University of California

Los Angeles

# Addressing Operational Challenges in Named Data Networking Through NDNS Distributed Database

A dissertation submitted in partial satisfaction

of the requirements for the degree

Doctor of Philosophy in Computer Science

by

Alexander Afanasyev

2013

Abstract of the Dissertation

# Addressing Operational Challenges in Named Data Networking Through NDNS Distributed Database

by

## Alexander Afanasyev

Doctor of Philosophy in Computer Science

University of California, Los Angeles, 2013

Professor Lixia Zhang, Chair

Named Data Networking (NDN) is a recently proposed Internet architecture. NDN retains the same hourglass shape as the IP architecture, but changes the narrow waist from delivery of IP packets to destinations to the retrieval of named and signed data chunks. This conceptually simple change allows NDN networks to use almost all the Internet's well-tested engineering properties to solve not only communication problems, but also digital distribution and control problems. The functionality of the narrow waist in NDN is fundamentally different from that in IP: it uses consumer-driven data delivery with a stateful data forwarding plane, implements built-in data security, and provides support for the extensive use of in-network storage. Preliminary experience shows that NDN bridges the gap between applications and network transport, simultaneously simplifying the application development process and addressing some of the Internet's most pressing problems in security, scalability, and sustainability. At the same time, the realization of the NDN architecture faces a number of brand new challenges. For example, all data packets must be signed by the original producers and verified by their consumers, bringing up the need for providing secure, resilient, and scalable support for public key distributions. Furthermore, since NDN eliminates the translation from application names to IP addresses and routes consumer data requests using

application data names directly, maintaining the scalability of the global routing system becomes another challenge.

This dissertation addresses the above challenges in moving NDN from an architecture blueprint to the operational reality. We designed and implemented a prototype of NDNS, a completely distributed database system that largely mimics the structure of the DNS system in today's Internet but operates within the NDN architecture. We show how NDNS can be used for cryptographic key distribution and routing scalability management. We believe that NDNS can also serve a number of other purposes during the development and deployment of the NDN architecture in coming years.

The dissertation of Alexander Afanasyev is approved.

Christos Papadopoulos

Mario Gerla

Leonard Kleinrock

Peter Reiher

Lixia Zhang, Committee Chair

University of California, Los Angeles

2013

# Table of Contents

# List of Figures

# Acknowledgments

I would like to gratefully and sincerely thank my academic advisor Prof. Lixia Zhang for providing invaluable support throughout my Ph.D. program. I also want to thank my colleagues from UCLA's Internet Research Laboratory Zhenkai Zhu, Yingdi Yu, and Wentao Shang for their support and valuable discussions, without which this work would not have existed. I am also enormously grateful to our collaborators Prof. Van Jacobson, Prof. Beichuan Zhang (University of Arizona), Prof. Lan Wang (University of Memphis), and many other members of the NDN team for insightful discussions that built up my understanding of the network architecture design in general and Named Data Network design in particular. Also, I have a special thanks to Allison Mankin (VeriSign) for providing many crucial insights on the current DNS protocol design, which influenced many decision made for the NDNS system design. I am also very much obliged to Neil Tilley and Janice Wheeler for the valuable input and suggested corrections to the draft of my thesis.

# Vita

| | |
|---|---|
| 2005 | B.Tech. (Computer Science), Bauman Moscow State Technical University, Moscow, Russia. |
| 2007 | M.Tech. (Computer Science), Bauman Moscow State Technical University, Moscow, Russia. |
| 2011 | Exemplary Reviewer IEEE Communication Letters |
| 2012 | M.S. (Computer Science), UCLA, Los Angeles, California. |
| 2011–2012 | Teaching Assistant, Computer Science Department, UCLA. |
| 2008–2013 | Graduate Research Assistant, Computer Science Department, UCLA. |
| 2012 | Research Intern, Palo Alto Research Center (PARC), Palo Alto, California. |
| 2012 | The IEEE Communications Society Best Tutorial Paper Award (for the "Host-to-host congestion control for TCP" paper) |

## Publications

A. Afanasyev, N. Tilley, P. Reiher, and L. Kleinrock, "Host-to-host congestion control for TCP," *IEEE Communication Surveys and Tutorials*, vol. 12, no. 3, 2010.

A. Afanasyev, J. Wang, C. Peng, and L. Zhang, "Measuring redundancy level on

the Web," in *Proc. of AINTEC*, 2011.

E. Kline, A. Afanasyev, and P. Reiher, "Shield: DoS filtering using traffic deflecting," in *Proc. of IEEE ICNP*, 2011.

V. Pournaghshband, L. Kleinrock, P. Reiher, and A. Afanasyev, "Controlling Applications by Managing Network Characteristics," in *Proc. of IEEE ICC*, 2012.

L. Wang, A. Afanasyev, R. Kuntz, R. Vuyyuru, R. Wakikawa, and L. Zhang, "Rapid Traffic Information Dissemination Using Named Data," in *Proc. of ACM NoM Workshop*, 2012.

C. Yi, A. Afanasyev, L. Wang, B. Zhang, and L. Zhang, "Adaptive Forwarding in Named Data Networking," *ACM Computer Communication Reviews*, vol. 42, no. 3, 2012.

A. Afanasyev, I. Moiseenko, and L. Zhang, "ndnSIM: NDN simulator for NS-3," *Tech.Report NDN-0005*, 2012

C. Yi, A. Afanasyev, I. Moiseenko, L. Wang, B. Zhang, and L. Zhang, "A Case for Stateful Forwarding Plane," *Computer Communications*, vol. 36, no. 7, 2013.

A. Afanasyev, P. Mahadevan, I. Moiseenko, E. Uzun, and L. Zhang, "Interest Flooding Attack and Countermeasures in Named Data Networking," in *Proc. of IFIP Networking 2013*, 2013.

Z. Zhu and A. Afanasyev, "Let's ChronoSync: Decentralized Dataset State Synchronization in Named Data Networking," in *Proc. of IEEE ICNP*, 2013.

# CHAPTER 1

# Introduction

The recently proposed Named Data Networking (NDN) architecture [JST09, EBJ10, SJ09] aims to replace the existing Internet infrastructure, fixing its long-standing problems and better supporting the current communication patterns. In particular, NDN shifts primary communication orientation from point-to-point channels (or connections) to data, which largely reflects recent changes in usage of the Internet. In most cases, we are no longer using the Internet to reach a particular host, instead we simply want to get a specific piece of data, be it today's news page, latest tweets on a particular topic, or a Netflix movie. Many complexities and kludges have been introduced to mitigate this inherent misalignment between the existing Internet architecture and its primary use today, such as DNS-based redirection towards content delivery networks [NSS10], load balancers in front of web servers, application-level (HTTP) caches, and many other systems. NDN architecture aims to provide a direct network-level support for such data-oriented application uses, providing in-network storage and caching opportunities, switching to a receiver-driven communication model, and, most importantly, incorporating a fundamental building block to improve security by requiring that all data packets be cryptographically signed.

## 1.1 Operational challenges in NDN

Although NDN as an architecture inherently provides numerous communication benefits, it is important to mention that several of these benefits come at an oper-

ational cost. The network is no longer completely application-agnostic, providing just a simple channel-based packet delivery; in fact, the NDN network provides an ecosystem for applications to function. While there is still Interest and Data packet forwarding that is largely independent from applications, the network uses application names directly (same as applications use network-friendly names) for forwarding purposes, and it is vital for NDN to provide adequate namespace regulation to avoid unexpected and highly undesirable namespace clashes or conflicts.

Another part of the ecosystem is cryptography support. While NDN defines and requires that each Data packet be signed with an application-selected cryptographic signature, it is still the architecture's responsibility to ensure proper publication, storage, and retrieval of the public keys and public key certificates, so the Data packet recipients can verify the signature and decide whether to use the packet or discard it.

Finally, NDN needs a scalable Interest forwarding mechanism. As NDN does not feature a separation between an IP address space for hosts and a namespace for domain names that are mapped to IP addresses using DNS, all communication happens using application-level names, which can be roughly seen as forwarding packets directly on DNS names. Given the existing problems with managing a very limited number of IP prefixes within a global routing system, it is unfeasible to assume that a conventional routing system would be able to handle upwards of billions of NDN names: the current BGP global routing system is struggling to support approximately 600,000 IP prefixes [ATL10], while there are already two billion more of second-level domain names [Ver12]. At the same time, it is unrealistic to assume that a routing system will not be used at all; flooding is not an option on a global scale. Thus, there is a great operational need to have a manageable solution for the scaling problem as NDN achieves wider adoption.[1]

---

[1]One completely different direction that is currently being explored inside the NDN team and is outside the scope of the current thesis is hyperbolic routing [PKB10, BPK10, KPK10], which is a type of a greedy routing that takes into account topology and topological properties

## 1.2 NDNS

Solutions to the operational challenges in NDN above assume the existence of some sort of authoritative database, which allows for storing namespace delegation information for NDN namespace regulation, cryptographic information for general NDN-based applications (Chapter 5), and mappings between routable or non-routable names (Chapter 6). All of these pieces of information use hierarchical names and namespaces to provide data scoping and localization: all data belonging to a particular user (e.g., his or her cryptographic credentials) can be stored under a single namespace. Therefore, the assumed database has to be directly based on names, and specifically on hierarchical names, to identify individual records and sets of records. Also, the management of these records has to be done in a distributed way, so each individual user or provider can independently decide how much and what data to be stored, without requiring any additional communication or coordination with other elements of the system. Clearly, the one system that exactly satisfies these criteria is the Domain Name System (DNS) [Moc87b, Eas99] infrastructure. While originally designed to provide a simple name-to-IP mapping service, DNS is essentially a large, highly scalable, and highly distributed database that is based on and managed using the hierarchy of names. More than 25 years of deployment and active use proved that DNS made the right designe choices, allowing its use in a wide variety of network and application related tasks.

The present work aims to design NDNS, a scalable and distributed database by applying DNS design choices and concepts within the NDN architecture. The objective of the work is not to merely port DNS to NDN, but to understand how DNS design principles can work within a pure data-centric communication model provided by NDN, which design choices can be borrowed directly, and what should

---

of Internet-like networks. The main objective of this research direction is to eliminate the need to maintain "full" routing tables and use special coordinates to enable guessing the path on which to forward the Interest, which is comparably good as if it was calculated based on the knowledge of the full topology using the shortest paths algorithms.

be redesigned in order to fully utilize the benefits of the NDN architecture.

### 1.2.1 NDNS design highlights

DNS, as a protocol, relies heavily on IP-based memoryless channels for packet exchange. For example, every time a request is sent out, it is directed to a specific name server: the selected root server (or its anycasted copy), the selected top-level domain (TLD) server, the selected second-level domain (SLD) server, etc. Every time the request is sent, the requester (iterative querier) asks the selected server the same question, while getting back potentially different answers: negative responses, referrals, or final answers, depending on the knowledge held by the server about the request.

Interest packets in NDN are almost equivalent to DNS queries. They both specify the requested name, and both expect to receive one answer. However, there are two fundamental properties of an NDN network that prevent direct translation of DNS queries into Interest packets. First, unlike DNS queries, Interest packets cannot be ambiguous: it is not possible to express exactly the same Interest and get two completely different answers. NDN's Interest must uniquely identify Data; otherwise, in-network storage and caching will prevent retrieval of more than one Data packet corresponding to the Interest.[2] Second, since NDN features destination-address-less communication, the Interest name needs to include directly or indirectly the authority in question. That is, if the query is intended for a SLD server, the SLD server's authority needs to be included in the Interest, along with a potential hint on how to forward the Interest on toward this authority (Chapter 6).

---

[2]Technically, within the current definition of NDN's Interest, it is possible to use various Interest selectors, including an exclude filter, to retrieve different Data packets using the same Interest name. However, usage of the Interest selectors should be as minimum as possible, as it most likely will result in "slow-path" processing, similar to the current processing of the non-standard IPv6 headers by high-speed routers.

NDNS is making an attempt to solve this problem by modifying some of the basic DNS elements, shifting channel-focused design to a pure data-centricity. In particular, NDNS explicitly separates zones and zone records (labels), making it necessary to send specific Interest packets to specific zones. This is seemingly in contrast to the existing DNS, where a domain name does not explicitly separate zone and labels. At the same time, the underlying iterative query process performs such separation implicitly, relying on IP-based memoryless channels. In other words, NDNS just brings implicit separation to the explicit plane.

The major benefit of the explicit separation is that there is no longer an ambiguity between different Interests. If the question is intended to be answered by a SLD's zone, the querier expresses the Interest for a SLD's zone and the appropriate label. The "location" of a specific zone can be hinted by including a forwarding hint, as described in Chapter 6. The Interest formatted in this manner does not really specify a specific destination, as rather an authority over who should have an authoritative answer to the question; it is still the network's task to get the requested data as fast as possible using all available means.

### 1.2.2   NDNS design benefits

Since one of the primary tasks of the DNS/NDNS system is to maintain proper delegations from the root zone toward any other zones and domains (Chapter 3), these delegations can be "abused" for global NDN namespace regulation (assuming that these delegations are properly secured, see Chapter 4). In other words, an application (e.g., NDN web-browser or email client) can restrict the set of acceptable NDN names and namespaces used in this application to ones that have proper NDNS delegation. For example, such an application can consider the Data packet "`/ndn/ucla/cs/alex/cv`" to be valid and present it to the user only if it is properly signed with the key, which is authorized by one of the higher-level NDNS zones, either by "`/ndn/ucla/cs/alex`", "`/ndn/ucla/cs/`", "`/ndn/ucla`",

"/ndn", or "/".

NDNS can also, similar to DNS, be used as a general-purpose storage system. The current DNS is to some extent "abused" to store various information, including DNSSEC keys, domain-mailserver mappings, IP blocklist, various annotations, and many others. Since NDN heavily relies on cryptography to secure the binding between name and data, it is crucial to provide a network-supported infrastructure for the storage and management of the cryptographic information, including public keys and public key certificates. While NDNS is not the only option for such an infrastructure, and an initial attempt discussed in Chapter 5 relies on another core NDN element called repo [BZA13], NDNS can provide a more robust, better organized, and simplified management for cryptographic data.

Finally, general-purpose NDNS-based storage can be utilized to solve the problem of scaling NDN's name-based routing. The existing DNS provides an excellent scalability solution using IP architecture: application/service developers use virtually an infinite number of domain names, which are mapped to IP addresses and used by clients to actually reach the application or service. While not actually deployed for various reasons, there were several proposals many years ago to scale IP routing based on a so-called map-n-encap approach [Dee96, MWZ07, JMY08, ASB99, GKR11]. In short, these proposals separated the IP space into provider-dependent and provider-independent subspaces, where only very limited subsets of provider-dependent addresses needed to be present in the global routing table. All other provider-independent addresses needed to be mapped to one or more provider-dependent ones, and an additional IP-IP encapsulation/decapsulation mechanism needs to be used for global communication. The required mapping could have easily been implemented using a DNS infrastructure and reverse DNS zone, and it can be naturally implemented in NDNS zones. That is, the NDNS zone can be used not only for namespace delegation, but also to provide "forwarding hints" on where the requested Data can be located. This hint can then

be used in an NDN adaptation of encapsulation, either by extending Interest selectors or by prepending the hint to the Interest's name (see Chapter 6). The actual lookup for such hints could be performed, for example, by the end-host's forwarding strategy, guessing (or trying several guesses in parallel) which Interests require additional "NDNS resolution" and which do not. While it may look this approach goes against core NDN principles of naming data, not locations, the hint does not really introduce anything new to the architecture; rather, it just helps the network layer facilitate discovery of the best way to satisfy the Interest. The Interest still has to uniquely identify the requested Data, and applications and application developers can still be completely ignorant of the particular deployment of the application.

## 1.3   Common NDN simulation platform (ndnSIM)

Understanding the pros and cons of the architectural mechanisms of NDN poses a challenge at this time. These include Interest forwarding strategies, Data packet caching and cache management, and a full understanding of application behavior within the new architecture. The existing NDN testbed is an important step in this direction. However, it may be hard or even impossible in certain cases to utilize the testbed to fully understand possible behavior nuances of prototyped applications: the testbed has a fixed, small-scale topology, and it could be hard to set up individual experiments and obtain desired metrics. In particular, the NDNS system introduced in this thesis aims to solve operational challenges, some of which do not yet exist and are not possible to be emulated on a testbed-scale.

Chapter 7 introduces another contribution of the current work and an important component needed for the new architecture: the ndnSIM simulation suite [AMZ12]. ndnSIM was originally designed and developed to understand just network-layer Interest/Data forwarding characteristics (e.g., evaluate options for

the Interest forwarding strategies and routing protocols). Since that development, it was significantly extended by our efforts, along with the help of the community, and was used in many evaluations at different levels of the network architecture: resiliency to various DDoS attacks, measuring performance of different caching strategies, and investigations at the application level, including evaluations of the NDNS system introduced in this thesis.

Since ndnSIM's public release in summer of 2012, we have witnessed a rapid growth of its user community. A number of indicators suggest that we have made a robust start towards the initial goal of making ndnSIM a common simulation platform for the community to investigate various NDN design choices and system properties. The project's mailing list was set up to facilitate exchanges among ndnSIM users; it gathered over 150 subscribers from multiple countries and has been generating a great amount of traffic, as can be seen from the list's archive (`http://www.lists.cs.ucla.edu/mailman/listinfo/ndnsim`). Just one year after the ndnSIM release, there have been at least 15 published papers based on the results from using ndnSIM (`http://ndnsim.net/ndnsim-research-papers.html`); four of these 15 papers are authored by the NDN team members, the rest contributed by others from the community. The community also started making contributions to ndnSIM development. In addition to bug reports and feature suggestions, there are 17 public forks on GitHub, and a number of users have contributed code development to ndnSIM (`https://github.com/NDN-Routing/ndnSIM/blob/master/AUTHORS`).

## 1.4   Contributions of this work

Contributions of this dissertation can be summarized as follows:

- Design, prototype implementation, and evaluation of NDNS, a distributed database system that largely mimics the structure of the DNS system in

today's Internet, but operates within the NDN architecture (Chapter 3).

- Design and prototype implementation of security policies that enable strict control over which Data packets can be signed by which key-certificates. The designed and implemented policies essentially provide a way to secure the NDNS design, as well as a simple and yet secure dynamic update protocol for NDNS (Chapter 4).

- Identification of a number of operational challenges to address in order to move Named Data Networking from an architectural blueprint to an operational reality, in addition to applying the designed NDNS database to address these identified challenges. This includes the proposal for NDN namespace regulation (authorization of NDN prefixes in the routing system) and the design for cryptographic credential management for NDN applications, preserving application control over the published Data (Chapter 5).

- Analysis of the future challenge of scaling NDN name-based routing, and NDNS-based design to mitigate this scaling problem in a global NDN deployment (Chapter 6).

- Design and implementation of ndnSIM, a common platform for simulation-based experimentation with NDN architecture (Chapter 7).

# CHAPTER 2

# Background

This chapter briefly introduces Named Data Networking (NDN) architecture, as well as the Domain Name System (DNS) protocol and its security extension (DNSSEC), which comprise the primary subject of this thesis. The following sections present a short description of the relevant parts of the architecture and protocols. A more detailed description can be found in specialized literature, such as [JST09, EBJ10] for NDN and [Moc87b, Eas99] for DNS.

## 2.1 Named Data Networking (NDN) architecture

This section gives an overview of NDN architecture, with a focus on its stateful forwarding plane (refer to [JST09, EBJ10, YAM13, YAW12] for more detail). NDN is a receiver-driven, data-centric communication architecture, where all communications in NDN are performed using two distinct types of packets: *Interest* and *Data*. Both types of packets carry a *name*, which uniquely identifies a piece of content that can be carried in one Data packet. Data names in NDN are hierarchically structured. An example name for the first segment of an HTML page for the "`ndnsim.net`" website would look like "`/net/ndnsim/www/index.html/%00`".

To retrieve Data, a consumer make a request by sending an Interest packet that carries the name of the content desired. Routers use this name to forward the Interest toward data sources, and a Data packet whose name matches the name in the Interest is returned to the consumer by following the reverse path of

10

the Interest. Similar to IP, Interest forwarding is based on longest name prefix match, but unlike IP, an Interest packet and its matching Data packet always take symmetric paths.

Each NDN router maintains three major data structures:

- A *Pending Interest Table (PIT)* holds all "not-yet-satisfied" Interests that have been sent upstream towards potential data sources. Each PIT entry contains one or multiple incoming and outgoing physical interfaces: multiple incoming interfaces indicate the same Data is requested from multiple downstream users; multiple outgoing interfaces indicate the same Interest is forwarded along multiple paths.

- The *Forwarding Interest Base (FIB)* maps name prefixes to one or multiple physical network interfaces, specifying directions where Interests can be forwarded.

- A *Content Store (CS)* temporarily buffers Data packets that pass through this router, allowing efficient data retrieval by different consumers.

When a router receives an Interest packet, it first checks whether there is a matching Data in its CS. If a match is found, the Data packet is sent back to the incoming interface of the Interest packet. If not, the Interest name is checked against the entries in the PIT. If the name already exists in the PIT, then it can be a duplicate Interest (identified by a random number carried by each Interest) and should be dropped, or it can be an Interest from another consumer asking for the same Data, which requires adding the incoming interface of this Interest to the existing PIT entry. If the name does not exist in the PIT, the Interest is added to the PIT and forwarded along the interface chosen by the strategy module, which uses FIB as input for its decisions.

When a Data packet is received, its name is used to look up the PIT. If a matching PIT entry is found, the router sends the Data packet to the interface(s)

from which the Interest was received, caches the data in the CS, and removes the PIT entry. Otherwise, the Data packet is deemed unsolicited and discarded. Each Interest also has an associated lifetime; the PIT entry is removed when the lifetime expires. Although the maximum lifetime is specified by users, it is ultimately a router's decision as to how long it keeps a PIT entry.

### 2.1.1 Repo

`repo` is a new communication element essential for NDN-based communications, providing permanent storage for any kind of NDN Data. Whenever a Data packet is put into repo, the packet can later be used to satisfy any incoming Interests for this Data, in exactly the same way an NDN cache would do. However, there are two fundamental differences between repo and a cache (packet buffer). First, repo, at least in its current definition and implementation in NDN platform codebase [NDN13a], never deletes any element from the underlying database, so that Data packets being put into the repo stay there virtually forever. The second fundamental difference is the mechanics of how Data packets get into the repo: repo expects an explicit command from the user. At the same time, repo is not exactly an application-level database or a fully managed storage. It is designed to store Data packets regardless of the originating application, and there is no way to issue commands to a specific repo instance.

The command interface that is currently defined requires the user to send an Interest for the Data that it wants to be put into the repo, attaching a special postfix that is recognized by the repo, configured to receive Interests for such prefixes. For example, if one wants to put an HTML page of `ndnsim.net` website into the repo, it needs to send Interest "**/net/ndnsim/www/index.html/%C1.R.sw/<nonce>**" to the NDN network, expecting that it will reach at least one repo. As soon as a configured repo receives and recognizes such an Interest, it may return a Data packet to the user (to ensure Interest/Data flow balance), remove the command

postfix "**/%C1.R.sw/<nonce>**", and express one or more Interests for the Data indicated by the user, e.g., "`/net/ndnsim/www/index.html/%00`". The number of required segments is determined using the segmenting protocol, which is currently defined to use the "LastBlockID" field in the returned Data packets, indicating the last sequence number of the last Data packet in the sequence.

## 2.2   Domain Name System (DNS)

The Domain Name System (DNS) [Moc87b] is a global public database system, providing various mapping services from hierarchical domain names. The common usages of the DNS system include mapping DNS names to IP addresses, mapping domains to the list of mail servers, reverse mapping from IP addresses to domain names, mapping for blacklisting purposes, and many others.
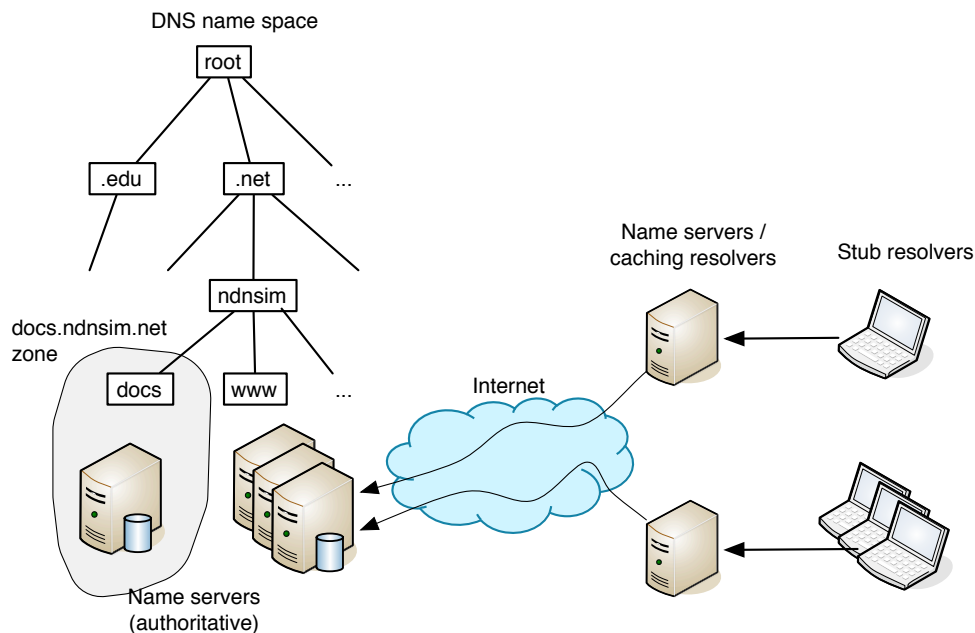


Figure 2.1: DNS components overview

On a high level, DNS consists of four major components: (1) domain name space, zones, domains and resource records (RRs), (2) the DNS protocol, (3) name

servers, and (4) resolvers (Figure 2.1). The domain name space and RRs basically define the name hierarchy and associate different type data to each leaf in this hierarchy ("A" RRs provide association to IP addresses, "MX" RRs associate mail servers to a domain, etc.). This information can be extracted via the DNS query protocol. Name servers are the programs that actually store parts of the domain name space and respond to incoming queries. Depending on the function, name servers are divided into *authoritative* name servers, which maintain definitive versions of RRs in a specific zone (zone is a unit of authoritative information), and *caching* name servers (also usually functioning as caching resolvers), maintaining cached versions of the RRs from diverse parts of the DNS name space. The last component of the DNS system—resolvers—are the programs that perform access to the stored mapping by DNS queries; in other words, they resolve the questions to the answers. Depending on the type of queries they generate, as well their position in the system, resolvers are divided into two categories: *caching resolvers* that perform iterative queries and *stub resolvers* that perform recursive querying (Figure 2.2).

The *iterative query* is a basic DNS query that allows discovery of the existing mapping (or the fact that such mapping does not exist) with minimal prior knowledge about the whole DNS database; it is usually performed by caching resolvers. In particular, the only information that is necessary to initiate an iterative query is the IP address of at least one root server. For example, if the ultimate question is to find the IP address of the "cs.ucla.edu" domain, the query for the "A" RR set for the domain "cs.ucla.edu" will be initially send to one of the root servers. The root server will simply provide a *referral* (names and IP addresses) to ".edu" name servers, which should know more about the question being asked. In this example, ".edu" name servers should deliver further referral to UCLA's name server(s). Each time the caching resolver receives a referral, it selects one of the name servers from the referral [YWL12] and sends it the original query.
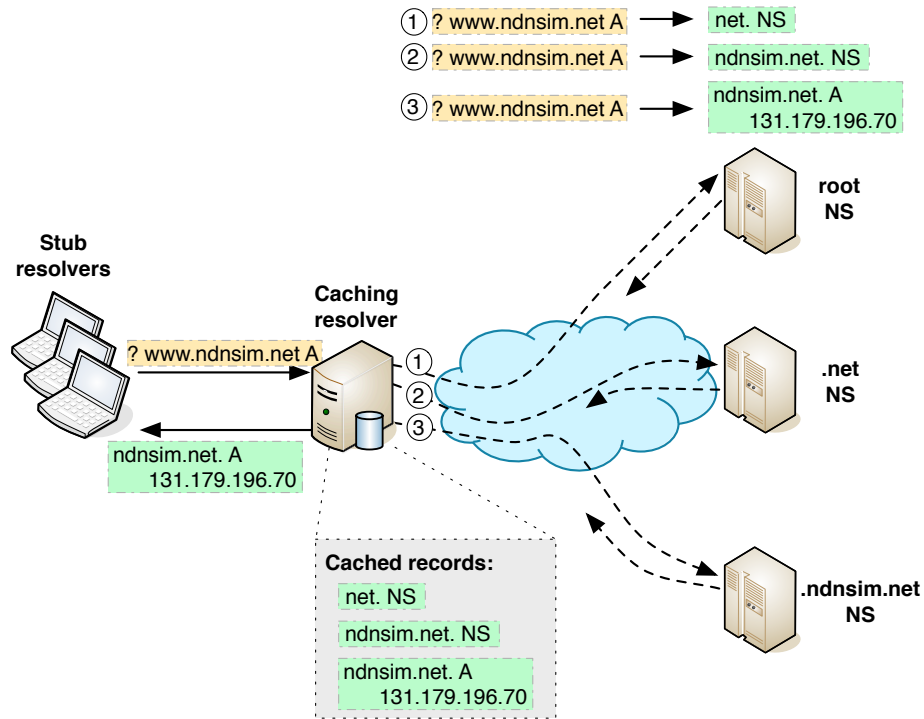
Figure 2.2: Example of DNS queries

Eventually, the iterative query process will be terminated, either when one of the name servers returns the queried RR set or authoritatively indicates that the queried RR set does not exist.

The *recursive querying* is performed usually by the end-host's stub resolvers and relies on the configured access to one or more caching resolvers, which perform iterative queries on the stub resolver's behalf. Each recursive query, unlike the iterative counterpart, is expected to return either the requested RR set or an indication that such an RR set does not exist. If the query does not return a valid answer (e.g., returns a referral), then it is assumed there is not a valid answer and no further operations will be performed, except possibly trying to send a recursive query to a different caching resolver.

The key advantages and success of DNS come from its excellent scalability properties. The zone owners can replicate authoritative name servers in different

parts of the world (as the case with the root zone), thus distributing the load and providing failure resiliency. Arguably more important, DNS has extensive caching of the query results by the caching resolvers. For instance, the caching resolver will refrain from repeating queries to the root authoritative server if it has a cached version of the referral from a root, TLD, or SLD name server.

### 2.2.1 DNS Security Extension (DNSSEC)

The original design of the DNS system contained limited measures to protect against malicious users and intentional attacks. To address these limitations and to provide additional assurances and resiliency against potential future attacks, the Internet Engineering Task Fork (IETF) developed DNSSEC [Eas99], a secure and backward compatible version of the DNS protocol. In essence, DNSSEC cryptographically signs each RR set and provides a way to verify signatures following the certification chains to the known trust anchor(s), such as the DNSSEC root zone trust anchor [IV09]. It is worth mentioning that DNSSEC leverages DNS as a general use database to store, manage, and retrieve all necessary cryptographic information to facilitate verification of all signed RR sets.

The following presents a simplified illustration of basic mechanics in DNSSEC protocol. When a DNSSEC-enabled name server replies to a query, it attaches "RRSIG" records to each of the returned autoritative RR set answers. The "RRSIG" records contain the actual cryptographic signature and additional identification of the key ("DNSKEY" RR set) that was used to produce this signature. The identified "DNSKEY" RR set can then be fetched by another DNSSEC query in exactly the same manner as any other DNS RR set, the fetched RR set having as well its own attached "RRSIG" information for further verification purposes.

To ensure that the key used to sign the records in the zone (so-called zone signing key, or ZSK) and the key that is used to sign other keys in the zone (key

signing key, or KSK) are properly authorized, DNSSEC uses Delegation Signer
("DS") records that are installed in the parent zone and specify a hash of the
authorized KSK for the zone (ZSK is indirectly authorized by providing a proper
signature using the directly authorized KSK). The example on Figure 2.3 shows
a DNSSEC delegation chain for a "TXT" RR set for "`ndnsim.net`": the "TXT"
RR set is signed with the "`ndnsim.net`" ZSK, which is signed by the zone's KSK,
which is authorized by the parent ".`net`" zone via a "DS" record, and so on.
This way, the zone preserves the ownership over the DNSKEY records (i.e., all
DNSKEY records are stored in the zone and only in the zone, and it is the author-
itative server's responsibility to maintain them), while the delegation information
is stored in the parent zone. The separation between KSK and ZSK was made
in an effort to optimize operational overhead in cryptographic key maintenance.
In other words, the more exposure of the key (the more frequently it is used),
the more often the key needs to be replaced. Being the most exposed, ZSK is
designed to be relatively easy to replace without the need to make any changes in
the parent zone. KSK usually uses a stronger cryptography, has limited exposure,
and can have longer lifetime, as each change would require additional modification
of the parent zone.
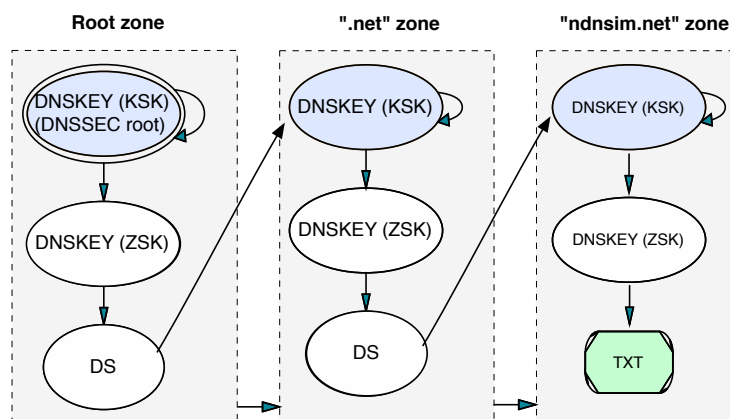


Figure 2.3: DNSSEC delegation for "`ndnsim.net`"

### 2.2.2 Dynamic updates in DNS

The basic idea behind dynamic DNS updates is to use a specialized form of the DNS message (DNS Update) containing the name of the zone to update and prerequisite, update, and additional sections. The prerequisite section contains the list of RR states, each indicating that a particular RR set or RR data should exist or should not exist in the zone. The update section contains the list of RRs (RR updates) that indicate addition or removal of the existing entries in zone, in case the zone satisfies conditions specified in the prerequisite section (for more detail, refer to RFC 2136 [VTR97]). The prerequisite section stems from the need to prevent update duplication, i.e., to some degree prevent potential replay attacks on the dynamically updated zone.

