For example, for Q_1^A, 'b' appears after 'a', 'e' appears after 'c'. But, 'g' does not appear at all, since one of its dependencies 'f' has not been received at A yet (even though it has reached user C, as seen by event 2 for user C). This is consistently true across all users. However, for un-orderable updates (*i.e.*, those with no causality relation), different users can put them in their Q_1 according to the order with which they were received. For example, updates 'c' and 'd' have no causal relation in the figure and are thus un-orderable; in Q_1^A, 'd' comes after 'c' while in Q_1^B, the reverse is true. Despite the difference, neither ordering is incorrect, as they do not violate causal ordering. Q_1 provides users with a Moderate View, which is useful as it provides a causal, meaningful, and "*a* correct" (rather than "*the* correct") view of the map, even though it might be different from another user's "also correct" Moderate View.

The level 2 (agreement) queue (Q_2) shows the updates in the agreed (by *all* users) order, provided by the consensus procedure. It resolves the differences of different users' Moderate Views regarding un-orderable updates, and provides users with a Strong View in the map that: a) honors the causal ordering, and b) is the same across all different users, as seen in Fig. 7.5. Each slot in Q_2 is filled via the result of a distinct consensus *session*. CoNICE first computes Q_1 and then feeds it to the consensus component as initial consensus *contributions* (*i.e.*, vote values). For example, user B's contribution for the second slot of Q_2 is 'd' , while users A and C vote for 'b'. Eventually, the majority value ('b' for slot 1) is *decided* and disseminated to everyone. This incremental approach to arriving at consensus provides several benefits:

1. Consensus starts from a point with potentially more nodes converged on the ordering of a larger number of events, compared to the alternative of jumping from Q_0 to Q_2.

2. Users are eventually provided with a Strong View that respects causal ordering (to perform a meaningful application order of updates), compared to the alternative of completely bypassing and ignoring causal ordering.

3. Users are provided with a useful, causally ordered Moderate View, while waiting for consensus sessions to be completed, compared to the alternative of arriving at a causal ordering after consensus. This is practically important in scenarios such as emergency response, since causal ordering is much less complex and less time-consuming (user-driven) than consensus (community-driven across multiple nodes).

7.4 Protocols for Consistent Dissemination

In CoNICE, three components, each with their own set of protocols, handle its three consistency levels (Fig. 7.4), all integrated with naming. The gossiping protocol propagates and replicates messages among users, the causal ordering protocol takes care of causal consistency of delivered updates, and consensus protocol ensures agreement and strong consistency.

7.4.1 Gossiping Protocol

We use epidemic propagation [185] for gossiping; note that it can be replaced with other information propagation methods given particular system assumptions, as long as they allow anycast propagation. Our assumption in CoNICE is that each user has a unique user ID (*UID*), which can be the mobile device's IMEI or a number provided by the CoNICE application at the time of installation. In CoNICE, each message has a *tag*, specifying its type. Each message has a message ID (*MID*) which is used to uniquely identify it in the network. Users buffer messages for epidemic propagation indexed by their MIDs. Users can create, relay (*i.e.*, store, carry, and forward), and receive messages. CoNICE makes this propagation selective via interest profiling, namely NBIP0 for gossiping. Typically, benevolent data mules help with relaying any message, regardless of what they are about, while other users (*e.g.*, first responders) can have a more fine-grained NBIP0 and only receive and relay messages matching their interest, while discarding others. Updates contain the ID of the region they belong to (R_B) and the (set of) regions they cover (R_C). R_B is integral across all levels, while R_C is only used at level 0 (*i.e.*, can be compared against NBIP0), and its purpose is to increase coverage. A user receiving a message based on its R_C, the R_B of which he is not subscribed to, can be considered to have subscribed to the R_B in the namespace hierarchy, to be able to also participate in its level 1 and level 2 procedures. This epidemic propagation can potentially cause excessive overheads, causing two challenges: 1) a message may keep travelling too many hops, causing unnecessary network and queuing overhead (*e.g.*, a user may receive the same content many times from multiple paths), and 2) too many messages will stay in user device buffers for excessive periods, causing unnecessary storage and processing overhead. To remedy these challenges, CoNICE uses hop count limits and cleaning buffers of obsolete messages (similar to [32, 185]).

7.4.2 Causal Ordering Protocol

Different updates that belong to the same region can potentially depend on each other. As an example, in Fig. 7.1, update '*d*' may depend on '*c*', as they both belong to $R121$, and the creator of '*d*' has seen '*c*' (may be the same creator); *e.g.*, '*d*' may remove some data that '*c*' has added, or modify the information provided by '*c*' about a particular disaster site. These dependencies need to be specified and considered both when it comes to creating and publishing updates, and receiving and processing them. The causal ordering component in CoNICE takes care of this, which helps with consistency level 1 (causality). We restrict Lamport's "happened before" relation [115] to only messages that belong to the same region, calling it "happened before in the same region", to capture causality. This is possible since the region ID is already carried in updates in CoNICE. Formally, we define it as the following:

Definition 2 *For updates u_1 and u_2, belonging to the same region, u_1 causally precedes u_2 ($u_1 \rightarrow u_2$) if and only if one of the following three conditions are true: 1) Some user U creates and publishes u_1, and then creates and publishes u_2 (*FIFO order*), 2) Some user U receives and delivers u_1, and then creates and publishes u_2 of the same region (*local order *or* network order*). 3) There is an update u_3 such that $u_1 \rightarrow u_3$ and $u_3 \rightarrow u_2$ (*transitivity*).*

We will now explain our protocol and how it ensures the aforementioned causality conditions.

The procedure for update creation is shown in Fig. 7.6(a). An update message in CoNICE is of the form shown in line 13. The update ID (*UpID*) consists of UID_A (user ID of update creator A), R_B (the region the update 'belongs to'), *seqNum* (sequence number), R_C (set of leaf-node regions 'covered' by the update), and $UpID_R$ (set of references for this update, which we explain later). The *data* element contains the map-related instruction for the update (*e.g.*, "mark house #1 as searched" or "need teams at building #2"). It is important to also include dependency information in the update, on top of basic gossiping, so that recipient will be able to order the received updates (which can be potentially delivered out-of-order), and achieve a consistent view across different mobile users. CoNICE updates contain dependencies in two ways: *implicit dependency* and *explicit dependency.* Implicit dependency pertains to the dependency of the update on its creator's previous updates on the same region (thus ensuring the FIFO ordering [40]). For a new update in region R_B, the creating user looks for the highest *seqNum* it has used for R_B so far, and assigns the next number to the update (line 8). Explicit dependency pertains to the dependency of the update on other users' previous updates on the same region (lines 9–12), that creator A has already causally delivered to higher layers The helper function *creators*() (line 9) traverses through A's level 1 queue and returns all the users who have contributed to it. (thus ensuring the local ordering [40]). For each $<user, R_B>$ pair, only the update ID with the highest sequence number is picked (line 10). To further reduce the message overhead, all those updates u in $UpID_R$ that precede an update u' existing in $UpID_R$, are removed from the references list (line 12). As a result, the reference list in CoNICE updates will be more compact than the full vector of VC, and also relieves users from having to maintain a global vector of

```
1  input:
2        $R_B$: belongs-to region; $R_C$: set of regions covered; $data$: update data
3  initialization:
4        $UID_A \leftarrow$ id of this user A
5        $M_A \leftarrow$ set of updates created by A
6        $M_{Q1} \leftarrow$ set of all updates at level 1 queues at A
7        $UpID_R \leftarrow \{\}$ /* set of reference updates to be included */
8  $seqNum \leftarrow nextSeqNum(R_B, M_A)$
9  foreach $user\ B\ in\ creators(M_{Q1})$ do        /* identify references */
10     $UpID_R \leftarrow UpID_R \cup latestUpdate(B)$
11 foreach $u \in UpID_R$ do        /* make reference list more compact */
12     if $\exists u' \in UpID_R : u \rightarrow u'$ then $UpID_R \leftarrow UpID_R - \{u\}$
13 $msg \leftarrow \langle UPDATE, UID_A, R_B, seqNum, R_C, UpID_R, data \rangle$
14 $publish\ msg$ /* send to gossiping module */
15 $M_A \leftarrow M_A \cup msg$
16 $M_{Q1} \leftarrow M_{Q1} \cup msg$
```

(a) Update Creation with Causality

```
1  input:
2        $R_B, R_C, data$: as in Alg. 1; $UID_C$: user id of the update creator;
3        $seqNum$: update sequence number; $UpID_R$: set of update references;
4        $UID_R$: user id of requestor in the response msg
5  initialization:
6        $UID_A \leftarrow$ id of this user A
7        $M_{Q0} \leftarrow$ set of all updates at level 0 queue at A
8        $M_{Q1} \leftarrow$ set of all updates at level 1 queues at A
9        $missing \leftarrow \{\}$ /* update IDs of missing prerequisites */
10 Upon receive
   ($msg$=$\langle UPDATE, UID_C, R_B, seqNum, R_C, UpID_R, data \rangle$) do
11     procUpdate($UID_C, R_B, seqNum, R_C, UpID_R, data$)
12 Procedure procUpdate($UID_C, R_B, seqNum, R_C, UpID_R, data$)
13     if $R_B \in NBIP1_A$ then
14         foreach $u' \in M_{Q0} \wedge u' \notin M_{Q1}$ do
15             if $dependencies(u')\ satisfied$ then $M_{Q1} \leftarrow M_{Q1} \cup \{u'\}$
16         $missing \leftarrow missing \cup missingImplicit(u, M_{Q1})$
17         $missing \leftarrow missing \cup missingExplicit(u, M_{Q1})$
18         if $missing = \{\}$ then $M_{Q1} \leftarrow M_{Q1} \cup \{u\}$
19         foreach $UpID_i \in missing$ do $publish \langle REQUEST, UpID_i \rangle$
20 Upon receive ($msg$=$\langle RESPONSE, UID_R, UID_C, R_B, seqNum,$
   $R_C, UpID_R, data \rangle$) do
21     procUpdate($UID_C, R_B, seqNum, R_C, UpID_R, data$)
22     if $UID_R = UID_A$ then $cancel\ msg$
```

(b) Update Receiving and Causal Ordering

Figure 7.6: Update creation and receipt in CoNICE

every user in the network. Finally, the created update will be published by sending it to the gossiping module, and will be added to the creating user's level 1 queue (lines 13–16).

The procedure for handling the receipt of updates and causally delivering them is described in Fig. 7.6(b). The processing of the incoming update u only proceeds at user A if the R_B 'belongs' to its NBIP1 (line 13). Pending updates that get satisfied, will be added to Q_1 (lines 14–15), and all (implicit and explicit) missing prerequisites of u will be collected in $missing$ (lines 16–17). If there are no missing prerequisites, u will be causally delivered and applied to its 'Moderate View' (line 18). In case of outstanding missing prerequisites, the VC algorithm typically waits till they are received. In a disconnected environment with gossiping, this may lead to starvation and indefinite waiting, since "gossips may die out" [83]. To remedy this, CoNICE adds a reactive recipient-driven procedure of requesting for those missing updates (line 19). The $REQUEST$ message identifies the update ID requested for, and the requester's ID (UID_R). Any user, not necessarily the creator of the update, who has that update buffered, can respond with a *RESPONSE* message, sent for the requester. When receiving a response, user A processes it in a similar manner to a normal *UPDATE* message, with one difference that if the response was meant for A, A will cancel the update and not propagate it in the network further (lines 20–22). CoNICE ensures the following key property:

Property 1. *Causal Order of Moderate View.* If user A applies (and delivers) update u to its moderate view, then A must apply every update causally preceding u before u.

Proof of Property 1. We can prove this property using induction. *Basis:* If A applies (to its moderate view) no updates, the property holds. If it applies only one update u_1 belonging to R_B, per Fig. 7.6(a) and 7.6(b), u_1 had no implicit references (*i.e.*, first update created by its creator B for R_B) and no explicit references (*i.e.*, no other user C has created an update for R_B that B had applied before creating u_1). *Inductive step:* Let us assume A has applied n updates $u_1, u_2, \ldots, u_n$, preserving the causal ordering property. An additional update u_{n+1} will only be applied at A, if and only if all causal prerequisites of u_{n+1} are already in $u_1, u_2, \ldots, u_n$ and there are no missing updates (per lines 14–19 in Fig. 7.6(b)), thus ensuring that Property 1 holds.

7.4.3 Consensus Protocol

CoNICE provides a consensus procedure with the goal of achieving agreement, so that users (*e.g.*, first responders) have the same consistent 'Strong View' of the situation (*e.g.*, map). The consensus solution in CoNICE builds on the One-Third Rule (OTR) algorithm [32]. We extend OTR in several ways, mainly with regards to naming and decision

```
input:
    $R_S$: region for this session S; $s_S$: slot number to be decided for S;
    $n_S$: user A's estimation of population for S
initialization:
    $Q_1^A \leftarrow$ user A's current level 1 queue
    $Q_2^A \leftarrow$ user A's current level 2 queue
    $UID_A \leftarrow$ id of this user A
    $contribs_s \leftarrow \{\}$                /* contributions multiset at A */
    $dec_S = \langle DECISION, UID_D, R_S, s_S, a_D, n_D, v_D \rangle \leftarrow \{\}$
    $solved_S \leftarrow false$ /* as $dec_S$ is empty initially */
    $v_I \leftarrow Q_1^A(R_S, s_S)$ /* Noop if null */
    if $v_I \neq Noop$ then startAttempt($1, 1, v_I$)
Procedure startAttempt($a, r, v$)
    $v_S \leftarrow v$
    $a_S \leftarrow a$
    startRound($a_S, r$)
Procedure startRound($r$)
    $r_S \leftarrow r$
    $publish\ msg = \langle CONTRIBUTION, UID_A, R_S, s_S, a_S, r_S, n_s, v_s \rangle$
    $contribs_S \leftarrow contribs_S \cup msg$
Upon receive ($msg = \langle CONTRIBUTION, UID, R_S, s_S, a, r, n, v \rangle$)
  do
    if $R_S \notin NBIP2_A$ then $cancel\ msg$
    switch $a$ do
        case $a > a_S$ do
            foreach $m \in contribs_S$ do $cancel\ and\ delete\ m$
            $n_S \leftarrow max(n_S, n)$
            $contribs_S \leftarrow \{msg\}$
            startAttempt($a, r$)
        case $a < a_S$ do $publish\ D_S$
    if $solved_S$ then
        $publish\ D_S$
        $cancel\ and\ delete\ msg$
    switch $r$ do
        case $r > r_S$ do
            foreach $m \in contribs_S$ do $cancel\ and\ delete\ m$
            $n_s \leftarrow max(n_S, n)$
            $contribs_S \leftarrow \{msg\}$
            startRound($r$)
        case $r < r_S$ do $cancel\ and\ delete\ msg$
        case $r = r_S$ do
            $contribs_S \leftarrow contribs_S \cup msg$
            $n_s \leftarrow max(n_S, n)$
            if $|contribs_S| > (2/3) \times n_S$ then
                $v_S \leftarrow smallest\ most\ frequent\ non\text{-}$
                  $Noop\ in\ contribs_S$
                if $all\ equal\ to\ \overline{V}\ in\ contribs_S\ \ excluding\ Noop$
                  then decide($R_S, s_S, a_S, v_S$)
```

(a) Consensus: Contributions

```
Procedure decide($UID, R_S, s_S, a, n, v$)
    if $\neg solved_S$ then
        if $\exists s' \neq s_S \wedge Q_2^A(R_S, s') = v$ then
            /* conflict with earlier existing decision */
            if $v_I = v$ then startAttempt($a_S + 1, 1, Noop$)
            else startAttempt($a_S + 1, 1, v_I$)
        $solved_S \leftarrow true$
        $v_S \leftarrow v$
        foreach $m \in contribs_S$ do $cancel\ and\ delete\ m$
        $contribs_S \leftarrow \{\}$
        $publish\ msg = \langle DECISION, UID_A, R_S, s_S, a, n, v_S \rangle$
        $D_S \leftarrow msg$
    else                                          /* need to invalidate */
        if $msg \neq D_S$ then
            if $a = a_D$ then
                if $n > n_D$ then $v'_D \leftarrow v$
                else if $n < n_D$ then $v'_D \leftarrow v_D$
                else if $n = n_D$ then
                    if $UID \geq UID_A$ then $v'_D \leftarrow v$
                    else if $UID < UID_D$ then $v'_D \leftarrow v_D$
                decide($max(UID_D, UID), R_S, s_S, a, n, v'_D$)
            if $a > a_D$ then
                startAttempt($a, 1, v_I$)
                decide($UID, R_S, s_S, a, n, v$)
        $Q_2^A(R_S, s_S) \leftarrow v$
        foreach $v' \in Q_2^A(R_S)\ that\ violates\ causality\ with\ v$ do
            $reorder\ locally\ through\ deterministic\ sort$
Upon receive ($msg = \langle DECISION, UID, R_S, s_S, a, n, v \rangle$)do
    if $R_S \notin NBIP2_A$ then $cancel$
    decide($UID, R_S, s_S, a, n, v$)
```

(b) Consensus: Decision

Figure 7.7: Consensus algorithms for CoNICE protocols

invalidations. The naming integration in CoNICE, makes sure all the interested users (even with overlapping interests) are involved in every consensus session relevant to them, which also systematically reduces the consensus participants to the interested ones, helping with faster reaching of decisions. CoNICE's decision invalidation procedures make sure to repair decisions if long-term fragmentation cases happen in the network, and also if the total and causal order of the final strong view are violated even after the OTR-based agreement is reached.

