NDNS: A DNS-Like Name Service for NDN

Alexander Afanasyev,* Xiaoke Jiang,† Yingdi Yu,* Jiewen Tan,* Yumin Xia,* Allison Mankin,‡ and Lixia Zhang*

*University of California, Los Angeles
Email: aa,yingdi,lixia@cs.ucla.edu
†Tsinghua University, China
‡Salesforce

***Abstract*—DNS provides a global-scale distributed lookup service to retrieve information of all types for a given name, be it IP addresses, service records, or cryptographic keys. The DNS service has proven essential in today's operational Internet. Our experience with the design and development of Named Data Networking (NDN) also suggests the need for a similar always-on lookup service. To fulfill this need we have designed the NDNS (NDN DNS) protocol, and learned several interesting lessons through the process. Although DNS's request-response operations seem closely resembling NDN's Interest-Data packet exchanges, they operate at different layers in the protocol stack. Comparing DNS's implementations over IP protocol stack with NDNS's implementation over NDN reveals several fundamental differences between applications designs for host-centric IP architecture and data-centric NDN architecture.**

***Index Terms*—Named Data Networking; DNS; Information-centric Networking; Future Internet Architecture**

I. Introduction

Named-Data Networking (NDN) [1], [2], [3] is a proposed Internet architecture, where data consumers send *Interest packets* to fetch desired *Data packets*. Our NDN design and development efforts over the last few years identified several needs for a DNS-like lookup service, e.g., to support NDN routing scalability [4], to provide rendezvous for mobile data producers [5], and to provide persistent storage of critical data, such as public key certificates [6].

To meet the above needs, we have designed NDNS (NDN DNS), a distributed, always-on distributed lookup service. The NDNS design leverages the experiences learned from the Domain Name Systems (DNS) and its security extensions (DNSSEC) [7], [8], [9], [10], [11], but also addresses the fundamental differences in the design requirements of NDN- and IP-based applications. As we discuss in Section III, NDNS and DNS(SEC) share the commonalities of retrieving secured data from hierarchically structured namespaces. However, using data names directly at the network level and securing network layer packets leads to fundamentally different implementations and different tradeoffs.

The contributions of this paper are threefold. First, we explain why the NDN architecture needs a lookup service, given it directly uses names to retrieve data at the network layer (Section II). Second, we design the NDNS protocol that provides DNS-like lookup service with enriched functionality using NDN Interest/Data primitives (Section IV). Third, we identify the fundamental differences between IP and NDN's network protocol architectures to explain why naming endpoints versus naming data results in the sharp differences between DNS and NDNS designs. We hope that our experience from the design and implementation of NDNS protocol provides insights and guidelines in adapting other IP-based application protocols to the NDN architecture.

II. The Necessity of a Lookup Service

This section presents a brief background of NDN and discuss the necessity of a lookup service in NDN. At the end of this section, we analyze the commonalities and differences between NDN and DNS.

A. NDN

Named Data Networking (NDN) [1], [2], [3] makes named and secured application *Data* packets the centerpiece of the network architecture—the "thin waist" of communication. To request Data, consumers send out *Interest* packets that include names or prefixes of what should be fetched. These names are then used by the NDN network forwarders to steer Interests towards available (closest) Data replicas and return them back to the requesters.

1) Name-based Forwarding: NDN's Interest-Data packet exchanges are *network-layer* operations, and network-layer's basic function is to make the forward decision for each packet by name, e.g., "`/net/ndnsim/download/pkg/_v=2`". Each NDN router maintains a Forwarding Information Base (FIB), which contains logically a table of (name prefix, interface list) tuples. Upon receiving an Interest packet, an NDN router uses *forwarding strategy* [12] to forward this request to one or more interfaces, using information from the matching FIB entry, previously recorded data plane performance for the data's prefix, and potentially other policy restrictions. The strategy decision is then recorded in the *Pending Interest Table* (PIT). Each Data packet is returned to the requester by utilizing the state recorded in routers' PIT, following the reverse path of the Interest.