The complementary RFC 2137 specification [Eas97], in addition to the basic protocol of the dynamic DNS update, defines the security provisions that enable dynamic updates (secure dynamic updates) for the DNSSEC-secured zones. The specification defines two operational modes that differ on how signatures attached to the updates are used by the authoritative name server. In both modes, the dynamic update generator, along with the updated (or removed) RR data, provides the corresponding "RRSIG" data, which is generated using a key that has a corresponding legitimately delegated "DNSKEY" record in the updated zone. The main difference between the modes is that in one mode, the updated record is installed and later served in the same form to the incoming requests, including the updater-generated "RRSIG" data, whereas in the other mode, the RRSIG supplied by the updater is used only for authentication and authorization purposes, and the authoritative name server uses its own zone-signing key to generate "RRSIG" data that is later attached to the updated record, when the record is requested.

The motivation for separating these modes is that in the first mode, the au-

thoritative name server does not require an online key, thus making it possible to generate stronger signatures. At the same time, not having the online key prevents generation of "NXT" (negative responses) records, and it may prevent AXFR transfers from working: dynamically updated records may need to be managed separately from static records. In the other mode, the server does not feature these limitations, but requires an online signing process, which usually means weaker signatures.

# CHAPTER 3

# NDNS: distributed database system for NDN

DNS is arguably one of the most successful elements in the current Internet architecture. It not only resolves application-layer domain names to network-layer host addresses, but also is used in many other name-data mapping solutions, including DNSSEC key management, mailserver management, and literally hundreds of other types of mappings. Given the great success of DNS, which is basically a highly distributed and extremely well-scaled online database system, the main goal of this thesis is to apply principles of DNS in the NDN architecture and enable its use to solve critical operational challenges in the NDN network, including the need for strict namespace management, NDN routing scalability, providing core infrastructure for cryptographic support in NDN, and potentially many others.

However, it is not mere a long history of success that drove our choice to base NDNS design on DNS, but a history-based proof that the design choices made by DNS are the right ones. Managing data using hierarchical namespaces and namespace (zone) delegation provides simple and distributed management, as well as ensures virtually unlimited database storage. Iterative query process that "walks down" the namespace delegation hierarchy guarantees access to the database elements with minimal a priory knowledge about DNS system (i.e., knowing just IP addresses of root servers is enough to access any element in the database). Dedicated *caching resolvers* aggregate and cache database queries from *stub resolvers* embedded in user operating systems, ensuring efficient, scalable, and failure-resilient access to the database entries. Therefore, the preservation of these

elements is essential for NDNS, and our choice to follow closely DNS model is just an attempt to minimize potential design errors. In other words, the primary objective is to build on DNS principles within a pure data-centric communication NDN-based infrastructure, and not to port blindly DNS to the NDN architecture.

Although the primary motivational example for NDNS was to scale NDN routing (see Chapter 6), the basic concepts of the designed protocol are independent of a specific usage. This chapter provides a detailed description of the NDNS protocol as a generic, highly scalable, distributed database system that is built on top of the NDN architecture, assuming that routers either contain full information about all application-level prefixes or have some other means to discover where to send an Interest for all application-specific prefixes (e.g., can use flooding to forward Interests). Also, this chapter does not discuss specific usage of the NDNS system or its place in the NDN ecosystem, deferring such discussions to the later chapters (Chapters 5 and 6). At the same time, it is our expectation that NDNS can be utilized as a universal database for a multitude of different network and application needs, including many existing uses of the DNS system, such as domain to mail server mapping, servicing as a database for various blocklists, providing service discovery, and many others.

In the rest of this chapter we introduce all elements of NDNS system and provide a large-scale simulation-based evaluation, showing several benefits of a tight integration between the application and network layers.

## 3.1 NDNS design

Similar to the conventional DNS system, NDNS consists of four major components: namespace, protocol, name servers, and resolvers (Figure 3.1).

The namespace used and regulated by the NDNS system is a subset of the general NDN namespace to ensure equivalency to the DNS namespace: an NDN
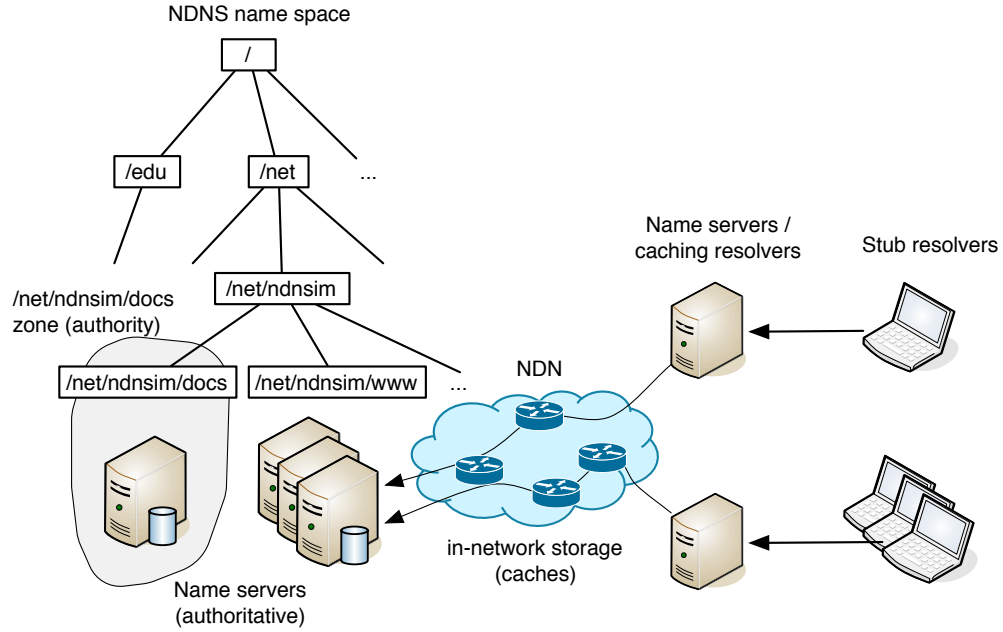
Figure 3.1: NDNS system overview

name used in NDNS should have an equivalent DNS name (by definition, every DNS name always has an NDN equivalent). Naming in NDN can be very flexible, and NDN imposes no restrictions on naming and name component content, basically allowing any name component to contain any binary blob, as long as the resulting name fits into the Interest and Data packet. However, in order to maintain equivalency to DNS (i.e., use DNS governance by ICANN and IANA), NDNS requires all NDNS-managed names to contain only lower-case DNS-compatible characters, rejecting any other names like "/EDU", "/EdU", and others. This requirement/restriction is completely non-technical and comes from the set practice of DNS system usage: typed names (in email addresses or in a web-browser) are expected to be case-invariant and contain only dashes and latin characters or conform to the internationalized domain names format [FHC03]. While it is not a direct goal of this thesis, such a requirement technically allows some interoperability between DNS and NDNS systems, where DNS can store and serve data stored in NDNS and vice versa.

The protocol part of the NDNS system contains many major changes compared to its DNS counterpart. These changes are due not only to the technical necessity to change the packet format so it conforms to the new communication architecture, but also to the fundamental changes in the communication paradigm. In particular, as shown in later sections, the switch from a channel-centric IP to a completely channel-less data-centric NDN requires a significant level of re-thinking and refactoring of the NDNS/DNS querying protocol. As pointed out in previous chapters, NDN does not feature the concept of "channels" or destination addresses. Instead, all data exchanges are performed via expressing Interests to the network for a specific piece of data (using the name). The network takes full care to *satisfy* these Interests with relevant Data packets using any means available, both by forwarding the Interest to the data producer and returning Data from the nearby cache. As a result, it is largely impossible for an end-host to send queries to a specific name server, which is an implicit requirement of the DNS protocol.

The two other parts of DNS—name servers and resolvers—are largely imported as is into the NDNS system design: name servers store and manage zones with domains and resource records, while resolvers perform querying operations either in an iterative (caching resolvers) or in a recursive (stub resolvers) style. However, it should be noted that NDNS contains an additional player that significantly affects the protocol: data-oriented network architecture. As pointed out later in Section 3.1.3 and confirmed via simulation-based evaluations (Section 3.2), there is a shifted importance (towards less significance) of the caching resolver. In other words, while in DNS the caching resolver is a major player that facilitates scalability—namely, prevents duplicate queries from reaching higher-level name servers—NDNS leverages NDN's in-network storage and caching ability for the same purpose. In essence, NDNS uses the inter-layer shared responsibility of caching resolvers (application layer, guaranteed cache space) and NDN caches

(network layer, best-effort cache space) to provide scaling. At the same time, the recursive resolver function of the caching resolvers, for simplifying and speeding up database queries from stub resolvers, remains as important as in DNS, or can be considered even more important.

### 3.1.1 Naming

Naming is a critical a part of any data-centric NDN application. Due to the fact that NDN operates directly on application names, i.e., it mashes the network and application layers, application developers must design the data naming model with care, so that it guarantees correct application functionality and leverages all the benefits of the NDN architecture. In particular, applications should be aware that returned Data can be cached anywhere in the network and that the subsequent Interests for the same name (or prefix) would yield exactly the same piece of Data: the application design should not include patterns whereby the same Interest is expected to be satisfied with different Data packets. Moreover, application developers should avoid a system design that mimics the IP-based practice of channel-based communication, e.g., assigning unique/random names to each Data packet and in this process foregoing the benefits of in-network caching. An ideal data-centric design guarantees that each unique piece of Data is named uniquely (retrieved using a unique name in the Interest); at the same time, Interests by independent parties for the same unique Data use identical names.

The primary conclusion from these considerations is that the Data should have the shortest possible (and therefore least specific) name that provides Data uniqueness, enabling potential sharing of these Data between different users in the NDN network. For example, a simple analysis of the iterative query protocol of DNS (see Section 2.2) can reveal that it goes against NDN requirements: the query does not uniquely identify the requested Data. In other words, while exactly the same DNS query is sent out to root, TLD, and SLD servers, the resulting Data can

be completely different, consisting of referrals to the next level or a final (positive or negative) answer.

NDNS design incorporates a substantially changed version of the iterative querying protocol (see Section 3.1.4.1), which includes an explicit separation of questions for the referrals and for the final answers from authoritative servers. The proposed protocol contains a certain amount of guesswork by the resolver and may result in an increased number of required iterations to get the final answer (e.g., the authoritative server cannot give the final answer until it is asked the final question), but the explicit requests completely solve data ambiguity and fully leverage potential of NDN caches. In other words, even though one caching resolver may need to spend several round-trip times to get an answer, another resolver would be able reuse intermediate results of the NDN caches.

### 3.1.1.1   NDNS naming model

Since an NDN name is used both by an application and by the network, it should contain all necessary parts for proper delivery of the Interests towards the intended Data producers (at least the first Interest for a unique piece of Data), for proper demultiplexing of the Interests to the right application at the producer node, and for a proper selection of the desired application data. In short, the name should contain a "routable" prefix, an application demultiplexer suffix, and the application-specific ending. Figure 3.2 shows an example of names for the iterative and recursive NDNS queries, each conforming to the general three-part naming model.

Depending on the query type (which technically are completely different applications), NDNS uses two different application demultiplexer suffixes: "`/DNS`" for iterative queries (the same as names for authoritative NDNS data) and "`/DNS-R`" for recursive queries (the same as names of the caching resolver's data). The

**Iterative queries**

| / | /DNS | /net/NS |
| /net | /DNS | /ndnsim/NS |
| /net/ndnsim | /DNS | /www/NS |
| /net/ndnsim | /DNS | /www/TXT |
| zone to route request | authority NDNS app | TXT query about www label in the zone |

**Recursive queries**

| / | DNS-R | /net/ndnsim/www/TXT |
| /com/google | /DNS-R | /net/ndnsim/www/TXT |
| scope to route request | caching NDNS resolver | TXT query about /net/ndnsim/www domain |
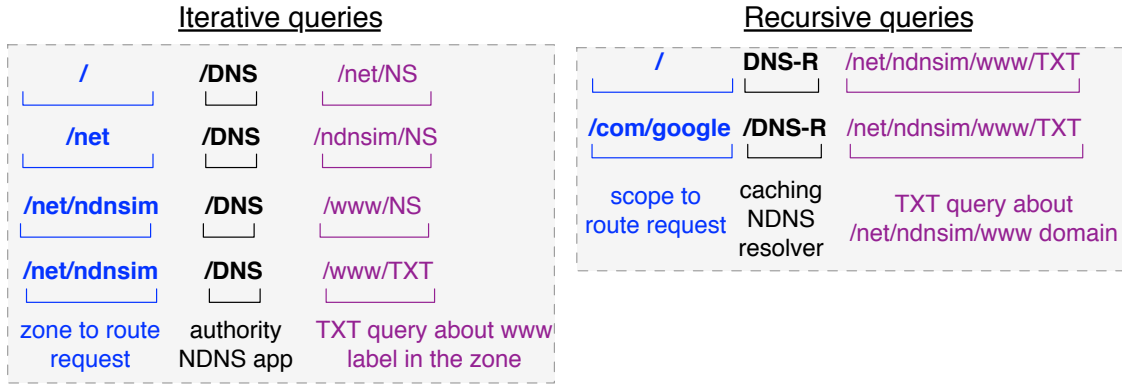
Figure 3.2: Example of iterative and reqursive query naming in NDNS

routable prefix in two different query types has a slightly different semantic meaning. In the case of an iterative query it specifies an authoritative zone name, which is assumed to be properly routed and/or anycasted towards one or more authoritative name servers (see Chapter 6 for a solution that eliminates this unrealistic assumption). In the case of a recursive query, the routable prefix is more or less equivalent to the scope, within which it is expected to be a functional caching resolver. In the above example (Figure 3.2), it is expected that the caching resolver exists in the network's Interest multicast domain, largely eliminating the need for additional caching resolver discovery, e.g., using a DHCP-like protocol. (This is also similar to the existing practice of using multicast DNS or simply mDNS [CK13] for local resource discovery.)

For better visual separation between NDNS-namespace names and supplementary information in Interests and Data packets, all such supplemental information (e.g., application demultiplexer and resource record type) is defined to be formatted in upper case letters.

### 3.1.1.2 Namespace conversions

As noted earlier, NDNS operates within a subset of the NDN namespace, which has unique bidirectional mapping to and from the DNS namespace. The following defines guidelines for an acceptable subset of NDN names used within the NDNS system, as well as conversion procedures to and from DNS names.

**NDN to DNS conversion (dnsification)**    NDNS requires use of NDN names that are convertible ("dnsifiable") to DNS format. In order for an NDN name to be convertible, it should adhere to the following constraints:

- Each component of the name is less than 64 bytes.

- The total length of all components plus lengths of implicit 1-byte separators is less than 253 bytes.

- Each component contains only characters from the DNS character set [Moc87b] and only in lower case.

- Unicode characters (if any) are explicitly mapped into the DNS character set using the Punnycode encoding system [FHC03].

Any name satisfying the restrictions listed above can be converted to a DNS equivalent name ("dnsified") using a simple procedure, as outlined in Figure 3.3: the DNS name is constructed by inverting the order of NDN name components, separating each component with a period (".") for the textual representation, or prepending a one-byte length value (or proper offset value, if a DNS compression mechanism [Moc87b] is used) for the DNS wire format.

**DNS to NDN conversion (ndnification)**    The procedure of converting DNS names to NDN names ("ndnification") is the opposite to dnsification (Figure 3.4),

```
 1: function DNSIFY(NDN name)
 2:     DNS name ← empty name
 3:     for component in NDN name in reversed order do
 4:         if component valid DNS component and is in lower case then
 5:             append component to DNS name
 6:         else
 7:             raise an error
 8:         end if
 9:     end for
10:     if DNS name valid DNS name then
11:         return textual representation or DNS wire format of DNS name
12:     else
13:         raise an error
14:     end if
15: end function
```

Figure 3.3: NDN to DNS name conversion (dnsification)

with two exceptions. First, all DNS names are convertible to NDN names, so the conversion operation always succeeds. Second, in order to chain `dnsify` and `ndnify` operations (the ndnified name should allow dnsification), each DNS name component is converted to lower case.

---

1: **function** NDNIFY(*DNS name*)

2:     *NDN name* ← empty name

3:     **for** *component* in *DNS name* in reversed order **do**

4:         append lowercase version of the *component* to *NDN name*

5:     **end for**

6:     **return** textual representation or NDN wire format of *NDN name*

7: **end function**

---

Figure 3.4: DNS to NDN name conversion (ndnification)

In any further description of the NDNS protocol, the NDN and DNS notation for zones and domains is used interchangeably, as is more appropriate for the particular context.

### 3.1.2 Name servers

The concept of an NDNS name server is almost entirely borrowed from the classic DNS design. As in DNS, NDNS name servers are the programs that authoritatively respond to incoming queries (Interests) in the served zones. NDNS zones can be physically hosted on one or more authoritative NDNS name servers, which can provide efficient horizontal scalability for the NDNS service, as well as ensure zone reliability in the face of potential software, hardware, and link failures and malfunctions.

### 3.1.2.1  NDN data packet management

There are several notable differences regarding how NDNS zones should be managed.[1]  NDNS extensively uses NDN Data packets, which are basically secure bundles of named data, providing not only assurances of data validity (see Chapter 4 for more detail), but potentially enabling internal data integrity maintenance and data corruption prevention.  In other words, if a Data packet for a particular RR set is created as soon as the RR set is modified (which implies it is properly signed), the fact of such creating "seals" the RR set, preventing many accidental and malicious modifications.

The process of signing a Data packet is another potentially important element in the implementation of NDNS name servers, as it always comes with a certain cost: the stricter the signature, the more computational overhead incurred in the signing process.  As a result, the implementation of NDNS authoritative servers should minimize the number of signing operations and do it proactively, immediately after an RR set is modified or RR set is requested to be resigned (periodically or after the zone's key change).  This model of proactive signing has been adopted by the implemented prototype of the NDNS name server daemon.

### 3.1.2.2  Zone data synchronization

One of the crucial functions of DNS/NDNS name servers is to synchronize the zone data between primary (hidden primary) and secondary name servers.  This synchronization requires very careful attention in the implementation.  While zone data synchronization is standardized within the DNS protocol in the form of DNS zone transfer queries ("AXFR"), this is largely an implementation issue.  Many existing DNS name server implementations, including the dynamically loadable

---

[1]As in DNS, NDNS does not specifically define the way name servers are implemented, and it is still choice of the particular implementation on how exactly to manage the underlying database of resource records.

zone (DLZ) extension of Bind name server [Sti02], require out-of-band methods to synchronize the underlying DNS databases, consciously prohibiting potential disproportionate protocol overhead. For example, if a zone contains billions of records and only one record has been modified, "AXFR"-style synchronization would needlessly require transferring all zone records individually to each secondary server.

Following the current operational trend, NDNS does not define a procedure for how the zone data should be synchronized between primary and secondary name servers. A direction that will be explored as part of future work is to apply the new data-centric application patterns for the zone data synchronization. In particular, NDN enables efficient, completely data-centric and decentralized dataset synchronization using the ChronoSync communication primitive [ZA13], which was successfully applied in simple multi-party text conferencing scenarios, as well as within more complex scenarios, such as distributed file sharing (ChronoShare [AZZ13]). Essentially, ChronoSync provides a way for a primary name server to notify secondaries that a new RR set has been generated, each RR set corresponding to an individual NDN Data packet. As soon as the secondaries receive such notification, they can immediately start the standard Interest/Data exchange to retrieve the updated records and install them in the local databases. This way, even if the zone has thousands of the secondaries and there are frequent zone updates (e.g., such as in the case of the ".com" zone), the master will never be overwhelmed with the incoming synchronization requests because of the natural multicast support by the NDN architecture, even without building additional synchronization hierarchies.

### 3.1.3 Resolvers

Following the model of DNS, the design of NDNS provisions two types of the resolvers: *caching resolvers* and *stub resolvers* (see Figure 3.5). Caching resolvers

are applications installed on dedicated servers inside the ISP networks that are responsible for performing iterative queries in response to the incoming recursive queries from the stub resolvers (see Sections 3.1.4.1 and 3.1.4.2). Stub resolvers are simple programs usually embedded inside applications that perform NDNS queries upon application requests.
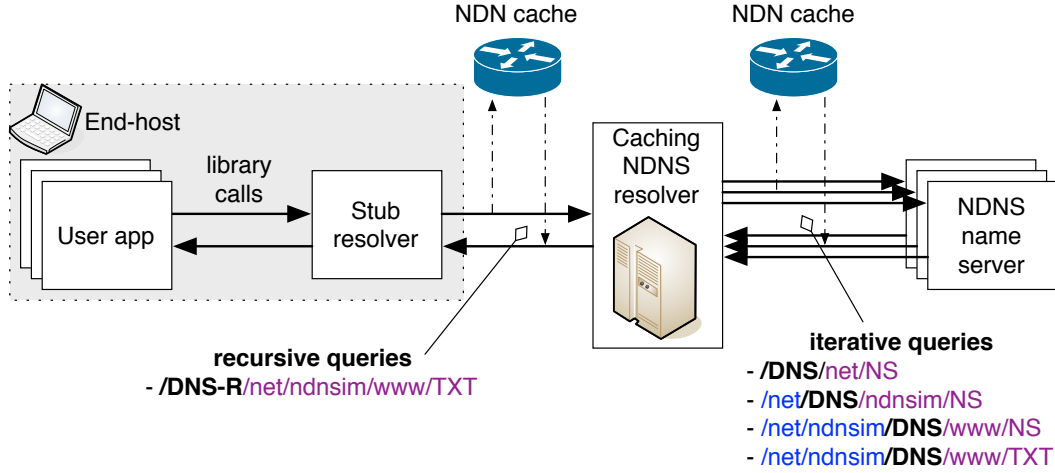


Figure 3.5: Interaction diagram between different types of resolvers in NDNS

Caching resolvers play a crucial role in the DNS protocol: they enable scalability and speed up query resolution. In NDN the scalability function is shared between the application layer of the caching resolvers and the network layer of in-network caches. This makes caching resolvers in NDNS, to some extent, less critical than in DNS. At the same time, the caching resolver is still a necessary component of the NDNS infrastructure, functioning to guarantee application-level cache (NDN's cache is best-effort and is shared between all NDN applications) and facilitate fast resolution of names from the stub resolvers. In other words, the data-centric iterative querying process (Section 3.1.4.1) may require more Interest/Data exchanges than the equivalent DNS process, because of the desire to leverage NDN's in-network caching and share intermediate query results. As a result, the caching resolver can effectively lessen the overhead of NDNS lookups and share query results between different users at the application level.

32

### 3.1.4 Query protocol

The initial goal in the design of NDNS is to borrow intact most of the existing DNS design concept, unless modification is necessary or there proves to be substantial benefits from modifying elements.

One major change necessary due to the communication paradigm shift is an almost complete redesign of the NDNS query format, and to some extent, both the iterative query and the recursive query protocol. In other words, because of the data-centric nature of the NDN architecture, any request in NDNS has to be in the form of an Interest, needs to be appropriately named, and cannot contain any DNS-like wire format payload. The response, on the other hand, is returned in the form of a Data packet that can contain any payload. In particular, in the NDNS python-based prototype implementation using PyNDN [KB11], all iterative queries yield a Data packet containing a properly formatted response message in DNS wire format as a payload, in addition to other required fields such as name (with version component), freshness, timestamp, and signature. The security and integrity of the NDNS system is discussed in detail in Chapter 4, but it is worth noting here that NDN also imposes another major change to the protocol (compared to DNSSEC): security granularity. NDN secures the Data packet, that is, the whole NDNS response message, not individual RR sets, as DNSSEC does. Although this somewhat restricts flexibility of the protocol, it assures data-centricity and all resulting benefits of the architecture (channel independence, efficient reuse of caches, natural multicast, etc.).

The rest of this section discusses necessary changes to (or new designs of) the iterative and recursive query protocols in detail.

### 3.1.4.1 Iterative query

Since NDN inherently does not have the concept of a destination address, there is a need to somehow bring the implicit communication with the intermediate name servers into the explicit plane. In other words, requests for the root zone data should be different from requests for an SLD's zone data. Also, keeping the objective of leveraging NDN caches, requests from different end-hosts towards the root server asking for a ".com" referral should always have the same name, regardless of what the original request was.

To address these challenges, the proposed NDNS design significantly changes how iterative queries are performed:

- Instead of asking a name server, Interests are expressed for data in a specific zone, with the potential of requesting a hint where the authoritative name server could be located (see Chapter 6).

- Each question to different zones is unique, even if asking about the same domain. This way, referrals can be clearly separated from the real answer or subsequent referrals.

- Each step of the iterative query progressively specializes the asked question: the least specific question is towards the root zone (asking for the SLD referral) and the most specific question is to the final zone authority. In this way, intermediate results can be effectively cached and then reused by later Interests from the same or other iterative resolvers.

The last change is the most intrusive and potentially most violating of basic DNS principles: name servers no longer receive the full question that iterative resolvers are searching for, rather a subquestion for intermediate information. The primary loss is that if the name server somehow knows the answer to the final question, it cannot return the answer, since it is being asked a completely

different intermediate question, requiring unnecessary iterative operations. While it is no doubt a shortcoming, this is not a foundational problem; in practice authoritative servers usually have very limited knowledge, therefore the process essentially requires a complete set of iterative queries to get to the final answer.

**Query structure**   The structure of the iterative query introduced in this section still follows the basic principles of the DNS system. There are zones, zones can be further delegated, resource record sets can be placed at all levels of the hierarchs, and access to these sets is performed through a query protocol. (The process of zone delegation itself is performed via installation of "NS" resource records in the parent zone.) The format of an NDNS iterative query is formally defined in Figure 3.6, with associated examples.
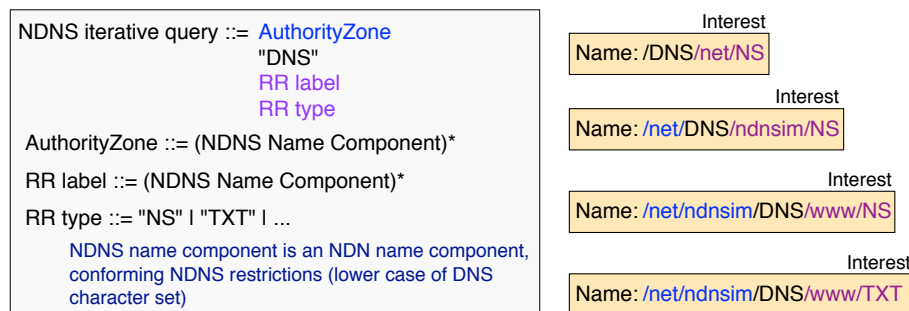


Figure 3.6: Definition and example of NDNS iterative query (naming structure)

The following description is based on a pure data-centric communication model, completely ignoring any location-specific semantics. In particular, even though the zone delegation process in NDNS still uses "NS" RRs, which specify names of NDNS name servers to which a corresponding subzone is delegated, in this chapter we do not associate any location-specific information (like "A" or "AAAA" RRs) with these names. Instead, the fact of the delegation itself is used as a piece of information necessary to proceed with the iterative query process. While it may seem redundant (e.g., why not simply define a new boolean-type RR indicating the fact of the delegation?), this choice allows for inter-operability between

existing DNS and NDNS systems: when NDNS data is delivered over IP, the DNS server would be able to include necessary "A" and/or "AAAA" type "glue" records. More importantly, as described in Chapter 6, even within NDN, names could be further mapped to dedicated "routable" name prefixes that can be used as *forwarding hints* for NDN routers as to which direction (which face) is the best way to find the requested Data. Note that NDN routers still retain full control on where Interests are actually forwarded or whether or not to use in-network storage and caches to satisfy these Interests.

**Query process**   The iterative query process, outlined in Figure 3.7, always begins with the most generic question: the initial question is always about the delegation of the top-level domain. For example, if the input to the iterative query is "/net/ndnsim/www" (or its equivalent dnsified version "www.ndnsim.net") and the objective is to discover the "TXT" RR set, the first question/Interest will be for "/DNS/net/NS" zone delegation Data. This question will be directed to any NDNS root server (directed to the root zone itself), which should be always properly routed (anycasted) in the NDN routers' FIBs. If the returned answer is positive (as should be in our example) and contains one or more "NS" RRs, the query process assumes that the zone ("/net" zone) has been properly delegated and all questions regarding domain names within this zone should be addressed to it directly. In other words, the iterative query process needs to discover whether a higher-level zone ("/net/ndnsim") is further delegated or not. More specifically, the second question in our example would be an Interest for "/net/DNS/ndnsim/NS" zone delegation Data, which NDN routers will direct towards one of the active "/net" zone authoritative name servers or to some in-network caches.

This process of delegation discovery proceeds until it is clear that no further delegation exists. For example, when "/net/ndnsim/DNS/www/NS" Inter-
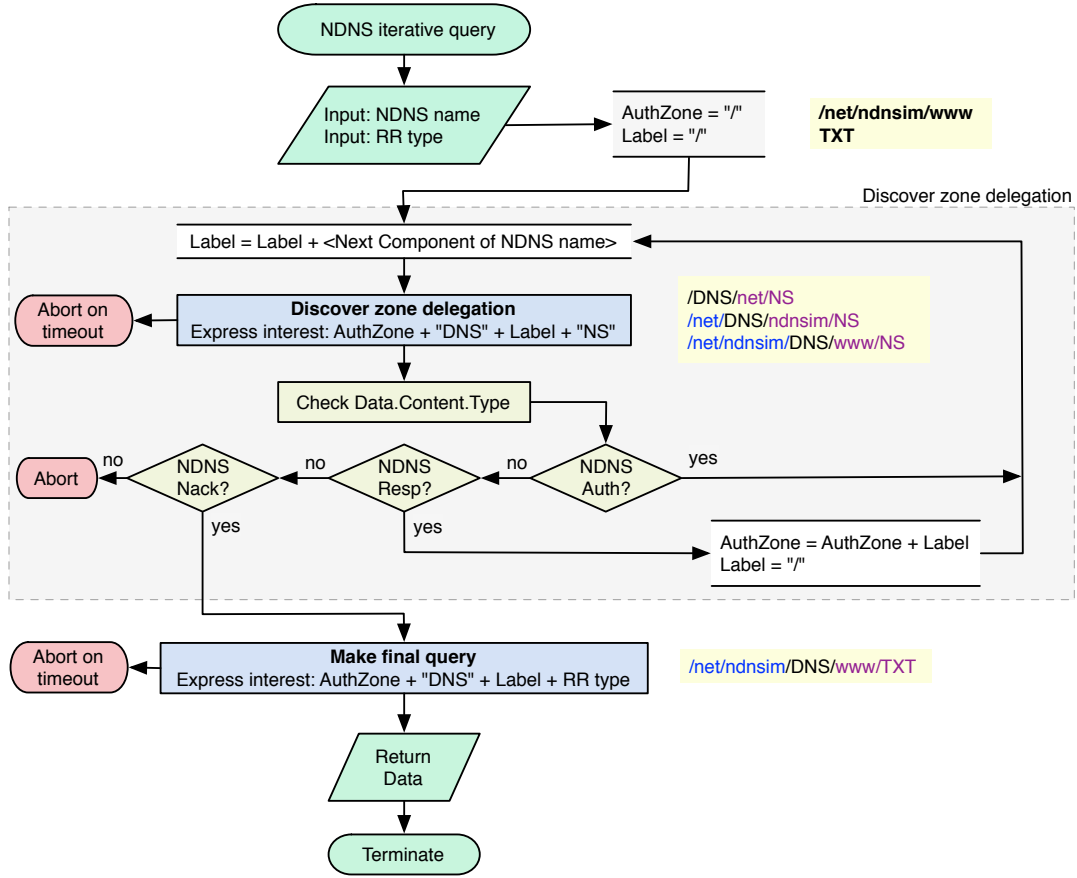
Figure 3.7: Flowchart of NDNS iterative query

est returns a negative autoritative NDNS response, it is an indication that the zone is not delegated, "`/net/ndnsim`" is fully responsible for any RR within the zone, and the iterative querier needs to format and express the final query to the zone. In our case, the querier would need to express the Interest for "`/net/ndnsim/DNS/www/TXT`" Data, which would be authoritatively satisfied either with a valid record or with a negative response.