The initialization and contribution procedures of CoNICE's consensus are described in Fig. 7.7(a). Each consensus session is associated with a region-slot pair ($<R_S, s_S>$), deciding the value v (*i.e.*, the update to be placed at the slot) to be inserted to $Q_2(R_S, s_S)$. To avoid scheduling complexities and overhead, we run consensus sessions for individual slots rather than the entire Q_2 content. Each session comprises multiple *attempts*, and each attempt comprises one or more *rounds*. We add the notion of attempt, because we may need to run another attempt of an already decided consensus session, due to the nature of our environment. Users initiate consensus with initial values (v_I) equal to their $Q_1(R_S, s_S)$ content (lines 11–12). If user A has no such content, its initial contribution will be a 'Noop' (or *null*). Any non-'Noop' contribution will be sent for round 1, containing the value (lines 13–20). The *CONTRIBUTION* message identifies the region, which will enable the subscribers of the region to participate in the consensus. Most consensus algorithms (including OTR), depend on knowing the consensus population (n_S) a priori. We enable a bootstrapping mechanism based on reachability beaconing (similar to [42]), for a user to get an estimate of the population; the number of users eligible to participate for R_S, are the number of total subscribers of R_S and its ancestor nodes in accordance with the namespace hierarchy, *e.g.*, Fig. 7.2. In a highly fragmented network environment that we consider, there is a chance this estimation will be incorrect. To remedy this, we allow the user to update its estimation of n_S, from the contributions it receives, to have an upper bound estimate of n_S.

It is important that users synchronize to be in the same attempt and round as much as possible. Upon receiving a contribution (lines 21–45), the user jumps to the attempt and round number of the contribution message if it is larger than its own (lines

24–28, 36–40). This helps users use the good period fuller when it occurs. Users remove obsolete contributions from the buffer, which helps with scalability and reduces the number of messages circulating in the network. Received contributions from older attempts and rounds will be discarded, with a possible response providing the decision that was already made. When the contribution is in the same attempt and round that the user is in (line 40), the user adds it to its contribution list (line 41). When user's received contribution set reaches the cardinality equal to $2/3 \times n_S$ (one-third rule, line 43), the user will either: 1) start a new round, sending a contribution with the value equal to the smallest (*i.e.*, earliest in terms of causality) most frequently received value (line 44), or 2) decided on a value, if all values in the set are the same (line 45). The decision procedure is described in Fig. 7.7(b). A decision message will be published as a result of reaching a decision (line 10). The value in the decision message, determines what value (update) should be inserted into everyone's Q_2 in that particular region's slot (lines 24–26). Another way of reaching a decision is to receive a decision message from someone who has already decided (lines 27–29). CoNICE's consensus protocol satisfies the following properties:

Property 2. *Consensus.* Every consensus session for a $<R, slot>$ pair in the Strong Views preserves the following: 1. **C1:** *Termination.* Every *correct* user (*i.e.*, that does not permanently crash or become unreachable) eventually decides. 2. **C2:** *Agreement.* No two users decide differently. 3. **C3:** *Validity.* Any value decided is some user's initial value. 4. **C4:** *Update Validity.* Any value decided is a valid update that was created. 5. **C5:** *Integrity.* No user decides twice.

Property 3. *Total and Causal Order of the Strong View.* If users A and B apply region R-bound updates u and u' to their strong views, the order of u and u' is the same in both A and B. This order also respect causality.

In normal situation, supported by basic OTR, the above properties are easy to prove [44]. Due to the nature of the environment and extended assumptions we consider, there are additional cases where the properties may get violated, which we provide remedies for in Fig. 7.7(b):

1. *Loss of causality.* Due to the lack of central orchestration and coordination in CoNICE, there may be cases that decided values for Q_2 may not respect causality; *e.g.*, having v_i for $<R, s_i>$ and v'_i for $<R, s_{i+k}>$ while $v'_i \rightarrow v_i$ (*i.e.*, v_i depends on v'_i). We can completely prevent this by running consensus sessions one by one, sequentially. However, this is not efficient and can lead to starvation, especially considering how time-consuming a consensus session can be. Thus, we provide a pragmatic solution to recover from this violation. In case this happens, the user swaps the values between slots, through a deterministic sorting algorithm (lines 25–26). This invalidation can be repaired entirely locally, without further messages.

2. *Long-term physical fragmentation.* Sometimes, more "intense" cases of fragmentation can occur, going beyond the "good periods" assumption of OTR: assume two disconnected shelters in a region, each with a number of users trapped in them, with no mules or paths between the two shelters for a very long time (much longer than consensus durations). As a result of such "long-term" fragmentation, users independently beacon, create, disseminate, and solve consensus within their partitions. In case a path appears sub-

sequently between the two shelters (*i.e.*, by a mule), and messages get exchanged between the two, the network will include two decisions with different values for the same $<R, s>$ pair. This will violate the correctness of consensus. To remedy this, we make use of *UID* and n fields of decision messages to invalidate decisions already-made (lines 12–23), and upgrade the decisions of users to one from the fragment with the higher population (and in case of tie, the one with a higher user ID). This invalidation can be repaired within the same attempt.

3. *Duplicate decisions.* There may be cases where the same value is picked for two different slots; *e.g.*, having v_i for $<R, s_i>$ and v_i for $<R, s_{i+k}>$. This shows that for some reason, one value was picked for two slots. This can be caused by divergences in users' initial values and is not fixed by the basic OTR. Thus, there will be a value that was missed during the consensus rounds, and got replaced by the same value picked in another session. As a result, consensus has to restart, albeit as the next attempt. The user detecting the two duplicates, will keep the first one (in this example, s_i), and starts a new session to re-do consensus for the second slot (s_{i+k}). The only catch is, if the initial value of the user for s_{i+k} (*i.e.*, from its Q_1) is v_i, it will pick 'Noop'. Otherwise, that user will pick $Q_1(R, s_{i+k})$. This way, the consensus will be performed again, giving v_i less chance to be picked for s_{i+k} at the end (lines 3–5). This invalidation can be repaired with a new attempt.

7.5 Proof of Consensus in CoNICE

In this section, we provide proof for CoNICE's consensus protocol against its properties, using the formalism of Heard-of model [44]. Since Property 1 is about basic causal ordering and not directly on consensus. This section only focuses on consensus-related properties. Note that *agreement* is the goal, and *consensus* is the procedure to provide it. The major novelty of our proof is extending the earlier proof methods for asynchronous consensus algorithms [44] to cases by also including long-term fragmentation among the users participating in the consensus.

7.5.1 Heard-Of Model

We use the *Heard-Of (HO)* model [44] for our proof formalism, as it allows for both node and link failures in asynchronous environments, which is appropriate for our system model. The HO model evolves from one communication-closed round to the next, where messages sent in each round are received and used in the same round. A *Heard-Of set*, denoted by $HO(\pi, r)$, represents the set of users (or *processes*) from whom user π "hears of" (*i.e.*, receives messages that were originated from) in round r. *Communication predicates* specify the features and requirements of a system under the HO model. A consensus problem is solved in the HO model by a *Heard-Of machine* defined as $M = (\Lambda, P)$, where Λ is an algorithm and P is a communication predicate. Work in [32] uses the HO model to prove the correctness of classic OTR in DTN environments; we extend that to also address invalidations and long-term fragmentation in CoNICE.

The procedure to prove a consensus protocol in the HO model is as follows: We first start by defining the system model and basic assumptions in the environment. We then formally specify communication predicates for network conditions that are necessary and/or sufficient for the specific protocol to work. Using various deductive or inductive reasoning methods, we prove the theorem of the form "under the given predicate, the given protocol solves consensus".

7.5.2 System Model

Each run (*i.e.*, attempt) of our consensus protocol pertains to a particular region R_S and a slot number s_S (for a consensus session S). The slot number for an update is the next available empty slot in the queue associated with R_S across different users. Its participants are the subscribers of region R_S, in accordance with the namespace. An important novelty of CoNICE is that it can solve consensus even in the case of long-term fragmentation, within the same run of the algorithm. Thus, we define two types of consensus solutions for each user within the same run: 1) *local consensus* (to reach *local agreement*), and 2) *global consensus* (to reach *global agreement*). Users constrained within the same long-term fragment (*e.g.*, trapped in a shelter during disaster) can reach local agreement amongst themselves for a $<R_S, s_S>$ pair through the one-third rule. Upon re-connecting of the shelter to the rest of the network after some time, if there are conflicts between the different values decided based on local agreements and as a result of invalidations of all but one of the values, users can reach global agreement throughout the whole network.

The basic assumptions of our system model are as follows:

- We assume each user in the network is equipped with the CoNICE protocol stack and follows the protocols honestly and correctly (performs reachability beaconing, counts group population per region, *etc.*).
- In a network without long-term fragments, proving our consensus is similar to that of [32]. Thus, we only focus on the fragmented scenarios here.

- We assume that we have multiple disjoint long-term fragments where each fragment Δ_i includes a set of users Π_i with n_i members who are subscribers of R_S. Π is the set of all subscribers of R_S throughout the whole network.

- No more that $\frac{n_i}{3}$ of users permanently crash. Also, in each round, at least one user hears from at least $\frac{2}{3}$ of the other users in the fragment:

$$\forall r, \, \exists \pi \in \Pi_i \, s.t. \, |HO(\pi, r)| \geq 2n_i/3 \tag{7.1}$$

  This is reasonable to assume for our environment, as we use the gossiping protocol that leverages mobility and buffering at users, and mules, to deliver messages with a high delivery probability, especially within a shelter. Furthermore, CoNICE's reachability beaconing and exchange of population counts makes sure that each user has a good enough approximation of n_i, which is needed for the OTR-based calculations.

- After a significant period of time, paths (and connections) appear among some of the fragments,

We first focus on solving local consensus within a fragment (*i.e.*, before there appears any paths between fragments). In that fragment, there is a set of participants Π_i running CoNICE's OTR-based algorithm for $<R_S, s_S>$:

$$\Pi_i = \{\pi_i | \pi_i \in \Delta_i \wedge \pi_i \in sub(R_S)\} \tag{7.2}$$

We specify the following communication predicate P_{otr}:

$$\begin{gathered} P_{otr} : \forall r > 0, \, \exists r_0 \geq r : \exists \Pi_0 \in \Pi_i \, s.t. \, |\Pi_0| \geq 2n_i/3 \\ \wedge \, \forall \pi \in \Pi_i : HO(\pi, r_0) = \Pi_0 \end{gathered} \tag{7.3}$$

which says that infinitely often, there will be rounds in which all the users will hear from a two-third majority subset of members, which is Π_0. If P_{otr} holds, CoNICE solves consensus since after several rounds, all users will deterministically converge their contribution to the same value (lines 40–45 in Alg. 7.7(a)) and eventually agree on a value and decide. Similar to what is discussed in [32], predicate P_{otr} is a sufficient condition to solve our consensus. Here, we complement that specification, *i.e.*, P_{otr}, by specifying a simpler predicate P_{L1} which is the necessary condition to achieve agreement in CoNICE:

$$P_{L1} : \exists r_0, v\, s.t. \forall \pi_i \in \Pi_i : |HO(\pi_i, r_0)| \geq 2n_i/3 \wedge v_i^{r_0} = v \tag{7.4}$$

where $v_i^{r_0}$ is the value picked for either contribution or decision at the end of round r_0 at user π_i. Predicate P_{L1} says that in at least one round, all users will pick the same value, and they hear it from at least $\frac{2}{3}$ of the users, even if it is not the same $\frac{2}{3}$ subset across everyone. If v is the smallest most frequent non-'Noop' value from received contributions (line 44 in Alg. 7.7(a)), but not $\overline{V}$ (as defined in line 45), another predicate is needed (P_{L2}) so that in the immediate next round, all non-crashed users could exchange their contributions and hear from a $\frac{2}{3}$ set:

$$P_{L2} : \exists r_1 > r_0\, s.t. \forall \pi_i \in \Pi_i : |HO(\pi_i, r_1)| \geq 2n_i/3 \tag{7.5}$$

We will show that with $P_L = P_{L1} \wedge P_{L2}$, CoNICE solves consensus in this case. Alternately, if v is the received decision value (lines 2–11 in Alg. 7.7(b)), CoNICE will also solve consensus.

To investigate global consensus, let us now assume that after a significant period of time, paths appear among some of the fragments, some of those Δ_i's have potentially reached their local decisions at round r_i^d on their respective value v_i^d. To follow the model's round-based specification, we define a special round r_∞, which involves every message exchange after the re-connection of all these fragments (which have reached local agreement), for $<R_S, s_S>$ for the rest of the lifetime of the whole network. All the invalidation procedures of decisions previously reached within the same attempt (line 13–20 in Alg. 7.7(b)) occur in r_∞. With respect to r_i^d and r_∞, each user in Δ_i can be in one of the following four categories, which determines their status of whether or not they can decide for local and/or global consensus:

1. user enters r_i^d, and then enters r_∞.
2. user never enters r_i^d, but enters r_∞.
3. user enters r_i^d, but never enters r_∞.
4. user neither enters r_i^d nor r_∞.

CoNICE's global agreement only involves users in categories 1 and 2, *i.e.*, those that enter r_∞, as interaction and contention between different decisions from different fragments is needed, *i.e.*, to further resolve to reach agreement. Users in categories 3 and 4 are permanently isolated in their fragments, and thus not relevant for global consensus, with the relative advantage of users in category 3 over those in category 4 being that the achievement of local consensus can be locally useful, *e.g.*, within a shelter, in category 3.

Among the different fragments that have reached local agreement and entered r_∞, let us define Δ_{max} to be the fragment with the highest population, namely n_{max}, *i.e.*, the "winning" fragment (line 15–16 in Alg. 7.7(b)). The decided value for Δ_{max}'s local consensus is denoted by v_{max}^d. If there is more than one fragment with the highest population, *i.e.*, "tie" situations (lines 17–19 in Alg. 7.7(b)), Δ_{max} is the fragment with the highest population *and* includes the non-crashed user with highest *UID*, denoted by U_{max}. We define the following predicates:

$$\begin{gathered} P_{G1} : \forall \Delta_j \neq \Delta_{max} : n_j < n_{max} \Longrightarrow \\ \forall \pi \in \Pi : \exists \pi' \in \Delta_{max} \, s.t. \, \pi' \in HO(\pi, r_\infty) \end{gathered} \tag{7.6}$$

$$\begin{gathered} P_{G2} : \exists \Delta_j \neq \Delta_{max} \, s.t. \, n_j = n_{max} \Longrightarrow \\ \forall \pi \in \Pi : U_{max} \in HO(\pi, r_\infty) \end{gathered} \tag{7.7}$$

Predicate P_{G1} says that if Δ_{max} is the uniquely most populated fragment, every other user will hear from some user in Δ_{max} in r_∞. Predicate P_{G2} says that if population of Δ_{max} is tied, all users hear from U_{max} in r_∞. It is worth noting that P_{G2} does not necessitate the permanent existence of the user U_{max} in the network; as long as U_{max} stays in the round r_∞ long enough to send its local decided value to some other user, the gossiping protocol enables that message to reach others through D2D relaying. As per Alg. 7.7(b), users who receive a message, first check the population count, and then the user ID carried in the decision message, to converge towards the correct global decision value. Also, no additional majority check or additional rounds are needed in the global agreement phase (due to pair-wise comparison in lines 14–20 in Alg. 7.7(b)). Thus, the communication predicate built by the conjunction of the two, namely $P_G = P_{G1} \wedge P_{G2}$, is necessary and sufficient for CoNICE to ensure global agreement.

7.5.3 Proving Property 2: Basic consensus Property

We now investigate if CoNICE's consensus protocol, in particular Alg. 7.7(a) and 7.7(b), satisfy the consensus property 2, and its sub-properties C1–5, under the communication predicates explained previously.

C1: Termination. For local consensus, assuming P_{L1} stands: 1) if all values v are mere contributions, then Alg. 7.7(a) ensures that each user reaches the decision mode in the next round (assuming P_{L2}), decides and terminates; 2) if at least one of those v's is a decision message, Alg. 7.7(b) (for the same and different attempt, respectively) ensures that the user jumps to the decision mode and terminates. For global consensus, if P_G stands and users do not permanently crash, Alg. 7.7(b) ensures that CoNICE always favors the decision received from the winning fragment/user; thus global consensus will eventually terminate since P_G ensures that the winning decision will be eventually heard of in r_∞ by all users.

C2: Agreement. Local agreement specifies that after some round r_i^d at Δ_i, the value for $<R_S, s_S>$ at each user in Δ_i (*i.e.*, set of users Π_i), will be the same, namely $v_i^d = v$. We assume P_L holds in our environment at some round r_0. We aim to prove local agreement is preserved at any round r after r_0. We prove that proposition using induction on the value of $r - r0$:

Basis: $r = r_0 + 1$. In this case, all users have picked value v at r_0; if v is the next contribution value that all users pick, then in the next immediate round $(r_0 + 1)$, as long as a user hears of "any" $\frac{2}{3}$ set of users, the conditions in both lines 43 and 45 in Alg. 7.7(a) will be true. Thus, it ensures all users go to decision, with value v; so at round r, all users will agree on v.

Inductive step: Suppose that at $r = r_0+n$, all users agree on value v. In round $r = r_0+n+1$, we assume users hear of each other. Given that users receive a duplicate decision value v in the same attempt, their decision does not change, ensured by condition of line 13 in Alg. 7.7(b) not being met. If users receive v as a contribution message, given that they already made a decision, then the decision status does not change, ensured by lines 30-32 in Alg. 7.7(a). As a result, in both cases, the decision value v within the same attempt remains unchanged, which proves the induction for $r = r_0+n+1$, proving CoNICE's local agreement.

For global consensus, we assume predicate P_G holds. As C1 shows, every user in the network terminates with the correct final value v, which is v^d_{max}, albeit possibly going through the incremental changes, based on the "highest winner seen so far" by a user (Alg. 7.7(b)). Since all users converge to termination on the same value (shown by C1), eventually every user in Π will agree on v^d_{max}. Any user that has not yet reached v^d_{max} is a user that has not yet terminated. Similarly, one that crashes permanently is not considered in the agreement requirement. This proves CoNICE's global agreement property. Therefore, any consensus run according to CoNICE's protocol, *i.e.*, all consensus steps that do not require a new attempt or a whole new session, preserves agreement.

C3: Validity. Alg. 7.7(a) ensures that every user starts a consensus session with an initial value. If it is not 'Noop', line 12 ensures that the initial value is then used to create contributions that are disseminated. As we can see in Alg. 7.7(a), no new value is created during the message exchange among users; thus the final value picked for a decision is necessarily some user's initial value, guaranteeing the validity property.

C4: Update Validity. For update validity, every user starts with an initial value that he has delivered in level 1, and thus is a valid update. It can also be a 'Noop'. Alg. 7.7(a) makes sure that 'Noop' would never pass the one-third rule test and proceed to the decision. With this fact and also considering C3, CoNICE ensures that update validity holds.