2) Per-Packet Authentication: NDN mandates Data producers to create digital signature on their Data packets at the time of creation. The cryptographic signature binds the name of Data with its content [13], so that the content authenticity can be verified independent from where a Data packet comes from. As a result, each NDN Data packet is immutable, any change to the carried content requires creation of a new Data packet with different (e.g., versioned) name and a new signature.

The signature field in each NDN Data packet includes a key locator field which names the signing key to be used for the signature verification. A key in NDN is simply another Data packet containing the public key bits and the corresponding signature [13], which in turn contains its own key locator. These key locators form a certification chain up to a trust anchor key that a consumer must have obtained and trust a priori.

3) Efficient Data Delivery: Naming and securing data at the network layer brings a number of important properties in data distribution. Since a Data packet is inherently secured and identified by the name at the network layer, NDN routers can cache Data packets (in-network caching). Multiple simultaneous requests for the same Data can be aggregated, providing built-in multicast Data delivery. The intelligent forwarding plane of NDN can forward requests along the best path based on realtime measurement.

B. Operational Needs of NDN

Our experience with NDN design and development since 2010 has identified three primary needs for an always-on lookup service: to scale network routing, to support mobility of Data producers, and to provide online persistent storage for keys when their owners go offline.

1) Routing Scalability: NDN routers forward Interest packets by names. However, the limited size of FIB implies that, in large-scale environments, only a small number of name prefixes can be stored in FIB. Therefore, NDN needs a way to map all application names that cannot retrieve data directly over the global Internet into names that can, i.e., that are present in FIBs and disseminated by the global routing protocols.

In a proposed secure namespace mapping solution [4], when necessary, content producers create *forwarding hints* to map their application data name prefixes to sets of "reachable" names. These hints [13] can be then attached to Interest packets to guide forwarding of the Interests towards a network region (or a node) where data can be found.

2) Mobile Publishing: While NDN's stateful forwarding provides a natural support for data retrieval by a mobile consumer, it is still a challenge to steer Interests towards Data when producers move. Several alternative approaches has been proposed to address this problem [5], including the use indirection based on the mapping service, inspired the existing mobile-IP solutions. In particular, instead of directly announcing its reachability through routing protocols, a producer can update its forwarding hint(s) in the lookup service. To retrieve data, consumers will lookup these sets and attach them to Interests, so that routers know where to find the requested Data.

3) Certificate Provisioning: The data-centric nature of NDN secures Data directly and enables asynchrony between Data production and consumption: the original Data producer may not be online when a piece of content is retrieved and needs to be verified. The always-on lookup service would make the public key certificates available to all consumers at all times, waiving the requirement of producer provisioning its own certificates. This service can be a default public key storage to facilitate Data verification and trust management, but does not restrict NDN applications from retrieving certificates and choose their trust model by other means.

III. NDN AND DNS

In this section, we present a brief summary of DNS [7], [8] and identify its commonality with NDN on "fetching Data" semantics and the differences from operating at application layer versus network layer, respectively.

A. DNS

DNS defines a globally unique tree-like hierarchical namespace used by all Internet applications. Each node or DNS domain of the hierarchical tree can be associated with several types of resource records (RR) under that name, grouped into RR sets. DNS namespace is managed in a distributed way by splitting it into so called *DNS zones*, which are responsible for hosting RR sets, including records for sub-domain delegation to child zones. Zones are usually hosted on multiple *name servers* that respond to relevant DNS queries.