**Necessary extensions**   As may be inferred from Figure 3.7, the process of finding proper delegation grows more complex than the above example. If it happens that the "`/net`" zone has delegated not "`/net/ndnsim`", but rather the "`/net/ndnsim/www`" zone (while unlikely in the case of top/second-level domains,

37

such delegation is common for third+ level domains), there is a certain ambiguity of what the "/net" zone should return as part of a "/net/ndnsim/DNS/www/NS" Data packet. On the one hand, the authoritative name server does not have the exact answer to the question, and it should return some negative response, which should not be mistaken for delegation termination. On the other hand, the name server has a more specific answer ("/net/DNS/ndnsim/www/NS" Data), but it cannot return this Data since it was asked a completely different question. To resolve this problem, NDNS defines a new "NDNS authority" response type, indicating that the zone does not have the exact match for the question, but it is expecting to receive a more specific question.

**Packet format optimization**  While it is possible to reuse the DNS message packet format for NDNS responses, and it provides full DNS-inherited flexibility, the NDN packet format itself provides a substantial level of flexibility, as well as mandates certain fields that may duplicate functionality of a DNS message. For example, an NDN packet must have a name which is at least the prefix of the corresponding Interest name. (In our prototype implementation the Data name is exactly the Interest name plus one component, indicating the "generation" or "version" of the requested RR set.) The Interest name already fully represents the original question: the zone, the label, and the resource record type, all of which will have to be duplicated as part of the DNS message that is put as part of the NDN Data packet content. Besides the name, NDN mandates other fields that are equivalent to DNS message elements. For example, the "freshness" field serves exactly the same purpose as TTL fields in a DNS message, but just on the network layer instead of the application layer. Since one of the main goals of NDNS design (at least with respect to the iterative query process) is to leverage NDN in-network storage as much as possible, but not to break application-level functionality, it is necessary that the freshness parameter be set to exactly the

same value as the TTL field for requested records (i.e., the NDN Data packet should be cached no longer than TTL). Finally, an NDN Data packet contains (explicitly or implicitly) a "Type" field that indicates the type of carried payload. There is nothing to prevent using/abusing this field to indicate the type of NDNS answer, instead of relying on inference-based distinguishing between negative and positive responses from the DNS message [Moc87b].
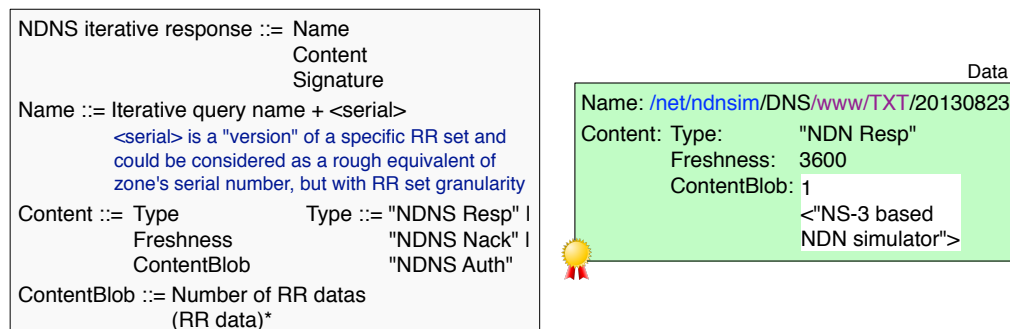


Figure 3.8: Definition and example of NDNS iterative query response

All of the above considerations lead to the conclusion that NDNS should use a more optimal format that not only eliminates unnecessary redundancies, but also is more NDN-friendly. For instance, application design using NDNS could be simplified significantly if there is no need for the application to understand wire format and complex semantics of the DNS protocol. The proposed format for an NDNS response message is illustrated in Figure 3.8. In essence, the content of the NDNS Data packet returned for the iterative query Interest is just a sequence of raw DNS-encoded RR data, while the number of items is indicated in the first byte of the content. The newly defined NDN Data packet types "NDNS Response," "NDNS Nack," and "NDNS Authority" correspond respectively to a positive NDNS response, negative response, and a request for a more specific question to the zone.

### 3.1.4.2 Recursive query

The primary purpose of the recursive query is to enable an optimized NDNS database lookup process and application-level caching of the results. The recursive query process is almost equivalent to the existing DNS protocol, with several exceptions. First, as with the iterative query, the DNS query message is completely replaced by the properly named Interest, indicating the scope (if any), query type, and parameters (Figure 3.9). (It is worth repeating that although the Interest can explicitly specify the intended scope, this scope will not necessarily result in forwarding Interest to a specific caching resolver: the Interest can be diverted or satisfied from caches, i.e., the scope is not the same as the destination address in IP architecture.)
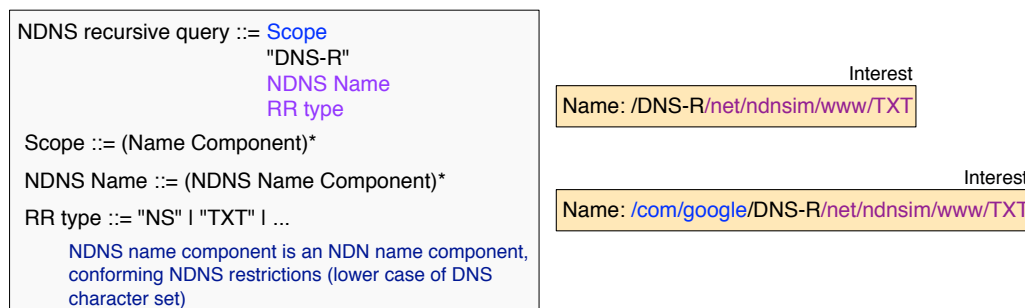


Figure 3.9: Definition and example of NDNS recursive query (naming structure)

The second major difference is the format of the returned Data packet. The NDNS response to the recursive query is explicitly distinguished from the iterative query response. In particular, the returned Data packet is not a DNS-formatted message or the optimized NDNS packet, but rather an NDN Data packet that encapsulates another NDN Data packet, which, as shown in Figure 3.10, is the final answer to the original query that the caching resolver looked up in its caches or requested from the proper authoritative name server.

For example, if one sends a recursive query with the scope "/" or "/localnet" (equivalent to using an ISP's caching DNS servers; for example, if the scope is
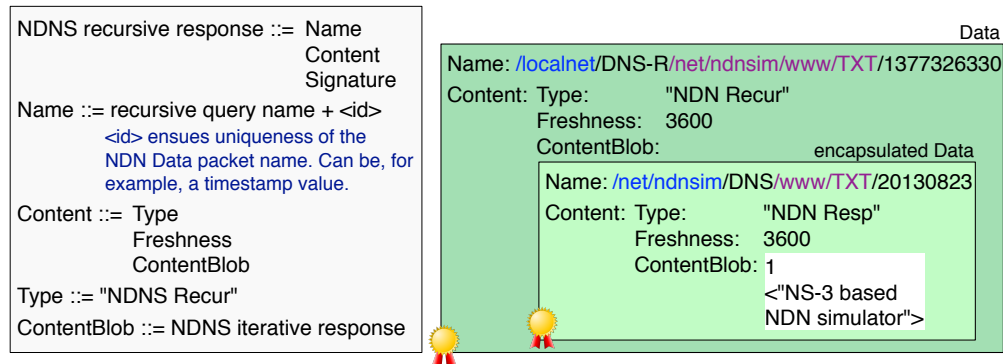
Figure 3.10: Data packet format for NDNS recursive query response

selected/configured to be "/com/google", then it would be equivalent to using Google's public caching DNS servers) with the query asking about the "TXT" record(s) for the "/net/ndnsim/www" domain, the actual query will appear as "/localnet/DNS-R/net/ndnsim/www/TXT", while the returned NDN packet will contain an internal payload with another Data packet "/net/ndnsim/DNS/www/TXT", which was looked up from the application cache or requested through the iterative query process.

There are two main reasons why we chose such a composite packet format (encapsulation) for the NDNS recursive query responses. First, the Data packet names must exactly match Interest prefixes; i.e., the answer names must include Interest prefixes (and since before an iterative query is performed, it is not known which zone has authority over the requested record, the stub resolver cannot ask the right question). Second, and maybe even more important, is the granularity of the security in NDN/NDNS (see Chapter 4). NDN signs the whole Data packet (= RR set); therefore, to preserve the security properties of the iterative query result, it is necessary to keep the whole packet intact. Therefore, a composite encapsulation-based packet format is the only option. This is conceptually different from DNSSEC, where several independent RR sets and their signatures can be included as part of one DNS message.

While the composite format results in a certain level of redundancy (e.g., the recursive query name and the encapsulated iterative query name largely repeat each other), in reality, the encapsulated iterative query Data packet name includes additional information, which was unknown before and could be utilized later. In other words, the name of the encapsulated iterative query explicitly specifies the zone responsible for the record, which, while we are not explicitly defining it in NDNS design, can potentially be utilized in implementations of the stub resolver; e.g., the resolver may elect to ask an iterative query question if it knows exactly to which zone/authority such a question should be addressed.

### 3.1.4.3 Recursive and iterative query interaction

Figure 3.11 illustrates an example of a recursive query, issued by the stub resolver, and the underlying iterative query resolution, performed by the designated caching resolver.

Note that each returned NDN Data packet contains proper security credentials, allowing any receiver to verify the legitimacy and accuracy of the answer. At the same time, in the case of a recursive query, the returned Data packet actually contains two signatures, which have differing importance. The one (the outer) signature is generated by the caching resolvers, specifically for the request (of course, the created Data packet can be later cached and used for error recovery, or shared between different users, if they happen to be interested in the same Data). The other (the inner) signature is the one that the authoritative name server has assigned to the RR set. Depending on the trust relations between the stub and caching resolvers, the stub resolver may either trust the answer from the caching resolver entirely (verifying only the caching resolver's signature, as is usually the case with local ISP caching resolvers) or trust only the original authoritative name server signature (e.g., in the case of a public NDNS server). Moreover, a particular implementation of the caching resolver may elect to use a
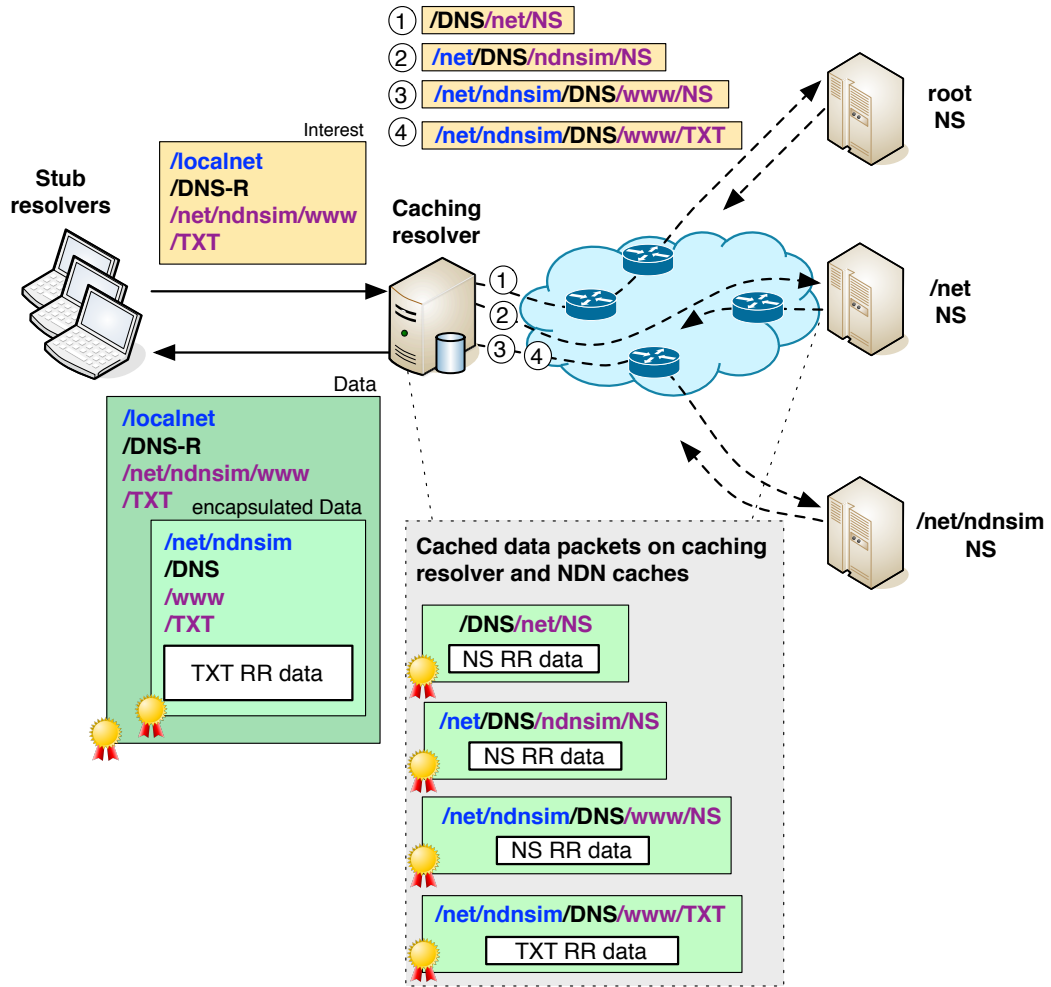
Figure 3.11: Example of DNS recursive and iterative query

very weak but fast signing algorithm (to provide basic assurance of the returned
Data packet integrity) without compromising the security of the NDNS protocol
in general, since it is always possible to double-check whether the returned answer
was indeed generated by a legitimate authoritative name server, which should be
pre-created in advance (see Section 3.1.2) and can use stronger cryptographic
signatures.

## 3.2 NDNS implementation and evaluation

We have implemented a python-based prototype of the NDNS system that is publicly available at `http://github.com/cawka/ndns`. The implementation includes an `ndns` python module, as well as a set of command-line tools that provide a simple and user-friendly command-line interface to the module functions. In particular, we have used a PyNDN module [KB11] to provide necessary NDN support for the module (e.g., creating NDN Data packets, expressing Interests, etc.), and a `dnspython` module to provide abstractions and wire format conversion for DNS protocol elements (DNS message, DNS name, RR data, RR set, etc.).

The iterative and recursive query process has also been implemented in JavaScript using the `ndn-js` library [STC13] and been made publicly available as well [Sha13]. While our immediate plan for future work is system deployment on an NDN testbed and enabling its use as a universal database for various application needs, this deployment may require implementation (or reimplementation) of the NDNS library in other languages (C, C++, Javascript, etc.) as well.

The available command-line tools in the implemented prototype can be divided into four groups, based on their function:

1. `ndns-daemon`: an implementation of the authoritative NDNS name server.

2. `ndns-dig`: a command-line tool providing an interface to the `ndns` module to execute iterative and recursive NDNS queries.

3. `ndns-create-zone`, `ndns-destroy-zone`, `ndns-zone-info`, `ndns-show-zone`, `ndns-add`, `ndns-rm`, `ndns-show`, and several others: command line tools for local management of the NDNS zone for the authoritative name server.

   These tools are necessary and the NDNS zones are not simply managed using plain-text zone files (which is a common practice in DNS), in that NDNS stores and manages the zone as a collection of pre-created NDN packets.

44

Therefore, adding or removing records in the zone involves operations of creating, modifying, and removing relevant NDN packets, while creating and destroying the zone itself requires proper security management (see Chapter 4).

4. `dyndns-keygen`, `dyndns-info`, `dyndns-add-rrset`, and `dyndns-rm-rrset`: a set of command-line tools that can be used for remote and secure management of the NDNS zone records using a dynamic NDNS update protocol (see Section 4.4 for more detail).
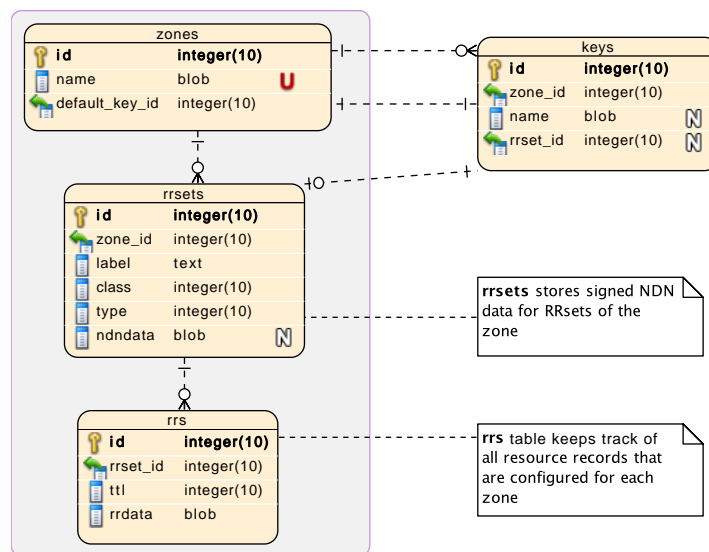


Figure 3.12: Database schema for the NDNS prototype of the authoritative name server

The zone management in our implementation is based on the `sqlite` database with the database schema outlined in Figure 3.12. The main idea behind the command-line tools for local zone management and the use of a `sqlite` database is to provide simple access to NDNS zone abstractions (zone, RR sets, individual records) and at the same time enable pre-creation of NDN data packets containing RR set data, which can be immediately returned when requested. This minimizes

(and potentially eliminates) the need for online NDN data packet creation, which can be prohibitively costly, e.g., when a very strong cryptographic signature algorithm is used.

Also, as pointed out in Section 3.1.2, NDN data packets provide additional assurances that the data stored in the database have not been corrupted maliciously or unintentionally (e.g., due to disk failure). In addition, the model used provides a partial database backup: the database structure is a re-generateable "index" to the stored NDN data packets with NDNS content.

### 3.2.1 Methodology

To perform our experimentation, we used our ndnSIM simulation platform (see Chapter 7), which was extended to support pure application-based evaluations.[2] In addition to enabling a full application API that conforms to the unified NDN API guidelines,[3] we added a PyNDN-compatible python interface to ndnSIM, allowing it to run the unmodified NDNS authoritative name server prototype and iterative query code within the simulated environment. To make sure the standard system-level python libraries, such as `time` and `random`, return simulation-aware data, `time.time()` returns current simulation time, not the real time, and `random.random()` returns simulation-run specific value, so that multiple runs of the same scenario produce matching results. This approach in general is equivalent to the NS-3 DCE effort [Lac10], but is aimed specifically for python-based applications. While the python-based application simulation is no doubt slower than the equivalent C++ implementations, python is a great tool for fast prototyping of applications. Evaluation of the real code in the large-scale simulated environ-

---

[2]Since the initial goal of the ndnSIM was to enable evaluation of network-layer operations of the NDN architecture, the initial implementation contained a very limited API for applications.

[3]As of the time of the writing, ndnSIM has been made compatible with the initial specification NDN API guidelines and is expected to be further modified as soon as the specification finalizes. When completed, the new (and ported existing) NDN applications will be source-code compatible with ndnSIM and easily run inside the simulation platform.

ments is not only important for understanding properties of the application and application-network interactions, but also an important way to ensure implementation correctness, address memory problems, and identify/correct computational bottlenecks.

To perform our evaluation we chose a modified version of a Rocketfuel-mapped AT&T topology [SMW02], consisting of 625 nodes and 2,101 links (Figure 3.13).
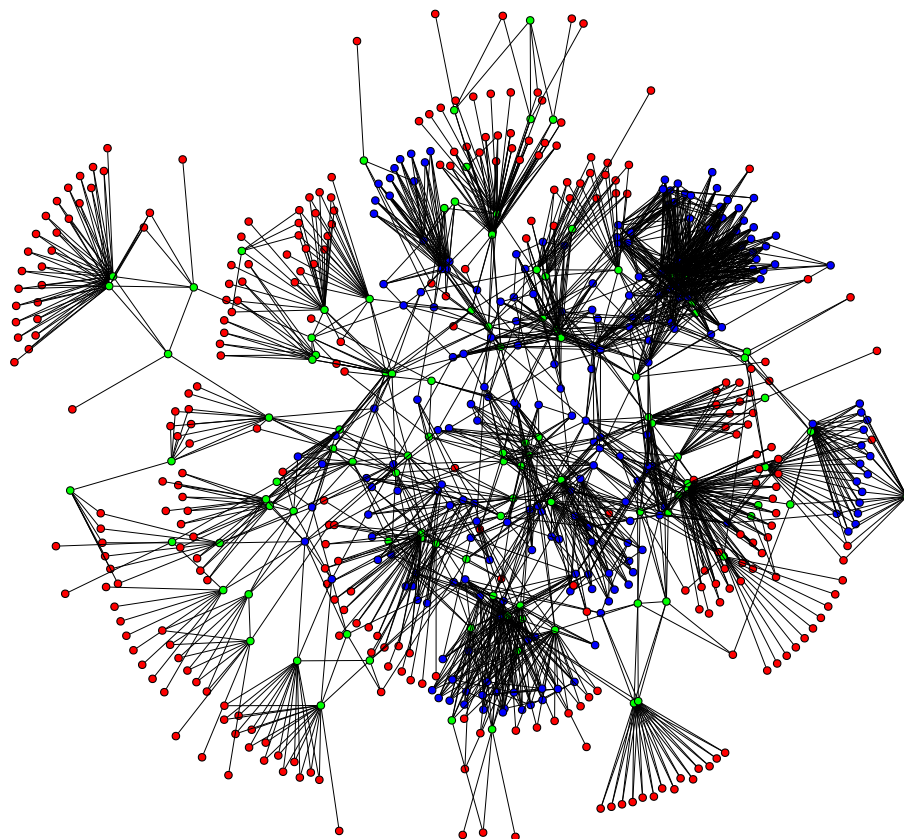


Figure 3.13: Modified version of Rocketfuel AT&T topology with assigned classes of nodes: clients (red), gateways (green), and backbone (blue)

Our modifications include removing a small number of disconnected nodes from the original measurement-based topology and adding a few additional links to ensure the desired topological parameters. We applied a simple heuristic (`http://`

`github.com/cawka/ndnSIM-sample-topologies`) to separate the topology nodes into three categories: clients, gateways, and backbone nodes. All nodes with degree one were assigned to the client category. All nodes with degree two that are not directly connected to other client nodes, were also assigned to the client category (296 client nodes in total). All nodes that are directly connected to clients were assigned to the gateway category (108 nodes), and the rest were assigned to the backbone category (221 nodes). In order to fully emulate the concept of the backbone (the core transport for the network), we added at random several links to ensure that a subset of backbone nodes forms a fully connected graph. This topological property allowed us a simple set up of Interest forwarding in the network that follows the "valley-free" policy [GR00] of inter-domain routing on the Internet: traffic between different ISPs (gateways) never goes through client networks (clients), and in most cases (unless there is an explicit peer-to-peer agreement) involves communication through the tier-1 network (backbone). In each run scenario, we assigned small random weights to backbone-to-backbone links, larger random weights for backbone-to-gateway links, even larger weights for gateway-to-gateway links, and the largest weights for client-to-gateway connections. After this link weight assignment, correct routing is set up through a shortest-path calculation using the Dijkstra algorithm for each node in the topology.

In our evaluation we used the relative evaluation metric, representing a relation of the number of Interests (iterative queries) that reach the authoritative name server(s) versus the total number of issued queries. This metric provides a measure of the network-level support that facilitates performance of the evaluated NDNS protocol.

### 3.2.2 Parameters

The main objective of the evaluations is to understand what benefits stem from the tight integration of an application and the NDN network architecture (i.e., a

better application-support within the NDN architecture), potentially making application implementation simpler and more scalable without requiring application-level dependencies (e.g., no need for additional load balancers). In particular, our primary interest was to estimate how NDN in-network caches can reduce the load on authoritative name servers, compared to the existing DNS deployment. For this purpose, we obtained a DNS query trace from a large ISP containing about 9 million queries issued within 10 minutes in May 2010. Without loss of generality and to simplify our experimentation setup, we removed all invalid queries and selected queries towards the ".com" zone only, totaling about 1 million queries.

As pointed out before, the caching resolver in DNS is the only mechanism that ensures scalability property, e.g., it aggregates the incoming recursive queries and prevents overloading the authoritative name servers with repeated requests. At the same time, if the users choose to use different caching resolvers, then even if the users send exactly the same sequence of queries, the authoritative name servers will inevitably observe repeated requests for the same records. NDNS has quite different behavior and properties. The NDNS version of the caching resolver shares caching responsibility with the whole NDN network, and in-network NDN caches in particular. The iterative queries, originating from different resolvers (be it a caching resolver or a stub resolver electing to send the iterative query), can be effectively resolved and shared using NDN's caching. Of course, caching capacity of the caching resolvers can be significantly larger and entirely dedicated to the NDNS functionality, while the NDN network shares cache space among all NDN applications. In other words, NDNS is still an important component of the NDNS infrastructure, but (as is our hope) a less critical component.

To understand the level of NDN in-network cache assistance, we designed the following simulation scenario. We selected 200 client nodes ($\approx$70% of all client nodes in the topology) to act as caching resolvers for users, who issue a subset of the trace-based queries. (For simplicity, we did not explicitly simulate users

and the recursive query process, directly feeding requests from the trace into the caching resolver.) The trace subsets were generated by randomly slicing the original DNS query trace, with different slicing for each individual simulation run. The positions of caching resolvers were also randomly assigned for each run of the simulation.

The whole NDNS infrastructure used in our simulation consists of one root zone NDNS server and 13 "/com" zone servers, which were anycasted NDN routing. Having multiple "/com" servers reduces the potential of cache sharing in the NDN network, since requests from different caching resolvers may be directed to different name servers, but we intentionally chose this setup, because it more realistically represents the network environment. Similar to the case of the caching resolvers, the positions of the root and the "/com" zone name servers were selected randomly for each individual simulation run, but instead of client nodes, we used a backbone node set, i.e., name servers belonging to the core of the network, not its edges. Figure 3.14 outlines the overall structure of our simulation experiments.
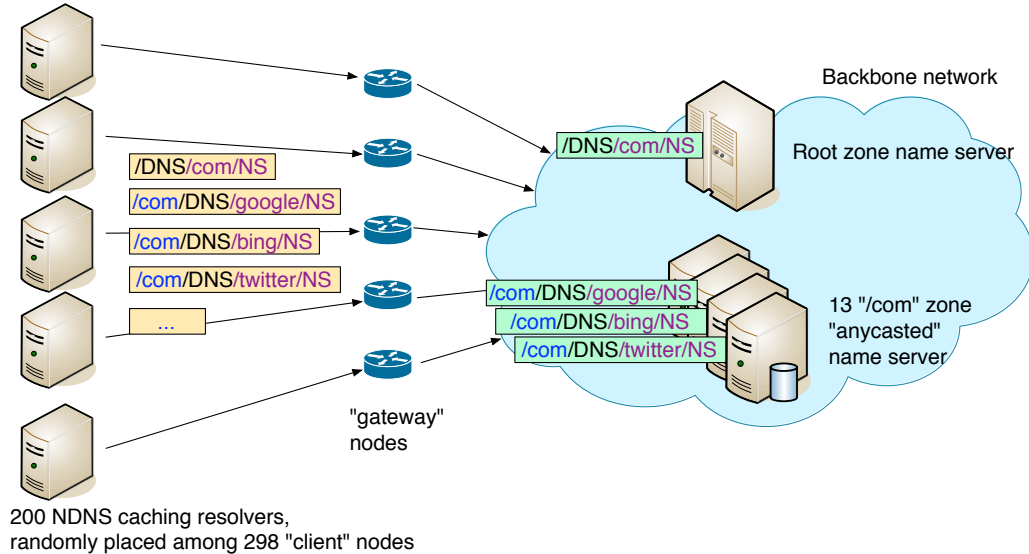


Figure 3.14: The overall structure of the NDNS simulation-based evaluation

In our evaluations we made another simplification that has a certain im-

pact on the results, but still preserves their generality. Our iterative trace-based queries were performed only partially, discovering only authority delegation for the second-level domain name. If the trace input is, for example, "`www.google.com`", in our experiment we issue a "`/DNS/com/NS`" query for the root zone and a "`/com/DNS/google/NS`" query for the "`/com`" zone. The first query actually reaches the root server only a few times during the whole simulation, since the returned Data packet is cached everywhere. The programmed reply for the second query is a negative answer, after which the iterative query process terminates, assuming that a "`/com`" zone has the authority over "`/com/google/www`" record(s).

### 3.2.3 Results

To measure the benefit that NDN in-network caches can provide, we varied the capacity of the in-network caching. In our simulation we did not have any other traffic, so the whole in-network cache capacity was devoted entirely to NDNS. While in real environments the cache will be shared among many different applications, if the NDNS system is widely used, NDN routes may elect to reserve certain amounts of in-network caching resources entirely for NDNS data caching, or employ adequate cache replacement policies that prioritize frequently accessed NDNS data. Also, to fully understand the caching effect, we varied cache capacity from very small (10 packets) to large (10000 packets), which allows us to see the impact (if any) of even small caches in the NDNS protocol.

It should be noted that we also have followed a general practice for caching resolvers in that we have not restricted the application-level capacity. In other words, if a portion of the trace selected for a particular caching resolver contains a duplicate query (the same second-level domain name), such a query is sent out to the network exactly once.

Figure 3.15 summarizes the cumulative effects of NDN caching on the NDNS

protocol in terms of the absolute number of requests (Interests) received by the authoritative NDNS servers for each selected in-network cache capacity. The presented values are averages among five independent simulation runs for each scenario, with 95% confidence intervals. Figure 3.16 gives an alternative representation of the same results, plotting the percentage of the requests that are satisfied using NDN caches. These graphs also show results for different caching strategies (i.e., replacement policies) that can be employed by NDN routers. The main point of showing different caching policies is not simply to show one caching policy is better than another, but to highlight that different cache decisions (potentially made in a collaborative fashion) can substantially improve the performance of frequently used NDN applications, such as we hope NDNS will be. Moreover, these decisions are completely application-independent (since they are performed at the network layer) and can potentially benefit any application running within an NDN architecture.

We also plotted per-node cache utilization for one of the simulations run, mapping the value onto the topology.[4] Figure 3.17 shows the results for one of the runs with LRU cache, but the general characteristic is common to all of the runs.