C5: Integrity. When there is no decision invalidation, C1 ensures integrity is also guaranteed, as no user will make a duplicate decision. In the case of invalidation, this property still holds, since at any point in time, only one decision is valid, even for the case of multiple attempts for a consensus session. This is ensured by line 9 in Alg. 7.7(a), as the variable dec_S by a user, only holds at most one value at a time, which is its latest valid value.

7.5.4 Proving Property 3: Total and Causal Ordering Property

In Property 2, we proved the consensus properties of CoNICE, regrading individual consensus sessions. Property 3 has two separate parts, both of which we will consider. The first part of property 3 specifies total order. Suppose two users π_1 and π_2 have established their final sequences of updates for region R in their strong views, without any holes in the sequence. By way of proof by contradiction, let us also suppose that π_1 has placed updates u and u' in slots s_1 and s_2 respectively (with $s_1 < s_2$), and user π_2 has placed updates u' and u in slots s_3 and s_4 respectively (with $s_3 < s_4$). In other words, we are supposing π_1 placed u before u', while π_2 placed u' before u, thus violating total ordering. We consider the following possible cases:

1. No invalidations are needed: This case would entail that π_1 decided u for $<R, s_1>$ while π_2 decided some update $u'' \neq u$ for $<R, s_1>$ (with similar situation for u').

This would mean that the consensus protocol reached two different decisions u and u'' for the same session $<R, s_1>$, violating the agreement property. Since there is no need for invalidation and CoNICE ensures agreement, this case is impossible.

2. There has been loss of causality, thus invoking the re-ordering procedure (invalidation case 1): This case would entail that update u was initially inserted into some slot s_5 at both π_1 and π_2 (with similar situation for u'), but the re-ordering due to loss of causality changed that. This would mean that the local re-ordering step (lines 25–26 in Alg. 7.7(b)) performed two different ordering outcomes at π_1 and π_2. As the re-ordering module used in Alg. 7.7(b) is deterministic (*i.e.*, same input necessarily leads to same output), this is impossible.

3. There has been long-term fragmentation, thus invoking the contention-breaking procedure for contention between fragments (invalidation case 2): This would mean that the consensus protocol reached two different decisions u and u'' for the same session $<R, s_1>$, after the re-connection of fragments, violating the agreement property. Since the invalidation procedure for long-term fragmentation picks one single value to eventually converge to (*i.e.*, value from the winning fragment/user), and CoNICE ensures agreement even in case of global consensus, this case is impossible.

4. There has been duplicate decisions and thus a new attempt procedure is invoked (invalidation case 3): Similar to the previous case, this would require that the consensus session for $<R, s_1>$ reach two different values u and u'' (with similar situation for u'), after a new attempt. Given that CoNICE ensures agreement in each of its attempts (no matter how many re-attempts happen), this is impossible.

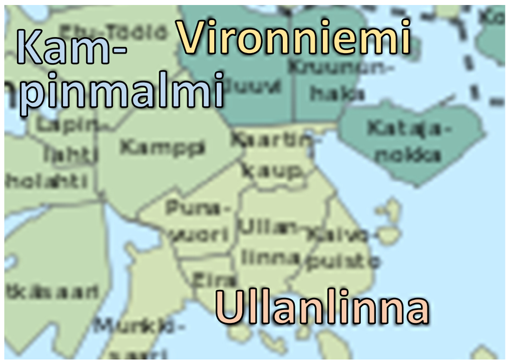


Figure 7.8: Map of our Helsinki-based simulation scenario

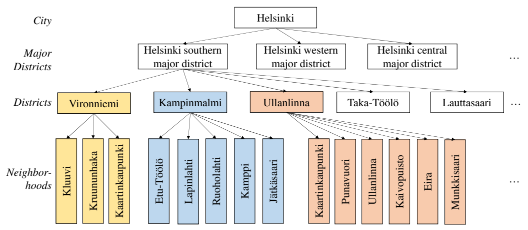


Figure 7.9: Namespace for our Helsinki-based simulation scenario

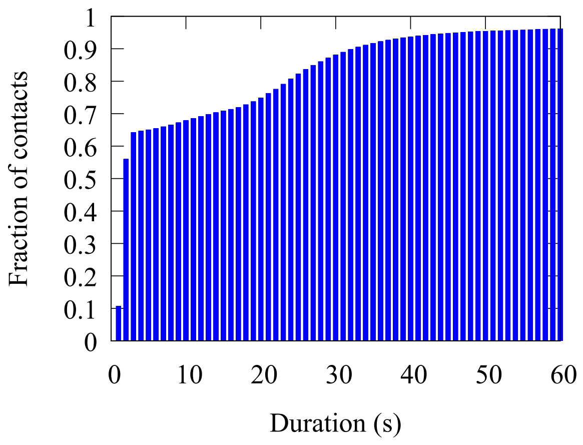


Figure 7.10: Contact duration CDF

Therefore, the initial hypothesis about lack of total order is contradicted, and CoNICE's total ordering is proved. As for causal order, since the outcome of the decided sequences get deterministically sorted (by the aforementioned sorting method in case 2 above) and re-ordered for causal order if necessary, causal order of the Strong view is also guaranteed.

7.6 Evaluation

To evaluate CoNICE, we perform a simulation based on a partial map of the city of Helsinki (Fig. 7.8, [192]) using the ONE simulator [108]. The associated hierarchically-structured namespace (Fig. 7.9) follows the "City$\rightarrow$Major districts$\rightarrow$Districts$\rightarrow$Neighborhoods" structure. Our simulation environment consists of the three districts (and hierarchically, the neighborhoods in them) highlighted in Fig. 7.8, and is 4500×3400 meters large. We model an emergency response scenario where there are 30 pedestrian first responder users (F-users), each dealing with one of the three districts: they are moving in the area, indicate an interest in events in them, and publishing updates for them. There is no networking infrastructure, but all users are equipped with D2D wireless capability. To increase message delivery, we place additional benevolent mules, namely 500 pedestrian civilians (C-users) and patrol vehicles (V-users). V-users move faster, have higher buffer capacity and wireless range than pedestrian users. Benevolent mules participate in relaying and causal delivery of every message they receive (regardless of region). However, they do not participate in any consensus sessions. We assume all users in these scenarios, *i.e.*, F-users, C-users, and V-users, behave honestly and non-maliciously. Mobility is based on map routes, with waiting times of at most 2 min. Each F-user creates three updates in the first half hour of the simulation (thus, total of 90 uniquely created updates), randomly belonging to one of the neighborhoods in their respective district. All messages are 1 KB. We report on two sets of scenarios, one with 1 hour in simulated time and another for 12 hours.

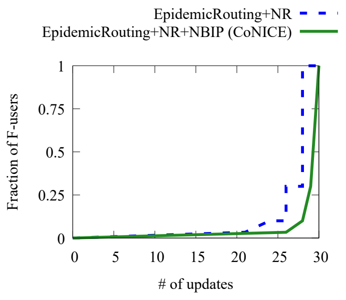


Figure 7.11: Relevant replication coverage CDF

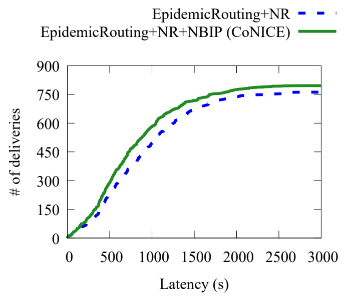


Figure 7.12: Relevant delivery latency (cumulative)

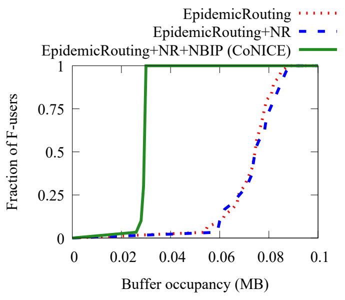


Figure 7.13: Buffer occupancy with level 0

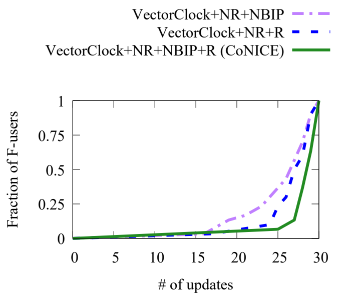


Figure 7.14: Relevant causal completeness CDF

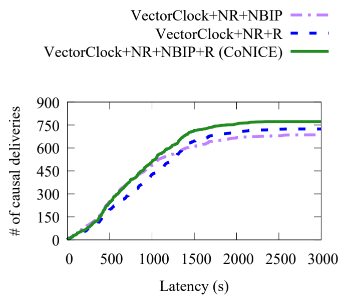


Figure 7.15: Relevant causal delivery (cumulative)

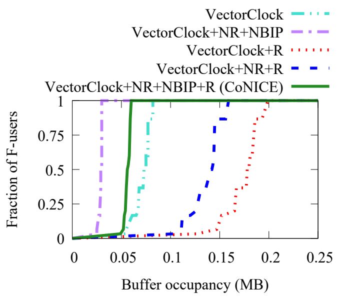


Figure 7.16: Buffer occupancy with levels 0 and 1

7.6.1 Experiments on Gossiping and Causal Ordering

To investigate level 0 and level 1 consistency, we use the 1-hour simulation scenario. There are a total of 59,558 D2D contacts during this time, and the cumulative distribution of contact durations is (partially) shown in Fig. 7.10. As Fig. 7.10 shows, 95 percent of contacts lasted less than 1 minute and 70 percent less than 10 seconds, which demonstrates the highly dynamic nature of the environment. The mobility and contact distribution is the same for all experiments in this sub-section.

Table 7.1: Results for level 0

Metric/Approach		Epidemic Routing	Epidemic Routing+NR	EpidemicRouting+ NR+NBIP (CoNICE)
F-users	Average RC_{rel}	N/A	28.40	29.53
	Average RC_{tot}	73.60	74.76	29.53
	Average RL_{rel} (s)	N/A	852.25	758.89
	Average RL_{tot} (s)	1,080.07	1,084.17	758.89
	Average Buffer Occupancy (MB)	0.07	0.07	0.02
Net-work	Total Relays	49,612	50,123	48,612
	Irrelevant Relays	N/A	1,393	0

First, we focus on gossiping only (*i.e.*, no causal ordering or consensus). We define *replication coverage* (RC) as a metric that shows how many of updates each node has received (albeit out of order). *Total* RC (RC_{tot}) denotes all updates a user received, while *relevant* RC (RC_{rel}) only considers the relevant ones pertaining to the F-user's tasks (can be at most 30). Note that always $RC_{rel} \leq RC_{tot}$, and with the right interest profiling, it is expected that $RC_{rel} = RC_{tot}$. CoNICE's gossiping enhances epidemic routing. As Table 7.1 shows, CoNICE achieves better RC_{rel} than 'epidemic routing+NR' (NR is name-based region-ing for publications), as it adds name-based interest profiling (NBIP). It also achieves better latency with more relevant deliveries (RL_{rel}), (at most can reach 30×30=900). This is due to naming which makes relays and queued messages more useful and relevant. Also, basic epidemic routing that uses no naming (thus, the notion of 'relevancy' is not applicable), receives lower RC_{tot} than when enhanced with NR. Its total relays value is similar to the rest while achieving less. CoNICE achieves higher coverage with lower buffer and network cost.

Table 7.2: Results for Level 1

Metric/Approach		Vector Clock	Vector Clock+ NR+NBIP	Vector Clock +R	Vector Clock+ NR+R	VectorClock+ NR+NBIP+ R (CoNICE)
F-users	Average CC_{rel}	N/A	25.73	N/A	28.30	28.70
	Average CC_{tot}	39.86	25.73	68.30	71.16	28.70
	Average CL_{rel} (s)	N/A	693.23	1088.16	800.18	729.31
	Average CL_{tot} (s)	1,093.01	693.23	1,119.02	1,084.79	729.31
	Average Buffer Occupancy (MB)	0.07	0.02	0.17	0.13	0.05
Net-work	Total Relays	49,612	98,485	108,289	88,134	89,792
	Irrelevant Relays	N/A	0	N/A	2,648	0

We then bring causal ordering into play. CoNICE's causal ordering enhances Vector Clock with the use of NR, NBIP, and Reactive mode (R), in addition to other minor optimizations such as variable-length vectors. Table 7.2 provides a comparison summary. Fig. 7.14 shows the CDF of relevant causal completeness (CC_{rel}), which denotes how many of updates have been causally applied at F-users. As Fig. 7.14 shows, CoNICE achieves better CC_{rel} compared to alternatives that enable NR, since CoNICE allows more selective use of causal ordering overhead (through NBIP) and requesting for unfulfilled prerequisites on demand using the reactive mode rather than waiting. It also achieves better causal latency (CL_{rel}) as Fig. 7.15 shows, and reasonable network overhead in terms of the number of relays (Table 7.2). For cases *without NR*, Table 7.2 shows that pure Vector Clock achieves the lowest RC_{tot}. This is because without name-based region-ing, every update can potentially depend on all others, which results in an extremely high number of references that have to be fulfilled and processed. Name-based region-ing makes appendices more selective, having to only depend on relevant updates. Fig. 7.16 shows CoNICE achieves better buffer usage than most of the alternatives, except 'VC+NR+NBIP' which does not use reactive mode

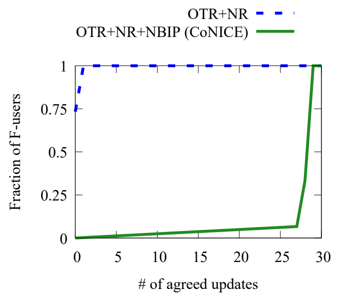


Figure 7.17: Relevant agreement completeness CDF

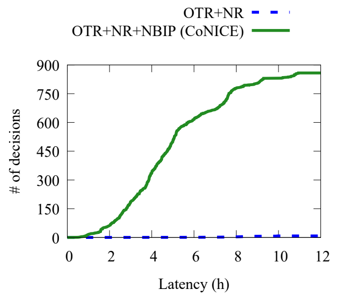


Figure 7.18: Relevant decision latency (cumulative)

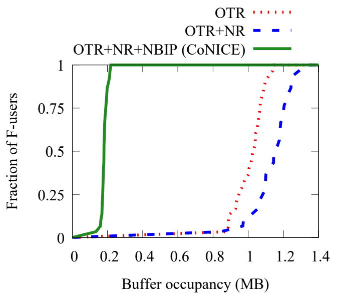


Figure 7.19: Buffer occupancy with levels 0–2

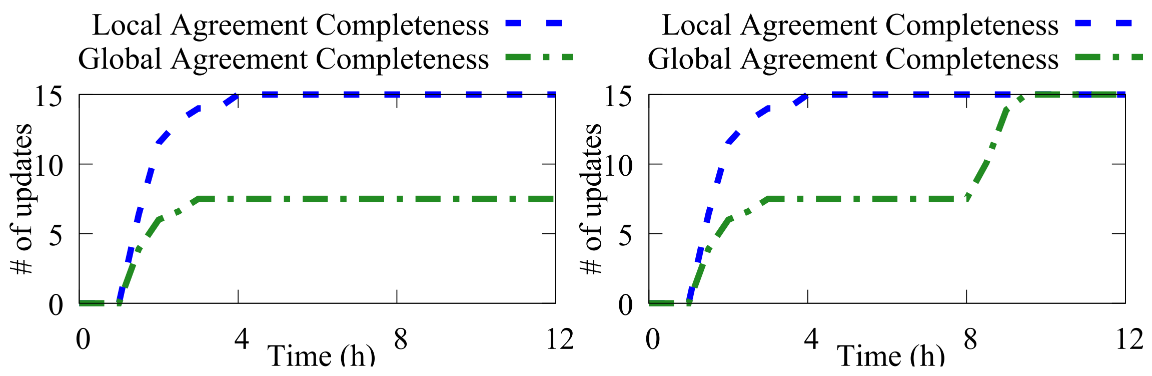
Table 7.3: Results for level 2

Metric/Approach		OTR	OTR+NR	OTR+NR+NBIP (CoNICE)
F-users	Average AC_{rel}	N/A	0.26	28.60
	Average AC_{tot}	0	0.93	28.60
	Average AL_{rel} (h)	N/A	8.29	4.91
	Average AL_{tot} (h)	None	7.51	4.91
	Average Buffer Occupancy (MB)	1.01	1.14	0.18
Network	Total Relays	3,489,035	3,512,598	3,504,557
	Irrelevant Relays	N/A	77,086	0
	Consensus Initiations	2,049	2,101	853
	Consensus Decisions	0	28	858

and achieves lower completeness. As seen, using causal ordering leads to slightly higher latency and buffer usage than pure level 0, but achieves causal order consistency.

7.6.2 Experiments on Consensus

To investigate consensus, we extend our scenario to 12 hours with the total of 683,876 D2D contacts. We now enable level 2, *i.e.*, consensus, on top of levels 0 and 1. After the passage of approximately one hour, users start to initiate consensus sessions for the slots they have content for. We compare CoNICE with the basic OTR, and show the impact of adding NR and NBIP. The *agreement completeness* (AC) metric shows how many of level 2 queue slots of users have been filled with agreed-upon updates. Just as before, we have AC_{rel} and AC_{tot}. As Table 7.3 shows, basic OTR fails to reach any decisions, and thus has zero AC. 'OTR+NR' is slightly better but is still not satisfactory. As the Table, and Fig. 7.17 (CDF of AC_{rel}) show, CoNICE achieves a dramatically better agreement completeness. Fig. 7.17 shows that with CoNICE, 90% of F-users agree on 26 or more updates, while with 'OTR+NR', 75% of F-users agree on zero updates, in the entire 12-hour simulation period. This is due to the fact that CoNICE uses NBIP, which limits the consensus participants only to those that are relevant, namely F-users dealing with neighborhoods within the same district. Table 7.3 also shows that OTR and 'OTR+NR' initiate much higher consensus sessions than CoNICE (2,049 and 2,101 *vs.* 853), but reach significantly fewer decisions (0 and 28 *vs.* 858). CoNICE even reaches more decisions than it initiates, which shows the improvement contributed by level 2 over level 1. This is because due to CoNICE's faster consensus convergence, some F-users can fill their slots in Q_2 the corresponding of which they do not have in Q_1 (as example in Fig. 7.5). Fig. 7.18 shows the cumulative latency of reaching relevant agreement decisions (AL_{rel}) across all F-users. As shown, CoNICE achieves considerably more. As can be seen (and previously shown in [32]), the latency of reaching consensus decisions is on the scale of hours in an intermittently-connected network, while CoNICE's causal order delivery is in the order of minutes (Table 7.2). This shows yet another benefit of going through level 1 first and then level 2: users will have a somewhat useful moderate view in the order of minutes while



(a) Basic OTR (no invalidations) (b) CoNICE (OTR + invalidations)

Figure 7.20: Average local and global agreement completeness

dealing with the incident, while waiting for possibly hours to reach consensus and build a strong view. CoNICE achieves far better agreement completeness, using the same level of relays as other alternatives (Table 7.3), and using much less buffer at F-users as shown in Fig. 7.19. These results show that CoNICE significantly improves on OTR, for achieving higher agreement completeness among users, while also using less buffer capacity. These improvements of CoNICE are greatly beneficial in practical situations such as geo-tagging in emergency response, as first responders can build their consistent strong views much faster and be able to deal with their critical tasks more effectively and efficiently.