Any RR set stored in DNS can be retrieved using a top-down approach (using so-called *iterative queries*), starting from the pre-configured name servers for the root zone. For example, to discover an IPv4 address of "`www.ndnsim.net`", a resolver would send "what is the "`A`" record of "`www.ndnsim.net`"?" query to one of the root zone DNS servers. Depending on the available information, each server responds either with the requested RR set or a referral to a child zone by returning a set of "`NS`" and "glue" "`A`" (and/or "`AAAA`" for IPv6) records. In the example, the root zone server would refer to a set of "`net`" nameservers, "`net`" nameservers would refer to a set of "`ndnsim.net`" nameservers, who will finally return the desired information. The servers can also return "`NSEC`" (or "`NSEC3`") records to indicate that the records does not exist.

To scale DNS resolution, in addition to replicating authoritative DNS servers for each zone, DNS also relies on *caching resolvers* between end hosts (which run *stub resolvers*) and name servers. The caching resolvers, as a response to *recursive queries* from stub resolvers, run the iterative querying on behalf of the end hosts and caches the responses that can be later reused to avoid repetitive iterative queries within a period defined by an RR set.

Over the years, DNS has significantly expanded along two dimensions: the types of RRs and the security of RRs. The first one is due to the fact that DNS is arguably the largest and most available distributed database on the Internet, therefore it has been used as a common lookup service of many other types of data not related to name resolution. At the time of this writing, IANA has recorded more than 80 RR types [14], including different types of host addresses, mail exchange information, server selection records, cryptographic keys, signatures, and many other types of information. The security extension

TABLE I
DATA SECURITY IN DNSSEC AND NDN

	DNSSEC	NDN
Security granularity	RR set	Data packet
Signature carrier	RRSIG RR set	Embedded in data packet
Key identifier	Key tag	Embedded in data packet (Key locator)
Key storage	DNSKEY record	Data packet
Security delegation	DS record in the parent zone	DKEY in the parent zone

(DNSSEC) [9], [10], [11] added authenticity and integrity checking of DNS replies through the use of digital signatures. As a self-contained design, DNSSEC introduced several new types of DNS RRs including RRSIG (for RRset signature) and DNSKEY (for public keys that can verify digital signatures). Each DNSKEY can be verified using the public key of its parent zone.[1] The overall DNSSEC trust is bootstrapped from a single pre-configured DNS root key.

B. Commonality and Differences between NDN and DNS

While both NDN and DNS fetch data using hierarchically structured names, DNS that operates at the application layer and NDN builds the data fetching semantics directly into the network layer. As a result, a DNS resolver has to explicitly select the name servers to fetch data [15] and NDN consumers just request data, unaware of where this data may be retrieved.

The name-based data retrieval allows both NDN and DNS to benefit from caching. However again, DNS has to implement an application-level caching by introducing caching resolvers into the system, while NDN can support caching (and any other kind of storage) at the network layer reducing complexity and improving scalability of applications.

DNS (with DNSSEC) secures data by attaching each RR set with a digital signature, which resolvers can validate through a strict trust model following the domain hierarchy. NDN, on the other hand, mandates per-packet signatures that can be validated by consumers or any network component that is aware of the trust model associated with data. Notably, many trust models in NDN, including the DNSSEC-like model described in Section IV-F, can be defined in terms of relationships between data names and signing key names using trust schemas [16] Table I highlights several other important differences between the data-centric security of DNSSEC and NDN.

IV. NDNS DESIGN

The design of NDNS inherits several basic concepts from DNS, including domains, zones, resource records (RR), name servers, caching and stub resolvers, and iterative and recursive queries. NDNS lookup also takes the top-down search, and relies on replication and caching to scale.

The architectural differences between IP and NDN dictate that the NDNS design takes a different form, most notably its iterative query mechanism. Because an NDN Interest packet is forwarded based on the name N carried in the Interest; to steer the Interest towards NDNS servers of a specific zone, the zone's name must be included in N's prefix. At the same time, since the returning data packet D goes back to its requester by following the PIT states left by the Interest packet, D's name also must contain N at least as a prefix. This means that a NDNS name server can only either return the exact answer to the request, either using pre-created Data packet or creating it on the fly. Moreover, because names are used directly at the network level, two Interests for the same name N most likely to bring the same data. In other words, once an Interest for name N succeeds in retrieving a Data packet, subsequent Interests for N may get the same data packet from network caches, even though there could be another Data packet with N as a prefix but a different suffix.