From the obtained simulation results we conclude that NDN can indeed provide substantial benefits for an NDNS infrastructure, even with relatively small in-network cache capacities. This finding confirms that a lot of critical functionality is shifted in NDN from the caching resolvers to the NDN network itself. While an ISP must still deploy a local caching resolver to further reduce duplicate NDNS traffic and improve the query resolution experience for users, the ISP may opt to invest more in general NDN in-network caching and storage, deploying a relatively low-capacity caching resolver. In turn, the deployed caching capacity will be used

---

[4]Node positions on both Figure 3.17 and Figure 3.13 were assigned using the "spring" model [KK89] that allows observing some topological properties of the network without any relation to the geographical coordinates of nodes in the actual AT&T network.
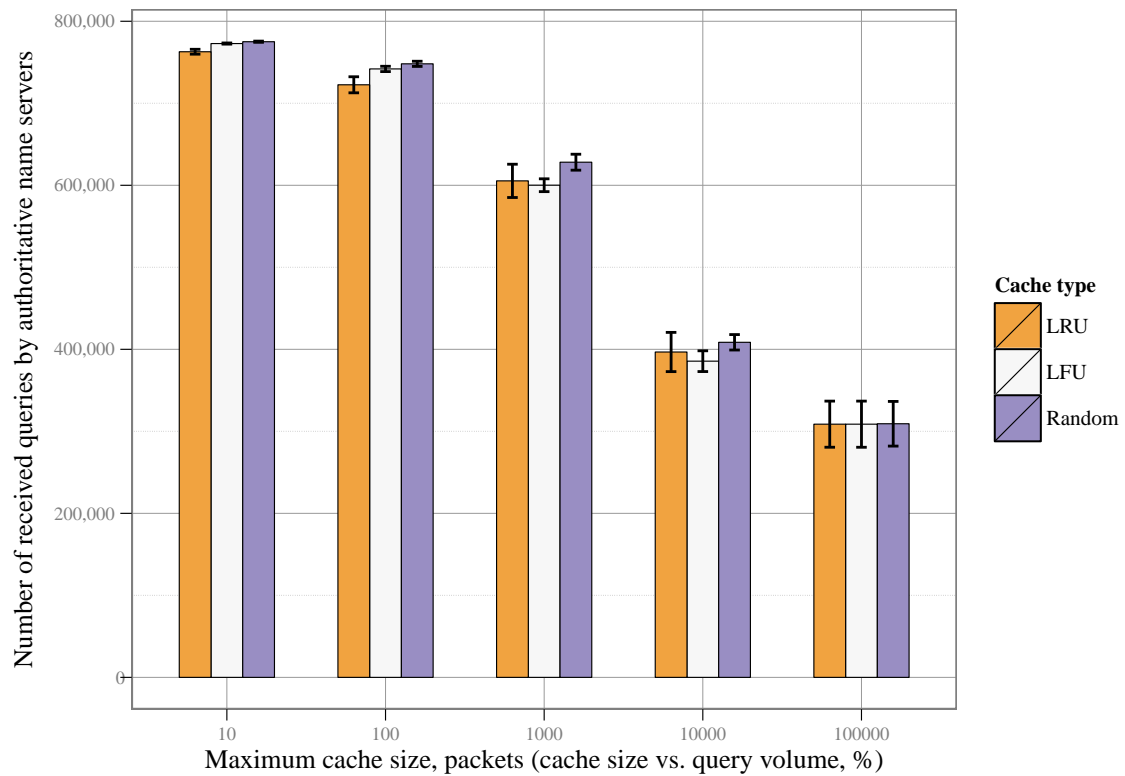
Figure 3.15: Absolute number of requests received by authoritative NDNS servers versus cache sizes
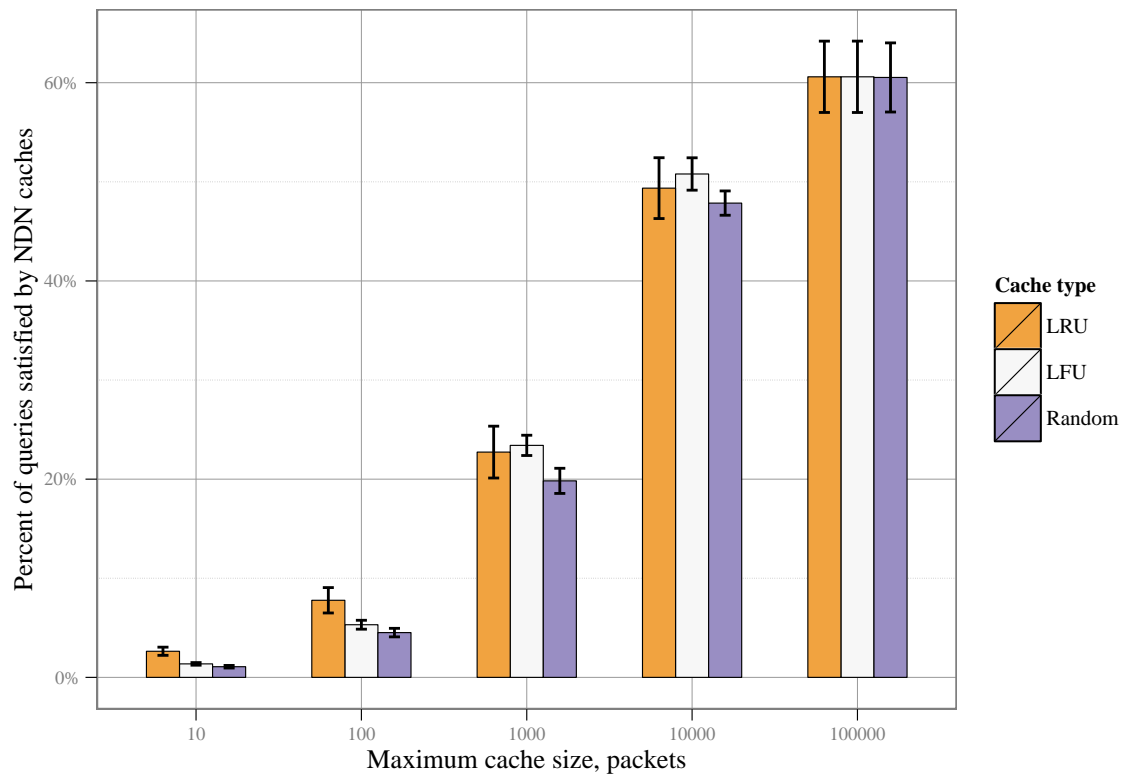
Figure 3.16: Percentage of queries from caching resolvers answered using NDN caches
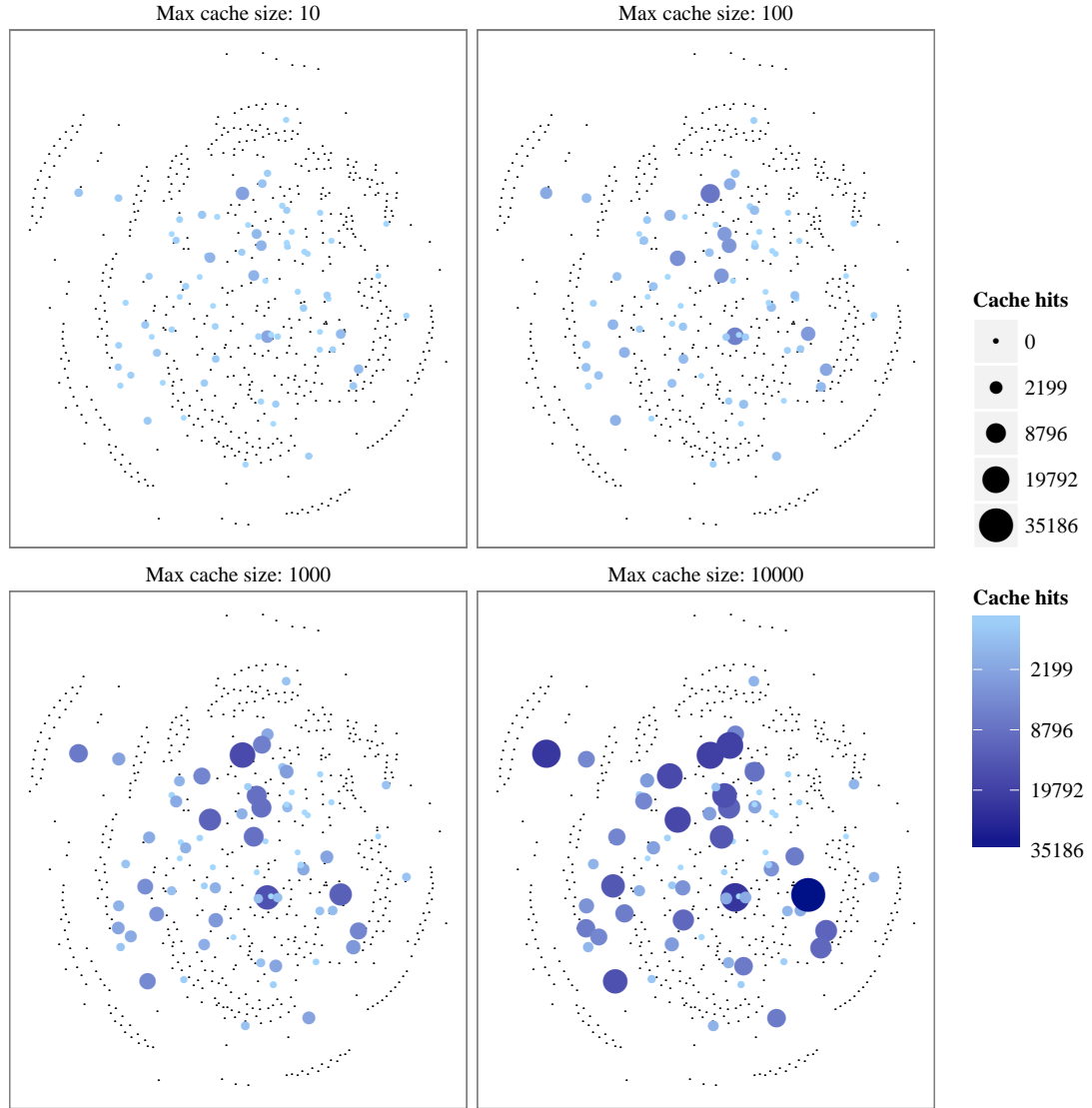
Figure 3.17: Example of per-node cache utilization (cache utilization concentrations) for one simulation run with LRU cache

not only for a particular NDNS application, but also for all other applications, and further reduce use of the costly inter-AS links.

# CHAPTER 4

# NDN-based security of NDNS

Security is the crucial element in any public database system such as DNS or NDNS. While in the DNS system security was not a part of the original design, the usage practices showed the need to incorporate a security extension (DNSSEC [Eas99], Section 2.2.1) as part of the protocol, eliminating various attacks on the database and database users. This experience was one of the basic motivations of incorporating security as a mandatory building block in NDN architecture (i.e., NDN requires each Data packet be properly cryptographically signed), requiring that we design security into the NDNS system.

The objective of the NDNS security design is to merge the best designs and operational practices of the established DNSSEC protocol with the security primitives provided by the NDN architecture.[1] This way we hope to support the same level of security not only in the application layer, but also in the network layer. (As noted previously, the separation between layers in NDN is relatively fuzzy, as both the network and application layers use the same basic communication units: the same names, Interest, and Data packets.)

Figure 4.1 lists the similarities and differences between security elements provided by the DNSSEC protocol and NDN architecture. In general, both largely overlap: the named pieces of data (RR sets in DNSSEC, Data packets in NDN)

---

[1] While our initial idea was to follow directly all the elements of the DNSSEC infrastructure in addition to the security elements provided by the NDN architecture, this choice inevitably creates an additional burden of maintaining two separate security infrastructures (already maintaining one is a notable challenge) and does not fully utilize all the benefits of a tight application-network integration provided by the NDN architecture.

|  | DNSSEC |  | NDN |
|---|---|---|---|

– each RRset is bundled with RRSIG

– each Data packet is bundled with a Signature and KeyLocator

– RRSIG "specifies/hints" DNSKEY **RR set** used to produce signature using "Key tag"

– NDN's KeyLocator refers to the **unique** key-certificate name used to sign data packet

– DNSKEY RRset is signed by KSK DNSKEY

– Keys-certificates are also Data packets, thus can be further signed, e.g., using KSK or ZSK

– KSK "belongs" to the zone and is authorized by parent's zone using DS record

   • KSK is self-signed

– KSK can still "belong" to the zone, but serve authorization via a proper signing chain

   • KSK signed by parent zone's key

Figure 4.1: Similarities between security elements of DNSSEC and NDN

in both cases are signed and include signature bits along with an identifier that specifies which key to use for the verification process (RRSIG RR data with key ID and key label in DNSSEC, and Signature block with KeyLocator in NDN). At the same time, there are several notable differences that have an impact on the NDNS protocol design. First, though it is a relatively minor difference, the DNSSEC specification does not provide a unique identity for the signing key: the key ID and key label provided in RRSIG RR do not uniquely identify a key in DNSKEY RR set, basically requiring verifier implementations to try keys from the RR set sequentially in order to check the validity of the signature (the number of keys in DNSKEY RR set is usually very small, which has little to no impact on the performance). Second and more important is the *security granularity.* If DNSSEC mandates signing of individual RR sets, which can be included together into one DNS message and returned to the requester within one DNS query, NDN signs the whole Data packet, which is the basic data exchange unit in NDN. Essentially, this requires either multiple explicit queries for specific RR sets (as in the case of the iterative NDNS queries; see Section 3.1.4.1) or double-signing and encapsulation techniques (as in the case of the recursive NDNS queries; see Section 3.1.4.2) to preserve the security properties assigned by the authoritative NDNS name servers.

Another aspect that can be considered a difference is that DNSSEC supports a richer set of the cryptographic signature algorithms than the currently supported initial NDN specifications and implementations. However, this is not a conceptual difference, and as NDN gains wider adoption, additional signature algorithms will be added to the specification and implementations, based on user and application demand.

## 4.1 NDNS extension for key-certificate storage

The KeyLocator field in a given NDN Data packet must uniquely identify another Data packet that contains the key-certificate needed to verify the given Data packet. This requirement may both appear at odds with the NDNS's way of storing the complete RR sets in each individual Data packet (e.g., NS RR set "`/net/DNS/ndnsim/NS`") and seemingly makes the NDNS system no longer applicable as a universal storage for the keys necessary to provide security for itself. However, it is trivial to define a new customized RR type ("NDNCERT") that requires exactly one RR data be associated with the RR set (i.e., a so-called *singleton RR set*), the same way the "SOA" singleton RR is defined in DNS.

As an additional design simplification and optimization, we define the content of all singleton NDNS Data packets to include just raw content of RR data, without additionally specifying the number of RR datas (as defined in Section 3.1.4.1) and without the length of RR data, as defined by DNS wire format specification [Moc87b]: the former can be automatically inferred from the RR type and additionally indicated by the "Subtype" field in Data packet meta-information section, and the latter can be directly taken from the "ContentBlob" length (see Figure 4.2). While this optimization puts an additional burden on the name server and general resolver (`dig` command) implementation, it really gives NDNS a much greater potential. In other words, it can be used as an underlying supporting

database for applications without requiring any understanding of NDNS semantics. The only operation needed is to define a new application-specific singleton RR type. The example on Figure 4.2 shows how the newly defined "NDNCERT" singleton RR is encoded into specialized version of the NDNS iterative response, which can be easily used by security enforcement implementations (security policies, discussed in the following sections).
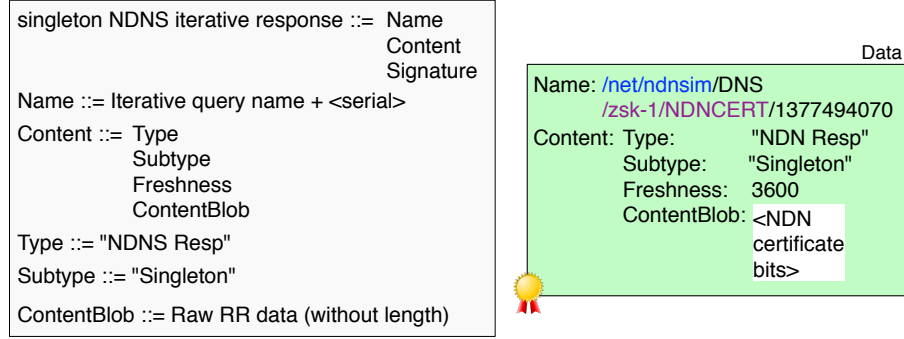


Figure 4.2: Singleton NDNS iterative query response

Note that the specific format of the key-certificate content is outside the scope of the present thesis, and in our prototype implementation, the content of NDNCERT is a simple DER-encoded RSA public key. This format is anticipated to be replaced before the initial rollout of NDNS with an extended certificate format [Yu13], which is currently in active development among the NDN project team.

## 4.2   Security policies

The general NDN security specification [NDN13b] mandates that Data packets be signed with a valid cryptographic signature and that the Data packet should specify the key name[2] (KeyLocator) needed to verify this Data packet. This

---

[2]The NDN key is actually a certificate, since each NDN data packet is properly signed and includes its own KeyLocators. The key as well is an ordinary Data packet, happen to carry cryptographic content. This applies also for self-signed keys-certificates, which include a

general specification provides just the basic mechanisms, which can be used to implement any model of the security infrastructure, including but not limited to the commonly used PKI [AL03] and DNSSEC [Eas99] security models.

The missing part, which is partially addressed in this chapter and is an active research area for the NDN project team, is the need for a unified (or standardized) way for the application and application developers to specify the *security policies*. In other words, in addition to the basic security elements of NDN, the real security model requires a number of additional parameters be satisfied in order to judge that a particular Data packet is indeed valid and signed with a proper signature. In particular, each NDN key-certificate Data packet, alongside the raw bytes of the corresponding public key, should contain encoded validity and other meta information, similar to how it is done in X.509-encoded certificates [CSF08]. This validity and meta information can be used by the security policy as an additional element to decide the validity of a particular signature.

Also, the key-certificate itself needs to be properly "authorized" to sign the specific piece of Data. The current practice with PKI certificates is to include the subject that the certificate is allowed to be used for as part of the meta information ("common name" field). For example, for HTTPS certificates, the "common name" field of the certificate issued by the certification authority includes one or more domain names (wildcarded domain names) that are authorized to be signed with the certified key. In other cases, the common name can include a listing of the email address for email certificates and a company's or developer's name for the code signing certificates.

In NDN, each Data packet, including the key-certificate Data packet is securely named, making it possible to use these names directly for authorization purposes, instead of extraneous fields inside the application-defined certificate format. For example, if one wanted to implement an NDN equivalent of the HTTPS

---

KeyLocator pointed to itself.

security model for the Data packet "`/net/ndnsim/www/index.html`", one would want to define a security policy that required this Data packet be signed with a key-certificate named "`/net/ndnsim/www`". (To follow this PKI security model, "`/net/ndnsim/www`" would further need to be signed with the key of one of the certification authorities.) Restated another way, the function of the "common name" field can be fully represented by the relation between names of the Data packet itself and the key-certificate Data packet, which is identified in the KeyLocator field.

The example mentioned above is not entirely realistic, since the actual key-certificate Data packets, as any other NDN Data packets, need to contain additional name components to ensure that the Interests are routed to the desired places, are demultiplexed to the desired applications, and contain enough information to identify what data is being requested (see Section 3.1.1). However, the described semantics can be easily captured with a slightly more complex application security policy that not only matches Data and key-certificate names, but also prior to the matching performs a deterministic name reductions (see Figure 4.3).
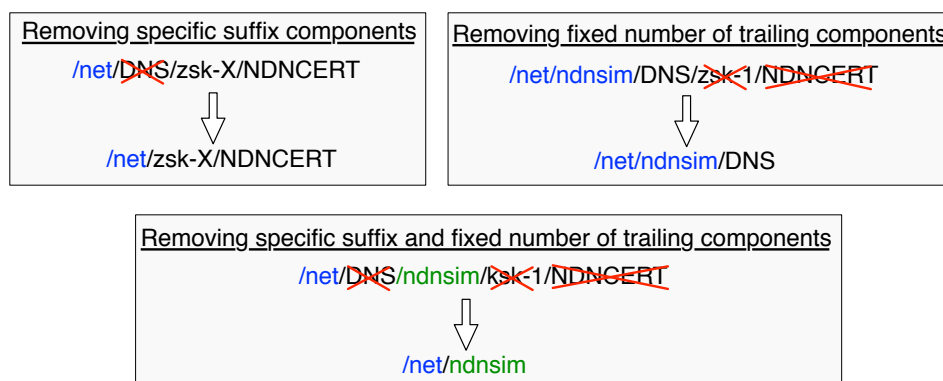


Figure 4.3: An example of deterministic NDN name reductions by the application security policy

The concept of a security policy is not a new and is ubiquitously used (at least in its implicit form) in all secure applications, including HTTPS, secure email com-

munication, and DNSSEC. For example, most browser applications are designed to notify the user when a page is not properly encrypted, not only when there is a cryptographic signature mistmatch, but also whenever there is mistmatch between the requested domain name and the "common name" field, honoring possible wildcards. In our case, we simply moved the policy definition closer to the network layer (following a general trend in many NDN applications).

For NDNS we have defined the security policy specified in Figure 4.4, which ensures that NDNS largely conforms to the DNSSEC security model: zone records are signed with zone signing keys that are signed with key signing keys that are authorized by the parent zone, down to the root zone key, which acts as a NDNS root trust anchor. Details of the security policy and insights behind the defined security delegation are explained in the next section. Also, note that the policy is defined using NDN regular-expression language [Yu13, AZY13], which is essentially a simplified form of general-use regular expressions, with name-component-based rule definition and minor extensions: "<>" matches any single NDN name component, "<>*" matches any number of NDN name components, and "<regexp>" matches any single NDN component that matches the regular expression "regexp".

| Trust anchors | | |
|---|---|---|
| Key name | NDNCERT data | Authorized namespace |
| /ndns-root/%01 | <raw NDNCERT RR> | / |
| ... | | |

| Namespace authorization rules | |
|---|---|
| Key-certificate name conversion rule | Data name conversion rule |
| (<>*)<DNS>(<>*)<><NDNCERT> => \1\2 | (<>*)<DNS>(<>*) => \1\2 |
| ... | |

Name conversion examples:

/net/ndnsim/DNS/zsk-1/NDNCERT
　　=> /net/ndnsim
/net/DNS/ndnsim/ksk-1/NDNCERT
　　=> /net/ndnsim
/net/DNS/zsk-X/NDNCERT
　　=> /net
/DNS/net/ksk-Y/NDNCERT
　　=> /net

Figure 4.4: NDNS security policy definition

## 4.3 NDNS security delegations

The objective of the NDNS zone delegation process is to combine the DNSSEC delegation model with a NDNS data storage model. The NDNS security policy should capture both the namespace structure and the zone authority of NDNS recursive query responses.

In NDNS, security design retains the concepts of zone signing keys (ZSK) and key signing keys (KSK), which are extensively used in DNSSEC, with several changes in the storage and ownership semantics. In particular, to ensure key uniqueness, each individual ZSK and KSK corresponds to a unique NDNS namespace label (e.g., "`/net/ndnsim/zsk-1`" and "`/net/ndnsim/ksk-1`" for the ZSK and KSK of the "`/net/ndnsim`" zone) and can easily be looked up through an iterative or recursive query process. The independent key function (ZSK or KSK) is moved into the key name, so that it can be captured by the name-based security policy.

In addition to ensuring the key name uniqueness within the NDNS namespace, we explicitly separate ownership over a KSK and ZSK: while the "zone" signing key is kept under the zone's authority (i.e., the corresponding NDNS iterative query response for the ZSK in our example would be named "`/net/ndnsim/DNS-/zsk-1/NDNCERT`"), the "key" signing key is moved under the parent's zone authority ("`/net/DNS/ndnsim/ksk-1/NDNCERT`" in our example). Essentially, this change replaces the need for a specialized delegation signer (DS) record that is used in DNSSEC, by directly storing the key signing key (not just a KSK hash, as in the DS/DNSSEC case), without changing any operational semantics. The zone owner keeps the same level of flexibility with the zone signing keys as in the current DNSSEC system, easily replacing ZSKs[3] without need to update records

---

[3]The exact procedure of the NDNS key replacement is outside the scope of the current thesis, but it is expected that this procedure will follow the same general guidelines defined for DNSSEC.

in the parent zone. (New ZSK records can be created and signed by the already delegated KSK.) At the same time, replacing the KSKs would require delegation updates in the parent zone, which in the NDNS case involves simply installing a new (unique) KSK NDNCERT record into the parent zone, which in turn will be signed (i.e., properly delegated) by the parent zone's ZSK.

The security policy defined in Figure 4.4 is applied to each Data packet within NDNS protocol and provides strict authorization as to whether a particular key-certificate object can sign a given Data packet. This authorization process basically involves extracting of the NDNS namespaces from NDNS iterative query response names (i.e., removing the "DNS" and RR-type components from the Data packet name, and removing the unique key identifier component from the NDNCERT records, when such a record is used as a key-certificate) and applying strict matching for the resulting namespaces. In other words, the key-certificate is considered legitimately authorized only if its namespace is equal to or shorter (per-component comparison) than the Data namespace. For example, "`/net/ndnsim/DNS/www/zsk-1/NDNCERT`" is allowed to sign any records within the "`/net/ndnsim/www`" namespace, but is invalid to sign anything outside it.

In addition to the namespace authorization rules, the policy also needs to specify one or more trust anchors and their namespace validity, so the verification process can be securely terminated, either with a positive or negative result. A negative response is assigned whenever the trust chain does not follow the security policy authorized path, ends with an untrusted anchor, or exceeds a pre-configured trust chain limit (in our prototype the default limit for the trust chain depth is set to 10). Figure 4.5 summarizes the example of NDNS delegation for a "TXT" resource record in the "`/net/ndnsim`" zone (i.e., the certification chain for the iterative query response "`/net/ndnsim/DNS/TXT`"). Compared to the DNSSEC example in Figure 2.3 for the same security delegation, NDNS follows exactly the DNSSEC model, but uses slightly different, more data-centric, means to achieve
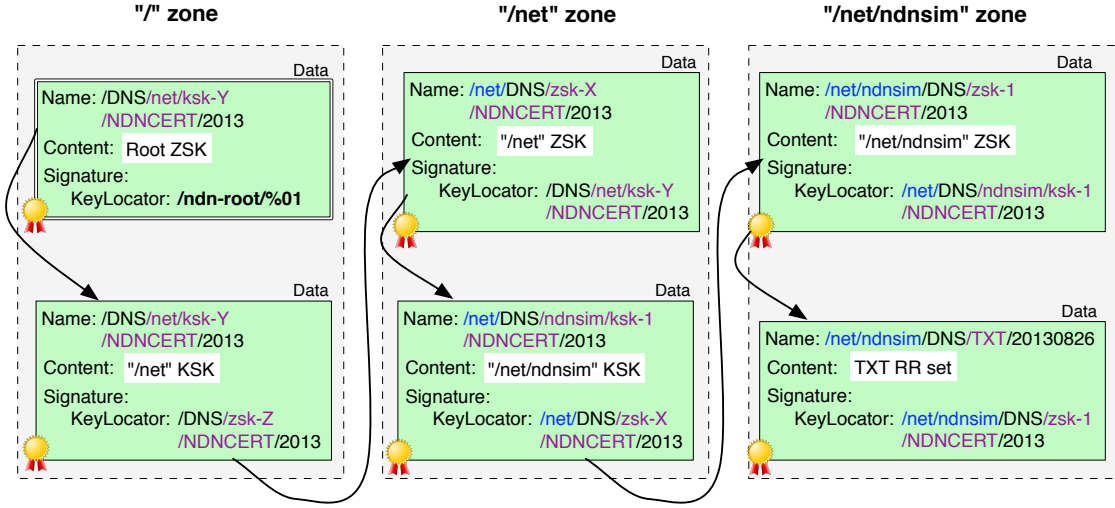
Figure 4.5: NDNS security delegation example

the same operational goal. Note that the root key-certificate "`/ndn-root/%01`" does not need to be part of the NDNS system (i.e., no need to create an NDNCERT record and store it in the root zone, since the name and the value of it is explicitly configured in the security policy).

## 4.4 Secure dynamic updates

The intended use of the NDNS system is as a general database service for NDN applications. Therefore, to ensure that such a database service not only provides a scalable read-only query interface, but also an interface for authorized parties to update database entries, it is crucial to have a secure interface for dynamic NDNS information updates. In general, this function is similar to the existing dynamic update functionality in DNS [VTR97, Eas97], and we have embedded such functionality into the NDNS protocol from the beginning. The dynamic update protocol for NDNS (DyNDNS, for short) is largely equivalent to its DNS counterpart, but again with several important necessary changes due to the network's

architecture switch.

### 4.4.1 DyNDNS updates

The design of DyNDNS tries to combine two usage modes of dynamic updates in DNS (refer to Section 2.2.2), applying NDN and NDNS design principles. First, the operation of creating pairs of RR sets and "RRSIG" blocks is directly equivalent to creating and signing Data packets that contain properly formatted NDNS iterative query replies. Second, an NDNS implementation inherently cannot have problems with AXFR zone transfers, since the AXFR-zone transfer concept has explicitly not defined in our NDNS specification, requiring instead out-of-band mechanisms for zone synchronizations between primary and secondary name servers (e.g., using ChronoSync as discussed in Section 3.1.2). Therefore, any authoritative server implementation should not have problems with installing updater-created records directly into the zone database, without first separating records into static and dynamic ones (all records can be considered dynamic). Finally, NDNS like a majority of other NDN applications ought to have the signing key available at all times (e.g., a short-term zone signing key), which in NDNS is necessary to generate negative responses. While it is not explicitly defined in the NDNS design (Chapter 3), it is assumed that negative responses are generated on demand using the short-term online ZSK, since it is impossible to pre-generate all possible combinations of negative responses for non-existing records: in NDN, the name of the response must match the name of the request.

Figure 4.6 illustrates dynamic DNS process, while Figure 4.7 formally defines operations that need to be performed by the update generator. As in the dynamic secure DNS update case, DyNDNS requires that the key used for the NDNS iterative response (dynamic update) generation has a corresponding "NDNCERT" entry in the updated NDNS zone. For clarity and to simplify zone management, DyNDNS requires that "NDNCERT" records for key-certificates used for dynamic
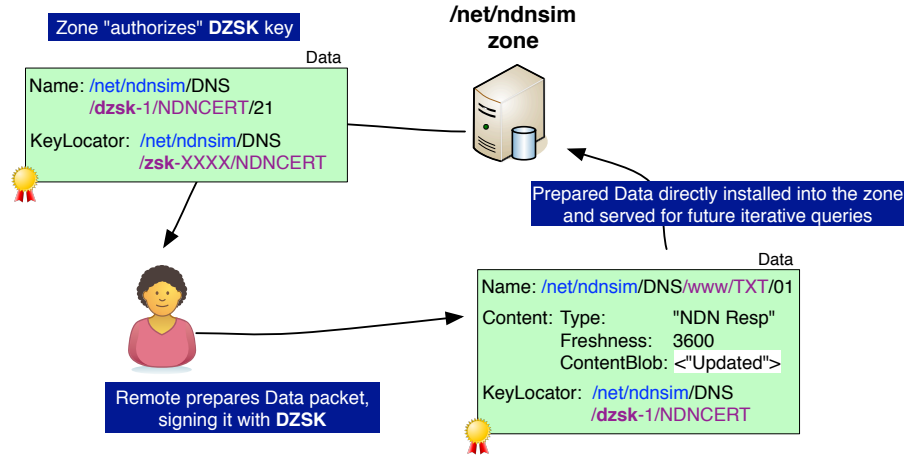
Figure 4.6: Dynamic update process in NDNS

updates contain a `dzsk-` prefix, distinguishing them from other types of key-certificates installed in the zone. Note that, unlike dynamic DNS updates, the NDNS zone owner has a much richer set of options to restrict dynamic updates to a specific namespace or sub-namespace. For example, if an issued dynamic zone signing key has the name "`/net/ndnsim/DNS/www/dzsk-2/NDNCERT`", the NDNS security policy, defined in Section 4.2, will effectively prevent this key from being used for any updates, except within the "`/net/ndnsim/www`" NDNS namespace.

### 4.4.2  Delivery of DyNDNS updates to the authority name server

As mentioned previously, standard dynamic DNS updates are carried inside special DNS update messages. These instead of the query are sent out to the authority server. The fact that the query in NDNS is an Interest packet seems to contradict the principles of dynamic updates: the update generator creates a fully-formatted Data packet/NDNS iterative query response, but it can only request data (express Interest) from the zone. However, there are several solutions to resolve this contradiction, each suited for a particular usage scenario, depending on the number and frequency of updates.
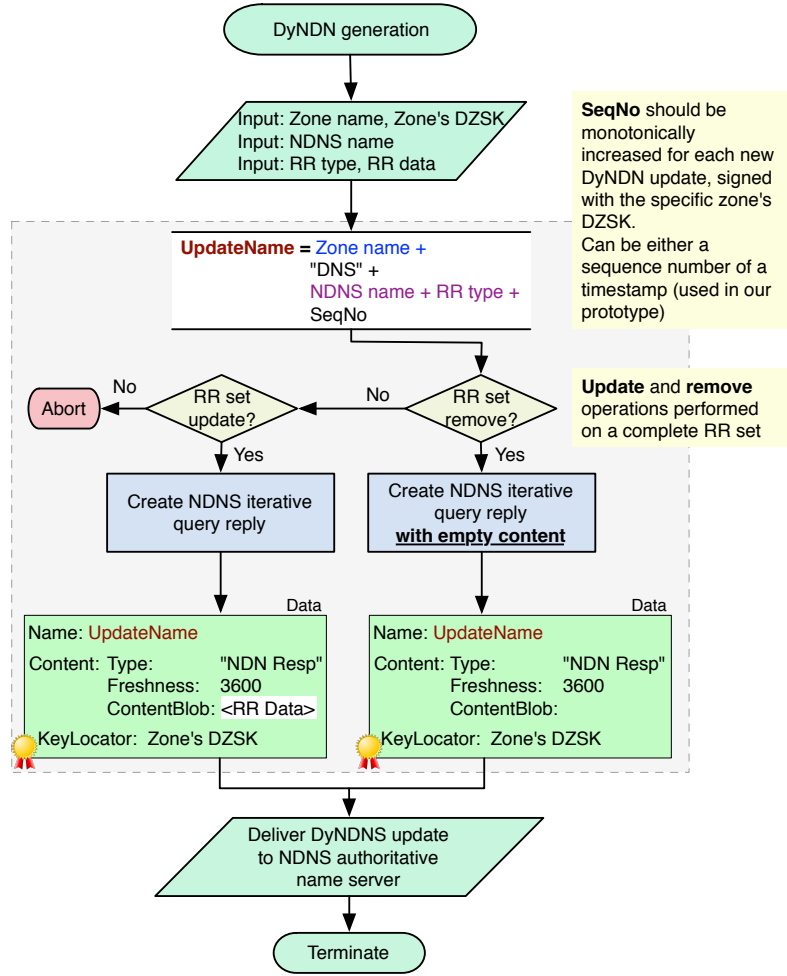
Figure 4.7: Dynamic update generation procedure

**Singular updates** First, if the number of queries is relatively small (e.g., an infrequent singular update), it is possible to define a simple protocol mimicking the style of DNS updates. In other words, we can define a special type of NDNS query (an Interest that can be sent out towards the authoritative NDNS name server), which instead of carrying the request contains a prepared Data packet of the RR set record to be installed or a signed command to remove a specific RR set from the zone. The formal definition of this Interest-based DyNDNS protocol is presented in Figure 4.8. To conform to the NDN delivery paradigm (i.e., each Interest needs to be satisfied, otherwise the forwarding strategy may infer that the

path is broken or there is an ongoing attack [AMM13, YAM13]), such dynamic update Interests need to be satisfied either with an "OK" Data packet or with a Data packet specifying a reason (optionally detailed) for the update rejection. Note that the authoritative name server may decide not to satisfy some dynamic update Interests, if there is an ongoing DDoS-style attack, which can be effectively prevented using the mechanisms embedded in the NDN architecture [AMM13].
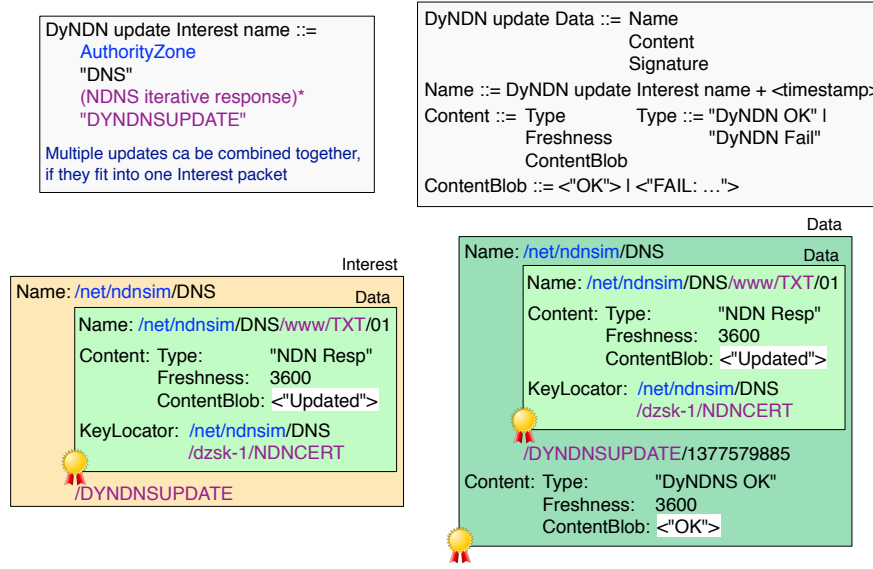


Figure 4.8: Definition of the Interest-based (singular) DyNDNS update

**Bulk updates**   In the case when the number of dynamic updates is relatively large or they are performed frequently, there is nothing that prevents the updater acting as a hidden secondary (or hidden master) name server, performing dynamic zone transfers using any of the available (in the future) zone synchronization mechanisms, such as ChronoSync [ZA13]. Although the particular implementation of such a zone transfer mechanism is not the immediate objective of the present work, such mechanisms need to be added either before or immediately following the initial deployment of the NDNS system.