7.6.3 Physical Fragmentation of Shelters During Disasters

We now conduct experiments to investigate the effect of long-term fragmentation. For this, we have two long-term fragmented shelters each circular with radius of 50 meters, with 500 meters distance between them. Each shelter contains 5 first responders (F-users), all dealing with the disaster in the 'Vironniemi' district. These F-users are constrained (or trapped) with very limited mobility within their respected shelters. Every F-user creates 3 updates. Thus, we will have a total of 30 updates. All benevolent mule activity is disabled during the first 8 hours of the experiment, with no connection between members of the two shelters. Due to this long-term fragmentation, the two shelters do not hear from each other for the 8-hour period, and we end up with 15 pairwise contentious consensus sessions, *i.e.*, same region-slot pair, from the two shelters.

Fig. 7.20 compares the use of basic OTR (without invalidation), (a), and CoNICE (OTR with invalidation), (b), for achieving both local and global agreement. Local agreement completeness indicates to what extent an average F-user reaches the same decision for an update as the others in the same shelter. Global agreement completeness, on the other hand, represents to what extent an average F-user reaches the "winning" global decision for an update just like the others in the whole network. Both local and global agreement completeness are shown as per-user average, and each can be at most 15.

As Fig. 7.20(a) shows for the basic OTR approach, it is easy for all the F-users to gradually reach complete local agreement before the 8th hour. Since one shelter has the winning advantage, at most half global agreement is reached (*i.e.*, those who are already in the winning shelter), without any invalidation procedure. We also observe the same outcome if we restart the whole consensus procedure with all 10 participants. This occurs because there is insufficient time to reach global agreement among all users. Thus, we do not make progress in achieving global agreement even after 12 hours. Alternatively, as Fig. 7.20(b) shows, with the addition of decision invalidation procedure, CoNICE achieves complete global consensus among all F-users in the network, gradually starting from the

8th hour, which is the time paths start to appear between shelters. This feature of CoNICE helps ensure that all first responders can get a chance to eventually get on the same page regarding the ordering of updates for their respective tasks, in a strongly consistent manner, even if they have been isolated in separate shelters for a long time.

Chapter 8

Leveraging Social Media Posts for Name-based Information Dissemination in Disasters

8.1 Introduction

In this chapter, we explore our work in the integration of social media into managing the disaster response. People (especially common citizens) use the normal forms of social media communications (Twitter, Facebook, etc.) to send and get information during disaster situations, since that is what they are familiar with. This was observed across different parts of the world. We first present an analysis of how people use social media during disasters, using the examples of Hurricanes Harvey and Irma in 2017.

We showed in previous chapters that name-based pub/sub can be quite useful for information dissemination during disasters. However, people are likely to have little knowledge or understanding of the notions of a structured namespace to determine where to publish information or ask for information (e.g., generate an Interest in NDN). It would be ideal to allow the use of these social media platforms in the manner people are used to, with free-form text, possibly enhanced with pull-down menus to determine key meta-information to associate with the message. This then would require mapping individual messages to the namespace for publishing, expressing interest or creating subscriptions. The approach we explore is to process the natural language in the social media posts (Tweets, Facebook posts, etc.) and map it to appropriate names in the namespace. This allows messages to be delivered to the correct entity (such as first responder or incident commander) based on the derived names. Additional handling of the message may be based on the decisions by those individuals (e.g., handling false positives).

We evaluate the applicability of our use of social media information using Tweets collected during (and just after) two recent wildfires in the state of California in the United States. DiRECT uses Natural Language Processing (NLP) to map the tweets to the right first responder(s) automatically. We demonstrate that it does so accurately up to 96% of the time. Even for the ones that are inaccurate (4%), they can be recovered by the help of the first responder, manually forwarding it to the correct recipient.

The contributions of this work are: 1) an analysis of the correlation between social media posts and real-time events during disasters; 2) a system that integrates critical components and actors in disaster scenarios, i.e., first responders, volunteers and social media, in a name-based, information dissemination model; 3) a social media engine that intelligently and automatically maps free-form social media posts to the right names for publication in a pub/sub framework, and 4) demonstration of the effectiveness of our social media engine through measurements from our evaluations.

8.2 The Evolving Nature of Disaster Management in the Internet and Social Media Era

First we analyze social media usage by people during disasters. Data collected from social media (in particular, Twitter) can be very informative about disaster-related issues as it has been widely used for asking and offering help, by government, volunteers, civilians, *etc.* This was observed anecdotally in several news articles soon after Harvey. We crawled the Tweets sent during and just after the hurricanes Harvey and Irma and see if they can potentially answer many useful questions such as *what*, *when* and *where* the need/offer for help occurs. Analyzing this data both qualitatively and quantitatively, we can get insights for the design of communication capabilities to complement traditional emergency service communication.

8.2.1 Keyword-Based Association of Tweets

We implemented an early-stage tweet processing algorithm that:

1. parses large collections of raw crawled tweets, and
2. identifies keywords and performs a phrase-based classification of tweets.

Table 8.1: Tweets per category for Harvey on Aug. 27 between 5:22:52 pm and 6:59:59 pm

Category	**Query phrase**	**#**
Total		561,187
Harvey-related	`harvey* hurricane*`	50,140
Deaths	`death* dead`	6,012
Shelter	`shelter*`	3,552
Damage	`damage*`	1,067
Search & Rescue	`(search)AND(rescue)`	852
Fire	`Fire`	736
Missing Persons	`Missing`	522
Collapsed Infrastructure	`collaps*`	876
Trapped	`Trapped`	382
Forward This Message	`(please this)AND(forward retweet)`	337
Outage	`((electricity power) AND (no out without outage* blackout*))` `outage* blackout*`	301
Shortage	`shortage* suffic* insuffic* ((run* ran are)AND(short low out))`	248
Distribution	`distribut*`	181
Earthquake & Aftershocks	`aftershocks AND earthquake* aftershock*`	157
Need Medical Equipment & Supplies	`(need*)AND(medic* suppl*)`	146
Human Remains	`remains bodies`	114
Looting	`loot*`	84

For the first phase, we use Java JSON Parser to extract those attributes of a tweet that we are most interested in, *i.e.*, mainly `createdAt` showing the time of the tweet, `text` showing the content of the tweet, and `geolocation` field of the tweet as a (latitude, longitude) pair. For the second phase, we use the Lucene [5] library, an open-source text mining engine to determine whether or not a tweet is associated to the disaster. We mine the `text` field of the tweets to get an understanding of what a tweet is about. Additionally, a dictionary construction program on the tweet pool gives us the frequency of each word and also the top k most frequent words, thus allowing us to learn what words are most popular in a tweet collection.

We analyze temporal and spatial distribution of incident-related tweets. For the tweets we crawled from the approximate one-week duration of Harvey, we identified Harvey-related tweets by counting the results of the `harvey* hurricane*` query (similarly for Irma, with the keyword `irma*`). We used the `createdAt` and `geoLocation` fields to plot the temporal and spatial distribution of the hurricane-related tweets.

Fig. 8.1 shows the percentage of the Harvey-related tweets during the crawling periods on Twitter, showing how the ratio of tweets related to Harvey was higher during the peak of the incident. Aug. 25th was the day Harvey made landfall while Aug. 27th was the day of considerable flooding. Fig. 8.2 shows the percentage of Irma-related tweets for continuous 3-hour periods

between the night of Sept. 9 and the evening of Sept. 11. We observe that Irma-related tweets tracked the progress of the hurricane. It is interesting to note the correlation

Table 8.2: Categories for Irma Tweets during Sept. 9th - Sept. 12th

Period start	**9/09 9:27pm**		**9/10 9:00am**		**9/10 9:00pm**		**9/11 9:00am**		**9/12 6:25pm**		**9/12 9:00am**	
Period end	**9/10 8:59pm**		**9/10 8:59pm**		**9/11 8:59am**		**9/11 7:59pm**		**9/12 8:59am**		**9/12 7:59pm**	
Category	#	**per Hour**	#	**per Hour**	#	**per Hour**	#	**per Hour**	#	**per Hour**	#	**per Hour**
Total	1939504	167922.42	3640540	303378.33	2350414	195867.83	3107882	282534.73	381807	147987.21	2960088	269098.91
Irma-related	214245	18549.35	464809	38734.08	252559	21046.58	217026	19729.64	15616	6052.71	88055	8005
Outage	6721	581.90	15031	1252.58	13937	1161.42	21097	1917.91	2439	945.35	13933	1266.64
Deaths	6407	554.72	10858	904.83	11817	984.75	19357	1759.73	1879	728.29	15641	1421.91
Shelter	11931	1032.99	26958	2246.5	7384	615.33	7673	697.55	359	139.15	2314	210.36
Damage	2188	189.44	8610	717.5	8017	668.08	15356	1396	1268	491.47	8287	753.36
Looting	200	17.32	10383	865.25	13782	1148.5	12172	1106.55	434	168.22	2142	194.73
Fire	3068	265.63	6377	531.42	4700	391.67	5251	477.36	678	262.79	6605	600.45
Forward This Message	186	16.10	489	40.75	307	25.58	403	36.64	320	124.03	15919	1447.18
Missing Persons	2022	175.06	2986	248.83	1851	154.25	3155	286.82	290	112.40	3086	280.55
Shortage	1363	118.01	2375	197.92	1116	93	2171	197.36	238	92.25	2169	197.18
Human Remains	2031	175.84	1997	166.42	1446	120.5	1838	167.09	190	73.64	1349	122.64
Collapsed Infrastructure	211	18.27	3784	315.33	705	58.75	752	68.36	28	10.85	547	49.73
Earthquake & Aftershocks	803	69.52	1745	145.42	481	40.08	430	39.09	63	24.42	384	34.91
Distribution	750	64.94	854	71.17	283	23.58	456	41.45	65	25.19	609	55.36
Trapped	230	19.91	473	39.42	391	32.58	319	29	40	15.50	273	24.82
Need Medical Equip.&Supplies	193	16.71	383	31.92	150	12.5	310	28.18	53	20.54	434	39.45
Search & Rescue	77	6.67	89	7.42	253	21.08	627	57	43	16.67	146	13.27
Road Blocked	16	1.39	127	10.58	141	11.75	345	31.36	93	36.05	177	16.09
Contaminated Water	122	10.56	157	13.08	151	12.58	215	19.55	6	2.33	136	12.36
Unstable	42	3.64	78	6.5	48	4	88	8	17	6.59	88	8
Rubble	26	2.25	62	5.17	38	3.17	66	6	6	2.33	39	3.55
Medical Emergency	22	1.90	36	3	18	1.5	99	9	4	1.55	51	4.64
Water Sanitation&Hygiene	7	0.61	50	4.17	18	1.5	92	8.36	2	0.78	36	3.27
Security Concern	5	0.43	20	1.67	23	1.92	25	2.27	5	1.94	12	1.09

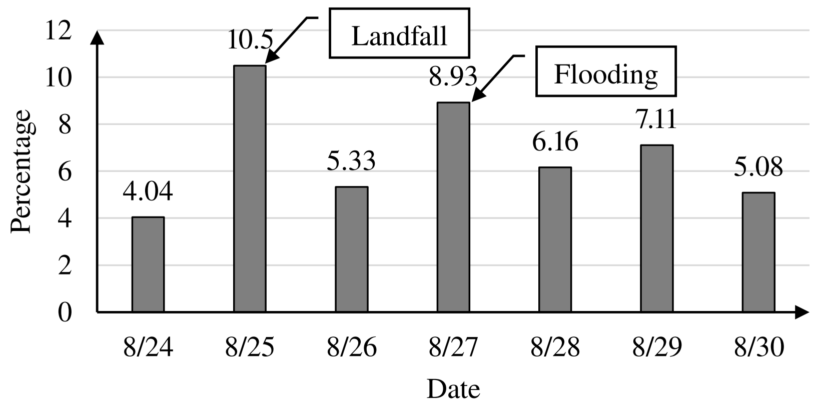


Figure 8.1: Percentage of Harvey-related Tweets over total collected tweets

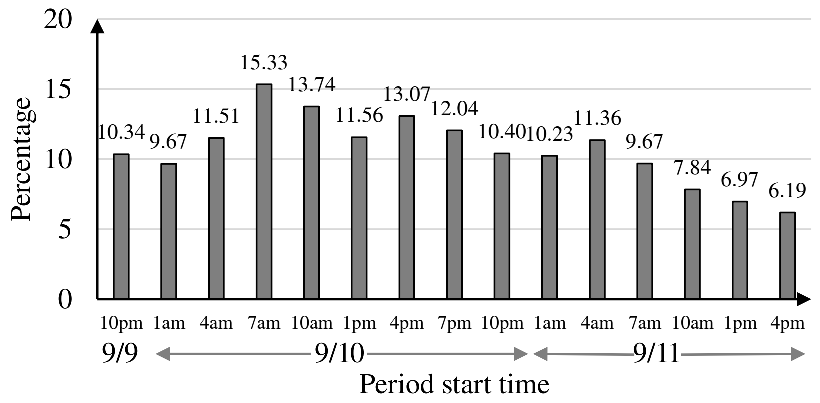


Figure 8.2: Percentage of Irma-related tweets during Sept. 9-11 over 3-hour periods

between these results and real events: According to [23], "… Irma was upgraded to a Category 4 …" on Sept. 10 and "… downgraded to a Category 1 …" on Sept. 11. As for Florida (where most crawled tweets are from), "Hurricane Irma pummeled the Florida Keys late Saturday (Sept. 9) into Sunday (Sept. 10) as a Category 4 and hit the Florida mainland as a Category 3 storm around 1pm eastern time Sunday …" [195]. While the frequency of tweets on a topic may rely on many different factors, we observed that Harvey-related tweets are more frequent during the peak of the hurricane.

8.2.2 Categorizing Tweets for Disaster Management

Once disaster-related tweets are identified, we classify the tweets according to what the tweeter is requesting/offering regarding the disaster, *e.g.*, requesting or offering aid, volunteering, reporting, or complaining. We identified a set of disaster-related categories and show their frequency in Tables 8.1 and 8.2 for Harvey and Irma, respectively. We picked the query phrase associated with each category after some trial and error to get a reasonable accuracy rate.

For Harvey, the tweet count for each category is shown in Table 8.1. We found the most frequent topics in Harvey tweets to be on "deaths", "shelter", and "damage". For Irma, we did a more comprehensive classification. Table 8.2 shows the tweet count for each disaster-related category for three days (Sept. 10, 11 and 12) in 12-hour intervals (starting from 9pm). We used the same category list and search keywords as we did for Harvey. As time values for different periods differ, we also show the *tweets per hour* which is the tweet count divided by the number of hours. This is very helpful for fair comparisons. Most of the trends observed in the table correlate with real progression of events. For example, It is interesting to note that the most frequent Irma tweet category was "outage" which was not a frequent category in Harvey. For "outage", we observe that the related tweets increase from Sept. 10 (1,252.58) to Sept. 11 (1,917.91) and decrease on Sept. 12 (1,266.64) during the daylight hours 9am–9pm (a similar pattern, with lower numbers, is seen during the night time, 9pm–9am). According to [65], power outage increased till Sept. 11, peaked, and then decreased after that. That seems to be similar to the numbers in our results. The aforementioned article also states ".. power outages peaked at 3pm on Sept. 11, affecting 64% of customers .." which also correlates with the results; as we see an increase-then-decrease pattern, peaking on Sept. 11. Looking at other categories, "Looting" reports go up first (immediately after event) and goes down afterwards – probably because of law enforcement. Likewise, "Shelter" requests go down over time. Similar trends can be observed for "Deaths" and "Damage" categories. There may be anomalies too; *e.g.* "Forwarding of message" goes strangely up on Sept. 12. This may be due to excessive retweets of one tweet.