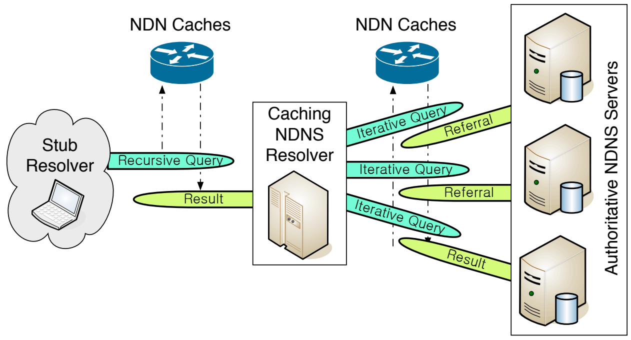

Fig. 1. NDNS operation overview, where queries carry only the names to be queried, and "NDN caches" represent caches at any network forwarders. Queries are forwarded based on their names, and can bring back matching answers from any intermediate caches.

Due to routing scalability concerns, we assume that only name servers of the top-level and popular second-level domains have "reachable" names. Retrieving information from NDNS servers of less popular domains would need to rely on the forwarding hint mechanism [4].

A. NDNS Design Overview

Figure 1 shows an overview picture of NDNS operations, highlighting resemblance to DNS/DNSSEC with several notable differences. The query process starts with stub resolvers sending recursive queries *towards* local NDNS caching resolver(s); we use the word "towards" instead of "to" here, because the query, which is an Interest packet, may bring back the requested answer from a router's cache before it reaches the local caching resolver. If the query reaches a caching resolver, the resolver either finds the answer from its internal caches, or otherwise issues a set of top-down iterative queries to find the answer.

All NDNS queries are represented as Interest packets that carry the names to be resolved, without specifying from which exact server to retrieve the data. Stub resolvers send queries under "`/NDNS-R`" prefix, which is a naming convention for "NDNS caching resolver service", and is announced to the routing system by the caching resolver instances.

[1] A zone usually has multiple keys for scalability and security reasons, including key signing key (KSK) and one or more zone signing keys (ZSKs).

As we explained at the beginning of this section, to retrieve data from a specific NDNS zone, Interest packets need to carry the name of that zone. Thus, a NDNS caching resolver has to split an incoming recursive query into a set of specific questions/Interests about the zone referrals and questions for data (see Section IV-B). More specifically, the questions have to start with determining whether the top-level zone has been delegated, then continue with whether the next-level zone has been delegated, and so on. This process continues until either all the name components in the original query are exhausted or it is determined that the sub-zone is no longer delegated or does not exist. In the first two cases, the resolver sends out a new query asking for the actual data requested by the stub resolver, requesting this information from the last discovered "authority" zone.

To allow data retrieval from specific zones, NDNS defines the following naming conventions for the iterative query Interests: "/<zone-name>/NDNS/<label(s)>/<record-type>". For example, Interest for "/NDNS/net/NS" requests "NS" records for "net" label from the root zone, while Interest for "/net/ndnsim/NDNS/foo/bar/TXT" requests "TXT" records for "foo/bar" label from "/net/ndnsim" zone.

The bootstrapping of the NDNS infrastructure can be accomplished by simply announcing the root zone prefix ("/NDNS"), top-level zone prefixes ("/net/NDNS", "/com/NDNS", etc.), and any popular second-level zone prefixes ("/com/google/NDNS", "/com/cnn/NDNS") into the routing system. Therefore, the caching resolvers do not need to be pre- or re-configured with information about NDNS servers. All they need to do is to generate query names by following proper naming conventions; the network can then take care of forwarding the query Interests towards the right data.