### 4.4.3 Replay attack prevention

The mechanisms discussed above provide simple and efficient ways to modify an NDNS zone, requiring strict authentication and authorization for each update. In other words, the updater is required to prepare and sign the NDNS iterative response (Figure 3.8), which will be subjected to the NDNS security policy (Section 4.2) before being installed into the zone and served as the query response, effectively barring any unauthorized updates in the zone. However, one should always consider the possibility of a replay attack. The fact that somebody in the network can intentionally capture legitimate updates and then replay them at a later point of time, or that the network can duplicate packets (e.g., the NDN strategy may retransmit an Interest if the response does not come within an expected time interval), should not cause "old" updates to be applied to the zone or repeated application of the same update. Therefore, the authoritative NDNS name server implementation that supports dynamic updates should have additional mechanisms to remember previously applied updates and prevent them being repeated.

While there could many possible solutions to the problem, we are proposing a solution that utilizes the NDNS database itself for all necessary storage functions. Since every dynamic update requires a proper dynamic zone key-certificate delegation (i.e., there should be an "NDNCERT" record for the `dzsk-` key-certificate used to sign updates), we decided to use a key-specific state as an attack prevention mechanism. In particular, the information about all previous updates made using a specific DZSK can be represented as just one value: the largest sequence number observed in updates signed using this DZSK key-certificate (see Figure 4.7).[4] Furthermore, this per-DZSK sequence number can be stored in an additional RR of a new "NDNCERTSEQ" type in the same zone, with the same

---

[4]This way of the dynamic updates state representation is largely inspired by the ChronoSync design [ZA13], which represents the knowledge about the items produced by a specific producer with a producer's prefix and the largest sequence number of the produced item.
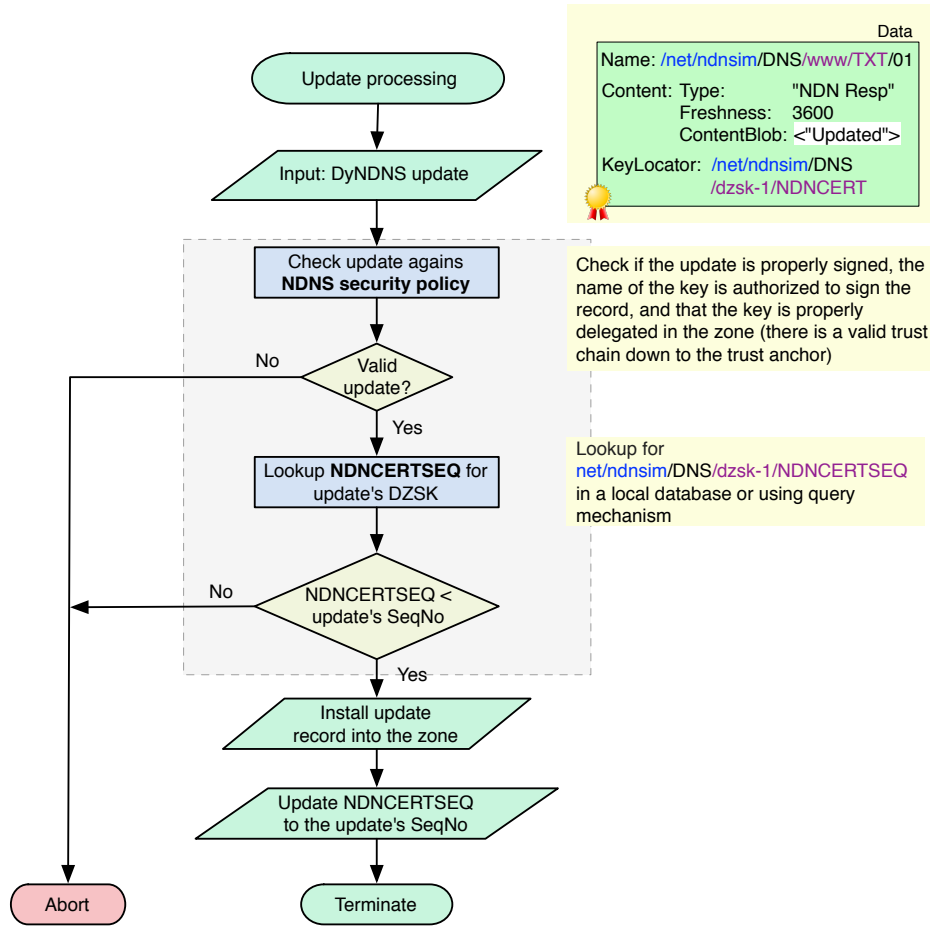
Figure 4.9: DyNDNS replay attack prevention mechanism

label as the "NDNCERT" record. (Since the "NDNCERT" record is defined to be a singleton RR, we define "NDNCERTSEQ" to be a singleton RR as well.) The procedure outlined in Figure 4.9 prevents reapplying "old" updates to the zone by executing additional checks for each received update, after the update is determined to conform to the NDNS security policy. Note that since the defined attack-prevention mechanism relies on the whole NDNS system and all its powerful mechanisms, it is largely irrelevant which authoritative NDNS name server the update reaches (and in which order). The decision outcome to accept or reject the update will be the same. As any other RR, "NDNCERTSEQ" is synchronized among the authoritative NDNS name server copies, leading to consistent security

decisions on each individual server.

It should be noted that it is critical to separate the state of the updates from the updates themselves. A naïve solution for example is simply to check the sequence number of the previous version of the updated RR set and reject any updates with "outdated" sequence numbers. Such a solution has a major security hole: as soon as a legitimate update carrying a "delete" message is captured by an attacker, he or she can easily install any other previously intercepted record, regardless of the sequence number, simply by replaying the delete-update sequence of the dynamic updates. Therefore, the state about previously seen updates must be preserved for as long as the other parameters of the "old" update remain valid. In our case, "NDNCERTSEQ" records must be preserved for the lifetime of the corresponding "NDNCERT" records. After the "NDNCERT" record is removed, there is no need to keep the "NDNCERTSEQ" record, since the "old" update will not be authorized anymore (i.e., there is no valid certification chain) by the NDNS security policy.

# CHAPTER 5

# NDNS use cases for NDN security

The previous chapter describes the design and security of the NDNS database, which preserves many properties of the existing DNS system that has made DNS so widely used. These include equivalent or even better scaling properties and allowing NDNS to be used in a generic way. Thus, except for certain limitations on the NDNS namespace (lowercase names compatible with DNS character sets) and the Data naming model ("ZoneName" + "DNS" + "Label" + "RR type"), NDNS has a large number of various usages. In this chapter, we identify and discuss two important use cases of NDNS that are related to the security of NDN architecture and aim to bring many theoretically defined elements of the NDN architecture closer to operational reality. First, we discuss how NDNS can be used to regulate the NDN namespace and ensure that the network and applications are working in tight collaboration with each other. Then we move to the discussion of how NDNS can be an effective solution to store and manage generic cryptographic credentials, crucial for all NDN-based communication. Separately in Chapter 6 we introduce another major operational challenge for the future deployment of the NDN architecture—scalability of name-based routing—and then show how this challenge can be addressed using the same NDNS database.

## 5.1 Namespace regulation in the global NDN routing system

With strong reliance on application-level names, the NDN architecture needs a mechanism or several mechanisms to regulate how these names are assigned and used. There should be a way to prevent two or more applications on the same network (local network or the global Internet) from using the same names and "collide" with each other. This problem can be effectively solved via a tight collaboration between the routing system and applications. Applications should suggest to the routing system directions where an Interest for specific prefixes should be directed (i.e., "register"/"announce" availability of Data within a specific NDN prefix). The routing protocol such as OSPFN [WHY12] or NLSR [HAA13] should propagate these suggestions through the NDN network, and NDN routers should use this information to forward Interests towards the application.

Theoretically, having different NDN applications use the same names, namespaces, and naming model is not a major problem in NDN, since the applications can (and should) use their own security policies to ensure that the retrieved Data packet is the expected one, re-expressing Interests for the same Data if there is a security mismatch (e.g., using a new Interest with enabled "Exclude" filter [NDN13b]). Since NDN routers are not necessarily obeying the routing information accumulated from application suggestions, the Interests eventually reach the right places (the Data producer or in-network caches) and return the expected data. In other words, NDN routers have an intelligent Interest forwarding layer [YAM13] that takes routing information as just one of the input recommendations, in addition to selectors in the Interest itself, the estimated per-prefix round-trip time for the outgoing interfaces ("faces" in NDN terms), the number of unsatisfied interests, and other data plane performance parameters.

However, it is highly desirable to ensure that the application and the network

are talking in the "same language": the network, the routing system, and the forwarding strategy should try their best to facilitate Data delivery that will likely satisfy the application, and limit Interest forwarding to undesired destinations. In other words, the prefixes that the routing system is announcing should be authorized by the application, and the application should be authorized to use these prefixes in the first place. In this regard, the NDNS system can provide crucial help. Similar to the ongoing ROVER effort [GMG12, GMO13], NDNS can be utilized to store authorization information for each application-announced prefix. In the context of ROVER, it is done by installing into DNS/DNSSEC the mappings from announced IP prefixes (using the reverse DNS zone "`in-addr.arpa`") to legitimate origin AS numbers. In the context of NDN, NDNS can store cryptographic authorization information that can be used in a similar way to the dynamic NDNS update protocol (Section 4.4).

One potential design, outlined in Figure 5.1, is to install into the destination NDNS zone an "NDNCERT" RR for a special routing signing key (RSK). This key then should be used to create a Data packet containing an application's request for prefix registration. This request/Data packet is propagated by the OSPFN, NLSR, or similar routing protocol throughout the network, while each individual router is able to validate the request through an NDNS query. OSPFN and NLSR have their own mechanisms to disseminate updates through the network and employ their own application-specific mechanisms to secure routing update data. The signed prefix registration request from the application, authorizing use of a prefix by a particular application, should be in addition to the security mechanisms specific to the routing protocol.

Note that the design, similar to DyNDNS, requires that the corresponding "NDNCERTSEQ" record, preventing "old" routing announcements from being validated by the routers, providing efficient protection from possible replay attacks. In the proposed design, the routers should invalidate any registration re-

76
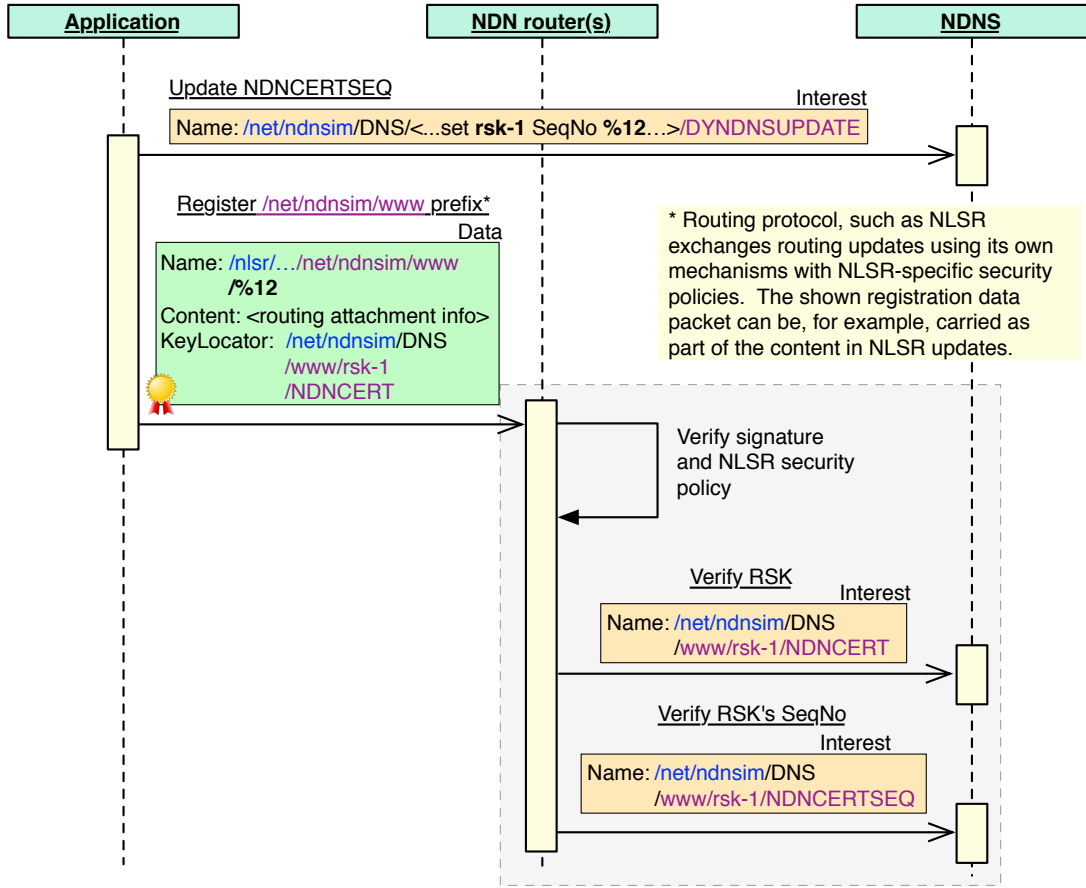
Figure 5.1: NDNS-based routing announcement authorization

quest, whose sequence number does not equal the "NDNCERTSEQ" record. This basically means that every time an application wants to generate a new announcement, it should first perform a (dynamic) update of "NDNCERTSEQ" and then generate a request with the updated sequence number.

The above described proposal is merely a rough design sketch, highlighting how NDNS can be utilized to secure announcement authorization in the routing system. Essentially, this means that NDNS can be used as a core mechanism to regulate which prefixes are getting injected into the routing system, so that the NDN network provides the best support only for applications that properly use delegated NDNS namespaces. As was noted earlier, other applications are still free to choose namespaces (i.e., it is always possible to determine application-

correctness of the received Data). Hopwever, these applications will experience additional operational penalties, such as potentially needing to employ multiple routing trip rounds to fetch the desired data, since the network will be basically working against the application.

## 5.2 Cryptographic key and certificate management for NDN applications

One of the biggest hurdles on the road to the full-fledged deployment of the NDN architecture, and specifically to writing and deploying fully secure NDN applications, is the need for a cryptographic credentials management infrastructure. In other words, since each Data packet is required to be properly signed, there should be a way to retrieve all necessary key-certificates that build up a trust chain for the proper application-specific Data packet verification. Generally speaking, each key-certificate object is an ordinary Data packet, having its own unique name, and a standard Interest/Data exchange can be used to fetch them. However, the biggest question is not how to fetch, but where such objects are physically stored, who put them there, how, and what kind of control this somebody has over the published key-certificate Data packets.

In the following sections we describe the initial attempt to provide a basic support for key-certificate storage [BZA13], relying on permanent data storage `repo` (see Section 2.1.1), another new communication primitive in the NDN architecture. We also highlight the operational problems that have occurred even in the present limited deployment scale of the NDN testbed and discuss how NDNS-based storage for cryptographic credentials can be a powerful alternative for the current approach.

### 5.2.1 Repo-based cryptographic credential management

The initial (and admittedly hacky) attempt to enable support for truly secure communications on the NDN testbed has relied on repo as a storage, ensuring that all used key-certificate Data packets are synchronized among and available from each participating site on the NDN testbed [BZA13]. All key-certificate Data packets has been assigned to the "`/ndn/keys`" namespace and each testbed router mandated to run a repo instance, participating in the "`/ndn/keys`" slice. In this way, we have been able to make sure that the key-certificates for each member of the NDN team are available at all times, including all other cryptographic credential information necessary to build and verify certification chains down to the root of trust on the NDN testbed. However, even such an initial attempt on a very small testbed scale has resulted in many unexpected hurdles, resulting mainly from the limited manageability and control over the selected storage mechanism—repo.

### 5.2.1.1 General limitation of repo

The specifics of the currently defined protocol to publish Data packets into repo as described in Section 2.1.1 introduce serious limitations how repo can be actually used. If repo happens to be installed on a local machine (or a first hub to which a user sends all the Interests, the default router), repo will be able to observe all commands from the user and perform all requested actions. However, even in this basic scenario the user will not be able to publish an arbitrary Data packet into the repo installed on the default router: the write protocol requires that repo send explicit Interest(s) for the "written" Data, and only Interests that have user-registered prefixes (on the current NDN testbed it is only the broadcast and local routable prefix) can reach back to the user and retrieve Data. In a more complex scenario, when repo is located several hops from the user, he or she has virtually

no control over which repo the write commands/Interests will reach: they will be routed towards the original Data producer, and if there is no repo present on this path (an Interest can get diverted to a place where repo may exist, but there are no guarantees this will happen), the command will have no effect at all.

Besides the above problem with the write control, the current implementation of repo lacks support of a delete operation, which has resulted in anecdotal repo usage patterns in NLSR [HAA13] and NDNVideo [KBZ13, KB12] applications: repo gets filled with Data for a predefined period of time, after which it is completely destroyed and re-instantiated from scratch. In this way, NLSR and NDNVideo applications are able to avoid overloading the permanent storage with useless data (old and irrelevant routing updates in the NLSR case and outdated video streams in the NDNVideo case), but require out-of-protocol interventions to the repo (i.e., physically stop the repo application, destroy the database, restart the repo application). This results in a loss of all Data in repo, including some that might still be useful.

In addition to the storage and the write protocol, repo also includes a new synchronization ("sync") mechanism [NDN13b], allowing efficient replication of stored Data packets in selected namespaces ("slices") among several repos. In other words, the repo-write protocol needs to be done only once (e.g., for the locally-run repo), after which Data packets are automatically and efficiently propagated to repos, participating in a specific slice. Although there is no doubt that the sync protocol provides a powerful mechanism enabling new pure data-centric application designs, the lack of the delete operation makes repo with sync an even less attractive implementation option than repo by itself. For example, if an undesired Data packet (intentionally or by accident) gets into repo and is synchronized everywhere else, it is close to impossible to remove this Data packet from the system. Even if one instance of repo is shut down and restarted from scratch as described above, the undesired Data packet will get back to the clean repo the

moment repo joins the sync slice. Basically, to delete one small piece, the whole repo infrastructure must be shut down and restarted from scratch at exactly the same time.

### 5.2.1.2 Limitation of repo-based cryptographic credentials management

Conceptually, the initial deployment of repo-based cryptographic credential management works quite well. The key-certificates are public, and everybody can fetch them from any place on the testbed and use them to verify Data packets, that are signed by these key-certificates. The problem arises when people actually start to use the system. The specific design for the naming model of the key-certificates has gone through several iterations, programs used for the key publishing have bugs, and people inevitably make mistakes using programs, forget passwords to private keys, or accidentally exposing private keys. All of these resulted in a large amount of erroneous Data packets being published to testbed repos, which can be deleted only if somebody coordinates an inter-site reconfiguration of all NDN testbed repos (and even afterwards, there is no guarantee that such action will not be required in the future).[1]

Also, if the private key is lost or accidentally exposed, there is no mechanism to "revoke" the corresponding published key-certificate. Such a revocation can be partially achieved through revocation lists, but there is no valid reason why the "authority" should serve the key-certificate as repo does, while it is known a priory that this certificate is invalid. Moving the revocation burden entirely to revocation lists is conceptually flawed, since the application can simply ignore the list, either because it cannot obtain/update it or there is a bug in the security implementation.

---

[1] As of August 2013, only 96 out of 1287 Data packets published in NDN testbed repos are useful Data. The rest are there only because it is virtually impossible to remove them.

### 5.2.2 NDNS-based cryptographic credential management

The above discussion highlights the need for a better storage model to manage cryptographic credentials for NDN application support. Such data must be available publicly and should ideally be replicated in several places to provide failure and error resiliency. At the same time, the owner of the cryptographic credential (either the owner of the private key or the entity that has issued the key-certificate) should have maximum control over the published data. NDNS is exactly the system that satisfies these requirements and can be used for cryptographic credential management purposes.

### 5.2.2.1 Properties as a general storage

From a storage perspective, NDNS is similar and largely equivalent to repo. It provides a universal storage for any kind of data, with just certain limitations on the naming model, but these limitations can be easily adopted by the applications by embedding a security policy or policies with the proper name reduction rules, similarly as defined in Section 4.2. In other words, the names for the stored Data packets need to follow an NDNS iterative response naming format, defined in Section 3.1.4.1, but the content of the Data can be anything that the application needs. Similar to repo with synchronized slices, NDNS provides functionality of the secondary authoritative NDNS name servers, in which Data packets can get synchronized using either the ChronoSync protocols [ZA13] or the same NDNx sync [NDN13b] protocol as is used in repo.

The more fundamental differences are with data management in zones ("slices" in terms of synchronized repos). While repo stores any Data packet under the configured namespace to which a user sends the write request (assuming that such write requests will actually reach repo and repo will be able to retrieve the requested Data), repo is not the authoritative source for the Data in the

namespace. While holding some pieces of Data permanently, repo assumes that this piece is just a copy of the Data and has no knowledge about any other Data in the same namespace, unless they are also present in the repo's database. On the contrary, NDNS is the logical authority for the NDNS zone data. Whenever there is a request to the zone, one of the authoritative NDNS name servers can give a definitive reply as to whether the requested Data exist or do not exist, returning the Data packet or one of the negative responses (refer to Section 3.1.4.1 for more detail). Moreover, being the authority allows NDNS to implement a better version of the write protocol—namely, secure dynamic NDNS updates. For example, with an Interest-based dynamic update delivery method (Section 4.4.2), the Interest containing the "command" will be directed strictly towards the closest authority NDNS nameserver for the zone, and not to an unknown place on the path towards the original data producer. Finally, NDNS allows deletions for any element from the zone.

Another big difference is the Data addressing granularity. If in repo a Data packet is identified by its full name, NDNS "uniquely" identifies elements (= RR sets) using a name prefix, containing just a zone name, domain label, and resource record type, while the actual Data packet representing an element/RR set can contain additional fields, such as the serial number or a version of the RR set. Note the word "uniquely" is in quotes for the following reason. For the lifetime of an RR set in the zone, there could be many versions (Data packets) of this RR set, either due to a change of the signature (e.g., when the zone changes its ZSK) or due to an RR set content update. However, at any particular point of time the authoritative NDNS name server will keep exactly one version of the RR set, since it knows exactly which version is the latest, and there is no valid reason for the authoritative server to return an old or invalid version of the RR set. Although similarly to the existing DNS the caching NDN resolvers and in-network NDN caches may result in multiple versions of a specific RR set available

in the network, this ambiguity is only transient and ceases after a record's TTL expiration within the caching resolvers and freshness expiration within NDN in-network caches.[2]

### 5.2.2.2 Key-certificate management on NDN testbed

All of these provide a strong ground and motivation for using NDNS as a basis for a key-certificate management solution. NDNS security described in Chapter 4 is just one example of such management, which actually can be generalized beyond securing just NDNS itself. For example, NDNS can be easily adopted, and we are planning to do it in the near future, to replace the initial repo-based solution [BZA13] with a user key-certificate management on the NDN testbed.

Each site of the NDN testbed will need to run the authoritative NDNS name server for the zone, representing an individual site's designated prefix, and be a secondary server for one or more other sites' zones (Figure 5.2). In addition to that, the root and "`/ndn`" zones should be hosted and replicated at some or all testbed sites.

Similar to the initial security management rollout, each NDN testbed user gets assigned a part of a site-specific NDNS namespace, which the user has an option either to further delegate to selected dedicated authoritative name servers or to keep hosted on the site's authoritative name server and its replicas. For example, the author of the present thesis can be assigned the "`/ndn/ucla/alex`" sub-namespace out of UCLA site "`/ndn/ucla`". In either case, the user retains full

---

[2]Although a scenario when a malicious node captures an old version of the RR set and then serves it to an incoming request is possible, it is unlikely and in most cases not detrimental. First, the routing system, as described in Section 5.1, will be acting as a first guard preventing random entities on the NDN network to "suck in" the Interest and respond in place of the authoritative name servers. Second, even if the malicious node is on the routing path, the forwarding strategy will take care of penalizing the wrongdoer, using one of the available methods of cache poisoning attack mitigation mechanisms. The worst case, when the malicious node is on the only path towards the authoritative server, is detrimental, but it is irrelevant to the system used, since the node on the path has full control over what data to return or not to return for each received Interest.
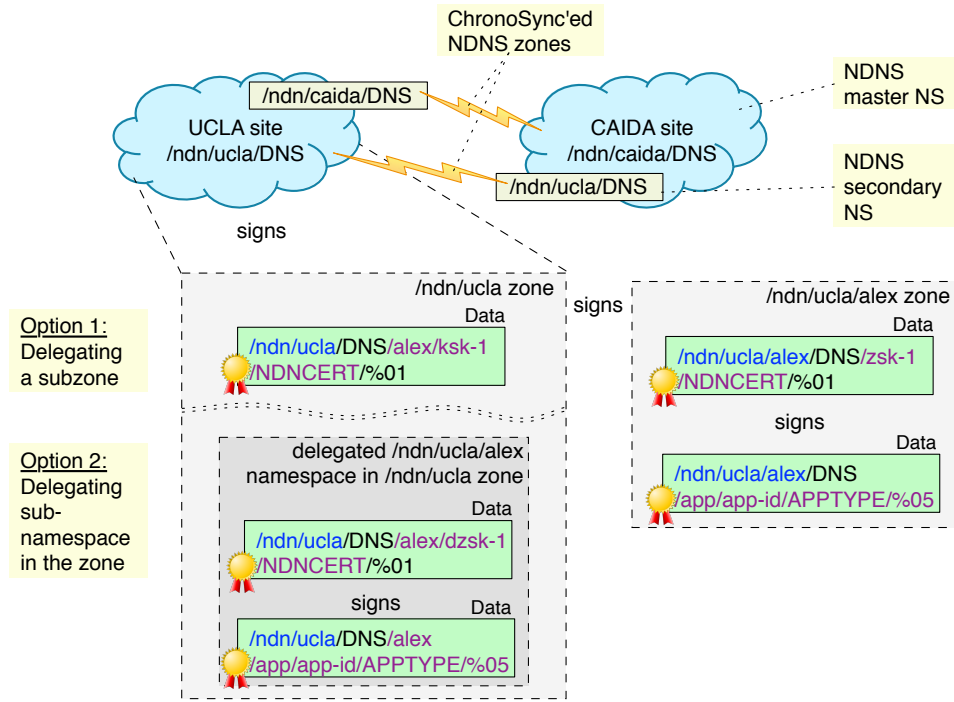
Figure 5.2: Proposed deployment of NDNS on NDN testbed

control over the namespace via proper NDNS and security delegation of the key signing key (if the user manually wants to manage NDNS name servers) or just security delegation of the dynamic zone signing key (if the user wants to keep hosting records in the site's zone). The same example on Figure 5.2 shows these two options of retaining full user-level control for using the namespace assigned to the author. Note that by the full control we mean that the user is able to "publish" and make publicly available any records, including "TXT", "MX", DANE-like records for application-specific roots of trust [HS12], or any other application-defined records that include the user's NDNS namespace prefix (e.g., the namespace "/ndn/ucla/alex" expressed either as "/ndn/ucla/DNS/alex/..." or "/ndn/ucla/alex/DNS/...").

### 5.2.2.3 Cryptographic credential revocation

As mentioned before, the ability to perform full-fledged revocation of the crypto-graphic credential is an essential part of the security management system. While key-certificates can be partially "revoked" with the help of revocation lists or specialized online certification validation methods (e.g., OCSP, an online certificate status protocol [MAM99] or pure NDN-based protocols [MV13]), these methods should be complementary to the authority's decision to make credentials publicly available, i.e., respond positively or negatively to incoming inquiries for the credentials.

NDNS allows retaining control over the issued certificates by both the key-certificate issuer and the certificate owner. Let us taking for example embedded security mechanisms in NDNS itself. The zone owner can always remove or replace the ZSK record in the zone, automatically invalidating (after expiration of all cached entries) any other records or delegations that were based on the trust chain using the key. This can be done either using the local zone management tools or using dynamic updates, if they are enabled for the zone. (While it is technically possible to revoke DZSK using a dynamic update that is signed by DZSK, a particular implementation of the authoritative name server could prevent such dangerous operation.) At the same time, the parent zone has full control over the zone's KSK, which, similar to a "DS" record, serves as a security delegation record. Therefore, if the parent zone (essentially an issuer or authorizer for the zone) discovers that a certain previously certified KSK (or the derivative ZSK or DZSK) is lost or compromised, it may decide to the remove zone's KSK record, effectively breaking any certification chain for the zone. Of course, this operation is potentially devastating and KSK issuers, such as the NDN testbed site and NDN root zone operators, must be very cautious and fully aware of possible consequences. Nevertheless, the ability to revoke shows that every security entity does not lose touch with previously issued security endorsements (as in the

case of repo-based credential management) and, if and when necessary, can act accordingly.

Note that while this presented example talks specifically about NDNS security, the same logic (with possible change of exact details) applies to any other application-based security that relies on NDNS. For example, if the application installs a DANE-like record [HS12] that specifies application's root of trust (after the user delegates such ability to the application, the same way as NDN sites delegate a user's namespace in the testbed case), the application itself can "revoke" the record by removing it, and the user himself can either remove/revoke the delegation or directly remove the record from the zone.

# CHAPTER 6

# Scaling NDN routing using NDNS

As Internet applications become increasingly data-centric, a number of new network architecture designs [CG00, KCC07, TAV09], including Named Data Networking (NDN) architecture [JST09, EBJ10]—the discussion in this thesis, have proposed a data-centric communication paradigm, which in part requires the routing system to work directly on Data names. In an NDN context, Interest packets carry Data names instead of host addresses, and routers forward these Interest packets based directly on their names. By naming Data explicitly and binding the name and Data by a cryptographic signature, NDN provides a number of benefits including Data security, in-network caching, natural multicast support, and in general better alignment between the desired application usages and the underlying Data delivery model. However, one frequently raised concern for NDN is scalability of the name-based routing scalability [BVR12, NO11]. Put simply, provided that the number of Data names is unbounded, how does name-based NDN routing scale?