8.3 System Model for DiRECT

The system we consider is one where people involved in a disaster (first responders, victims and volunteers willing to help) can communicate easily using a name-based communication framework. Each participant may have an associated *role*, and the namespace captures the relationships among these roles. We expect that the *incident commander* creates, updates and manages the namespace, which may follow a structure that is similar to the specifications in the National Incident Management System (NIMS) [71], reflecting the

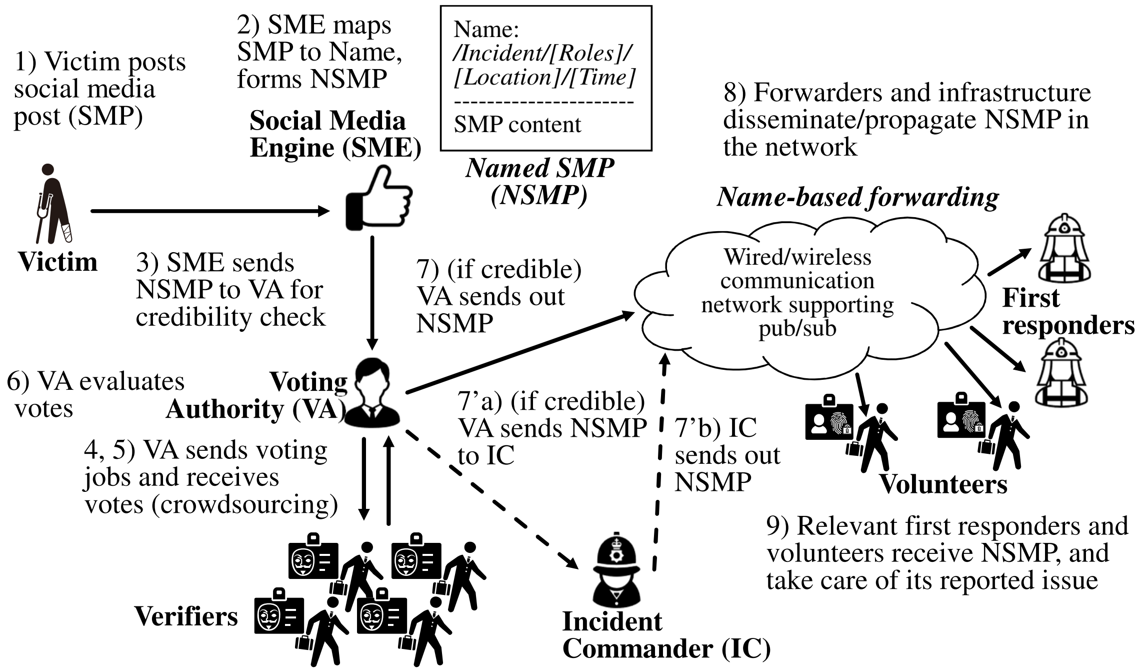

1) Victim posts social media post (SMP)
2) SME maps SMP to Name, forms NSMP
Name: /Incident/[Roles]/[Location]/[Time]
SMP content
Named SMP (NSMP)
Social Media Engine (SME)
8) Forwarders and infrastructure disseminate/propagate NSMP in the network
Name-based forwarding
Victim
3) SME sends NSMP to VA for credibility check
7) (if credible) VA sends out NSMP
Wired/wireless communication network supporting pub/sub
First responders
6) VA evaluates votes
Voting Authority (VA)
7'a) (if credible) VA sends NSMP to IC
7'b) IC sends out NSMP
Volunteers
4, 5) VA sends voting jobs and receives votes (crowdsourcing)
9) Relevant first responders and volunteers receive NSMP, and take care of its reported issue
Verifiers
Incident Commander (IC)

Figure 8.3: System model

organizational roles of participants. Volunteers, first responders and incident commanders publish into the namespace, with messages that are relevant to the disaster and intended to reach the relevant people. They also subscribe to names or name prefixes based on their role and the information they expect to receive. Information can be accessed by the name, using an Interest/Data framework as in NDN [203] as well as using a publish/subscribe framework such as the recipient-based pub/sub in [48]. An important consideration is that the namespace is likely to dynamically change as new incidents [48] occur or when new roles are added or assigned to specific first responders and volunteers. Generally, exchanged messages refer to *events*. An event is defined as a 3-tuple of $\langle$*Type*, *Time*, *Location*$\rangle$; an example for an event is $\langle$*Fire*, *11am April 1*, *Yamadaoka Suita-city*$\rangle$. The event type is a task or issue associated with a role.

The overall procedure is shown in Fig. 8.3. A *victim* (or any *data initiator*) posts a disaster-related message (report, update, etc.) on social media, e.g., tweeting, in free-form text. A *social media engine* (SME) collects, analyzes and maps *social media posts* (SMPs) to a name, of form "/Incident/[Role]/[Location]/[Time]" according to the incident namespace (described in §8.4.1), using NLP/ML procedures (described in §8.4.2).

After the name-mapping procedure, and for the purpose of fact-checking social media posts, the SME sends the "named SMP" (NSMP) to the *verification service* which is a crowdsourcing service involving a set of volunteers that vote on the veracity of posts, referred to as *verifiers*, who help in establishing the trustworthiness (credibility) of the NSMP. The *voting authority* is a trusted third party which manages the verification service. The voting authority eventually sends the verified NSMP to the incident commander or to those first responders that are appropriate (with the choice depending on the real-time-criticality of the matter) who are part of the namespace. The *trust management* evaluates the trustworthiness of verifiers in performing the verification task, which helps with ensuring the spread of only credible information in the network. A *first responder* receives a command from the incident commander as well as a verified NSMP from the voting authority.

In addition, we also consider *volunteers* who seek to actively participate in the disaster response. A volunteer registers herself/himself with a volunteer service by providing credentials, including potentially biometric information, under an assumption that volunteers who provide their biometrics would never behave maliciously. Since these volunteers are trusted, they are allowed to publish messages and subscribe to names in the incident namespace.

Our design assumes the initial interaction between users and social media servers (e.g., Twitter) over the Internet. However, the communications in the pub/sub framework, e.g., between first responders and volunteers, can be a combination of infrastructure-based (e.g., through fixed routers) or infrastructure-less (e.g., device-to-device, through data mules and/or drones) communication links. The use of information-centric dissemination allows for information delivery over such diverse links [166], especially to deal with situations such as mobility and network link failures. We assume that the incident commander and first responders are honest, while volunteers may not all be honest. First responders and volunteers are assumed to have identifiers and credentials. The incident commander and first responders have certificates issued by an incident authority, that is a trusted third party and works as the root certification authority. Volunteers are assumed to not have certificates issued by the trusted third party and their credentials are self-certified. Moreover, volunteers may have multiple pairs of identifiers and credentials.

8.4 Architectural Components

To facilitate the desired bridging between social media platforms and publish/subscribe frameworks, DiRECT uses a namespace that unifies and organizes the information exchanges, a social media engine that maps free-form social media posts to the right names through NLP/ML techniques, and a verification service that assesses the credibility of social media posts through crowdsourcing verifiers and reputation systems. This section explains these key components.

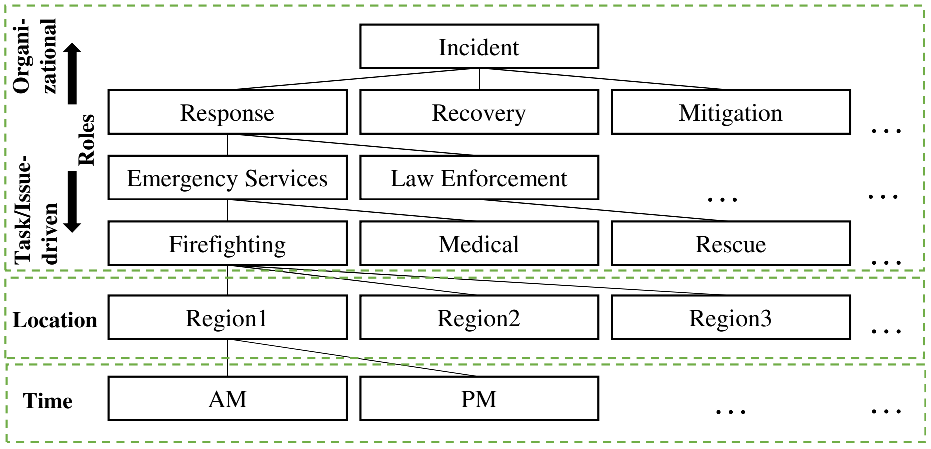


Figure 8.4: Incident management namespace

8.4.1 Naming Schema

Naming is a key component of DiRECT, as it unifies the interactions between all different actors (civilians, first responders, etc.) and guides the subscription and publication paths. DiRECT's namespace follows an NDN-style, hierarchical structure [203] (it can be extended to a graph-based namespace as well, as proposed in [93]). The namespace represents entities related to and critical in incident management, and captures complex relations among them. An example namespace is (partially) depicted in Fig. 8.4. The namespace in DiRECT has the structure of "/Incident/[Role]/[Location] /[Time]", where elements within brackets "[...]" can be any number of name components. Each dimension may correspond to some aspect of an incident and can contain any number of name levels in the namespace hierarchy. For example, the 'Role' dimension can consist of organizational roles (e.g., NIMS [71]) towards the root, and task- or issue-driven (i.e., incident-specific) roles towards the leaves of the tree, to cover the critical components for managing the incident. This design is suitable to model the *what*, *where*, and *when* aspects of content, which are critical aspects of incident information.

The namespace follows a recipient hierarchy for dissemination, as proposed in CNS [48] for a recipient-based pub/sub. The paths followed for publication/subscription for recipient-based pub/sub is the reverse of topic-based pub/sub [49]. In DiRECT, subscribing to a prefix, implicitly means also subscribing to all ancestors of that prefix. For example, a subscriber of "/Incident/Response/EmergencyServices/Firefighting", would receive all publications corresponding to that name, as well as publications to names above it, i.e., "/Incident/ Response/EmergencyServices", etc. Conversely, a content published to a prefix, implicitly means it will also be disseminated towards all subscribers of the descendants of the prefix. For example, a content published to "/Incident/ Response/EmergencyServices/Firefighting/Region1" will also be received by subscribers of "/Incident/ Response/EmergencyServices/Firefighting/Region1/AM", etc.

First responders and volunteers subscribe to prefixes; e.g., a fireman dispatched to fight a fire in Region1 during AM hours, subscribes to "/Incident /Response/EmergencyServices/Firefighting/Region1/AM". The use of recipient-based pub/sub is very beneficial in our architecture since it allows the most relevant first responders, at the finest granularity possible, to receive incident-related publications, without causing distraction and information overload for 'non-relevant' first responders that may be busy dealing with other tasks. Also, leveraging the hierarchical structure of the namespace, and allowing subscription/publication to any desired granularity (and not necessarily each individual below/above a prefix) greatly decreases the number of pub/sub messages and state that has to be maintained in the network, as shown in [48, 49].

An incident commander manages, creates and updates the namespace. At the beginning of an incident, the initial namespace is derived from an a priori template. This template is pre-defined and follows the incident command chains designed specifically for that particular type of incident, e.g. [71]. A benefit of having this template is that it is also accompanied by a trained data set to enable supervised classification performed by the Social Media Engine (SME) during the incident, as we explain in §8.4.2. As the incident progresses and new issues and tasks arise, either from monitoring the incident or suggestions/offers from volunteers, the incident commander dynamically modifies the (more fine-grained parts of the) namespace and notifies the nodes that have it or use it (such as rendezvous points [93]) for synchronization. For example, upon receiving an aid offer from a volunteer team suggesting that they want to help with firefighting in Region4 during PM hours, the incident commander adds the sub-tree "Region4 $\rightarrow$ PM" as a child of "Firefighting" in the namespace. For encryption and authentication, we envisage the use of Attribute-Based Encryption (ABE) [33] with rules and attributes following our namespace. However, we are unable to focus on those issues due to limited space.

8.4.2 Social Media Engine

DiRECT's recipient-based pub/sub allows social media posts (SMP), e.g., Tweets, to be sent as publications and disseminated in the network. However, the correct delivery of each publication depends on the name it has been published to. A civilian (e.g., a victim reporting an emergency and seeking help) may not have knowledge of or access to the namespace to pick the right prefix. Having users download the whole namespace (or have it pushed to them proactively) is costly both in terms of network usage and storage on a user device. Users would also have to manually go through a potentially large namespace to pick the right name for their publication which can be time-consuming and error-prone. DiRECT solves these problems by employing a social media engine (SME) that intelligently maps a SMP to the right part(s) of the namespace using natural language processing and machine learning.

Fig. 8.5 shows the overall architecture of SME: The incoming SMP, possibly including latitude/longitude, and timestamp, in addition to the text, goes through a sequence of stages to get mapped to a (set of) name(s) of the namespace structure shown in Fig. 8.4. This pipeline processes SMPs in an online fashion, i.e., as SMPs arrive and are captured from each social media platform. Using trained models for text (data from previous and/or similar incidents), the classification procedure maps the textual part of the SMP to the right roles, depending on what tasks and/or issues the SMP is referring to. The classes (labels) of this classification are the leaves of the 'Roles' part of the namespace only; i.e., tasks and issues only. Using maps and other geo-related databases, the localization step maps lat/long (and possibly location names mentioned within the text) to the right location names under the previously derived role names. Finally, the temporalizing step maps the timestamp in the SMP to the right time-related name, i.e., which time interval this particular timestamp belongs to. Having formed a complete name, the SME appends the name to the SMP and sends it forward, to the voting authority to be further processed and disseminated.

It is worth noting that SME does not determine the importance of an SMP. It merely takes care of the mapping to a name for directing the delivery. Despite this map-

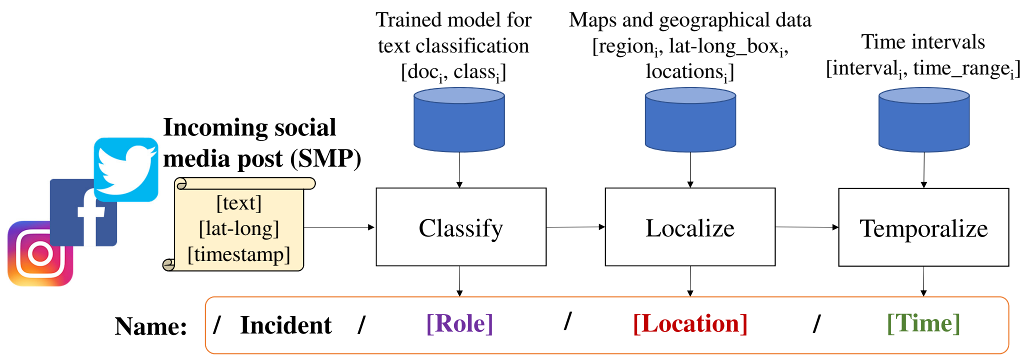
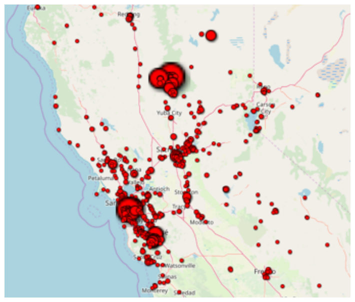

Trained model for text classification [doc$_i$, class$_i$]
Maps and geographical data [region$_i$, lat-long_box$_i$, locations$_i$]
Time intervals [interval$_i$, time_range$_i$]
Incoming social media post (SMP)
[text]
[lat-long]
[timestamp]
Classify
Localize
Temporalize
Name:
/ Incident / [Role] / [Location] / [Time]

Figure 8.5: Social media engine

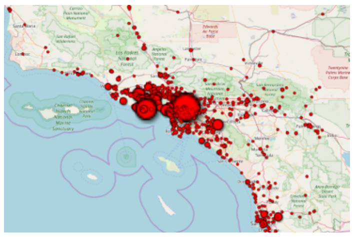
ping, there is chance for inaccuracy, i.e., an SMP is mapped to the wrong name, and thus delivered to the wrong first responder(s). If that happens, e.g., if a medical doctor receives a report regarding an urgent need for firefighting, he/she can either: 1) re-publish the SMP to the network picking the right names (as he/she has access to the namespace as a first responder); or 2) send it as a unicast message to his/her incident commander, who then can forward it as appropriate. This step recovers DiRECT from inaccurate deliveries. However, using a good classification/learning approach will greatly reduce these inaccuracies, so that only a very small percentage require subsequent correct forwarding.

8.5 Preliminary Evaluation

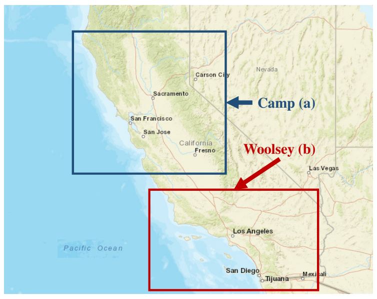
In this section, we provide preliminary results of our evaluation of our trust management and social media engine.



(a) Camp (Northern California)



(b) Woolsey (Southern California)



(c) Bounding boxes for Camp and Woolsey

Figure 8.6: Spatial distribution of fire-related tweets for Camp Fire (a) and Woolsey Fire (b) – (size of circles correlates with the number of tweets)

8.5.1 Social Media Engine

The SME processes social media posts using NLP & ML methods to map incoming posts to the right names. To evaluate the performance of SME in our context, we use Tweets collected from two disasters in California in 2018, namely the Camp [190] and Woolsey [193] wildfires. Crawling Twitter, we captured many Tweets (with no restrictions on keywords) sent from Nov. 7th to Nov. 26th, 2018 (several million) for each fire into the "Camp" and "Woolsey" tweet *pools* (un-processed collections). The geo-location bounding boxes for the two pools are shown in Fig. 8.6(c). The geo-location restrictions were chosen according to the facts of the two wildfires [190, 193].

The Camp and Woolsey pools have 959,740 and 1,961,131 tweets respectively. We show the spatial and temporal distributions of fire-related tweets, doing a keyword-based text mining for tweets that include combinations of the words "fire" or "Camp/Woolsey", using the Apache Lucene API [5]. The analysis results are shown in Fig. 8.6 and Fig. 8.7, indicating strong correlation of tweet patterns with the actual progression of the events

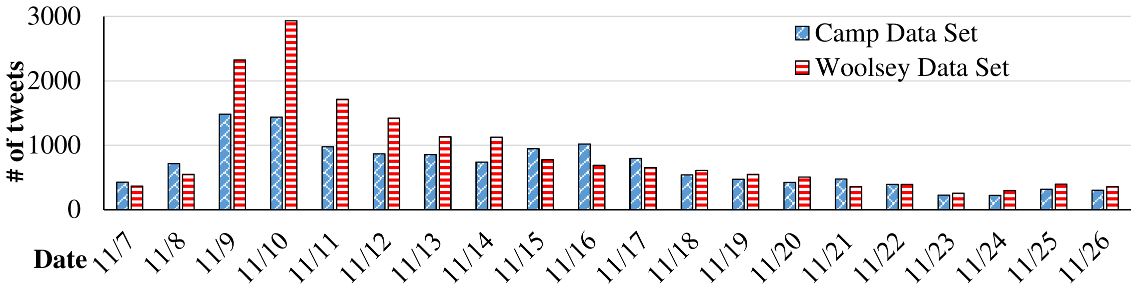


Figure 8.7: Temporal distribution of fire-related tweets

during the two wildfires: peaking around Nov. 9-10 and getting contained around Nov. 21-25 [190,193]. Fig. 8.6 shows that the density of fire-related tweets is higher at areas most affected by the wildfires, i.e., Paradise, CA (Camp fire), and Thousand Oaks and Malibu, CA (Woolsey fire) [190,193].