In cases where zone (e.g., "/net/ndnsim/NDNS") is not a directly "reachable" name, the parent zone's referral response (as part of "NS" records) must contain a forwarding hint, so that the query Interests can find the data.

By running on top of NDN, NDNS takes advantages of all built-in NDN features, including (1) efficient query Interest forwarding taking into account server and network availability information at network layer; (2) aggregation of same queries and multicasting query results; (3) in-network caching of query results; and (4) built-in authentication for query results carried in NDN data packets.

A comparison between DNS and NDNS is presented in Table II.

B. NDNS Namespace and Naming

Borrowing the DNS concept of *zones*, NDNS also uses zones as units of administrative management of namespaces. Different from DNS, NDNS uses a set of naming conventions to steer NDNS query interest towards the servers of specific zones (Figure 2). NDNS zones have names that are constructed by appending "NDNS" component to the corresponding zone namespace ("/NDNS" for root namespace zone or just root zone, "/net/NDNS" for "/net" namespace zone, etc.).

TABLE II
COMPARISON OF IMPLEMENTATION IN DNS AND NDNS

	DNS	**NDNS**
Namespace	DNS namespace	Reflection of NDN namespace
Scalability and availability	Replication and application-level caching	Replication, in-network caching, application-layer caching
Robustness	Best name server selection by caching resolvers	Retrieval best answer from the NDN network
Security	DNSSEC extension (application level)	Built-in NDN (network level)
Data format	Resource record set	Data packet (Section IV-C)
Iterative lookup	From root to leaves, implicit	From root to leaves, explicit

Each zone publishes data its own namespace in the form of data packets, which represent specific resource records in the zone. The zone data is served by one or more authoritative NDNS name servers that synchronize their records using, for example, the ChronoSync protocol [17].

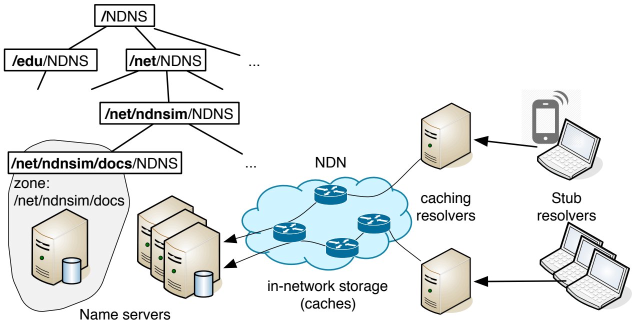

Fig. 2. NDNS elements: tree-organized zones, name servers, caching resolvers and stub resolvers.

Any NDN name (or in DNS terms, any domain name) belongs to exactly one NDNS zone that manages information associated with that name. For example in Figure 2, "/net/ndnsim/www" domain belongs to the "/net/ndnsim/NDNS" zone, while "/net/ndnsim/docs/..." domains are managed by the "/net/ndnsim/docs/NDNS" zone, i.e., the longest match zone name takes precedence.

NDNS resource records, or RRs for short, are *typed* data associated with the domain name that are retrieved during the iterative NDNS query process. The name of NDNS RR consists of the zone name, label, data type, and auxiliary parts such as version and segment components (Figure 3).[2] By design, the zone name without "NDNS" component concatenated with the label part of the RR name equals to the domain name this record is associated with. In a way, NDNS RR is a metadata associated with the specific NDN name that is stored in a specific place inside NDNS infrastructure.

[2] Version and segment number fields are optional in interest names, by default the server returns data with latest version.