A brief analysis of Internet history [MZF07] reveals that the scalability issue with the name-based routing in NDN is not that new: a similar question has been asked previously about IP. Given the finite space of IP addresses ($2^{32}$ for IPv4 and $2^{128}$ for IPv6), how can IP routing scale if even today's routers cannot hold such volumes of information? The current answer to this question employed on the Internet is IP address aggregation. In particular, at the edge of the Internet, hosts and small networks get addresses from their access providers; these addresses,

being from the same provider, can be easily aggregated into prefixes (i.e., these are *provider-aggregatable*), and only aggregated prefixes need to go into routing tables instead of individual IP addresses. At the same time, over the years there has been an increasing demand for *provider-independent* addresses [RIP09], so that large customers can avoid network renumbering after switching to different providers and support connecting to multiple providers at the same time (multihoming). As a result, many IP prefixes used today cannot be effectively aggregated along with the provider's prefixes and must be announced separately, leading to an increased routing table size [Hus05, MZF07, ATL10]. In addition, other factors contribute to prefix de-aggregation, including various types of network-layer traffic engineering, load balancing [ATL10], and mechanisms to mitigate ongoing distributed denial of service attacks and recover from prefix hijacks [Arb11].

## 6.1 Map-n-encap for IP

An alternative solution, proposed long ago but never actually deployed due to difficulties in retrofitting the new solution into the operational Internet, is map-n-encap [Dee96]. The idea behind this solution, as exemplified in Figure 6.1, is that the global inter-AS routing system (the so-called Default-Free Zone, or DFZ) exchanges and maintains only provider-aggregatable addresses, while the routing protocols within each individual provider (intra-AS routing) maintain information about all "attached" client (provider-independent) prefixes. In addition to that, all provider-independent addresses are mapped using a dedicated mapping service, for example using a reverse zone in DNS, to provider-aggregatable addresses. As a result, any inter-AS communication requires contacting this mapping service to get the "routable" address, and then use some form of tunneling, such as IP-in-IP encapsulation [Per96], to deliver IP packets destined for a provider-independent address using a routable, provider-aggregatable address (Figure 6.2). Note that
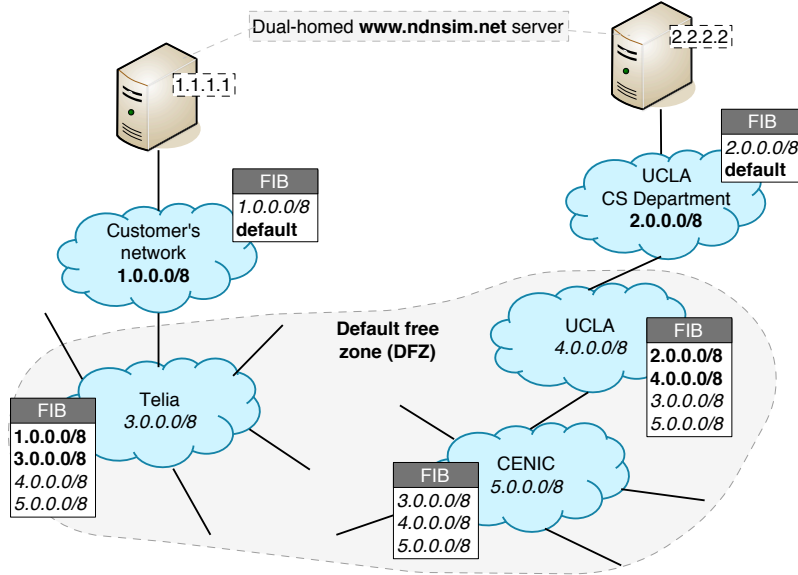
Figure 6.1: Example of FIB state in the Internet with map-n-encap
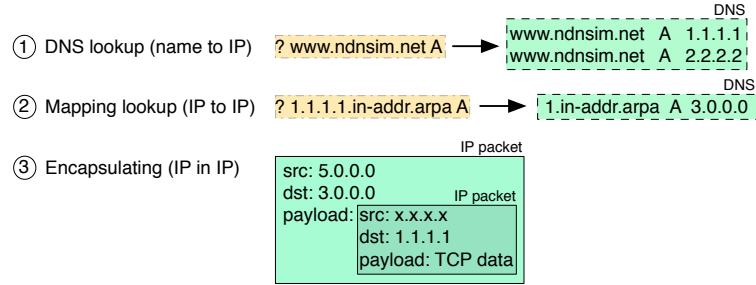


Figure 6.2: Example map-n-encap communication sequence in IP

if the mapping system returns multiple addresses (i.e., the destination is multihomed), as in our example, the sender is free to choose any of the returned options. The key advantage of the map-n-encap solution is that the core can maintain only a limited number of IP prefixes and have easily managed routing tables (e.g., limited number of routing updates, updates only about connectivity, not all attachments as in the currently used BGP [RL06]), while ensuring the efficient global-scale communication. This idea has led to a number of specific designs including 8+8 [OD96], LISP [Far07], ILNP [ABH09], and APT [JMM08], to name a few.

90

## 6.2   Map-n-encap for NDN

NDN is inherently a data-centric architecture, thus every application is supposed to name its Data in a provider-independent way, relying on the network to deliver Interests to the right places based on Data names. This naming should still provide enough information, including a routable prefix, application de-multiplexor, and application-specific information (see naming example for NDNS in Section 3.1.1), to enable proper delivery of the Interest to the intended destinations and applications, and identify the requested application Data, but all these naming model elements are independent of the specific location or the network attachment. The applications just need to indicate to the NDN network that they are using a selected application-specific "routable" prefix, for example, in a way described in Section 5.1. Therefore, given the number of different applications and application instances that can be run on the NDN network, it is infeasible to assume that the conventional routing system would be able to maintain a virtually unbounded volume of application-specific provider-independent prefixes.

In this work, we propose to apply the old map-n-encap idea in order to scale NDN routing, which should not encounter any of IP's retrofitting hurdles, since the NDN architecture is just in its first steps toward global adoption. Similar to the IP's world with map-n-encap described above, the global inter-AS NDN name-based routing system (DFZ) will need to maintain a limited number of NDN provider-specific (AS-specific) name prefixes (see Figure 6.3). The number of these prefixes, assuming that each AS on the current Internet will be injecting a few NDN prefixes, will be, pessimistically, on the order of one million and, realistically, no more than 100,000 (i.e., $\approx$ two prefixes on average for each AS on the current Internet [ATL10, Hus05]).[1]

---

[1] According to our analysis of BGP data in 2006 [MWZ07], the number of IP prefixes belonged to ISPs was 22,000 out of a total 209,000 prefixes in the global routing table. Assuming one name prefix for each IP prefix for ISPs, the name-based routing table would have had only tens of thousands of entries; the vast majority of NDN names used by end-sites will not show up in
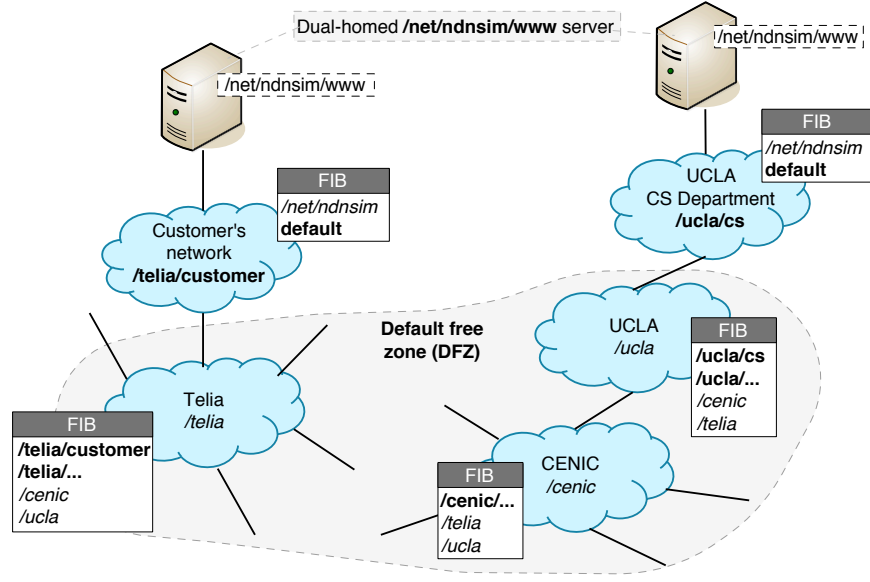
Figure 6.3: Example of FIB state in NDN network with map-n-encap
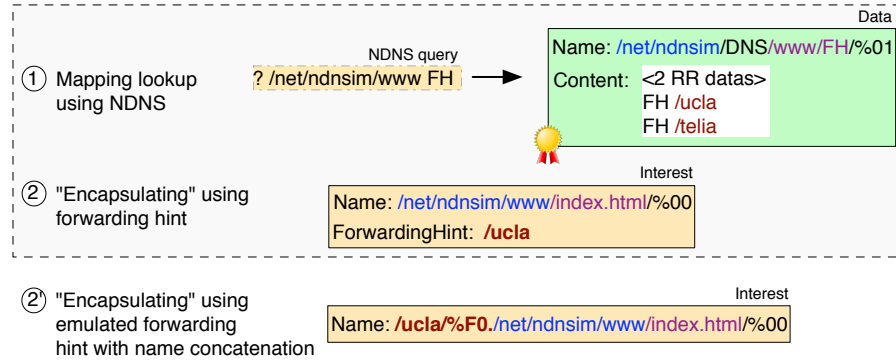


Figure 6.4: Example map-n-encap communication sequence in NDN

In this way, the global NDN routing system will maintain just connectivity between providers/ASes, including all routing policy-specific information about alternative inter-AS paths, while a separate mapping system, such as NDNS, will store the information about current attachments of the application-specific prefixes to the provider-specific ones. At the same time, each provider internally should have enough resources to ensure that each "registered" application-specific

the global routing table. Another analysis of BGP data shows also that the growth rate of ISP ASes is only 20% of total AS growth rate [ATL10, OZZ07].

prefix is fully routable within the provider's network, using OSPFN [WHY12], NSLR [HAA13], or similar intra-AS NDN routing protocols.

Similar to the inter-AS communication in map-n-encap in IP if an application wants to request Data that is not available inside the AS,[2] it needs to do a lookup into the mapping system to find the corresponding attachment point or points, and then express an "encapsulated" Interest for Data directed to one of the attachment points (see example in Figure 6.4). Such "encapsulation" can be achieved by providing additional field in the Interests, which we call a *forwarding hint*, that can suggest to NDN routers the direction where the requested Data may be found (see Section 6.3). Unlike in real IP-in-IP-style encapsulation, NDN routers are free to ignore the hint if either they know exactly where to look for Data (e.g., they have a routing entry for the requested Data) or the Data is available in local in-network storage or a packet buffer cache.

However, since the forwarding hint concept requires certain modifications of Interest processing logic on NDN routers, it is also possible to emulate it without resorting to any processing logic modifications. For instance, one could prepend the forwarding hint to the original Data name and express the Interest for a concatenated name, which will first be directed to the right provider and then the right application inside the provider's network. This *concatenation* approach, discussed in Section 6.3.1, is simpler to implement and deploy (and in fact has already been implemented and used in NDNS prototype and ChronoShare applications [AZZ13]). However, it has certain limitations and to some extent violates NDN principles: the same Data could be "named" differently (encapsulated in differently named responses) and caches may not be too effective if the Data producer is multi-homed to many different providers. However, in this work we want to provide all implementation alternatives that can effectively solve the scalability

---

[2]This decision can be made either within the application after a certain number of retries, or within the local or first-hop NDN router.

problem. Additionally, in practice, consumer networks are connected to a limited number of providers (unless this consumer is a giant such as Google or Twitter that are most likely going to be a part of the global routing system and will not require forwarding hints at all), and cache efficiency will largely be unaffected even with concatenation implementation of the forwarding hint concept.

The resulting overall picture is that (a) the prefix aggregation happens at the edge of the Internet, where application Data names are getting collapsed into the provider's prefixes, and (b) forwarding-hint ("encapsulation") provides a way to direct Interests for application Data toward the right provider. This architecture essentially moves the scalability issue from the routing to the mapping system: if the mapping from application names to forwarding hints can scale, then the routing task is trivial. As the mapping happens at the edge of the network, it will have no impact on the network core. The NDNS database system introduced in this thesis, having scalability properties as good as or better than the existing DNS system, is one of the best candidates for such a mapping system.

The following sections in this chapter discuss specific design decisions for the forwarding hint concept implementation, as well as details how NDNS can be coupled with forwarding hints, in other words, which changes in the NDNS query process are needed and which resource records should be used.

## 6.3 Encapsulating using forwarding hint

In addition to the application name, NDN allows Interests to include so called *Interest selectors*, which can be used by applications to request NDN routers and Data producers to return a specific version of the Data packet, provided that there are multiple versions of the Data packet available and the Interest name does not uniquely identify one (e.g., when an Interest specifies only a prefix of the Data name). The selectors currently defined and implemented in the NDN platform

codebase include "`MinSuffixComponents`" and "`MaxSuffixComponents`" to specify how many additional name components the acceptable Data packet should have, "`Exclude`" to specify what additional name components should not be present in the retrieved Data name, and several others [NDN13b]. In order to achieve application correctness, all these selectors need to be processed by each router supporting in-network transient (packet buffer cache) and permanent (repo) storage, as well by each Data producer that is receiving the specified Interest. Therefore, it is possible to add a new *forwarding hint* selector (see example in Figure 6.5), which would serve a similar but slightly different purpose. Instead of selecting which exact Data can satisfy the Interest, it would "select" (suggest or hint) the path, through which the requested Data packet (conforming to all other selectors) can be found.
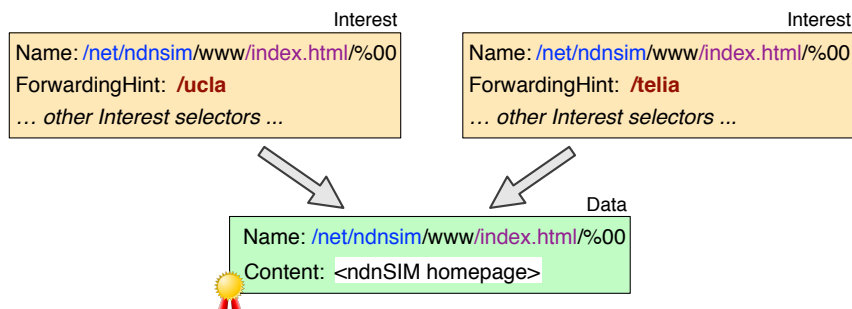


Figure 6.5: Forwarding hint as an additional Interest selector

In our example map-n-encap NDN environment in Figure 6.3, to retrieve the application Data "`/net/ndnsim/www/index.html`" from, for example, CENIC network, the Interest needs to include either a "`/telia`" or "`/ucla`" forwarding hint selector. When NDN routers in the CENIC network receive such an Interest, they may have no idea how to process it using just the Data name, since the specified name in the Interest or any prefix of this name is not part of DFZ's routing tables. At the same time, the included forwarding hint selector will contain a prefix that is guaranteed to be present in the router's FIB, thus simplifying the Interest forwarding decision for the router. In one sense, the forwarding se-

lector represents a "locator" of the Data, with the exception that NDN routers do not have to strictly follow this "locator" if they know how to forward the Interest toward the actual Data name (e.g., within Telia and UCLA networks) or if they have a pre-cached version of the Data to satisfy the Interest. Note that this forwarding hint selector can be provided either by the end-host (the requesting application, such as NDN-based WEB browser) or any other router on the path (e.g., home router), provided that it has enough memory and processing power to do the lookup.

The only "drawback" from the new forwarding hint Interest selector is that the processing logic on NDN routers, even for ones that do not feature in-network storage and caches, needs to be slightly amended. Instead of doing just one FIB lookup after the optional step of checking the cache and applying Interest selectors, the router needs to do an additional FIB lookup using the forwarding hint selector, if it is present in the Interest packet. Figure 6.6 summarizes the amended Interest processing logic for NDN routers, highlighting the necessary additions:

1. Look up the application name in: cache, pending interest table (PIT), and routing table (FIB).

2. If any lookup of step (1) returns a positive result, proceed with the standard Interest processing. In other words, the Interest will either be satisfied with previously cached Data, be recorded in the existing PIT entry, or be forwarded according to the forwarding strategy.

3. *If none of the lookups of step (1) returns a positive result, look up the hint in the routing table.*

4. *If the lookup of step (3) returns a positive match, forward the Interest accordingly.*

Note that the specified addition will have no effect on Interests for Data that

```
 1: function PROCESS(Interest)
 2:     if Data ← Cache.Find(Interest.Name, Interest.Selectors.*) then
 3:         Return(Data)
 4:     else if PitEntry ← PIT.Find(Interest.Name) then
 5:         Record(PitEntry, Interest)
 6:     else if FibEntry ← FIB.Find(Interest.Name) then
 7:         Forward(FibEntry, Interest)
 8:     else if FibEntry ← FIB.Find(Interest.ForwardingHint) then
 9:         Forward(FibEntry, Interest)
10:     else
11:         Drop Interest (Interest cannot be satisfied)
12:     end if
13: end function
```

Figure 6.6: Interest processing

NDN routers know how to retrieve: either there is a FIB entry for the specified Data name (as in the case inside Telia and UCLA networks) or there is a cached version of the Data. Depending on cache implementation, this could be referring either to just the packet-buffer cache that is inherent for each NDN routers or to a bigger, possible collaborative [WBW13], dedicated in-network cache inside the network.

Depending on the specific implementation, and subject to the further research that is outside the scope of the current thesis, Interests for the same Data name, but specifying different forwarding hint selectors may be either collapsed into one PIT entry or treated completely separately, when such Interests arrive to the NDN router at about the same time. In the first case, the router will simply ignore the suggesting specified in the Interest that arrived later, while in the latter case both suggestions could be followed, potentially opening more possibilities to retrieve

Data from many alternative sources, as well as to abuse or to create denial of service attacks.

### 6.3.1 Forwarding hint emulation using name concatenation

If or when modification of Interest processing logic on NDN routers is not acceptable, it is still possible to use the concept of the forwarding hint. However, instead of adding it as a separate element to the Interest, the "hinted" provider-specific prefix can be prepended to (concatenated with) the original Data name of interest (Figure 6.7). To avoid potential ambiguities for the Data naming (e.g., in "/ucla/some/app", does "/ucla" refer to the forwarding hint directing interest towards "/some/app" inside the UCLA network, or is it an actual application in UCLA directly using "routable" names?), we separate concatenated parts of the name with an arbitrary-selected name component "%F0."[3]
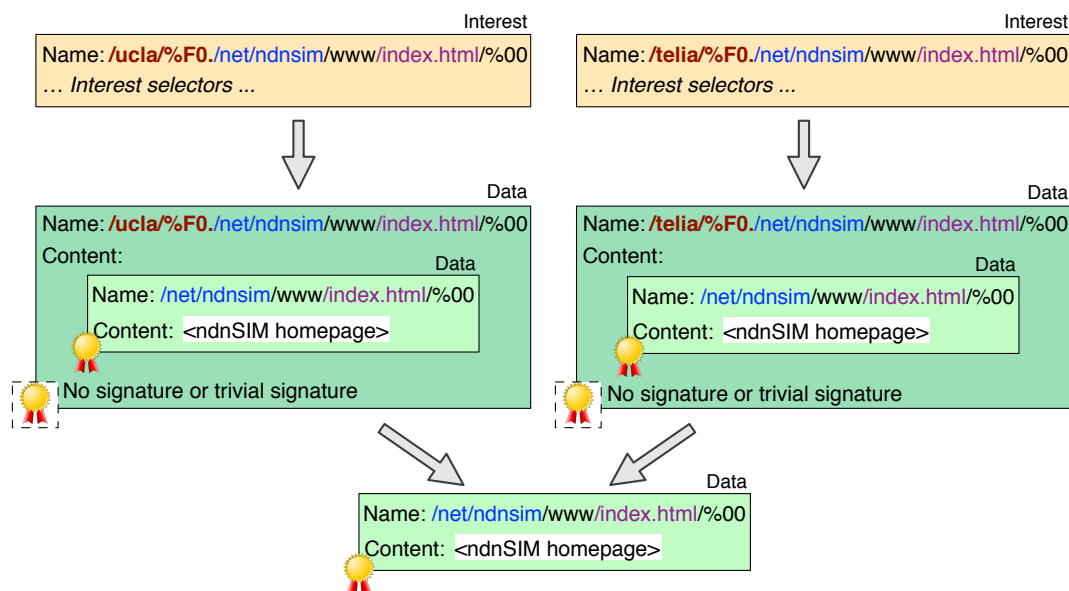


Figure 6.7: Interests in concatenation approach

---

[3]Because the concatenated forwarding hint is irrelevant for NDN routers, a particular choice for this delimiter is arbitrary. Our choice is motivated by the fact that "%F0." is likely to be unique, is short, and to some extent represents the first two letters in the "FOrwarding hint" term.

With this forwarding hint emulation approach, NDN routers do not need to alter any existing logic and will perform exactly one FIB lookup if the Data is not found in caches and there is not an active PIT entry for the requested Data. At the same time, this approach requires additional processing to be performed by the original Data producer or its delegate (e.g., local NDN daemon or NDN daemon on the home router, if configured to do so): when an encapsulated Interest is received, one needs to strip off the hint part, perform the application-specific Data lookup, and then encapsulate the retrieved Data, so it can be returned to the requester. In other words, since Data packets always follow the "breadcrumb" path set up by the forwarded Interests, the returned Data packets have to contain, at least as prefixes, full names of the incoming Interests, including the forwarding hint, if present. Because each individual Data packet is a secure bundle of the name and content, to preserve validity, authenticity, and provenance properties, the original Data packet needs to be included in another forwarding-hint-specific Data packet (see Figure 6.7) that optionally has its own signature and provides additional network-level validity, authenticity, and provenance for the encapsulated bundle. For example, the border router at UCLA's network may need encapsulate the retrieved Data of "`/net/ndnsim/www/...`" under the name "`/ucla/%F0./net/ndnsim/www/...`", optionally sign it with UCLA's key, and forward encapsulated Data packets back to the requesters.

Note that the outer Data packet does not really need to be additionally secured, since all the application-level security can be deduced and inferred from the retrieved encapsulated Data. However, the Data producers or their designated agents may still employ a lightweight signature generation to ensure network-level integrity and validity checks, e.g., in the case routers on the return path elect to check if the packet has been generated by a legitimate source.

Although the forwarding hint emulation using name concatenation achieves the goal of forwarding packets in the map-n-encap'ed network environment, it

fundamentally changes the names used for fetching data, and this change leads to a number of undesirable problems. First, there could be extra signing overhead because the original content is signed over twice, i.e., first by the producer's key (inner packet) and then by the ISP's key (outer packet). Such extra signing, if selected, can lead to serious scalability issues, since routers would need to sign packets at line-speed.

Second, and probably more critical, is that the name concatenation breaks Data uniqueness: the same piece of Data can become available via a different topology-specific name, reducing the level NDN-provided scalability and data availability benefits that rely on efficient in-network caching and Interest aggregation. For example, Interests directed to one ISP (e.g., "`/ucla/%F0./net/ndnsim/www`") cannot be satisfied by the router's cache, if the previously cached Data is originally requested using another ISP prefix (e.g., "`/telia/%F0./net/ndnsim/www`"). However, in practice most consumers have very limited scale of multi-homing (usually two, one primary and the other providing a back-up channel), effectively alleviating the caching efficiency and data availability problems.

## 6.4   Mapping using NDNS

As pointed out earlier, map-n-encap "outsources" the scalability problem from the routing system to a mapping system. In the IP Internet, DNS has become the de facto best, most widely used, highly scalable, reliable, secure (with DNSSEC extension), and easy-to-use general-use mapping system. NDNS is specifically designed to retain most of these DNS properties and aims to serve a similar universal database purpose in the NDN world. Therefore, NDNS is the ideal candidate for the mapping service for the map-n-encap approach to scale the name-based routing system in NDN, allowing NDN to become a practically deployable and usable system on the scale of the current Internet or larger.

The concept of the routing hint, which becomes an essential part of the NDN environment with an enabled map-n-encap scalability solution, fundamentally changes some communication patterns in NDN, including ones used and relied upon in NDNS iterative query implementation. The assumption which we used in Chapter 3 that all prefixes used by NDNS zones are directly routable does not hold anymore. The only element in the map-n-encap NDN world that can be assumed about the routing tables is that prefixes for the DNS root ("`/DNS`") and some dedicated top-level domain names (e.g., "`/ndn`" for NDN testbed, "`/com`", "`/net`", and some others for the Internet-scale NDN deployment) will be present in DFZ FIB. Any other communication will require (in general, since some Data can be fetched directly from NDN in-network caches) the presence of the forwarding hint, either using a new Interest selector or the concatenation approaches, described in Section 6.3. Therefore, there is a need to do certain amendments to NDNS, ensuring the tight integration and coupling of the NDNS iterative queries and forwarding hints.