SME's NLP and ML procedures use the NLTK [35] and Scikit-learn [154] toolkits. Our implementation is in Python, and we evaluate the performance using a machine with Ubuntu 14.04.6 LTS using Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz dual-socket with 14 cores each with hyper-threading enabled with 252GB RAM. To evaluate the performance, we use a subset of our Camp and Woolsey pools; to produce our Camp and Woolsey *data sets*. We identify a total of ~35K tweets across the two pools related to the wildfire incidents. We identified 13 classes (tasks/issues in namespace) and annotated them based on keywords related to each class. The classes picked are based on what we felt were the most important issues and roles during wildfires, in accordance with FEMA reports [70]. Fig. 8.8 shows these classes and the number of instances for each class in the two data sets, combined. 'Firefighting' has the most tweets associated with it (around 75%), which is reasonable since the incident is fire-related. Inference, i.e., assignment of a

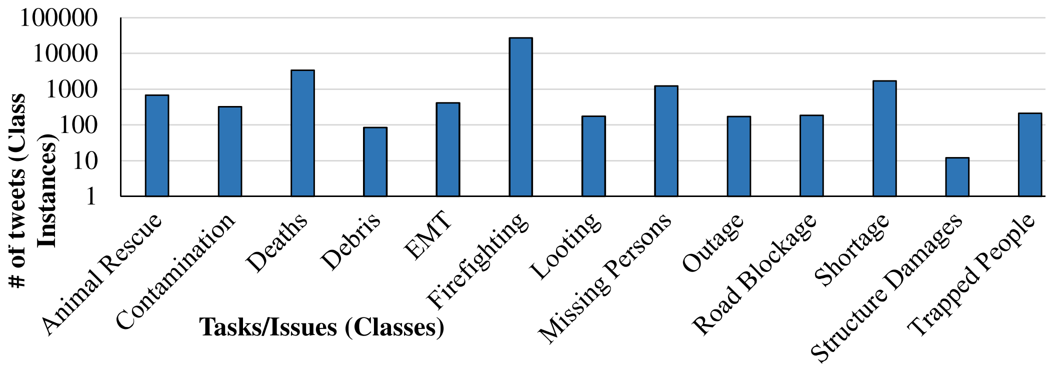
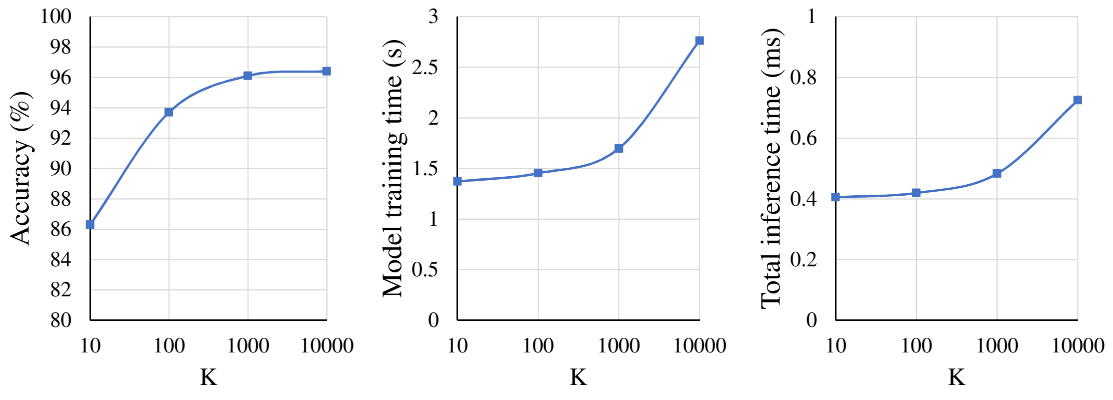


Figure 8.8: Per-class distribution of disaster-related tweets

tweet to a class, is followed by mapping it to a name. Each class corresponds to a name in the namespace; e.g., a tweet classified as a 'Firefighting' instance, would be mapped to "/Incident/Response/EmergencyServices/Firefighting/..." (according to the namespace, e.g., Fig. 8.4). The time and location of the name can be derived from geo-location and timestamp of the tweets. Then the SMPs are sent to verifiers for accuracy and credibility. Additionally, methods similar to content-based fake news detection [81] can be leveraged for faster, more thorough, and automated fact checking of SMPs by the SME.

Our learning procedure consists of tokenization, filtering out unwanted tokens (stopwords, etc.), stemming, and vectorization. For vectorization, we use tf-idf [154], allowing n-grams of size 1 and 2. For inference, we use Random Forest classification. An important feature of DiRECT is its processing of tweets in an online way; thus, we use training data from a previous and/or similar incident, to enable an accurate and fast classification of new tweets. In our experiment, we use the Woolsey data set (with ~23K tweets) for training, and the Camp data set (with ~12K tweets) for testing. Tweets from the Camp data set (i.e., test set) are processed one by one, classified into one of the classes using



(a) Accuracy (b) Model training (c) Total inference

Figure 8.9: Performance of the SME procedures

Table 8.3: Inference metrics for K=1000

Metric	Precision	Recall	F1-score
Micro average	0.96	0.96	0.96
Macro average	0.88	0.81	0.84

the trained model based on the other similar data set, namely the Woolsey data set, to be mapped to an appropriate name.

Feature selection is an important step in learning, as a good feature selection prevents overfitting and reduces processing time. We use K-Best feature selection using the chi2 method [154], which intelligently picks the top K most relevant features, and perform training and classification based on them. There are ∼20K features in our classification We use values of 10, 100, 1000, and 10000 for the value of K in the 'K-best' feature selection process and compare the results, as displayed in Fig. 8.9. The tf-idf vectorization takes

2.68 seconds. Fig. 8.9(a) shows the accuracy (calculated according to [154]) for different values of K: it shows that for values of K at or above 100, we reach accuracies of above 93%, which we believe is very good. For K from 100 to 10000, the accuracy does not change much, while the model training time (shown in Fig. 8.9(b), excluding tf-idf vectorization step) keeps increasing for increasing K. Fig. 8.9(c) shows the total inference (classification) time for all (12,697) test tweets; per-tweet average inference time would be the values shown divided by 12,697. The results show the increasing growth rate and also reasonably low execution times. These latency values (both training and inference times) are important in online settings for the server-based SME. The small inference times demonstrate that using DiRECT, we can quickly (in the order of microseconds) classify a tweet and map it to the right names (and therefore to the right first responders who can help), with high accuracy. Generally, the training data can be either from another incident having high similarity with the current incident (as we do here), or new tweets arriving from the current incident. In the former case, training needs to be done only once. In the latter case, which uses a more relevant training set (as every incident may have its unique characteristics), the training needs to be periodically done with the new data added, i.e., re-training. For faster model re-training, using incremental learning methods such as [163] can be leveraged, albeit sacrificing some accuracy.

Taking into account accuracy, training and classification times (Fig. 8.9), we pick K=1000 as a reasonable value for the feature set. It achieves 96% classification (and thus mapping to the correct name) accuracy which is very good: it means that out of 12,697 tweets, 12,189 of them get to the correct first responders with the publisher. Note that we are assuming that the civilian user does *not* know anything about the namespace, and we are performing the mapping automatically. Only 508 tweets would be inaccurately delivered, which can be appropriately forwarded manually afterwards; this shows the significant benefit of using DiRECT. Table. 8.3 shows other metrics of our algorithm which are important as they answer different questions about the inference performance and its practical usefulness in name mapping. These metrics are Precision (e.g., of all messages sent to firefighters, how many were actually about fire?), Recall (e.g., of all the messages actually about fire, how many did we send to firefighters?) and F1-score (weighted average of Precision and Recall), both as micro and macro averages (calculated according to [154]). Macro average metric values are a sum of metrics for all classes, divided by the number of classes. Micro average metrics, on the other hand, take into account the number of per-class instances, thus giving a more fine-grained averaging. As seen in the table, macro average values are less than micro average values. Note that micro average values (all 96%) are a better metric for our data set since our data set is not a balanced one (75% of instances belong to a single class, namely 'firefighting'). These results show the good performance of our learning/inference at the social media engine, indicating the effectiveness of DiRECT in mapping social media posts to the right names leading to the relevant first responders and volunteers.

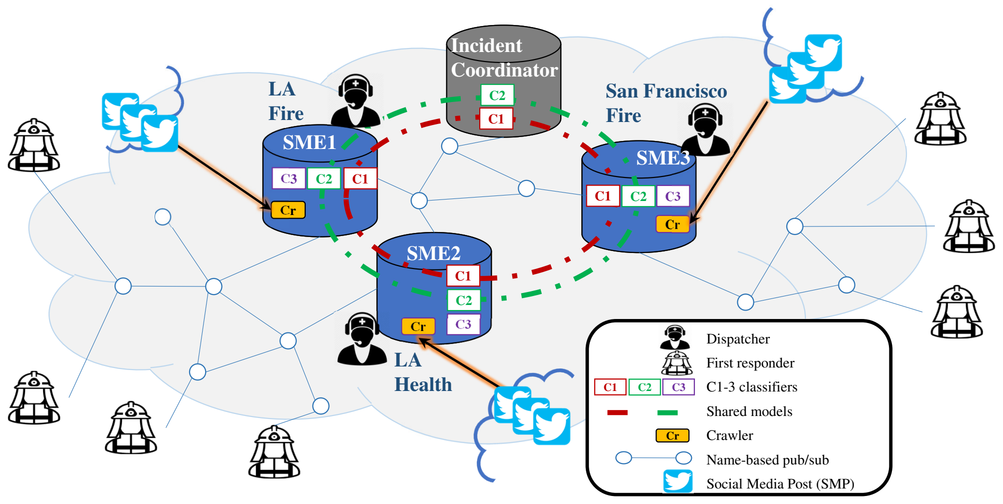


Figure 8.10: Social media engine capable of federated and active learning

8.6 Extending Social Media Engines with Federated and Active Learning

The design we presented in DiRECT can be extended for more effective and efficient learning and classification, integrated with name-based delivery. We have designed additional systems in [135, 136], that introduces federated and active learning for the SME.

A high-level view of the architecture of this framework is presented in Fig. 8.10. This is a framework that uses 'Social Media Engines' (SMEs) to map social media posts (SMPs), such as tweets, to the right names. SMEs perform natural language processing (NLP)-based classification and exploit a number of machine learning capabilities, in an online real-time manner. Unlike the previous version, here we have multiple SMEs, each associated with a particular emergency response department/organization. In order to reduce the manual labeling effort required for learning during the disaster, we leverage active learning complemented by dispatchers. Each dispatcher is associated with a particular

SME/department, and has specific domain-knowledge regarding that department. This system also leverages federated learning across the departments/SMEs with specialized knowledge to handle notifications related to their roles in a cooperative manner. Similar to DiRECT, the system is integrated with a disaster management namespace design that guide the name-based delivery (to first responders and appropriate components). The classification performed in the data path for SMPs is broken into three different classifiers, namely C1 (incident relevance predictor), C2 (organization predictor), and C3 (fine-grained role predictor), and each SME is associated with a specific subset of the namespace graph. More details of this work are presented in [135, 136].

Chapter 9

Conclusion and Future Work

The major demands in today's Internet are content-oriented services, which led to the proposals and developments for the new paradigm of Information-Centric Networks (ICN), such as Named Data Networks (NDN) and MobilityFirst (MF). ICN treats content as first-class citizens and supports names in networking, different from today's heavily location/address-based IP networks. In this dissertation, we explore the various aspects of naming, and propose solutions for correctness of name-based networks, improving the efficiency and scalability of name-based delivery and dissemination, and enhancing its applicability with real-world applications with the integration of machine learning.

First, we developed solutions for checking the correctness of ICNs. Chapter 4 proposes Name Space Analysis (NSA), a data plane verification framework for NDN, which includes essential NDN-specific verification applications of content reachability test (to detect name space conflicts, content censorship-freedom, *etc.*), name-based loop detection, and name leakage detection. We also design a name registry method to detect and resolve name space conflicts in the data plane. Applied to the NDN testbed, we found a number of data plane errors through NSA's automatized verification. Our evaluation results on various test cases show the effectiveness, efficiency, and scalability of NSA.

Chapter 5 paper proposes COIN, a content-oriented interoperability solution as a pragmatic approach to manage evolution towards future Internet architectures. COIN does not change existing architectures (of IP and different ICNs), preservers and uses their key features, enables their co-existence, and is flexible for extensibility and evolvability. Through various scenarios and experiments, COIN was shown to make essential content-oriented services (static and dynamic content retrieval) available to consumers across multiple domains, with reasonable efficiency. Additionally, the chapter presented an Alloy-based formal analysis model for the interoperability framework. We showed how model finding can be used to analyze basic (reachability and returnability) properties across domains. Additionally, our proposed model counting approach analyzes failure and mobility scenarios, which we used to prove the negative impact of certain routing policies (particularly, reverse path forwarding), and the helpfulness of certain mobility-handling mechanisms (particularly, late binding), providing necessary confidence and guidelines for Future Internet interoperability.

In chapter 6, we proposed POISE, an architecture for recipient-based pub/sub for disaster management, supporting free-form graph-based namespaces and automatic load splitting to eliminate traffic concentration based on a novel hybrid graph partitioning algorithm. Our results show that POISE is efficient and scalable, compared to alternatives: its graph-based namespace outperforms the state-of-the-art hierarchical namespace of NDN; its overall network architecture extends the recipient-based pub/sub framework of CNS; its partitioning outperforms the popular graph partitioner METIS/ParMETIS. We also showed that POISE's RP-based name expansion and core migration are effective and beneficial.

Chapter 7 proposed CoNICE, a framework to ensure consistent dissemination of updates among users in intermittently-connected, infrastructure-less environments. It exploits naming and multi-level consistency for more selective and efficient preservation of causal ordering and consensus. We proved CoNICE guarantees consensus with causal and total ordering properties. Additionally, we showed that CoNICE can also solve consensus even in case of long-term fragmentation. Our simulation experiments on an application of map-based geo-tagging in emergency response show that CoNICE achieves a higher degree of replication coverage and causal completeness than state-of-the-art vector clocks with epidemic routing, even while using less network resources. More significantly for consensus, CoNICE achieves a considerably higher degree of agreement completeness than the state-of-the-art asynchronous consensus algorithm, OTR, as it exploits naming, showing the applicability of CoNICE in practical, intermittently-connected scenarios.

Finally, in chapter 8, we proposed DiReCT, a framework to coordinate disaster response with first responders that receive timely relevant information and trusted volunteers. DiReCT bridges civilian-oriented social media platforms with a pub/sub information dissemination architecture for first responders and volunteers. It uses a hierarchical naming schema, and NLP/ML-based social media analysis in social media engines which assign the right name to incoming textual content. Results from our evaluation with real-world incident-related datasets show that DiReCT is effective and efficient in providing accurate mapping between free-form text and pub/sub-based names.

9.1 Future Work

Some potential future research directions are as follows:

- *Graph-based Consistent Pub/Sub Platform for Datacenters.*

  Datacenters networks are the backbone of enterprises. Many pieces of content need to be sent to a set of relevant servers in a datacenter in a one-to-many pattern; that piece of content can be a file (replica) that needs to be stored on some nodes, a database update that needs to be applied to some nodes, a piece of data that needs to be used in computations at some nodes, etc. We can use the graph-based pub/sub of POISE and name-based consensus of CoNICE, which fits the aforementioned use cases of datacenters. Having said that, datacenter networks have unique characteristics and challenges to be considered. One challenge is that a datacenter network is often a tightly coupled topology, prone to high node/link bottlenecks. To address this, we can take topology into account when making graph partitioning decision for the namespace, so packets do not travel too long. Another aspect is leveraging network management specifics (*e.g.*, SDN controllers) for faster consensus solutions, to have stricter real-time guarantees, which is important in datacenter networks.

- *ICN Control Plane Verification and Real-Time update Collection.* The control plane of ICN is becoming increasingly complex, with multiple routing and forwarding strategies at each router. Our work in NSA made the first attempt in ICN verification space, but another layer of verification to explore is ICN control plane verification. We can extend NSA's formalism to describe ICN router configurations, and check for name-based properties (*e.g.*, host-to-content reachability and loop-freedom) across all possible data planes resulted from the configuration, and also take into account changes in the state of the network. Another important aspect is collection changes in network topology, configuration and/or forwarding state. We can leverage network management tools, and potentially leverage centralize controllers to collect new states and forward to the machine performing the verification task.

- *Social Media-Driven Namespace Design.* The work in DiRECT focuses on mapping tweets to the right name in a manually created namespace (*e.g.*, by incident commanders). It can be interesting to also explore the reverse path, *i.e.*, to use social media content to inform and prepare the namespace. This can be helpful as it allows the structure of information flows and the mobilization of first responders to follow the (latent) patterns of reports and help requests in tweets. It is a challenging task as tweets are often noisy and the majority of them are irrelevant. We can leverage Natural Language Processing/Understanding with a combination of supervised and un-supervised learning methods for an intelligent and automatic preparation of incident namespace driven by civilian's social media content.

Bibliography

[1] https://www.cs.ucr.edu/~mjaha001/ICI.zip.

[2] 24. Hash Library – Data Plane Development Kit 20.08.0-rc2 documentation. https://doc.dpdk.org/guides/prog_guide/hash_lib.html.

[3] 6. RCU Library – Data Plane Development Kit 20.08.0-rc2 documentation. https://doc.dpdk.org/guides/prog_guide/rcu_lib.html.

[4] Alloy: A Language and Tool for Relational Models. http://alloy.mit.edu/alloy/.

[5] Apache Lucene. https://lucene.apache.org/.

[6] COIN. https://github.com/SAIDProtocol/ICNInteroperability.

[7] Data Plane Development Kit. https://www.dpdk.org/.

[8] MF Software Release. http://mobilityfirst.orbit-lab.org/wiki/Proto.

[9] NSF Future Internet Architecture Project. http://www.nets-fia.net/.

[10] ORBIT. http://www.orbit-lab.org/.

[11] Poise simulator. https://github.com/SAIDProtocol/NetworkSimulator.

[12] Kubernetes. https://kubernetes.io/, 2019.