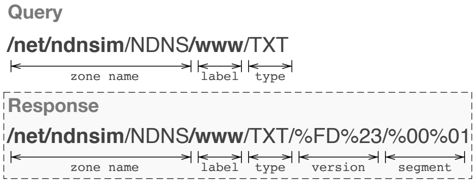

Fig. 3. NDNS query/response naming example

The type component in the RR name is an application-defined data blob. Currently, NDNS reserves "NS" type to store (potentially empty) sets of forwarding hints to indicate delegation of the subzones, "CERT" type to store NDNS public key certificates, "APPCERT" to store applications certificates as encapsulated data, and "TXT" type to store a collection of free-formed text records. We expect applications to introduce new types of records over time, as what happened to DNS RRs. While applications are free to select any name for the custom record type, in near future we plan to establish a formal process to to ensure uniqueness and promote standardized naming conventions and operations.

NDNS defines additional naming conventions for the recursive queries, with an objective to reach the closest instance of the caching resolver (Figure 4). More specifically, all names carried in recursive queries start with prefix "NDNS-R" that is disseminated by the routing protocols within local networks. Following the "/NDNS-R" prefix, the query includes the domain name together with the requested record type. In cases where clients are willing to use caching resolvers provided by a different provider (e.g., Google): users can either use a dedicated prefix (e.g., "/com/google/NDNS-R") or rely on forwarding hints to steer Interests towards that provider, bypassing the local caching resolvers.

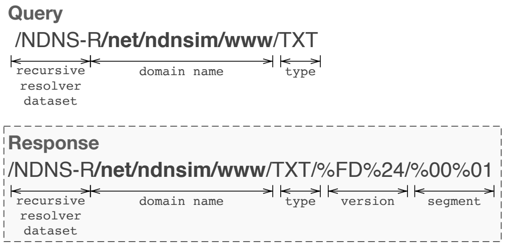
Query
/NDNS-R/net/ndnsim/www/TXT
recursive resolver dataset
domain name
type
Response
/NDNS-R/net/ndnsim/www/TXT/%FD%24/%00%01
recursive resolver dataset
domain name
type
version
segment

Fig. 4. Recursive query naming example

C. Data Format

NDN retrieves data in the unit of packets. Each NDNS data packet contains enough information to indicate with which NDN name it is associated, to which zone the record belongs, and what the record type is (Figure 5).

Interpretation of the carried payload in the NDNS data packet depends on the value of ContentType field. In particular, NDNS distinguishes the following semantic interpretations:

- ContentType "NACK" indicates that the requested record does not exist. A NACK can optionally include an error code as part of its content.
- ContentType "NDNS_AUTH" indicates that the requested record does not exist, but the child zone is delegated. "NDNS_AUTH" only applies for "NS" records and has no defined meaning for other record types. See a more detailed description why NDNS needs this in Section IV-D.
- Any other value of the ContentType field is interpreted as a positive response for the iterative query. The interpretation of the payload is application-defined and may or may not rely on ContentType value.

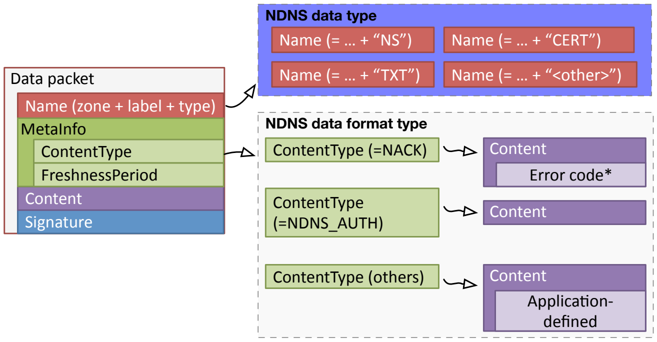

Fig. 5. NDNS data packet format

```
NsContent ::= CONTENT−TYPE TLV−LENGTH
              ForwardingHint∗

ForwardingHint ::= FORWARDING−HINT−TYPE(=31) TLV−LENGTH
                   Preference
                     Name

Preference ::= LINK−PREFERENCE−TYPE(=30) TLV−LENGTH
               nonNegativeInteger
```

* When NsContent includes zero hints, the child zone is delegated and its data can be retrieved directly, i.e., without the need to attach forwarding hints to the Interest.