The iterative NDNS queries as defined in Chapter 3 aim to discover the proper authority of the input NDNS namespace. However, a map-n-encap NDN environment requires, in addition to this discovery (i.e., discovery of the fact that there is an "NS" record for a label in NDNS zone), the name of the NDNS name server returned by the "NS" record that needs to be further mapped to a forwarding hint, which may need to be used to actually fetch the desired Data. This process to some extent should be equivalent to mapping the name server names in the DNS protocol to IPv4 and IPv6 addresses using "A" and "AAAA" records. Thus, we define a new resource record type using an "FH" mnemonic (Figure 6.8), which holds the NDN name of the desired forwarding hint, with an additional "priority" parameter. This additional parameter can be used by Data producers to express the local (routing) policies and define the recommended order of the forwarding hints to try: the lower priority record should be tried first.

```
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                             priority
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                  Forwarding Hint (NDN Name)
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
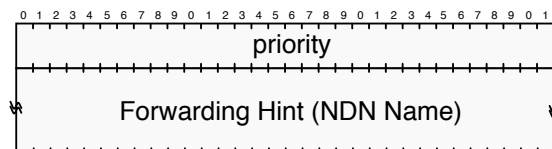
Figure 6.8: New FH resource record type to hold forwarding hint information

The current DNS/DNSSEC specification defines that the response for an "NS"-type query, including referral "NS" responses, should include "glue" records as part of the additional section in a DNS message [Moc87b]. These "glue" records include known "A" and "AAAA" records for the returned name server names, whereas in-zone records are configured during the delegation, and out-zone records can be maintained in name server cache. For example, when delegating "`ndnsim.net`" using the "`ns1.ndnsim.net`" name server name, the IP address for "`ns1.ndnsim.net`" should be explicitly configured in the ".`net`" zone. With the help of glue records, DNS avoids potential loops and reduces the number of round-trips to get to the final answer. At the same time, the glue records are not secured using DNSSEC (i.e., the additional section of the DNS reply does not contain corresponding "RRSIG" records), and a separate DNS query will be necessary to ensure validity of the delegation.

Although a similar process of bundling "glue" records with "NS" responses is possible in NDNS, we consider it as just an optimization technique that can be implemented in the future.[4] In the initial design we follow a pure data-centric model, requiring each individual piece of iterative NDNS information to be explicitly requested using a uniquely-named Interest. The querier first discovers the zone delegation information ("NS" RR set), after which it sends one or more queries for "FH" records to the same zone using if needed the same forward-

---

[4]Such an implementation can track changes in in-zone and out-zone FH records, recreating and re-signing an NS bundle, containing an "NS" RR set and relevant "FH" information. Although such an implementation could be relatively simple, there could be unexpected performance drawbacks when one of the domain records is "unstable": if even one "FH" record changes, the whole "NS" RR needs to be recreated, limiting in-network cache efficiency and creating an additional burden on the zone synchronization protocol.

ing hint as before or invoking a separate iterative query if the "FH" record is out of the zone. In a similar example, when "/net/ndnsim" is delegated using "ns1.ndnsim.net" (a.k.a. "/net/ndnsim/ns1"), the querier first sends a "/net/DNS/ndnsim/NS" query, and after discovering the name server's name it sends another "/net/DNS/ndnsim/ns1/FH" query to the "/net" zone. We argue that while this way may require additional round-trips to discover the final answer, it does not create a new circular dependency problem (impossible with in-zone delegation, possible the same way as in the existing DNS with out-of-zone delegations), provides a clean and pure data-centric data retrieval process that can extensively benefit from NDN's in-network storage and caches (potentially mitigating additional round trips), and guarantees integrity and provenance of Data at each step of iterative process. The resulting iterative NDNS querying process coupled with the forwarding hint concept is defined in Figure 6.9. Note that the definition assumes use of new forwarding hint selectors in Interest packets, but it can be applied (and is actually implemented this way in our NDNS system prototype) to the forwarding hint emulation using the name concatenation approach.

### 6.4.1 Security considerations

It should also be noted that the map-n-encap version of NDNS requires additional considerations related to the security. The NDNS security protocol defined in Chapter 4 uses a "KeyLocator" to identify necessary key-certificate object to verify response's signature. However, this "KeyLocator" is specified as a fully-formed NDNS iterative query (e.g., "/net/ndnsim/DNS/zsk-1/NDNCERT"), which may not be directly possible to express, since the specified zone in the query may not be directly routable through DFZ. At the same, the "KeyLocators" actually specify either the same zone, if the record being verified is signed by ZSK or DZSK, or a parent zone, in case the record is "NDNCERT" and it was signed by KSK. Therefore, there is no additional communication overhead to fetch the

103
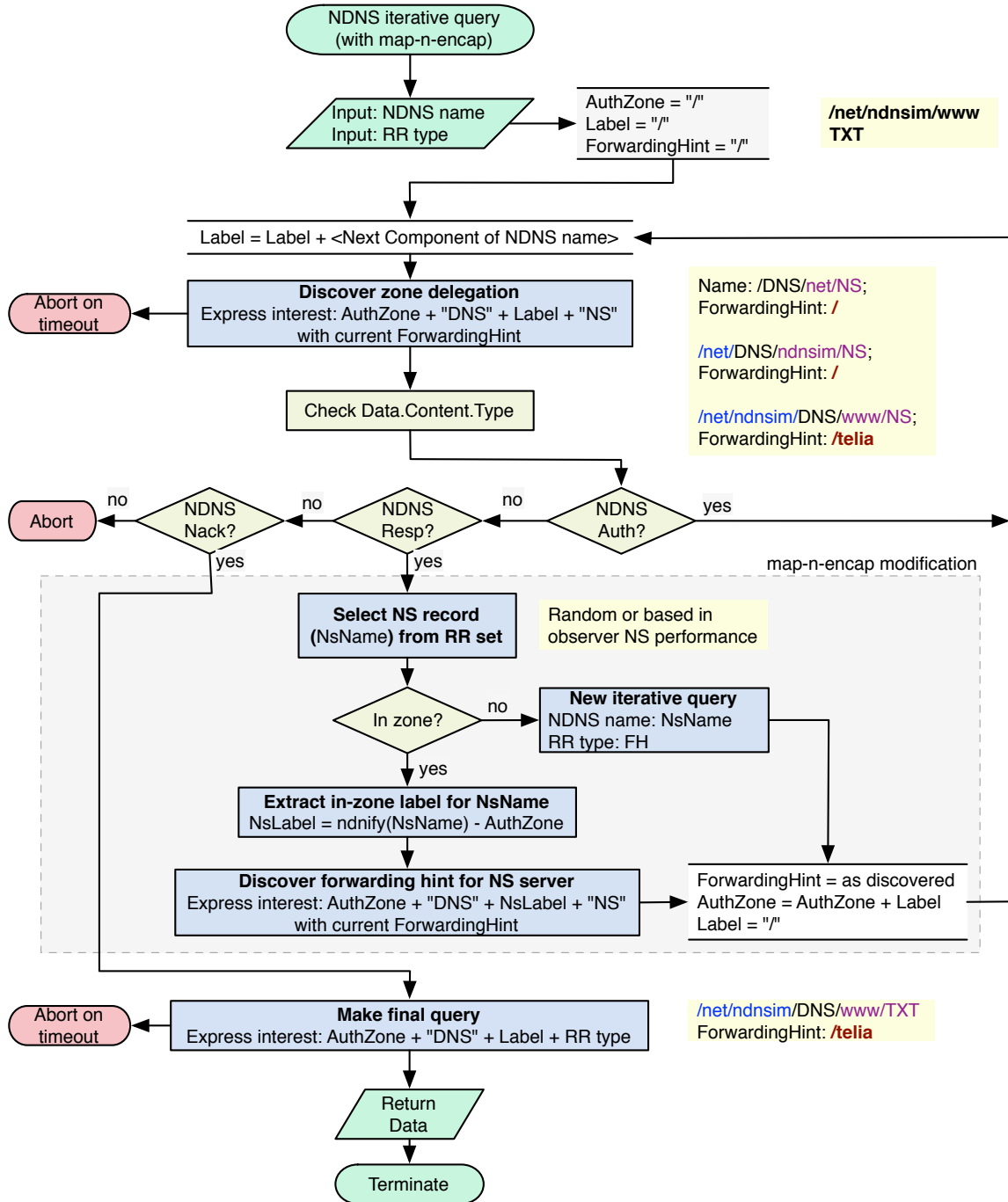
Figure 6.9: Iterative NDNS query process in map-n-encap NDN environment

required "NDNCERT" records, since the querier performing the NDNS iterative query process will have enough information at each step of the process to format the proper Interest, if necessary with a forwarding hint, which can be successfully routed through DFZ.[5]

### 6.4.2 Zone delegation discovery: special case of iterative query

Since one of the primary aims of NDNS to be a general-use universal database, applications may directly use NDNS iterative query format (see Section 3.1.4.1) to name their Data, explicitly separating zone, label, and record type. Although these Data names can be directly used in an Interest, assuming the zone's prefix is directly routable everywhere, map-n-encap environment requires an additional piece of information, the forwarding hint, to be attached to such Interests. This by and large highlights the need for a special iterative query type: a zone's forwarding hint discovery. We define this query as an abridged version of the general-purpose NDNS iterative query, shown in Figure 6.9, with the removed final part—there is no question for the final answer. This final question will in fact be replaced by the application's Interest. Both NDNS and the application will perform a full iterative query process, except that NDNS discovers delegation and forwarding hint information about the zone's delegation, and the application asks for the record in the zone.

### 6.4.3 Mobile producer and forwarding hint updates

NDNS defines a simple, secure, and easy-to-use dynamic update mechanism (Section 4.4), which can be employed to update "FH" records when a mobile producer

---

[5]There are two options for the NDNS iterative query verification: (1) verification of each response and (2) verification of only the final answer. The second approach is close to the existing DNSSEC definition and may result in fewer verification steps. At the same time, the first approach is more NDN-centric and guarantees correctness at each step of the iterative query process, i.e., incorrect or corrupted delegation will be ignored. Our prototype NDNS implementation supports both verification options, using the second approach by default.

moves from one network to another. Indeed, one currently deployed mobility solution is to rely on DNS as the mapping service to keep up-to-date information about a mobile's latest location [ZWC11, Zhu13]. In the context of a mobile producer we are taking more than a reasonable assumption that authoritative NDNS name servers that are responsible for maintaining "FH" records for mobile producers have a stable location. In other words, while the producer may change location relatively quickly, the place (essentially a form of a rendezvous point) which stores a producer's attachment information is fixed. In this way, both a mobile producer and a producer's client can send NDNS queries to the zone, the former to perform Interest-based dynamic updates and the latter to discover the latest information about the producer.

Similar to DNS, NDNS relies on caching to enable scalability, which means that records should not change too fast. At the same time, a particular mobile producer and its designated NDNS authority server are free to choose lifetimes for NDNS iterative responses (Freshness in terms of NDN Data packets), essentially controlling the time granularity for possible forwarding hint changes.

Finally, a potentially slow pace of forwarding hint updates does not preclude successful operation of the highly mobile Data producers. Remember that the forwarding hint is devoted entirely to directing the Interest packets through the DFZ and does not (in general) represent location inside the destination network. Such a location can be tracked inside the network using any existing mobile node tracking mechanisms, and the network can implement proper handover and redirection mechanisms to deliver an Interest to constantly moving producers, e.g., in the same way it is done in cellular networks.

## 6.5 Discussion

There is a well known Rekhter's law that relates routing scalability to the congruency between addressing and topology: "Addressing can follow topology or topology can follow addressing. Choose one." [MZF07] In other words, no matter how a routing system is designed, it can scale only if either (1) the address structure reflects the topology, or (2) the topology is built based on the addresses.

A number of proposed routing protocols, including ROFL [CCK06] and VRR [CCN06], fall into the second category. These designs perform routing operations over virtual (overlay) topologies, which are built based on node IDs (e.g., hashes of node names). As a result, their routing table size can scale on the order of $\log N$, where $N$ is the maximum number of node IDs. However, because the topology is virtual, the actual forwarding paths selected by these routing protocols may have a significant stretch. From a network operators' perspective, the resulting virtual topology does not obey administrative boundaries nor allow traffic engineering.

Routing schemes from the first category, such as Compact routing [TZ01], Landmark routing [Tsu88], and many others, including the existing Internet routing system, are more operator-friendly and easier to implement. In these systems the routing table size scales by letting nodes use topology-dependent addresses (names). Map-n-encap is a simple and effective way to obtain topology-dependent names. For example, a recent work [GKR11] argued that one could successfully scale routing with flat data names if one just concatenates the (also flat) names of aggregation entities in front of data names, as another example of the map-n-encap idea [Dee96].

We believe that routing in NDN should be performed on the physical topology. Based on Rekhter's law, our only option is to scale routing by topology-dependent names. While users and applications need topology-independent names, this conflict can be solved by employing the well known map-n-encap approach, introduced

almost twenty years ago.

Although the forwarding hint represents some form of a location related to the Data producer, we do not consider it to be a step back from data-centricity of NDN. This hint is nothing more than a suggestion of where the requested content may reside. For example, if the client expresses an Interest for a Data packet that is available locally (e.g., the producer resides inside the same provider and there are corresponding FIB entries on provider's routers), then this Interest will be forwarded to the local producer, independent of whether it carries a forwarding hint or not. Also, Interests will pull Data back from the very first router that cached the desired Data, independent of how many providers the data producer may connect to.

The binding between the application name and the hint is not signed. The forwarding hint can be easily modified in transit, seamlessly redirecting Interests out of the requested way. This can be used for good as well as for bad. For example, routers may modify aliases to enforce traffic engineering agreements or to select better paths to producers. At the same time, a misconfigured or compromised router can request all passing-through Interests go to unintended destinations, in an attempt to black-hole or eavesdrop traffic, or to DDoS another ISP's network. However, given NDN Interest packets are not signed in general, routers can already modify data names without being detected and perform the same "evil" activities. In short, introducing the forwarding hint does not introduce any new vulnerabilities into the system.

The forwarding hint approach is an effective mechanism to deliver packets under map-n-encap without impacting the major advantages of NDN communication paradigm, including in-network caching. We believe that, for the time being, forwarding hint represents the best tradeoffs among the options to scale an unlimited application name space of NDN with the use of conventional routing.

108

# CHAPTER 7

# ndnSIM platform for simulation-based evaluations of NDN deployment and NDN-based applications

The fundamental changes introduced by the NDN architecture to the Internet communication paradigm call for extensive and multidimensional evaluations of various aspects of the NDN design. While the existing implementation of NDN (NDN platform code base [NDN13a]) along with the testbed deployment give invaluable opportunities to evaluate both the NDN infrastructure design as well as its applications in a real-world environment, it is both difficult to experiment with different design options and impossible to evaluate the design choices in large scale deployment. To meet such needs and provide the community at large with a common experiment platform, we have developed an open source NDN simulator, ndnSIM, based on NS-3 network simulator framework [ns 11].

The design of ndnSIM has the following goals in mind:

- Being an open source package to enable the research community to run experimentations on a common simulation platform.

- Being able to faithfully simulate all the basic NDN protocol operations.

- Maintaining optional packet-level interoperability with NDNx codebase implementation [NDN13a], to allow sharing of traffic measurement and packet analysis tools between NDNx and ndnSIM, as well as direct use of real NDNx

traffic traces to drive ndnSIM simulation experiments.

- Being able to to support large-scale simulation experiments.

- Facilitating network-layer experiments with routing, data caching, packet forwarding, and congestion management.

- Providing a simple way to evaluate the performance of NDN-based applications, such as NDNS, in large-scale environments.

Following the NDN architecture, ndnSIM is implemented as a new network-layer protocol model, which can run atop any available link-layer (point-to-point, CSMA, wireless, etc.), network-layer (IPv4, IPv6), and transport-layer (TCP, UDP) protocol models. This flexibility allows ndnSIM to simulate various homogeneous and heterogeneous deployment scenarios (e.g., NDN-only, NDN-over-IP, etc.).

The simulator is implemented in a modular fashion, using separate C++ classes/interfaces to model and abstract behavior of each network-layer entity in NDN: pending Interest table (PIT), forwarding information base (FIB), content store, network and application interfaces, Interest forwarding strategies, etc. This modular structure allows any component to be modified or replaced easily with little or no impact on other components. In addition, the simulator provides an extensive collection of interfaces and helpers to perform detailed tracing behavior of every component and NDN traffic flow.

The ndnSIM implementation effort started in the fall of 2011. Since then the initial implementation has been used both by ourselves for various NDN design and evaluation tasks and by external alpha testers. The first release of ndnSIM as an open-source package has been available since June of 2012, and we still continue active development, adding new features that extend functionality of the simulator and address needs of the community. For example, in the summer of 2013, we have

released an updated version of ndnSIM (version 0.5), featuring replaceable NDN wire format, exclude filter support, and extended support for application-level simulations (i.e., a full-featured API for simulated applications). More detailed information about the release, code download, basic examples, and additional documentation is available on the ndnSIM website `http://ndnsim.net/`.

## 7.1 Design

The desire to create an open source NDN simulation package largely dictated our selection of the NS-3 network simulator [ns 11] as the base framework for ndnSIM. Although NS-3 is relatively new and does not have everything that the commercially available Qualnet or legacy ns-2 simulator has (e.g., NS-3 does not have native support for simulating conventional dynamic IP routing protocols[1]), it offers a clean, consistent design, extensive documentation, and implementation flexibility.

In this section we provide insights about main components of ndnSIM design, including a description of protocol implementation components.

### 7.1.1 Design overview

The design of ndnSIM follows the philosophy of network simulations in NS-3, which devises maximum abstraction for all modeled components. Similar to existing IPv4 and IPv6 stacks, we designed ndnSIM as an independent protocol stack that can be installed on a simulated network node. In addition to the core protocol stack, ndnSIM includes a number of basic traffic generator applications and helper classes to simplify creation of simulation scenarios (e.g., helper to installing the NDN stack and applications on nodes) and tools to gather simulation statistics for measurement purposes.

---

[1]`http://www.nsnam.org/docs/release/3.18/models/html/routing-overview.html`

The following list summarizes the component-level abstractions that have been implemented in ndnSIM. Figure 7.1 visualizes the basic interactions between them:

- **ndn::L3Protocol**: implementation of the core NDN protocol interactions: receiving Interest and Data packets from upper and lower layers through Faces;

- **ndn::Face**: abstraction to enable uniform communications with applications (ndn::AppFace) and other simulated nodes (ndn::NetDeviceFace);

- **ndn::ContentStore**: abstraction for in-network storage for Data packets (e.g., short-term transient, long-term transient, long-term permanent);

- **ndn::Pit**: abstraction for the pending Interest table (PIT) that keeps track (per-prefix) of Faces on which Interests were received, Faces to which Interests were forwarded, as well as previously seen Interest nonces;

- **ndn::Fib**: abstraction for the forwarding information base (FIB), which can be used to guide Interest forwarding by the forwarding strategy;

- **ndn::ForwardingStrategy**: abstraction and core implementation for Interest and Data forwarding. Each step of the forwarding process in ndn::ForwardingStrategy—including lookups to ContentStore, PIT, FIB, and forwarding Data packets according to PIT entries—is represented as virtual function calls, which can be overridden in particular forwarding strategy implementation classes (see Section 7.1.7);

- reference NDN applications, including simple traffic generators and sinks.

Each component with the exception of the core ndn::L3Protocol has a number of alternative implementations that can be arbitrarily chosen by the simulation scenario using helper classes (see ndnSIM online documentation `http://`
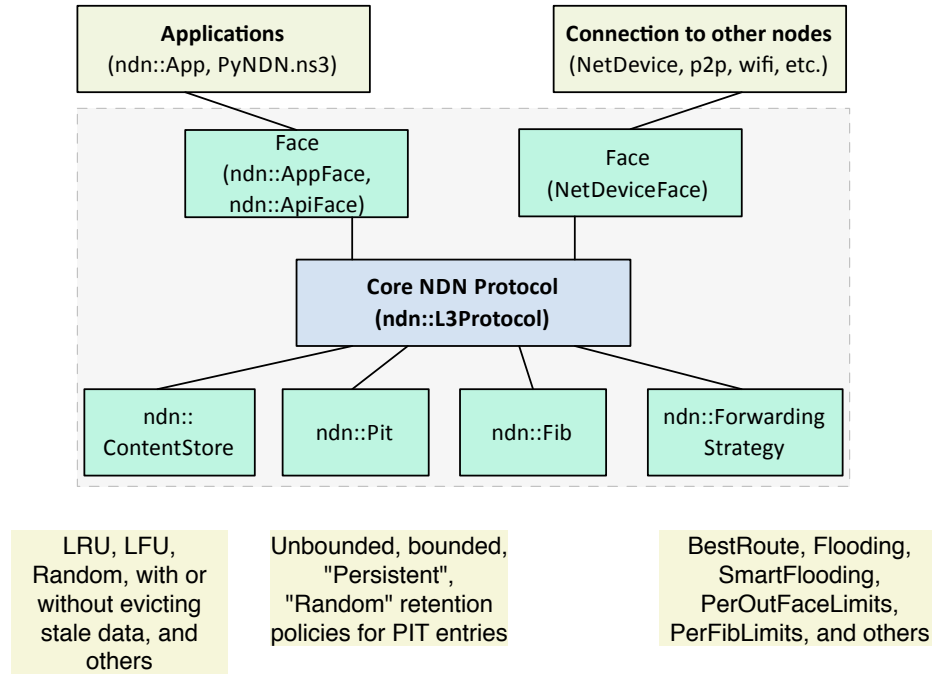
Figure 7.1: Block diagram of ndnSIM components

`ndnsim.net/helpers.html`). For example, ndnSIM currently provides implementations for ContentStore abstraction with Least-Recently-Used (LRU), First-In-First-Out (FIFO), and random replacement policies for cached Data, unbounded and bounded implementations of PIT abstraction with persistent and random PIT entry retention policies, and a number of forwarding strategy options listed in Figure 7.1.

By default, in order to optimize simulation run-time, ndnSIM uses a simplified version of the wire format for Interest and Data packets. However, it allows switching to the wire format of the current NDN implementation in the NDN platform codebase (NDNx Binary XML Encoding[2]) either statically inside the simulation scenario or dynamically for each simulation run. This option allows reusing the existing traffic analysis tools (ndndump,[3] wireshark NDNx plugin[4]),

---

[2]`http://named-data.net/doc/0.1/technical/BinaryEncoding.html`
[3]`https://github.com/named-data/ndndump/`
[4]`https://github.com/named-data/ndnx/tree/master/apps/wireshark`

as well as drive simulations using real NDNx traffic traces. The implemented abstractions allow for addition of new packet formats in the future, e.g., if one wants to simulate NDN behavior with alternative packet formats or if there is a change in the default packet format of the NDN platform.

Design of ndnSIM contains a number of optional modules, including (1) a place-holder for data security (the current code allows attaching of a user-specified "signature" to Data packets), (2) an experimental support of negative acknowledgments (Interest NACK) to provide fast feedback about any data plane problem, (3) a pluggable Interest rate limit and interface availability component in the form of specialized versions of the NDN forwarding strategies, and (4) an extensible statistics module. Interested readers may refer to [YAM13, YAW12] for more detail on Interest NACKs and the Interest rate limit.

### 7.1.2 Core NDN protocol implementation

ndn::L3Protocol in ndnSIM is a central architectural entity and stands in the same line of class hierarchy as the corresponding Ipv4L3Protocol and Ipv6L3Protocol classes of the NS-3 framework that implement IPv4 and IPv6 network-layer protocols. ndn::L3Protocol is a logical component aggregator for all available communication channels with both applications and other nodes (Face abstraction, see Section 7.1.3), and it performs basic handling of incoming packets from Faces to a forwarding strategy.

The ndn::L3Protocol class defines the API to manipulate the following aspects of the NDN stack implementation:

- **AddFace/RemoveFace**: to register a new Face realization to NDN protocol or remove an existing Face;
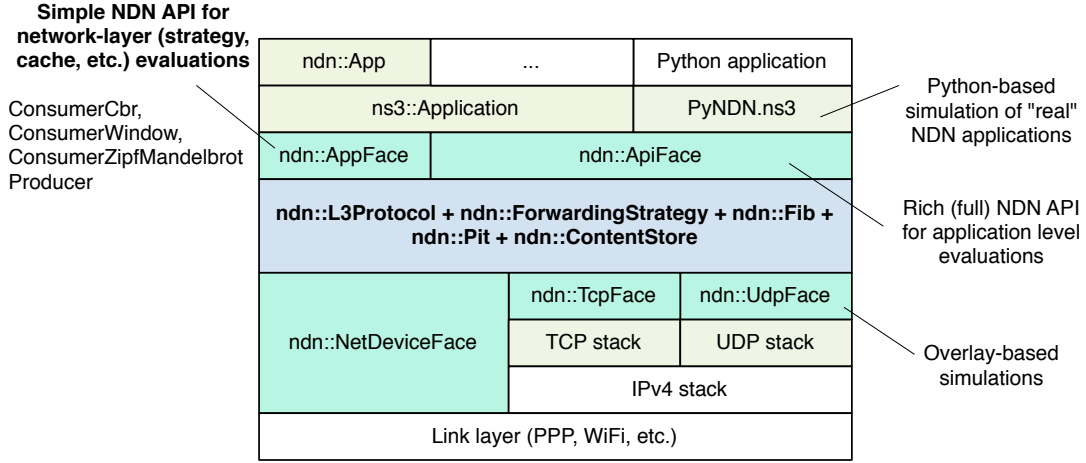
**Figure 7.2: Communication-layer abstraction for ndnSIM scenarios**

### 7.1.3 Face abstraction

To achieve our goal of providing maximum flexibility and extensibility, we make the ndnSIM design independent from the underlying transports through abstracting the inter-layer interactions. All communication between the core protocol implementation (ndn::L3Protocol), network, and applications is accomplished through a *Face* abstraction (ndn::Face), which can be realized in various forms (see Figure 7.2): a link-layer face (ndn::NetDeviceFace) for inter-node communication directly over the link layer, a network-layer face (ndn::Ipv4Face and ndn::Ipv6Face) and a transport-layer face (ndn::TcpFace and ndn::UdpFace) for inter-node overlay communication, and application-layer faces (ndn::AppFace and (ndn::ApiFace) for intra-node (application-to-local-NDN stack) communications. The difference between the last two faces lies in the amount and richness of the supported NDN API. The AppFace interface is designed and primarily targeted for network-layer simulations, where the only requirement for applications is to ensure flow of Interest and Data packets with predetermined parameters (e.g., a fixed rate). The ApiFace is a recent addition that provides a rich NDN API that allows writing/porting and simulating behavior of full-featured NDN applications.

The Face abstraction defines the following API:

- **SendInterest** (realization-specific): to pass an Interest packet from NDN stacks to the underlying layer (network or application).

- **SendData** (realization-specific): to pass a Data packet from NDN stacks to the underlying layer (network or application).

- **ReceiveInterest** (realization-specific): to pass an Interest packet from the underlying layer (network or application) towards the NDN stack.

- **ReceiveData** (realization-specific): to pass a Data packet from the underlying layer (network or application) towards the NDN stack.

- **RegisterProtocolHandlers/UnRegisterProtocolHandlers** (realization-specific): to enable forwarding of packets from the underlying layer (network or application) to the NDN stack.

- **SetMetric/GetMetric**: to assign and get Face metrics that can be used, for example, in routing calculations.

- **IsUp/SetUp**: to check if Face is enabled and to enable/disable Face.

- **GetFlags**: to get flags associated with the Face. Currently face flags can be used to determine whether the Face is for intra-node communication or inter-node communication.

### 7.1.4 Content Store abstraction

The Content Store at each NDN router enables in-network storage, providing efficient error recovery and asynchronous multicast data delivery. ndnSIM provides an interface to plug in different implementations of Content Store that can implement different indexing and item look up designs, different size limiting features, and different cache replacement policies.

The current version of ndnSIM contains three realizations of the Content Store abstraction with Least-Recently-Used (**ndn::cs::Lru**), First-In-First-Out (**ndn::cs::Fifo**), and Random replacement policies (**ndn::cs::Random**). Each of these realizations is based on a dynamic trie-based container with an (optional) upper bound on its size, and hash-based indexing on Data names (per-component lookup on a trie). Other Content Store modules are expected to be implemented either by ourselves or with the help of the community as the need arises.

The Content Store abstraction provides the following operations:

- **Add** (realization specific): caching a new or promoting an existing Data packet in cache.

- **Lookup** (realization specific): performing a lookup for previously cached Data.

The Content Store abstraction does not provide an explicit data removal operation. Lifetime of the Content Store entries depends on traffic patterns, as well as on the Data packet freshness parameter supplied by data producers.

### 7.1.5 Pending Interest table (PIT) abstraction

PIT (ndn::Pit) maintains state for each forwarded Interest packet in order to provide directions for Data packet forwarding. Each PIT entry contains the following information:

- the name associated with the entry;

- a list of incoming Faces, from which the Interest packets for that name have been received, with associated information (e.g., arrival time of the Interests on this Face);

- a list of outgoing Faces to which the Interest packets have been forwarded with associated information (e.g., time when the Interest was sent on this Face, number of retransmission of the Interests on this face, etc.);

- time when the entry should expire (the maximum lifetime among all the received Interests for the same name); and

- any other forwarding strategy specific information in the form of forwarding strategy tags (any class derived from ndn::fw::Tag).

The current version of ndnSIM provides a templated realization of PIT abstraction, allowing optional bounding of the number of PIT entries and different replacement policies, including

- persistent (**ndn::pit::Persistent**)—new entries will be rejected if the PIT size reaches its limit;

- random (**ndn::pit::Random**)—when the PIT reaches its limit, random entry (could be the newly created one) is removed from PIT;

- least-recently-used (**ndn::pit::Lru**)—the least recently used entry (the oldest entry with the minimum number of incoming faces) will be removed when the PIT size reaches its limit.

All current PIT realizations are organized in a trie-based data structure with hash-based indexing on Data names (per-component lookup on a trie) and additional time index (by expiration time) that optimizes removal of timed out Interests from the PIT.

A new PIT entry is created for every Interest with a unique name. When an Interest is received with a name that has been seen previously, the "incoming Faces" list of the existing PIT entry is updated accordingly, effectively aggregating (suppressing) similar Interests.

The PIT abstraction provides the following realization-specific operations:

- **Lookup**: find a corresponding PIT entry for the given content name of an Interest or Data packet;

- **Create**: create a new PIT entry for the given interest;

- **MarkErased**: Remove or mark a PIT entry for removal;

- **GetSize**, **Begin**, **End**, **Next**: get the number of entries in the PIT and iterate through the entries.

### 7.1.6 Forwarding information base (FIB)

An NDN router's FIB is roughly similar to the FIB in an IP router except that it contains name prefixes instead of IP address prefixes, and it (generally) shows multiple interfaces for each name prefix. It is used by the forwarding strategies to make Interest forwarding decisions.

The current realization of FIB (ndn::fib:FibImpl) is organized in a trie-based data structure with hash-based indexing on Data names (per-component lookup on a trie), where each entry contains a prefix and an ordered list of (outgoing) Faces, through which the prefix is reachable. The order of Faces is defined as a composite index, combining the routing metric for the Face and data plane feedback. Lookup for a match is performed on variable-length prefixes in a longest-prefix match fashion, honoring any exclude filter [NDN13b] if it is specified in the Interest.

#### 7.1.6.1 FIB population

Currently, ndnSIM provides several methods to populate entries in the FIB. A first is to use a simulation script to configure FIBs manually for every node in a simulation setting. This method gives the user full control over what entries are

present in which FIB, and it works well for small-scale simulations. However, it may become infeasible for simulations with large topologies.

The second method is to use a central global NDN routing controller to automatically populate all routers' FIBs. When requested (either before a simulation run starts, or at any time during the simulation), the global routing controller obtains information about all the existing nodes with the NDN stack installed and about all exported prefixes. The controller uses this information to calculate shortest paths between every node pair and updates all the FIBs. Boost.Graph library (`http://www.boost.org/doc/libs/release/libs/graph/`) is used in this calculation.

In the current version, the global routing controller uses Dijkstra's shortest path algorithm (using Face metric) and installs only a single outgoing interface for each name prefix. To experiment with multipath Interest forwarding scenarios, the global routing controller needs to be extended to populate each prefix with multiple entries. However, it is up to the particular simulation to define the exact basis for multiple entries, e.g., whether entries should represent paths without common links. We welcome suggestions and/or global routing controller extensions, which can be submitted on the GitHub website (`https://github.com/NDN-Routing/ndnSIM`).

Finally, a simple method to populate the FIB is to install a default route (route to "/"), which includes all available Faces of the NDN stack. For example, this method can be useful for simulations that explore how well an Interest forwarding strategy can find and maintain paths to prefixes without any guidance from the routing plane.

### 7.1.7 Forwarding strategy abstraction

Our design enables experimentation with various types of forwarding strategies, without any need to modify the core components. This goal is achieved by introducing the forwarding strategy abstraction (ndn::ForwardingStrategy) that implements core handling of Interest and Data packets in an event-like fashion. In other words, every step of handling an Interest or Data packet, including Content Store, the PIT, the FIB lookups, is represented as a virtual function that can be overridden in particular forwarding strategy implementation classes.

More specifically, the forwarding strategy abstraction provides the following set of overrideable actions:

- **OnInterest**: called by ndn::L3Protocol for every incoming Interest packet;

- **OnData**: called by ndn::L3Protocol for every incoming Data packet;

- **WillErasePendingInterest**: fired just before a PIT entry is removed;

- **RemoveFace**: called to remove references to a Face (if any used by a forwarding strategy realization);

- **DidReceiveDuplicateInterest**: fired after detection reception of a duplicate Interest;

- **DidExhaustForwardingOptions**: fired when the forwarding strategy exhausts all forwarding options to forward an Interest;

- **FailedToCreatePitEntry**: fired when an attempt to create a PIT entry fails;

- **DidCreatePitEntry**: fired after a successful attempt to create a PIT entry;

- **DetectRetransmittedInterest**: fired after detection of a retransmitted Interest. This event is optional and is fired only when the "DetectRetrans-

missions" option is enabled in the scenario. Detection of a retransmitted Interest is based on observing that a new Interest matching an existing PIT entry arrives on a Face that is recorded in the incoming list of the entry. Refer to the source code for more details.

- **WillSatisfyPendingInterest**: fired just before a pending Interest is satisfied with Data;

- **SatisfyPendingInterest**: actual procedure to satisfy a pending Interest;

- **DidSendOutData**: fired every time a Data packet is successfully sent out on a Face (can fail during congestion or if the rate limiting module is enabled);

- **DidReceiveUnsolicitedData**: fired every time a Data packet arrives, while there is no corresponding pending Interest for the Data's name. If the "CacheUnsolicitedData" option is enabled, then such Data packets will be cached by the default processing implementation.

- **TrySendOutInterest**: fired before actually sending out an Interest on a Face;

- **DidSendOutInterest**: fired after successfully sending out an Interest on a Face;

- **PropagateInterest**: basic Interest propagation logic;

We anticipate that in future releases more events will be added to the forwarding strategy abstraction. At the same time, additional events can be created in an object-oriented fashion through class inheritance. For example, an Interest NACK extension (see [YAM13, YAW12] for more detail) is implemented as a partial specialization of the forwarding strategy abstraction.

Figure 7.3 shows a partial hierarchy of the currently available forwarding strategy extensions (Nacks, GreenYellowRed) and full forwarding strategy implementations (Flooding, SmartFlooding, and BestRoute) that can be used in simulation scenarios. While all current realizations are inherited from a Nack extension that implements additional processing and events to detect and handle Interest NACKs [YAM13, YAW12], NACK processing is disabled by default and can be enabled using "EnableNACKs" option.
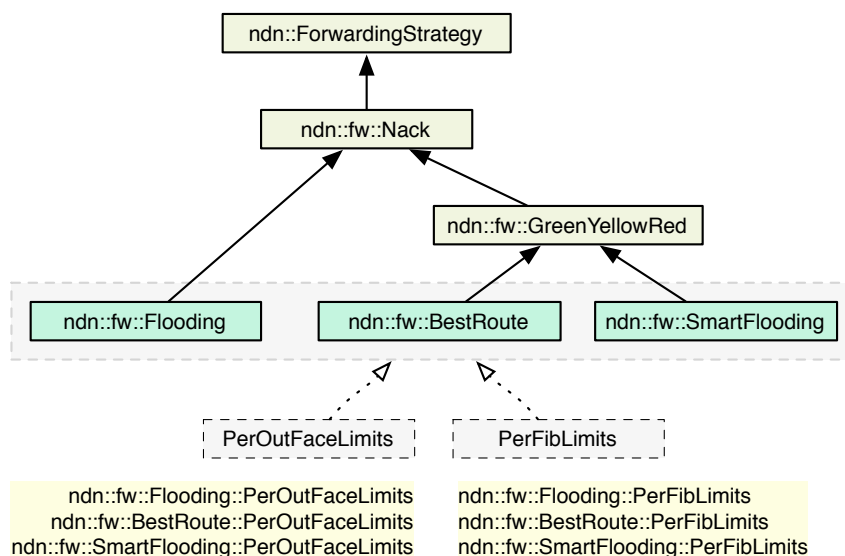


Figure 7.3: Available forwarding strategies (Flooding, SmartFlooding, and BestRoute are full realizations, which can be wrapped over PerOutFaceLimits or PerFibLimits extensions)

SmartFlooding and BestRoute realizations rely on color-coding for the status of each Face, based on observed data plane feedback [YAM13, YAW12].

- **GREEN**: the Face works correctly (e.g., if an Interest is sent to this Face, a Data is returned);

- **YELLOW**: the status of Face is unknown (e.g. it may have been added recently or has not been used in a while);

- **RED**: the Face is not working and should not be used for Interest forwarding.

The status information is attached to each Face in a FIB entry and initialized to a YELLOW status. Every time a Data packet is returned as a response to a previous Interest, the corresponding Face in the FIB entry is set to GREEN. Every time an error occurs (a PIT entry timeout or, if enabled, a NACK-Interest is received), the Face is turned back to YELLOW status. If a Face has not been used for long enough, it also is turned back to YELLOW status. RED state is assigned to the Face when the lower layer notifies the NDN stack of a problem (link failure, connection error, etc.).

The following list summarizes the processing logic in the currently available full forwarding strategy implementations:

- **Flooding strategy** (ndn::fw::Flooding): an Interest packet is forwarded to all Faces that are available in the FIB entry for the Interest's prefix, except for the incoming Face of that Interest.

- **Smart flooding strategy** (ndn::fw:SmartFlooding): if a FIB entry contains at least one GREEN Face, an Interest is forwarded only to the highest-ranked GREEN Face. Otherwise, all YELLOW Faces will be used to forward the Interest. RED Faces are not used for Interest forwarding.

  This strategy mode can be used in simulations without routing input, and the data plane can use Interest packets to discover and maintain working paths.

- **Best-Route strategy** (ndn::fw::BestRoute) forwards Interest packets to the **highest-ranked** GREEN (if available) or YELLOW face. RED Faces are not used for Interest forwarding.