[13] Wikipedia: Outline of knowledge. https://en.wikipedia.org/wiki/Portal:Contents/Outlines, 2019.

[14] S. S. Adhatarao, J. Chen, M. Arumaithurai, X. Fu, and K. K. Ramakrishnan. ORICE: An Architecture for Object Resolution Services in Information-Centric Environment. In *LANMAN*, 2015.

[15] Sripriya S Adhatarao, Jiachen Chen, Mayutan Arumaithurai, Xiaoming Fu, and K. K. Ramakrishnan. Comparison of naming schema in icn. In *LANMAN*, 2016.

[16] Alex Afanasyev and Sanjeev Kaushik Ramani. Ndnconf: Network management framework for named data networking. In *ICC Workshops*, 2020.

[17] Alexander Afanasyev et al. Packet fragmentation in ndn: Why ndn uses hop-by-hop fragmentation. *NDN Memo, Technical Report NDN-0032*, 2015.

[18] Alexander Afanasyev, Xiaoke Jiang, Yingdi Yu, Jiewen Tan, Yumin Xia, Allison Mankin, and Lixia Zhang. Ndns: A dns-like name service for ndn. In *ICCCN*, 2017.

[19] Alexander Afanasyev, Junxiao Shi, et al. Nfd developer's guide. *Technical report, NDN-0021, NDN*, 2016.

[20] Alexander Afanasyev, Cheng Yi, Lan Wang, Beichuan Zhang, and Lixia Zhang. Snamp: Secure namespace mapping to scale ndn forwarding. In *INFOCOM Workshops*, 2015.

[21] Bengt Ahlgren, Christian Dannewitz, Claudio Imbrenda, Dirk Kutscher, and Borje Ohlman. A survey of information-centric networking. *IEEE Communications Magazine*, 50(7):26–36, 2012.

[22] Kawakib K Ahmed, Mohd Hasbullah Omar, and Suhaidi Hassan. Survey and comparison of operating concept for routing protocols in dtn. *Journal of Computer Science*, 12(3):141–152, 2016.

[23] Kimberly Amadeo. Hurricane Irma: Facts, Damage, and Costs . `https://www.thebalance.com/hurricane-irma-facts-timeline-damage-costs-4150395`.

[24] Mostafa Ammar. Ex uno pluria: The service-infrastructure cycle, ossification, and the fragmentation of the internet. *SIGCOMM CCR*, 2018.

[25] Renzo Angles and Claudio Gutierrez. Querying rdf data from a graph database perspective. In *European semantic web conference*, pages 346–360, 2005.

[26] Hitoshi Asaeda, Xun Shao, and Thierry Turletti. Contrace: Traceroute facility for content-centric network. `https://tools.ietf.org/html/draft-asaeda-icnrg-contrace-042`, 2018.

[27] Onur Ascigil, Vasilis Sourlas, Ioannis Psaras, and George Pavlou. A native content discovery mechanism for the information-centric networks. In *Proceedings of the 4th ACM Conference on Information-Centric Networking*, 2017.

[28] Peter Bailis, Alan Fekete, Ali Ghodsi, Joseph M Hellerstein, and Ion Stoica. The potential dangers of causal consistency and an explicit solution. In *SOCC*, 2012.

[29] Roberto Baldoni, Roberto Beraldi, Vivien Quema, Leonardo Querzoni, and Sara Tucci-Piergiovanni. TERA: topic-based event routing for peer-to-peer architectures. In *DEBS*, 2007.

[30] Sebastien Barré, John Ronan, and Olivier Bonaventure. Implementation and evaluation of the shim6 protocol in the linux kernel. *Computer Communications*, 34(14):1685–1695, 2011.

[31] Ryan Beckett, Aarti Gupta, Ratul Mahajan, and David Walker. A general approach to network configuration verification. In *SIGCOMM*, 2017.

[32] Abdulkader Benchi, Pascale Launay, and Frédéric Guidec. Solving consensus in opportunistic networks. In *ICDCN*, 2015.

[33] John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-policy attribute-based encryption. In *SP*, 2007.

[34] Abhinav Bhatele, Sébastien Fourestier, Harshitha Menon, Laxmikant V Kale, and François Pellegrini. Applying graph partitioning methods in measurement-based dynamic load balancing. Technical report, Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States), 2012.

[35] Steven Bird, Ewan Klein, and Edward Loper. *Natural Language Processing with Python*. O'Reilly Media, Inc., 1st edition, 2009.

[36] Rob H Bisseling and Wouter Meesen. Communication balancing in parallel sparse matrix-vector multiplication. *Electronic Transactions on Numerical Analysis*, 21:47–65, 2005.

[37] Fatemeh Borran, Ravi Prakash, and André Schiper. Extending paxos/lastvoting with an adequate communication layer for wireless ad hoc networks. In *SRDS*, 2008.

[38] Francisco Brasileiro, Fabíola Greve, Achour Mostéfaoui, and Michel Raynal. Consensus in one communication step. In *PaCT*, 2001.

[39] F. Bronzino, C. Han, Y. Chen, et al. In-Network Compute Extensions for Rate-Adaptive Content Delivery in Mobile Networks. In *ICNP*, 2014.

[40] Christian Cachin, Rachid Guerraoui, and Lus Rodrigues. *Introduction to Reliable and Secure Distributed Programming*. Springer Publishing Company, Incorporated, 2nd edition, 2011.

[41] Giovanna Carofiglio, Luca Muscariello, Jordan Augé, Michele Papalini, Mauro Sardara, and Alberto Compagno. Enabling icn in the internet protocol: Analysis and evaluation of the hybrid-icn architecture. In *ICN*, 2019.

[42] David Cavin, Yoav Sasson, and André Schiper. Consensus with unknown participants or fundamental self-organization. In *AdHoc-Now*, 2004.

[43] Supratik Chakraborty et al. A scalable approximate model counter. In *CP*, 2013.

[44] Bernadette Charron-Bost and André Schiper. The heard-of model: computing in distributed systems with benign faults. *Distributed Computing*, 22(1), 2009.

[45] Mark Chavira and Adnan Darwiche. On probabilistic inference by weighted model counting. *AI*, 172(6-7), 2008.

[46] Jiachen Chen, Mayutan Arumaithurai, Xiaoming Fu, and K. K. Ramakrishnan. Coexist: Integrating Content Oriented Publish/Subscribe Systems with IP. In *ANCS*, 2012.

[47] Jiachen Chen, Mayutan Arumaithurai, Xiaoming Fu, and K. K. Ramakrishnan. G-copss: A content centric communication infrastructure for gaming applications. In *2012 IEEE 32nd International Conference on Distributed Computing Systems*, pages 355–365, 2012.

[48] Jiachen Chen, Mayutan Arumaithurai, Xiaoming Fu, and K. K. Ramakrishnan. Cns: content-oriented notification service for managing disasters. In *Proceedings of the 3rd ACM Conference on Information-Centric Networking*, 2016.

[49] Jiachen Chen, Mayutan Arumaithurai, Lei Jiao, Xiaoming Fu, and K. K. Ramakrishnan. Copss: An efficient content oriented publish/subscribe system. In *2011 ACM/IEEE Seventh Symposium on Architectures for Networking and Communications Systems*, pages 99–110, 2011.

[50] Jiachen Chen, Mohammad Jahanian, and K. K. Ramakrishnan. Black ice! using information centric networks for timely vehicular safety information dissemination. In *2017 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*, pages 1–6. IEEE, 2017.

[51] Alberto Compagno, Mauro Conti, Paolo Gasti, and Gene Tsudik. Poseidon: Mitigating interest flooding ddos attacks in named data networking. In *LCS*, 2013.

[52] Alberto Compagno, Xuan Zeng, Luca Muscariello, Giovanna Carofiglio, and Jordan Augé. Secure producer mobility in information-centric network. In *ICN*, 2017.

[53] Mauro Conti, Ankit Gangwal, Muhammad Hassan, Chhagan Lal, and Eleonora Losiouk. The road ahead for networking: A survey on icn-ip coexistence solutions. *arXiv preprint arXiv:1903.07446*, 2019.

[54] Ellen Cranley. 'Our government failed us': Bahamians were left to coordinate rescue efforts on social media after Hurricane Dorian. `https://tinyurl.com/y3kvvmce`, September 2019.

[55] Jon Crowcroft, Steven Hand, Richard Mortier, Timothy Roscoe, and Andrew Warfield. Plutarch: an argument for network pluralism. *ACM SIGCOMM CCR*, 33(4), 2003.

[56] Frank Dabek, Russ Cox, Frans Kaashoek, and Robert Morris. Vivaldi: a decentralized network coordinate system. In *SIGCOMM*, 2004.

[57] Christian Dannewitz, Dirk Kutscher, BöRje Ohlman, Stephen Farrell, Bengt Ahlgren, and Holger Karl. Network of information (netinf)–an information-centric networking architecture. *Computer Communications*, 36(7), 2013.

[58] Raimundo José de Araújo Macêdo. Causal order protocols for group communication. In *SBRC*, 1995.

[59] Daniele Di Sarli and Filippo Geraci. Gfs: A graph-based file system enhanced with semantic features. In *Proceedings of the 2017 International Conference on Information System and Data Mining*, pages 51–55, 2017.

[60] Yanlei Diao, Shariq Rizvi, and Michael J. Franklin. Towards an internet-scale XML dissemination service. In *VLDB*, 2004.

[61] Klaus Doppler, Mika Rinne, Carl Wijting, Cássio B Ribeiro, and Klaus Hugl. Device-to-device communication as an underlay to lte-advanced networks. *IEEE Communications Magazine*, 47(12), 2009.

[62] Christos Douligeris and Aikaterini Mitrokotsa. Ddos attacks and defense mechanisms: a classification. In *ISSPIT*, 2003.

[63] Dragos Dumitrescu, Radu Stoenescu, Matei Popovici, Lorina Negreanu, and Costin Raiciu. Dataplane equivalence and its applications. In *NSDI*, 2019.

[64] Zayan EL Khaled and Hamid Mcheick. Case studies of communications systems during harsh environments: A review of approaches, weaknesses, and limitations to improve quality of service. *IJDSN*, 2019.

[65] Energy Information Administration. Hurricane Irma cut power to nearly two-thirds of Florida's electricity customers. `https://www.eia.gov/todayinenergy/detail.php?id=32992`.

[66] Kevin Fall. A delay-tolerant network architecture for challenged internets. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, 2003.

[67] Dino Farinacci, Vince Fuller, David Meyer, Darrel Lewis, et al. Locator/id separation protocol (lisp). RFC 6830, January 2013.

[68] Seyed Kaveh Fayazbakhsh, Yin Lin, Amin Tootoonchian, Ali Ghodsi, Teemu Koponen, Bruce Maggs, KC Ng, Vyas Sekar, and Scott Shenker. Less pain, most of the gain: Incrementally deployable icn. *ACM SIGCOMM CCR*, 43(4), 2013.

[69] Andreas Emil Feldmann and Luca Foschini. Balanced partitions of trees and applications. *Algorithmica*, 71(2):354–376, 2015.

[70] FEMA. State and federal partners respond to the california wildfires. `https://www.fema.gov/news-release/2018/11/17/4407/state-and-federal-partners-respond-california-wildfires`, November 2018.

[71] FEMA. National incident management system. https://www.fema.gov/national-incident-management-system, 2019.

[72] B. Fenner, M. Handley, H. Holbrook, and I. Kouvelas. Protocol Independent Multicast - Sparse Mode (PIM-SM): Protocol Specification (Revised). RFC 4601, August 2006.

[73] J Fidge. Timestamps in message-passing systems that preserve the partial ordering. In *Proc. 11th Australian Comput. Science Conf.*, 1988.

[74] Sergio Flesca and Sergio Greco. Querying graph databases. In *International Conference on Extending Database Technology*, pages 510–524, 2000.

[75] Nikos Fotiou, Pekka Nikander, Dirk Trossen, and George C Polyzos. Developing Information Networking Further: from PSIRP to PURSUIT. In *BROADNETS*. 2012.

[76] Eli Gafni. Round-by-round fault detectors (extended abstract) unifying synchrony and asynchrony. In *PODC*, 1998.

[77] Paolo Gasti, Gene Tsudik, Ersin Uzun, and Lixia Zhang. DoS and DDoS in named data networking. In *ICCCN*, 2013.

[78] GDdata. Graph Drawing. `http://www.graphdrawing.org/data.html`.

[79] Chavoosh Ghasemi, Hamed Yousefi, and Beichuan Zhang. Far cry: Will cdns hear ndn's call? In *Proceedings of the 7th ACM Conference on Information-Centric Networking*, pages 89–98, 2020.

[80] Carla P Gomes, Ashish Sabharwal, and Bart Selman. Model counting: A new strategy for obtaining good bounds. In *AAAI*, 2006.

[81] Gisel Bastidas Guacho, Sara Abdali, Neil Shah, and Evangelos E Papalexakis. Semi-supervised content-based detection of misinformation via tensor embeddings. In *ASONAM*, 2018.

[82] Cenk Gündoğan, Peter Kietzmann, Thomas C Schmidt, and Matthias Wählisch. Information-centric networking for the industrial internet of things. In *Wireless Networks and Industrial IoT*, pages 171–189. 2021.

[83] Zygmunt J Haas, Joseph Y Halpern, and Li Li. Gossip-based ad hoc routing. *IEEE/ACM Transactions on networking*, 14(3), 2006.

[84] Toru Hasegawa. A survey of the research on future internet and network architectures. *IEICE transactions on communications*, 96(6):1385–1401, 2013.

[85] Laura Heath, Henry Owen, Raheem Beyah, and Radu State. Clip: Content labeling in ipv6, a layer 3 protocol for information centric networking. In *ICC*, 2013.

[86] AKM Hoque, Syed Obaid Amin, Adam Alyyan, Beichuan Zhang, Lixia Zhang, and Lan Wang. Nlsr: named-data link state routing protocol. In *Proceedings of the 3rd ACM SIGCOMM workshop on Information-centric networking*, 2013.

[87] Farzin Houshmand and Mohsen Lesani. Hamsaz: replication coordination analysis and synthesis. *POPL*, 2019.

[88] Pan Hui, Jon Crowcroft, and Eiko Yoneki. Bubble rap: Social-based forwarding in delay-tolerant networks. *IEEE Transactions on Mobile Computing*, 10(11), 2010.

[89] IETF. Network configuration. https://datatracker.ietf.org/doc/charter-ietf-netconf/, 2020.

[90] Hasan MA Islam, Dmitrij Lagutin, Andrey Lukyanenko, Andrei Gurtov, and Antti Ylä-Jääski. Cidor: content distribution and retrieval in disaster networks for public protection. In *2017 IEEE 13th international conference on wireless and mobile computing, networking and communications (WiMob)*, pages 324–333, 2017.

[91] Daniel Jackson. Alloy: a lightweight object modelling notation. *TOSEM*, 11(2), 2002.

[92] V. Jacobson, D. K. Smetters, J. D. Thornton, et al. Networking Named Content. In *CoNEXT*, 2009.

[93] Mohammad Jahanian, Jiachen Chen, and K. K. Ramakrishnan. Graph-based namespaces and load sharing for efficient information dissemination in disasters. In *2019 IEEE 27th International Conference on Network Protocols (ICNP)*, pages 1–12. IEEE, 2019.

[94] Mohammad Jahanian, Jiachen Chen, and K. K. Ramakrishnan. Formal verification of interoperability between future network architectures using alloy. In *International Conference on Rigorous State-Based Methods*, pages 44–60. Springer, 2020.

[95] Mohammad Jahanian, Jiachen Chen, and K. K. Ramakrishnan. Managing the evolution to future internet architectures and seamless interoperation. In *2020 29th International Conference on Computer Communications and Networks (ICCCN)*, pages 1–11. IEEE, 2020.

[96] Mohammad Jahanian, Toru Hasegawa, Yoshinobu Kawabe, Yuki Koizumi, Amr Magdy, Masakatsu Nishigaki, Tetsushi Ohki, and KK Ramakrishnan. Direct: Disaster response coordination with trusted volunteers. In *2019 International Conference on Information and Communication Technologies for Disaster Management (ICT-DM)*, pages 1–8. IEEE, 2019.

[97] Mohammad Jahanian and K. K. Ramakrishnan. Name space analysis: verification of named data network data planes. In *Proceedings of the 6th ACM Conference on Information-Centric Networking*, pages 44–54, 2019.

[98] Mohammad Jahanian and K. K. Ramakrishnan. Conice: Consensus in intermittently-connected environments by exploiting naming with application to emergency response. In *2020 IEEE 28th International Conference on Network Protocols (ICNP)*, pages 1–12. IEEE, 2020.

[99] Mohammad Jahanian and K. K. Ramakrishnan. Name Space Analysis. `https://github.com/mjaha/NameSpaceAnalysis`, 2020.

[100] Mohammad Jahanian and K. K. Ramakrishnan. Name space analysis: Verification of named data network data planes. *IEEE/ACM Transactions on Networking*, 29(2):848–861, 2021.

[101] Mohammad Jahanian, Yuxuan Xing, Jiachen Chen, K. K. Ramakrishnan, Hulya Seferoglu, and Murat Yuksel. The evolving nature of disaster management in the internet and social media era. In *2018 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*, pages 79–84. IEEE, 2018.

[102] David S Johnson, Cecilia R Aragon, Lyle A McGeoch, and Catherine Schevon. Optimization by simulated annealing: An experimental evaluation; part i, graph partitioning. *Operations research*, 37(6):865–892, 1989.

[103] George Karypis. METIS - Serial Graph Partitioning and Fill-reducing Matrix Ordering, Version 5.1.0, 2013.

[104] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing*, 20(1):359–392, 1998.

[105] George Karypis and Kirk Schloegel. ParMETIS - Parallel Graph Partitioning and Fill-reducing Matrix Ordering, Version 4.0. `http://glaros.dtc.umn.edu/gkhome/metis/parmetis/overview`, 2013.

[106] Peyman Kazemian, Michael Chang, Hongyi Zeng, George Varghese, Nick McKeown, and Scott Whyte. Real time network policy checking using header space analysis. In *NSDI*, 2013.

[107] Peyman Kazemian, George Varghese, and Nick McKeown. Header space analysis: Static checking for networks. In *NSDI*, 2012.