Fig. 6. Payload definition for "NS" type.

```
KeyContent ::= CONTENT−TYPE TLV−LENGTH
               Data(=Application's NDN certificate)+
```

Fig. 7. Payload definition for "KEY" type

```
TxtContent ::= CONTENT−TYPE TLV−LENGTH
               RrData+

RrData ::= RR−DATA−TYPE(=191) TLV−LENGTH
           BYTE∗
```

Fig. 8. Payload definition for "TXT" type

The payload for the reserved "NS", "KEY", and "TXT" types is defined in Figure 6, 7, and 8. Refer to NDN packet specification [13] for additional information. The "CERT" type is interpreted as NDN certificate [18].

Each NDNS record/data packet is signed at the time of its creation. NDN data packets also include FreshnessPeriod field that defines how long a cached packet is considered "fresh" in router caches and can be used to satisfy interests requesting the same data. Effectively, FreshnessPeriod plays the role of TTL field in DNS.

Table III summarizes how data-centric nature of NDN affects the design of NDNS, capturing differences between NDNS record and DNS packet formats.

TABLE III
DATA FORMAT COMPARISON BETWEEN DNS AND NDNS

DNS	NDNS
DNS packet, containing identifier, flags, codes, question count, answer record count, authority record count, additional record count	NDN data packet, containing a single NDNS record (equivalent to answer section)
Answer section containing one or multiple (RR)	Type- and application-specific content that can contain multiple items
RR name	Part of data packet name
Record type	Part of data packet name
Record class	Not defined (can be part of data packet name)
TTL	FreshnessPeriod
Resource data	Data packet's Content

D. Iterative Resolution

The overall NDNS iterative query process is defined by the flowchart in Figure 9.

To illustrate the process of an iterative resolution we will use the lookup for a TXT record of the "/net/ndnsim/www" domain (Figure 10). Assuming a caching resolver initially has an empty cache, it starts the process by expressing a query Interest with name "/NDNS/net/NS" to retrieve the referrals (NS record) to the zone "/net/NDNS" from a root server. Since the root zone ("/NDNS") is globally reachable, routers can forward this Interest directly using the name carried in the Interest.