There are additionally two experimental forwarding strategy extensions, Per-OutFaceLimits and PerFibLimits, that can be wrapped over existing or new forwarding strategy realizations. These extensions attempt to avoid congestion in the network and maximize network utilization by taking into account the Interest rate limits [YAM13, YAW12] in the Face selection process at different granularities: either individually for each outgoing face (PerOutFaceLimits) or individually per each outgoing face in each FIB entry (PerFibLimits). For example, if the highest-ranked Face has reached its capacity—more specifically, the number of pending Interests for this Face has reached a set maximum limit—the strategy selects the next Face in rank order that is under the limit.

### 7.1.8 Reference applications

Applications interact with the core of the system using the ndn::AppFace realization of the Face abstraction. To simplify implementation of a specific NDN application, ndnSIM provides a base ndn::App class that takes care of creating ndn::AppFace and registering it inside the NDN protocol stack. It also provides default processing for incoming Interest and Data packets.

Listed below is the set of reference applications that are currently available in ndnSIM:

- **ndn::ConsumerCbr**: an application that generates Interest traffic with a predefined frequency (constant rate, constant average rate with inter-Interest gap distributed uniformly at random, exponentially at random, etc.). Names of generated Interests contain a configurable prefix and a sequence number. When a particular Interest is not satisfied within an RTT-based timeout period (same as TCP RTO), this Interest is re-expressed.

- **ndn::ConsumerBatches**: an on-off-style application generating a specified number of Interests at specified points of a simulation. Names and

125

retransmission logic is similar to the ndn::ConsumerCbr application.

- **ndn::Producer** a simple Interest-sink application that replies to every incoming Interest with a Data packet of a specified size and name matching the Interest.

### 7.1.9 PyNDN compatible interface

ndnSIM and NS-3 in general provide the opportunity to run NS-3 simulations using python bindings. Generally, this applies just to writing the scenarios, defining scenario parameters, and requesting specific C++-implemented applications run at selected simulation times. ndnSIM provides a way not only to start C++ code from python, but also to use python as a driving force for the simulation, i.e., it is possible to simulate python-based applications directly.

In particular, we have implemented an interface that emulates the functionality of PyNDN, python bindings for NDN [KB11]. Using this interface, it is possible to run the real code that has been written against PyNDN directly, but run it in a virtual simulated network environment. Of course, it is still not possible to run any arbitrary application, since a simulation applies certain limitations such as restrictions on using synchronous network operations. In a sense, the implemented interface follows the idea used in the DCE module for NS-3 [Lac10], but applies this idea to Python applications instead of compiled versions of C/C++ code.

Of course, the run-time performance of such simulations will not be perfect, since there are multiple redirections from python and C++ code, that have different calling conventions and memory management. At the same time, PyNDN ndnSIM interfaces can be invaluable for fast prototyping and evaluating prototypes of NDN applications. In fact, as described in Chapter 3, we have successfully used this extension to evaluate our python-based prototype implementation of the NDNS system within a large-scale network.

## 7.2 Summary

ndnSIM is designed as a set of loosely bound components that give a researcher an opportunity to modify or replace any component, with no or little impact on the other parts of ndnSIM. Our simulator provides a set of reference application and helper classes, allowing evaluation of various aspects of NDN protocol under many different scenarios. The first version of ndnSIM has been publicly released since June of 2012. More detailed information about the release and additional documentation is available on the ndnSIM website `http://irl.cs.ucla.edu/ndnSIM/`.

# CHAPTER 8

# Related work

The core of the present thesis is based on the concepts and design choices of the Domain Name System (DNS) [Moc87a, Moc87b], applying them in the context of a pure data-centric Named Data Networking architecture [JST09, EBJ10]. Therefore, the first major related work is DNS itself and its usages by the Internet community as a universal database, addressing a large number of operational needs, discussed in Section 8.1.

The second part of the thesis applies NDNS to address the problem of routing resource authorization, which is an ongoing problem even in the context of the IP Internet. Some existing proposals to provide such authorization for IP prefixes are listed and briefly discussed in Section 8.2.

In the third part, the thesis addresses the growing problem of global routing scalability. While in our work we explicitly focus on scaling name-based NDN routing, the proposed solutions borrow many ideas from the existing body of work, summarized in Section 8.3.

Finally, Section 8.4 shows a number of alternative attempts that have been made to allow flexible experimentation with the Named Data Networking architecture. All these attempts are largely complementary to each other and to the ndnSIM platform presented in the last part of the current thesis, providing essential paths for building a better understanding of different parts of the architecture (applications, forwarding strategy, cache) and at different levels (micro-, macroscopic).

## 8.1 DNS usage as distributed DB

While the original motivation and driving force to design and implement DNS was to address the need to simplify and optimize mapping between host names and network addresses (the way of keeping track of all host name and address mapping in "`hosts.txt`" was no longer adequate [Moc87a]), DNS was recognized from its initial design as a much more powerful system. DNS can not only provide a solution for name-address mapping, but also be used as a highly-scalable general-use database system for various network and application-level purposes. In fact, there are more than 80 officially IANA-registered resource record types [IAN13] to store various types of information, which can be used for a huge range of different application and services. For example, a simple "TXT" resource record can map a domain name to an opaque text string, which can be used from a simple domain annotation to domain-owner authorization purposes (e.g., to authorize usage of Google Webmaster tools [Goo13]).

There is also an application security-related use of DNS/DNSSEC known as DANE [HS12], which is currently getting more and more attention, because of occurrence of numerous accidents with the commonly-used PKI-based Certification Authorities (CA) certifications, including a recent semi-successful attack on the Comodo registration authority [Rob11] that resulted in issuance of several phony certificates. DANE defines another new "TLSA" resource record, which can be used by the service owners to define specific or set restrictions on which CA (and which specific CA's key) is authorized to issue certificates for the service.

There is also a variation of the DNS protocol, multicast DNS (or mDNS in short) [CK13], for resource discovery services within local environments. In mDNS, requests/queries are sent out in a multicast fashion and each device receiving and understanding the request can send a proper mDNS response, announcing its presence and the requested services. There is actually an ongoing effort based

on community demand to enable mDNS services on a larger DNS-like scale, since it became popular and extremely useful service nowadays.

All of these highlight the universality of the DNS protocol, which we hope is fully captured and extended in a pure data-oriented manner within the NDN architecture by an NDNS system designed in this thesis.

## 8.2 Securing global routing resources

Routing security is one of the hottest areas in network research [NM09]. The closely related routing area for which NDNS, designed in the present thesis, provides a solution is authorization of routing resources, such as name prefixes in NDN and IP prefixes on the existing Internet. While several global routing security proposals, such as Secure BGP (S-BGP) [KLS00, KLM00], provide a comprehensive security solution for the global routing system, these solutions are too hard to deploy and use in practice.

There are also several alternative efforts to provide less comprehensive security solutions, providing ways to ensure that prefixes are announced only by the legitimate entities. These solutions can be easily deployed within the current Internet infrastructure and provide a reasonable protection against current security threats and misconfigurations. One such effort, known as Resource Public Key Infrastructure (RPKI) [MVK12], is supported by the Regional Internet Registries (RIRs) to provide strict certification for AS numbers and IP prefixes. RPKI uses the standard X.509 Public Key Infrastructure (PKI) trust model [AL03, CSF08] and allows flexible delegation and re-delegation of resources. The usual operational concern about the RPKI way of authorizing resources is related to the storage of the actual resource certificates, called route origination authorizations. This storage is based on a hierarchy of dedicated RPKI repositories, which means that networks need to maintain a dedicated storage infrastructure, if they want

to sub-delegate RIR-assigned resources, while the original certifications are stored in RIR-provided repositories [ARI13].

A separate effort that motivates by and large the inspiration for the proposal of the present thesis is the ROVER project [GMG12, GMO13], which enables routing resource certification similar to RPKI, but leverages the existing DNS/DNSSEC infrastructure. In short, ROVER defines a way for network providers to designate which prefixes (sub-prefixes) can be present in the global routing system and which ASes are allowed to announce these prefixes. This information is recorded in the already delegated reverse DNS zone "`in-addr.arpa`" in the form of several new resource record types ("SRO" and "RLOCK"). While the ROVER effort can be considered complimentary to RPKI, it better fits the existing infrastructure, including the existing DNS/DNSSEC deployment and trust model.

## 8.3  Routing scalability

Routing scalability has long been a recognized problem of the present Internet [ATL10]. A number of currently enforced regulations reduce the problem (e.g., limiting the maximum prefix size in the global routing table), but unfortunately cannot completely eliminate it. A common approach for solving this problem is through address aggregation, which requires address allocation to reflect the network topology, either directly or indirectly. Existing solutions can be categorized into two groups: namespace elimination and namespace separation.

Identifier-Locator Network Protocol (ILNP) [AB12] is a global routing scalability solution most closely related to the proposal in this thesis. ILNP is a new network protocol that switches from an IP communication model, where IP addresses (both in IPv4 and IPv6) are overloaded with the functionality of network locators and host identifiers, leading to many existing problems with application session maintenance as the network topology changes. In some sense, ILNP ex-

plicitly "untangles" usages of IP addresses at different layers by mandating the use of separate network locators for packet forwarding, host identifiers for transport sessions, and DNS names within applications (e.g., not possible to use IP address in a WEB browser, instead of the domain name). Similar to our proposal, ILNP relies on the existing DNS/DNSSEC deployment to map from application-level domain names to node identifiers, which are then mapped to the network locators.

Although in our proposal we have a similar separation that uses application-specific names to uniquely identify pieces of the requested data and forwarding hints to suggest locations where the Data can be, there are several notable differences. First, the suggested location by the forwarding hint is not mandatory (e.g., not needed for the local data retrieval, either when Data producer is local or Data was previously cache) and can be completely ignored by NDN routers. Second, in NDN generally, the same names are used throughout the application, transport, and network layers. As a result, the application names can be directly mapped using NDNS to the forwarding hints, instead of requiring an additional name to host identifier mappings.

The scalability problem in conventional routing can be solved only through efficient aggregation. Over the last two decades, the research community has introduced a number of approaches to enable efficient aggregation, which can be divided into two big categories. First, there are proposals that call for clear separation between addresses that appear in the global routing table and those addresses (or names) that are used by end-hosts [Dee96, MWZ07, JMY08, ASB99, GKR11], while some additional service (e.g., DNS) is used to map end-host addresses to (a set of) routable addresses. A second category contains proposals that call for an extended use of multiple provider-dependent addresses, while upper layers (transport layers [Tsu91, HWB08] or a shim layer between IP and TCP [NB07]) need to take care of managing multiple addresses within a single connection. When estab-

lishing connection, upper layers may either obtain multiple addresses through DNS by mapping an application-requested domain name to a set of resource records, or negotiate during the connection handshake (Shim6). In both cases, addresses that appear in the global routing table are only addresses of ISPs, which are limited in number and easy to aggregate.

Our proposal belongs to the first category and is in the same spirit as the map-n-encap approach [Dee96] for scaling the IP routing system. However, our work ensures that the design is consistent with NDN's name-based data retrieval model: (1) Interests are sent by data consumers towards data producers, leaving a trail for returning Data packets; and (2) an Interest can retrieve data as long as the names match, even if the Interest does not reach the producer.

An alternative way to solve the routing scalability problem is to replace the conventional routing system with, as an example, Hyperbolic routing [PKB10, BPK10, KPK10]. While this method does not require a global routing table, it still requires an additional mapping service to map names to coordinates. In this regard, the solutions are conceptually similar, but there is still a question about how well Hyperbolic routing can work and how can it handle existing complex routing policies between ISPs.

## 8.4 NDN architecture evaluation tools

Over the last few years several efforts have been devoted to the development of evaluation infrastructure for NDN architecture research.

One effort by the NDN project team is support of NDN within the Open Network Lab (ONL) [ZEB11]. ONL currently contains 14 programmable routers and over 100 client nodes, connected by links and switches of various capability. Every node and router runs an NDNx implementation from the NDN platform code base. Users have full access to the hardware and software state of any node

of ONL. It is also possible to run and evaluate NDNx implementation on nodes of DeterLab testbed [DET00]. Having a programmable non-virtualized testbed is a valuable option, though its capability is limited to evaluate relatively small-size networks. For larger-scale experiments, researchers may need to resort to simulations.

Rossi and Rossini [RR11] developed ccnSim to evaluate the caching performance of NDN. ccnSim is a scalable chunk-level simulator of NDN that is written in C++ under the Omnet++ framework, which allows assessing NDN performance in large-scale scenarios (up to $10^6$ chunks) on a standard consumer-grade computer hardware. ccnSim was designed and implemented with the main goal of running experimentations of different cache replacement policies for the NDN routers content store. Therefore, it is not a fully featured implementation of the existing NDN protocol. In the current version of ccnSim, PIT and FIB components are implemented in the simplest possible way, thus it is unable to evaluate different data forwarding strategies, different routing policies, or different congestion control schemes.

Another NDN simulator has been written at Orange Labs by Muscariello and Gallo [MG11]. Their Content Centric Networking Packet Level Simulator (CCNPL-Sim) is based on SSim that is a utility library which implements a simple discrete-event simulator. Combined Broadcast and Content-Based routing scheme (CBCB) [CRW04] must run as an interlayer between SSim and CCNPL-Sim to enable name-based routing and forwarding over generic point-to-point networks. Though a canonical NDN model was completely reimplemented in CCNPL-Sim in C++, this solution has the drawback of using a custom discrete-event simulator that is unfamiliar to most researchers. Additionally, an obligatory usage of CBCB narrows the possible experimentation area, making it impossible to evaluate other routing protocols, such as OSPF-N (OSPF extension for NDN) or routing on Hyperbolic Metric Space [PKB10, BPK10, KPK10].

A completely different approach has been taken by Urbani et al. [ns 13]. They provide support of Direct Code Execution (DCE) for NDNx implementation inside the NS-3 simulator. The general goal of DCE NS-3 module is to provide facilities to execute existing implementations of user-space and kernel-space network protocols within the NS-3 simulated environment. The main advantage of this approach is that simulations can use the existing unmodified NDNx code directly, thus providing maximum realism and requiring no code maintenance (as new versions are supposed to run in DCE NS-3 without much effort). However, this approach also raises a few concerns. First, the real implementation, including NDNx code, is rather complex, difficult to modify to explore different design approaches, and contains a great deal of code that is irrelevant for simulation evaluations. Second, there is a known scaling problem, because each simulated node has to run a heavy DCE layer and a full-sized real NDNx implementation.

# CHAPTER 9

# Conclusions

The recently proposed Named Data Networking (NDN) architecture can arguable bring a number of advantages in support of the current communication patterns, compared to the existing Internet infrastructure. At the same time, moving from the blueprint of the proposed architecture to the actually deployed system faces a number of operational hurdles, some of which are addressed in this thesis by the use of NDNS, the scalable general-use database system.

The developed NDNS database system applies the best principles of the existing Domain Name System (DNS) protocol to a pure data-centric communication model within NDN architecture. Although DNS by itself seems to be data-centric and implementing DNS within NDN looks trivial on the surface, the presented analysis reveals that many elements of the DNS design rely heavily on the connection-oriented nature of the current communication model of the Internet. For instance, DNS caching resolvers rely on point-to-point connectivity to the authoritative name servers to ask the same question about the resolved domain and receive different answers, such as referrals to next-level authoritative name servers, rejections, and final answers. While such an operation can be emulated in NDN as well, it would inevitably violate principles of data centricity and not be able to leverage benefits of in-network caching and storage provided by NDN. The designed NDNS protocol solves this problem by requiring the caching NDNS resolvers to issue specific questions to the network, iteratively and progressively inferring which NDNS zone is authoritative to the resolved question. While this

change increases the complexity of the caching resolver logic and potentially increases the number of iterative queries necessary to resolve the question, NDNS is able to effectively utilize all the benefits of the NDN, ensuring that NDNS is scalable on the same level or even better than the existing DNS.

We anticipate that the designed NDNS will become not just a big, secure database, but also a new ecosystem for the NDN network itself and many NDN-based applications. With the help of NDNS it is possible to address many of the existing and future operational challenges, including routing resource authorization, providing the core for security credential management, and providing the base for scaling name-based routing in NDN through application of the map-n-encap concept.

In addition to that, the present work is one of the first attempts to apply successful design principles of a protocol, designed for the existing channel-based network architecture, within a pure data-centric NDN-based network environment. While it is possible to apply the existing IP-based designs in NDN by emulating a concept of the channel-based communication (e.g., by including "source" name as part of the Interest name and ensuring uniqueness of each Interest via a sequence number), such designs will not be able to take full advantage of the NDN-provided benefits. An efficient data-centric application or protocol must follow a very carefully crafted naming model for Interests and Data, ensuring that NDN's build-in multicast, in-network caching, and other benefits are fully utilized. This observation highlights the enormous power of names within NDN architecture. In the designed NDNS system, the naming model (see Chapter 3) provides enough information to guide Interests towards the appropriate network locations, to dispatch Interests to the appropriate applications, and to identify specific pieces of application-specific information requested by the Interests. At the same time, the designed naming ensures that each piece of NDNS Data has unique (stable) name, and any parties requesting the same Data ("NS" referrals

137

or final answers) use the same names. Essentially, this guarantees that if there are multiple Interests for the same Data, only one of them will actually reach the authoritative name server.

Another high-level result from the present work is a case-based proof that build-in security of NDN architecture can unify and simplify design of application, such as NDNS. In particular, by the means of cryptography and name-based security policies NDNS effectively implement DNSSEC-like security model within the network architecture, without requiring any additional protocol-specific mechanisms.

## References

[AB12]     RJ Atkinson and SN Bhatti. "Identifier-Locator Network Protocol (ILNP) Architectural Description." RFC 6740, 2012.

[ABH09]    Randall Atkinson, Saleem Bhatti, and Stephen Hailes. "ILNP: mobility, multi-homing, localised addressing and security through naming." *Telecommunication Systems*, **42**(3), 2009.

[AL03]     Carlisle Adams and Steve Lloyd. *Understanding PKI: concepts, standards, and deployment considerations*. Addison-Wesley Professional, 2003.

[AMM13]    Alexander Afanasyev, Priya Mahadevan, Ilya Moiseenko, Ersin Uzun, and Lixia Zhang. "Interest Flooding Attack and Countermeasures in Named Data Networking." In *Proceedings of IFIP Networking*, May 2013.

[AMZ12]    Alexander Afanasyev, Ilya Moiseenko, and Lixia Zhang. "ndnSIM: NDN simulator for NS-3." Technical Report NDN-0005, NDN, October 2012.

[Arb11]    Arbor networks. "Worldwide Infrastructure Security Report." Volume VII, `http://www.arbornetworks.com/research/infrastructure-security-report`, 2011.

[ARI13]    ARIN. "RPKI frequently asked questions." `https://www.arin.net/resources/rpki/faq.html`, 2013. Accessed on September 6, 2013.

[ASB99]    W. Adjie-Winoto, E. Schwartz, H. Balakrishnan, and J. Lilley. "The design and implementation of an intentional naming system." In *SIGOPS Operating Systems Review*, volume 33, 1999.

[ATL10]    Alexander Afanasyev, Neil Tilley, Brent Longstaff, and Lixia Zhang. "BGP Routing Table: Trends and Challenges." In *Proceedings of the 12th Youth Technological Conference "High Technologies and Intellectual Systems"*, 2010.

[AZY13]    Alexander Afanasyev, Zhenkai Zhu, and Yingdi Yu. "NDN.cxx: C++ NDN API." `https://github.com/named-data/ndn.cxx/`, 2013.

[AZZ13]    Alexander Afanasyev, Zhenkai Zhu, and Lixia Zhang. "The story of ChronoShare, or how NDN brought distributed file sharing back." Under submission, 2013.

[BPK10]    Mariana Boguna, Fragkiskos Papadopoulos, and Dmitri Krioukov. "Sustaining the Internet with hyperbolic mapping." *Nature Communications*, **1**(62), 2010.

[BVR12]    Akash Baid, Tam Vu, and Dipankar Raychaudhuri. "Comparing Alternative Approaches for Networking of Named Objects in the Future Internet." In *Proceedings of NOMEN Workshop*, 2012.

[BZA13]    Chaoyi Bian, Zhenkai Zhu, Alexander Afanasyev, Ersin Uzun, and Lixia Zhang. "Deploying Key Management on NDN Testbed." Technical Report NDN-0009, Revision 2, NDN, 2013.

[CCK06]    Matthew Caesar, Tyson Condie, Jayanthkumar Kannan, Karthik Lakshminarayanan, and Ion Stoica. "ROFL: routing on flat labels." In *Proceedings of SIGCOMM*, 2006.

[CCN06]    Matthew Caesar, Miguel Castro, Edmund B. Nightingale, Greg O'Shea, and Antony Rowstron. "Virtual ring routing: network routing inspired by DHTs." In *Proceedings of SIGCOMM*, 2006.

[CG00]     David R. Cheriton and Mark Gritter. "TRIAD: A new next-generation Internet architecture." Technical report, TRIAD project, 2000.

[CK13]     S. Cheshire and M. Krochmal. "Multicast DNS." RFC 6762, 2013.

[CRW04]    Antonio Carzaniga, Matthew J. Rutherford, and Alexander L. Wolf. "A Routing Scheme for Content-Based Networking." In *Proceedings of IEEE INFOCOM*, 2004.

[CSF08]    D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile." RFC 5280, 2008.

[Dee96]    Steve Deering. "The Map & Encap Scheme for scalable IPv4 routing with portable site prefixes." *Presentation Xerox PARC*, 1996.

[DET00]    DETER project. "DeterLab: Cyber-Security Experimentation and Testing Facility." http://www.isi.deterlab.net, 2000.

[Eas97]    D. Eastlake. "Secure Domain Name System Dynamic Update." RFC 2137, 1997.

[Eas99]    D. Eastlake. "Domain Name System Security Extensions." RFC 2535, March 1999.

[EBJ10]    Lixia Zhang Deborah Estrin, Jeffrey Burke, Van Jacobson, James D. Thornton, Diana K. Smetters, Beichuan Zhang, Gene Tsudik, kc claffy,

Dmitri Krioukov, Dan Massey, Christos Papadopoulos, Tarek Abdelzaher, Lan Wang, Patrick Crowley, and Edmund Yeh. "Named Data Networking (NDN) Project." Tech.Report NDN-0001, PARC, October 2010.

[Far07]     D. Farinacci. "Locator/ID separation protocol (LISP)." Internet draft (draft-farinacci-lisp-00), 2007.

[FHC03]     P. Faltstrom, P. Hoffman, and A. Costello. "Internationalizing Domain Names in Applications (IDNA)." rfc3490, 2003.

[GKR11]     Ali Ghodsi, Teemu Koponen, Jarno Rajahalme, Pasi Sarolahti, and Scott Shenker. "Naming in content-oriented architectures." In *Proceedings of SIGCOMM Workshop on ICN*, 2011.

[GMG12]     Joseph Gersch, Dan Massey, Michael Glenn, and Christopher Garner. "ROVER BGP Route Origin Verification via DNS." NANOG55, 2012.

[GMO13]     J. Gersch, D. Massey, C. Olschanowsky, and L. Zhang. "DNS Resource Records for Authorized Routing Information." Internet draft (draft-gersch-grow-revdns-bgp-02), 2013.

[Goo13]     Google. "Webmaster Tools. Verification: Domain name provider." `https://support.google.com/webmasters/answer/176792?hl=en`, 2013. Accessed on September 6, 2013.

[GR00]     Lixin Gao and Jennifer Rexford. "Stable Internet routing without global coordination." *SIGMETRICS Perform. Eval. Rev.*, **28**(1):307–317, June 2000.

[HAA13]     AKM Hoque, S. O. Amin, A. Alyyan, B. Zhang, L. Zhang, and L. Wang. "Named-data Link State Routing Protocol." In *Proceedings of the ACM SIGCOMM ICN Workshop*, 2013.

[HS12]     P. Hoffman and J. Schlyter. "The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA." RFC 6698, 2012.

[Hus05]     Geoff Huston. "Growth of the BGP table, 1994 to present." `http://bgp.potaroo.net`, 2005. Accessed on August 30, 2013.

[HWB08]     Mark Handley, Damon Wischik, and Marcelo Bagnulo Braun. "Multipath Transport, Resource Pooling, and implications for Routing." Presentation at IETF-71, July 2008.

[IAN13]     IANA. "Domain Name System (DNS) Parameters." `http://www.iana.org/assignments/dns-parameters/dns-parameters.xhtml`, 2013.

[IV09]        ICANA and Verisign Inc. "Root DNSSEC: Information about DNSSEC for the Root Zone." `http://www.root-dnssec.org/`, 2009. Accessed on August 30, 2013.

[JMM08]     Dan Jen, Michael Meisel, Daniel Massey, Lan Wang, Beichuan Zhang, and Lixia Zhang. "APT: A Practical Tunneling Architecture for Routing Scalability." Technical Report 080004, UCLA Computer Science Department, 2008.

[JMY08]     Dan Jen, Michael Meisel, He Yan, Dan Massey, Lan Wang, Beichuan Zhang, and Lixia Zhang. "Towards a New Internet Routing Architecture: Arguments for Separating Edges from Transit Core." In *Proceedings of HotNets*, 2008.

[JST09]      Van Jacobson, Diana K. Smetters, James D. Thornton, Michael F. Plass, Nicholas H. Briggs, and Rebecca L. Braynard. "Networking Named Content." In *Proceedings of CoNEXT*, 2009.

[KB11]       Derek Kulinski and Jeff Burke. "Python wrapper for NDNx." `https://github.com/named-data/PyNDN`, 2011.

[KB12]       Derek Kulinski and Jeff Burke. "NDNVideo: Random-access Live and Pre-recorded Streaming using NDN." Technical Report NDN-0007, NDN, 2012.

[KBZ13]     Derek Kulinski, Jeff Burke, and Lixia Zhang. "Video Streaming over Named Data Networking." *IEEE COMSOC MMTC E-Letter*, **8**(4), July 2013.

[KCC07]     Teemu Koponen, Mohit Chawla, Byung-Gon Chun, Andrey Ermolinskiy, Kye Hyun Kim, Scott Shenker, and Ion Stoica. "A Data-Oriented (and Beyond) Network Architecture." In *Proceedings of SIGCOMM*, 2007.

[KK89]       Tomihisa Kamada and Satoru Kawai. "An algorithm for drawing general undirected graphs." *Information processing letters*, **31**(1):7–15, 1989.

[KLM00]    S. Kent, C. Lynn, J. Mikkelson, and K. Seo. "Secure Border Gateway Protocol (S-BGP)—Real World Performance and Deployment Issues." In *Proceedings of the Network and Distributed System Security Symposium (NDSS 2000)*, 2000.

[KLS00]      S. Kent, C. Lynn, and K. Seo. "Secure Border Gateway Protocol (S-BGP)." *IEEE Journal on Selected Areas in Communications*, **18**(4):582–592, 2000.

[KPK10]   Dmitri Krioukov, Fragkiskos Papadopoulos, Maksim Kitsak, Amin Vahdat, and Mariana Boguna. "Hyperbolic geometry of complex networks." *Physical Review E*, **82**, 2010.

[Lac10]   Mathieu Lacage. *Experimentation tools for networking research*. Ph.d. thesis, Ecole doctorale Stic, Université de Nice Sophia Antipolis, 2010.

[MAM99]   M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams. "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol—OCSP." RFC 2560, 1999.

[MG11]    Luca Muscariello and Massimo Gallo. "Content Centric Networking Packet Level Simulator." `https://code.google.com/p/ccnpl-sim/`, 2011.

[Moc87a]  P. Mockapetris. "Domain names—concepts and facilities." RFC 1034, 1987.

[Moc87b]  Paul Mockapetris. "Domain names—implementation and specification." RFC 1035, 1987.

[MV13]    Giulia Mauri and Giacomo Verticale. "Distributing Key Revocation Status in Named Data Networking." In *Advances in Communication Networking*, pp. 310–313. Springer, 2013.

[MVK12]   T. Manderson, L. Vegoda, and S. Kent. "Resource Public Key Infrastructure (RPKI) Objects Issued by IANA." RFC 6491, 2012.

[MWZ07]   Daniel Massey, Lan Wang, Beichuan Zhang, and Lixia Zhang. "A Scalable Routing System Design for Future Internet." In *Proceedings of SIGCOMM IPv6 and the Future of the Internet workshop*, 2007.

[MZF07]   D. Meyer, L. Zhang, and K. Fall. "Report from the IAB Workshop on Routing and Addressing." RFC 4984, 2007.

[NB07]    E. Nordmark and M. Bagnulo. "Shim6: Level 3 Multihoming Shim Protocol for IPv6." Internet draft (draft-ietf-shim6-proto-09), 2007.

[NDN13a]  NDN Project. "NDN Platform." `http://named-data.net/codebase/platform/`, 2013.

[NDN13b]  NDN Project. "NDNx Technical Documentation." `http://named-data.net/doc/0.1/technical/`, 2013.

[NM09]    M. Nicholes and B. Mukherjee. "A survey of security techniques for the border gateway protocol (BGP)." *IEEE Communications Surveys and Tutorials*, **11**(1):52–65, 2009.

[NO11]    Ashok Narayanan and David Oran. "NDN and IP routing: Can it scale?" Proposed Information-Centric Networking Research Group (ICNRG), Side meeting at IETF-82, November 2011.

[ns 11]    ns-3. "Discrete-event network simulator for Internet systems." `http://www.nsnam.org/`, 2011.

[ns 13]    ns-3. "NS3 DCE CCNx Quick Start." `http://www.nsnam.org/overview/projects/direct-code-execution/`, 2013.

[NSS10]    Erik Nygren, Ramesh K Sitaraman, and Jennifer Sun. "The Akamai network: a platform for high-performance internet applications." *ACM SIGOPS Operating Systems Review*, **44**(3), 2010.

[OD96]    Mike O'Dell. "8+8—An alternate addressing architecture for IPv6." Internet draft (draft-odell-8+8-00), 1996.

[OZZ07]    Ricardo Oliveira, Beichuan Zhang, and Lixia Zhang. "Observing the Evolution of Internet AS Topology." In *Proceedings of SIGCOMM*, 2007.

[Per96]    C. Perkins. "IP Encapsulation within IP." RFC 2003, 1996.

[PKB10]    Fragkiskos Papadopoulos, Dmitri Krioukov, Mariana Boguna, and Amin Vahdat. "Greedy Forwarding in Dynamic Scale-Free Networks Embedded in Hyperbolic Metric Spaces." In *Proceedings of IEEE IN-FOCOM*, 2010.

[RIP09]    RIPE NCC. "IPv6 Address Allocation and Assignment Policy." `http://www.ripe.net/ripe/docs/ripe-481#_8._IPv6_Provider`, 2009.

[RL06]    Y. Rekhter, T. Li, , and S. Hares. "A Border Gateway Protocol 4 (BGP-4)." RFC 4271, 2006.

[Rob11]    Paul Roberts. "Phony SSL Certificates issued for Google, Yahoo, Skype, Others." `http://threatpost.com/phony-ssl-certificates-issued-google-yahoo-skype-others-032311/75061`, March 2011.

[RR11]    Dario Rossi and Giuseppe Rossini. "Caching performance of content centric networksunder multi-path routing (and more)." Technical report, Telecom ParisTech, 2011.

[Sha13]    Wentao Shang. "JavaScript version of NDNS query API." `http://github.com/wentaoshang/ndns-client-js`, 2013.

[SJ09]    Diana K. Smetters and Van Jacobson. "Securing Network Content." Technical report, PARC, 2009.

[SMW02]   Neil Spring, Ratul Mahajan, and David Wetherall. "Measuring ISP topologies with Rocketfuel." In *Proceedings of SIGCOMM*, 2002.

[STC13]   Wentao Shang, Jeff Thompson, Meki Cherkaoui, Jeff Burke, and Lixia Zhang. "NDN.JS: A JavaScript Client Library for Named Data Networking." In *Proceedings of IEEE INFOCOMM 2013 NOMEN Workshop*, April 2013.

[Sti02]   Stichting NLnet. "Bind DLZ: Dynamically Loadable Zones." `http://bind-dlz.sourceforge.net/`, 2002.

[TAV09]   Sasu Tarkoma, Mark Ain, and Kari Visala. "The Publish/Subscribe Internet Routing Paradigm (PSIRP): Designing the Future Internet Architecture." *Towards the Future Internet*, 2009.

[Tsu88]   Paul F. Tsuchiya. "The landmark hierarchy: a new hierarchy for routing in very large networks." In *Proceedings of SIGCOMM*, 1988.

[Tsu91]   Paul F. Tsuchiya. "Efficient and robust policy routing using multiple hierarchical addresses." In *Proceedings of SIGCOMM*, 1991.

[TZ01]   Mikkel Thorup and Uri Zwick. "Compact routing schemes." In *Proceedings of the 13th annual ACM symposium on Parallel algorithms and architectures (SPAA)*, 2001.

[Ver12]   Verisign Inc. "The VeriSign domain report." The domain name industry brief, `http://www.verisigninc.com/assets/domain-name-brief-dec2012.pdf`, 2012.

[VTR97]   P. Vixie, S. Thomson, Y. Rekhter, and J. Bound. "Dynamic Updates in the Domain Name System (DNS UPDATE)." RFC 2136, 1997.

[WBW13]   Sen Wang, Jun Bi, and Jianping Wu. "Collaborative Caching Based on Hash-Routing for Information-Centric Networking." SIGCOMM Poster, 2013.

[WHY12]   Lan Wang, A K M Mahmudul Hoque, Cheng Yi, Adam Alyyan, and Beichuan Zhang. "OSPFN: An OSPF Based Routing Protocol for Named Data Networking." Technical Report NDN-0003, NDN, 2012.

[YAM13]   Cheng Yi, Alexander Afanasyev, Ilya Moiseenko, Lan Wang, Beichuan Zhang, and Lixia Zhang. "A Case for Stateful Forwarding Plane." *Computer Communications*, **36**(7):779–791, 2013.

[YAW12]   Cheng Yi, Alexander Afanasyev, Lan Wang, Beichuan Zhang, and Lixia Zhang. "Adaptive Forwarding in Named Data Networking." *ACM Computer Communication Reviews*, **42**(3):62–67, July 2012.

[Yu13]      Yingdi Yu. "NDN Security Library." `http://irl.cs.ucla.edu/`
            `~yingdi/web/pub/Trust-Management-Library-v4.pdf`, 2013.

[YWL12]     Yingdi Yu, Duane Wessels, Matt Larson, and Lixia Zhang. "Author-
            ity Server Selection of DNS Caching Resolvers." *ACM SIGCOMM
            Computer Communication Reviews*, April 2012.

[ZA13]      Zhenkai Zhu and Alexander Afanasyev. "Let's ChronoSync: Decen-
            tralized Dataset State Synchronization in Named Data Networking."
            In *Proceedings of the 21st IEEE International Conference on Network
            Protocols (ICNP 2013)*, 2013.

[ZEB11]     Lixia Zhang, Deborah Estrin, Jeffrey Burke, Van Jacobson, James D.
            Thornton, Ersin Uzun, Beichuan Zhang, Gene Tsudik, kc claffy, Dmitri
            Krioukov, Dan Massey, Christos Papadopoulos, Tarek Abdelzaher,
            Lan Wang, Patrick Crowley, and Edmund Yeh. "Named Data Net-
            working (NDN) Project 2010 - 2011 Progress Summary." Technical
            report, PARC, http://www.named-data.net/ndn-ar2011.html, Novem-
            ber 2011.

[Zhu13]     Zhenkai Zhu. *Support Mobile and Distributed Applications with Named
            Data Networking.* PhD thesis, UCLA, June 2013.

[ZWC11]     Zhenkai Zhu, Ryuji Wakikawa, Stuart Cheshire, and Lixia Zhang.
            "Home as you go: an engineering approach to mobility-capable ex-
            tended home networking." In *Proceedings of Asian Internet Engineer-
            ing Conference (AINTEC)*, 2011.