[108] Ari Keränen, Jörg Ott, and Teemu Kärkkäinen. The one simulator for dtn protocol evaluation. In *SIMUTools*, 2009.

[109] Siham Khoussi, Davide Pesavento, Lotfi Benmohamed, and Abdella Battou. Ndn-trace: a path tracing utility for named data networking. In *ICN*, 2017.

[110] Ahmed Khurshid, Xuan Zou, Wenxuan Zhou, Matthew Caesar, and P Brighten Godfrey. Veriflow: Verifying network-wide invariants in real time. In *NSDI*, 2013.

[111] R Koch, R Moser, and P Melliar-Smith. Global causal ordering with minimal latency. In *ICPADS*, 1998.

[112] Miika Komu, Mohit Sethi, and Nicklas Beijar. A survey of identifier–locator split addressing architectures. *Computer Science Review*, 17:25–42, 2015.

[113] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica. A data-oriented (and beyond) network architecture. In *SIGCOMM*, 2007.

[114] Jun Kurihara, Kenji Yokota, and Atsushi Tagami. A consumer-driven access control approach to censorship circumvention in content-centric networking. In *ICN*, 2016.

[115] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7), July 1978.

[116] Leslie Lamport. Password authentication with insecure communication. *Communications of the ACM*, 24(11):770–772, 1981.

[117] Leslie Lamport. The part-time parliament. *ACM Transactions on Computer Systems (TOCS)*, 16(2), 1998.

[118] Daniel Le Berre and Anne Parrain. The sat4j library, release 2.2, system description. *Journal on Satisfiability, Boolean Modeling and Computation*, 7, 2010.

[119] Craig A Lee, Zhiyi Zhang, Yukai Tu, Alex Afanasyev, and Lixia Zhang. Supporting virtual organizations using attribute-based encryption in named data networking. In *CIC*, 2018.

[120] Halikul Lenando and Mohamad Alrfaay. Epsoc: social-based epidemic-based routing protocol in opportunistic mobile social network. *Mobile Information Systems*, 2018.

[121] Sugang Li, Jiachen Chen, Haoyang Yu, Yanyong Zhang, Dipankar Raychaudhuri, Ravishankar Ravindran, Hongju Gao, Lijun Dong, Guoqiang Wang, and Hang Liu. Mf-iot: A mobilityfirst-based internet of things architecture with global reach-ability and communication diversity. In *IoTDI*, 2016.

[122] Suoheng Li, Jie Xu, Mihaela Van Der Schaar, and Weiping Li. Popularity-driven content caching. In *INFOCOM*, 2016.

[123] Tianxiang Li, Zhaoning Kong, Spyridon Mastorakis, and Lixia Zhang. Distributed dataset synchronization in disruptive networks. In *MASS*, 2019.

[124] Yahui Li et al. A survey on network verification and testing with formal methods: Approaches and challenges. *IEEE Communications Surveys & Tutorials*, 21(1), 2018.

[125] Ying-Dar Lin, Nai-Bin Hsu, and Chen-Ju Pan. Extension of rp relocation to pim-sm multicast routing. In *ICC 2001. IEEE International Conference on Communications. Conference Record (Cat. No. 01CH37240)*, volume 1, pages 234–238, 2001.

[126] Sheng Luo, Shangru Zhong, and Kai Lei. Ip/ndn: A multi-level translation and migration mechanism. In *NOMS*, 2018.

[127] R. Mahajan, N. Spring, D. Wetherall, and T. Anderson. Inferring Link Weights using End-to-End Measurements. In *IMW*, 2002.

[128] Haohui Mai, Ahmed Khurshid, Rachit Agarwal, Matthew Caesar, P Brighten Godfrey, and Samuel Talmadge King. Debugging the data plane with anteater. *ACM SIGCOMM Computer Communication Review*, 41(4), 2011.

[129] Spyridon Mastorakis, Jim Gibson, Ilya Moiseenko, Ralph Droms, and David Oran. Icn ping protocol draft-mastorakis-icnrg-icnping-00. `https://tools.ietf.org/html/draft-mastorakis-icnrg-icnping-02`, 2017.

[130] Spyridon Mastorakis, Jim Gibson, Ilya Moiseenko, Ralph Droms, and David Oran. Icn traceroute. `https://tools.ietf.org/id/draft-mastorakis-icnrg-icntraceroute-01.html`, 2017.

[131] F Mattern. Virtual time and global states of distributed systems. *Parallel and Distributed Algorithms*, 1989.

[132] Douglas Mauro and Kevin Schmidt. *Essential SNMP: Help for System and Network Administrators.* O'Reilly Media, Inc., 2005.

[133] James McCauley et al. Enabling a permanent revolution in internet architecture. In *ACM SIGCOMM*, 2019.

[134] Victor S Miller. Use of elliptic curves in cryptography. In *CRYPTO*, 1985.

[135] Viyom Mittal, Mohammad Jahanian, and K. K. Ramakrishnan. Flare: Federated active learning assisted by naming for responding to emergencies. In *Proceedings of the 8th ACM Conference on Information-Centric Networking*, 2021.

[136] Viyom Mittal, Mohammad Jahanian, and K. K. Ramakrishnan. Online delivery of social media posts to appropriate first responders for disaster response. In *Adjunct Proceedings of the 2021 International Conference on Distributed Computing and Networking*, pages 13–18, 2021.

[137] Ilya Moiseenko and Dave Oran. Tcp/icn: carrying tcp over content centric and named data networks. In *Proceedings of the 3rd ACM Conference on Information-Centric Networking*, pages 112–121. ACM, 2016.

[138] Ilya Moiseenko, Mark Stapp, and David Oran. Communication patterns for web interaction in named data networking. In *Proceedings of the 1st ACM Conference on Information-Centric Networking*, pages 87–96. ACM, 2014.

[139] Edo Monticelli, Benno M Schubert, Mayutan Arumaithurai, Xiaoming Fu, and K. K. Ramakrishnan. An information centric approach for communications in disaster situations. In *2014 IEEE 20th International Workshop on Local & Metropolitan Area Networks (LANMAN)*, pages 1–6, 2014.

[140] Waldir Moreira, Paulo Mendes, and Susana Sargento. Opportunistic routing based on daily routines. In *WoWMoM*, 2012.

[141] Waldir Moreira, Paulo Mendes, and Susana Sargento. Social-aware opportunistic routing protocol based on user's interactions and interests. In *ADHOCNETS*, 2013.

[142] Robert Moskowitz, Pekka Nikander, P Jokela, et al. Host identity protocol (hip) architecture. RFC 4423, May 2006.

[143] Irene Moulitsas and George Karypis. Partitioning algorithms for simultaneously balancing iterative and direct methods. Technical report, Minnesota Univ Minneapolis Dept of Computer Science, 2004.

[144] S. Mukherjee, F. Bronzino, S. Srinivasan, J. Chen, and D. Raychaudhuri. Achieving Scalable Push Multicast Services Using Global Name Resolution. In *GLOBECOM*, 2016.

[145] David Naylor, Matthew K Mukerjee, Patrick Agyapong, et al. XIA: Architecting A More Trustworthy and Evolvable Internet. *SIGCOMM CCR*, 2014.

[146] NDN. NDN Packet Format Specification 0.3 documentation. `http://named-data.net/doc/NDN-packet-spec/current/`, 2019.

[147] NDN. NDN Regular Expression. `http://named-data.net/doc/ndn-cxx/current/tutorials/utils-ndn-regex.html`, 2019.

[148] NDN. NDN Testbed. `http://ndndemo.arl.wustl.edu/ndn.html`, 2019.

[149] NDN. ndnSIM. `http://ndnsim.net`, 2019.

[150] Erik Nordmark and Marcelo Bagnulo. Shim6: Level 3 multihoming shim protocol for ipv6. RFC 5533, June 2009.

[151] Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In *USENIX ATC*, 2014.

[152] Aurojit Panda, Ori Lahav, Katerina Argyraki, Mooly Sagiv, and Scott Shenker. Verifying reachability in networks with mutable datapaths. In *NSDI*, 2017.

[153] José Rolando Guay Paz. Introduction to azure cosmos db. In *Microsoft Azure Cosmos DB Revealed*. 2018.

[154] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 2011.

[155] Charles Perkins et al. Ip mobility support for ipv4, revised. RFC 5944, November 2010.

[156] Charles Perkins et al. Mobility support in ipv6. RFC 6275, July 2011.

[157] Ali Pınar and Bruce Hendrickson. Partitioning for complex objectives. In *Proceedings of the 15th International Parallel and Distributed Processing Symposium (CDROM), IEEE Computer Society, Washington, DC, USA*, 2001.

[158] Lucian Popa, Ali Ghodsi, and Ion Stoica. Http as the narrow waist of the future internet. In *Hotnets*, 2010.

[159] Ioannis Psaras, Lorenzo Saino, Mayutan Arumaithurai, KK Ramakrishnan, and George Pavlou. Name-based replication priorities in disaster cases. In *2014 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 434–439. IEEE, 2014.

[160] Dipankar Raychaudhuri, Kiran Nagaraja, and Arun Venkataramani. Mobilityfirst: A robust and trustworthy mobility-centric architecture for the future internet. *SIGMOBILE*, 2012.

[161] Paul Resnick and Rahul Sami. Sybilproof transitive trust protocols. In *EC*, 2009.

[162] Erik Rolland, Hasan Pirkul, and Fred Glover. Tabu search for graph partitioning. *Annals of operations research*, 63(2):209–232, 1996.

[163] Amir Saffari, Christian Leistner, Jakob Santner, Martin Godec, and Horst Bischof. On-line random forests. In *ICCV Workshops*, 2009.

[164] Mark Saunders. Social media: California wildfires force thousands to evacuate. `https://www.10news.com/news/social-media-california-wildfires-force-thousands-to-evacuate`, November 2018.

[165] P Savola and T Chown. A survey of ipv6 site multihoming proposals. In *Proceedings of the 8th International Conference on Telecommunications, 2005. ConTEL 2005.*, volume 1, pages 41–48. IEEE, 2005.

[166] Jan Seedorf, Atsushi Tagami, Mayutan Arumaithurai, Yuki Koizumi, Nicola Blefari Melazzi, Dirk Kutscher, Kohei Sugiyama, Toru Hasegawa, Tohru Asami, K. K. Ramakrishnan, et al. The benefit of information centric networking for enabling communications in disaster scenarios. In *2015 IEEE Globecom Workshops (GC Wkshps)*, pages 1–7, 2015.

[167] B. Segall, D. Arnold, J. Boot, M. Henderson, and T. Phelps. Content Based Routing with Elvin. In *AUUG2K*, 2000.

[168] Wentao Shang, Jeff Thompson, Meki Cherkaoui, Jeff Burkey, and Lixia Zhang. NDN.JS: A JavaScript Client Library for Named Data Networking. In *NOMEN*, 2013.

[169] Susmit Shannigrahi, Chengyu Fan, and Craig Partridge. What's in a name? naming big science data in named data networking. In *Proceedings of the 7th ACM Conference on Information-Centric Networking*, pages 12–23, 2020.

[170] Susmit Shannigrahi, Chengyu Fan, and Greg White. Bridging the icn deployment gap with ipoc: An ip-over-icn protocol for 5g networks. In *NEAT*, 2018.

[171] Susmit Shannigrahi, Dan Massey, and Christos Papadopoulos. Traceroute for named data networking. `https://named-data.net/publications/techreports/ndn-0055-2-trace`, 2017.

[172] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The hadoop distributed file system. In *2010 IEEE 26th symposium on mass storage systems and technologies (MSST)*, pages 1–10, 2010.

[173] Mukesh Singhal and Ajay Kshemkalyani. An efficient implementation of vector clocks. *Information Processing Letters*, 43(1), 1992.

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

[175] Isabelle Stanton and Gabriel Kliot. Streaming graph partitioning for large distributed graphs. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1222–1230, 2012.

[176] Kai Su, Francesco Bronzino, KK Ramakrishnan, and Dipankar Raychaudhuri. Mftp: A clean-slate transport protocol for the information centric mobilityfirst network. In *ICN*, 2015.

[177] Atsushi Tagami, Tomohiko Yagyu, Kohei Sugiyama, et al. Name-based push/pull message dissemination for disaster message board. In *LANMAN*, 2016.

[178] Dominic Tarr, Erick Lavoie, Aljoscha Meyer, and Christian Tschudin. Secure scuttlebutt: An identity-centric protocol for subjective and decentralized applications. In *ACM ICN*, 2019.

[179] Pouyan Fotouhi Tehrani, Eric Osterweil, Jochen H. Schiller, Thomas C. Schmidt, and Matthias Wählisch. The missing piece: On namespace management in ndn and how dnssec might help. In *ICN*, 2019.

[180] Emina Torlak and Daniel Jackson. Kodkod: A relational model finder. In *TACAS*, 2007.

[181] Reza Tourani, Satyajayant Misra, Joerg Kliewer, Scott Ortegel, and Travis Mick. Catch me if you can: A practical framework to evade censorship in information-centric networks. In *ICN*, 2015.

[182] Reza Tourani, Satyajayant Misra, Travis Mick, and Gaurav Panwar. Security, privacy, and access control in information-centric networking: A survey. *IEEE communications surveys & tutorials*, 20(1):566–600, 2017.

[183] Dirk Trossen, Martin J Reed, Janne Riihijärvi, Michael Georgiades, Nikos Fotiou, and George Xylomenos. Ip over icn-the better ip? In *2015 European Conference on Networks and Communications (EuCNC)*, pages 413–417. IEEE, 2015.

[184] Bora Uçar and Cevdet Aykanat. Encapsulating multiple communication-cost metrics in partitioning sparse rectangular matrices for parallel matrix-vector multiplies. *SIAM Journal on Scientific Computing*, 25(6):1837–1859, 2004.

[185] Amin Vahdat, David Becker, et al. Epidemic routing for partially connected ad hoc networks, 2000.

[186] Roland van Rijswijk-Deij, Mattijs Jonker, and Anna Sperotto. On the adoption of the elliptic curve digital signature algorithm (ecdsa) in dnssec. In *CNSM*, 2016.

[187] Liang Wang, Suzan Bayhan, Jörg Ott, Jussi Kangasharju, Arjuna Sathiaseelan, and Jon Crowcroft. Pro-diluvian: Understanding scoped-flooding for content discovery in information-centric networking. In *Proceedings of the 2nd ACM Conference on Information-Centric Networking*, pages 9–18. ACM, 2015.

[188] Lijing Wang, Yongqiang Lyu, Jian Liu, Wentao Shang, Wenbo He, Dongsheng Wang, and Geyong Min. Naxos: A named data networking consensus protocol. In *HPCC/SmartCity/DSS*, 2018.

[189] Sen Wang, Jun Bi, Jianping Wu, Xu Yang, and Lingyuan Fan. On adapting http protocol to content centric networking. In *Proceedings of the 7th International Conference on Future Internet Technologies*, pages 1–6. ACM, 2012.

[190] Wikipedia. Camp fire (2018). `https://en.wikipedia.org/wiki/Camp_Fire_(2018)`.

[191] Wikipedia. Category:Disaster management. `https://en.wikipedia.org/wiki/Category:Disaster_management`.

[192] Wikipedia. Subdivisions of helsinki. `https://en.wikipedia.org/wiki/Subdivisions\_of\_Helsinki`.

[193] Wikipedia. Woolsey fire. `https://en.wikipedia.org/wiki/Woolsey_Fire`.

[194] Wikipedia. Tabu search. `https://en.wikipedia.org/wiki/Tabu_search`, 2018.

[195] World Vision. Hurricane Irma: Facts, FAQs, and how to help. `https://www.worldvision.org/disaster-relief-news-stories/hurricane-irma-facts`.

[196] Liang Xiao, Xiaoyue Wan, Xiaozhen Lu, Yanyong Zhang, and Di Wu. Iot security techniques based on machine learning: How do iot devices use ai to enhance security? *IEEE Signal Processing Magazine*, 35(5), 2018.

[197] L Yan. A survey on communication networks in emergency warning systems. *Sci. Comput*, 2011.

[198] Yingdi Yu, Alexander Afanasyev, David Clark, Van Jacobson, Lixia Zhang, et al. Schematizing trust in named data networking. In *Proceedings of the 2nd ACM Conference on Information-Centric Networking*, 2015.

[199] P. Zave. A formal model of addressing for interoperating networks. In *FM*, 2005.

[200] P. Zave. A practical comparison of alloy and spin. *Form. Asp. Comput.*, 27(2), March 2015.

[201] Hongyi Zeng, Peyman Kazemian, George Varghese, and Nick McKeown. Automatic test packet generation. In *CONEXT*, 2012.

[202] Guoqiang Zhang, Yang Li, and Tao Lin. Caching in information centric networking: A survey. *Computer Networks*, 57(16), 2013.

[203] Lixia Zhang, Alexander Afanasyev, Jeffrey Burke, Van Jacobson, Patrick Crowley, Christos Papadopoulos, Lan Wang, Beichuan Zhang, et al. Named data networking. *ACM SIGCOMM CCR*, 44(3), 2014.

[204] Yu Zhang, Alexander Afanasyev, Jeff Burke, and Lixia Zhang. A survey of mobility support in named data networking. In *INFOCOM Workshops*, 2016.

[205] Yu Zhang, Hongli Zhang, and Lixia Zhang. Kite: a mobility support scheme for ndn. In *Proceedings of the 1st ACM Conference on Information-Centric Networking*, 2014.

[206] Zhiyi Zhang, Yingdi Yu, Haitao Zhang, Eric Newberry, Spyridon Mastorakis, Yanbiao Li, Alexander Afanasyev, and Lixia Zhang. An overview of security support in named data networking. *IEEE Communications Magazine*, 56(11):62–68, 2018.

[207] Liang Zhou, Laxmi N Bhuyan, and K. K. Ramakrishnan. Goldilocks: Adaptive resource provisioning in containerized data centers. In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, pages 666–677, 2019.

[208] Ming Zhu, Dan Li, Fangxin Wang, Anke Li, K. K. Ramakrishnan, Ying Liu, Jianping Wu, Nan Zhu, and Xue Liu. Ccdn: Content-centric data center networks. *IEEE/ACM Transactions on Networking*, 24(6):3537–3550, 2016.