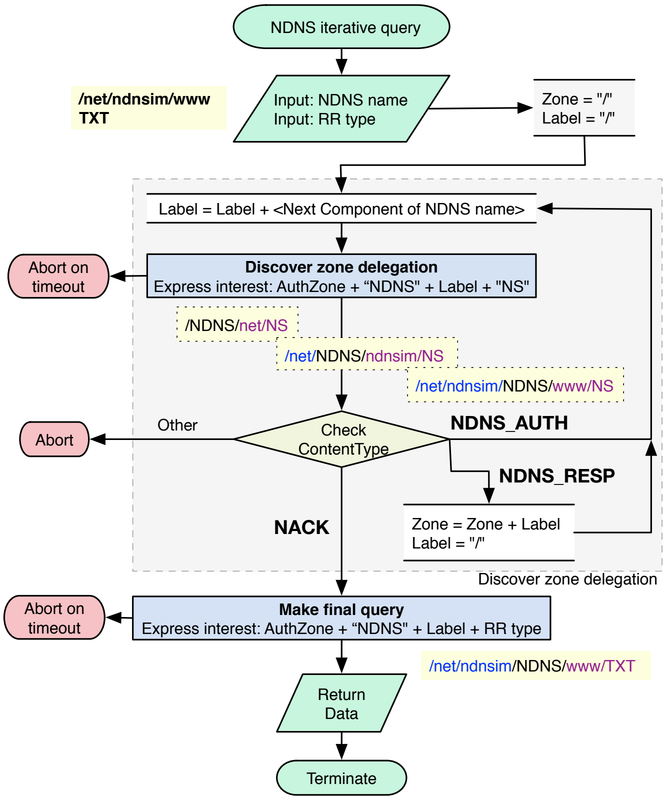

Fig. 9. NDNS iterative query flowchart

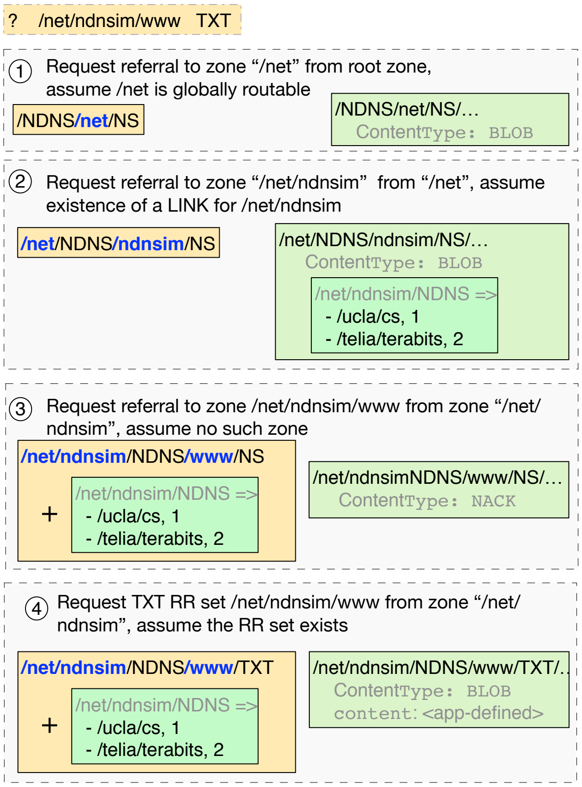

Fig. 10. NDNS iterative resolution example

When the caching resolver receives the referral data packet back, the packet should contain information on how to reach the NDNS servers of next level zone "/net/NDNS". Depending on whether the name "/net/NDNS" is announced to the routing system, the referral "NS" data packet may contain either (1) an empty set of forwarding hints or (2) a set of forwarding hints to steer an Interest with prefix "/net/NDNS" toward its data. Note that if it is known that "/net/NDNS" zone is globally reachable, this step could have been omitted.

The caching resolver can then continue the process to retrieve the referral to the servers of the next level zone ("/net/ndnsim/NDNS"). In the first case, the caching resolver simply expresses a query Interest "/net/NDNS/ndnsim/NS". In the second case, the caching resolver expresses the same query Interest but attaches the forwarding hints from the retrieved "NS" record, so that the Interest can be properly forwarded toward the zone "/net/NDNS". In our example, we assume routers have "/net/NDNS" prefix in their FIB to further forward the Interest to one of the "/net" NDNS servers.

The process of discovering the authoritative zone of a target data stops when the query for the referrals to the *target* domain name brings back a NACK ("/net/ndnsim/NDNS/www/NS" in our example), indicating that the zone is not delegated further. At this moment, the resolver can construct the *final question*, e.g., "/net/NDNS/ndnsim/www/TXT", to retrieve the target information, which will be returned from either the authoritative server or a router cache.

One must also consider the case where a domain does not have its own NDNS server and stores its data in its parent domains, e.g., a hypothetical case when the root zone has delegated "/net/ndnsim/NDNS" zone directly (no delegation for "/net/NDNS" zone). In this case, the result of the first query must return a special "NDNS_AUTH" response, indicating that the zone in question has not been delegated, but there are child zones of the requested namespace that are delegated. When such response is received, the iterative resolver asks a question to the same zone, but using a more specific label for query ("/NDNS/net/ndnsim/NS" instead of "/NDNS/net/NS").

E. Recursive Resolution

The NDNS recursive query is expressed using Interest with names that follows the structure "NDNS-R" + "domain name" + "requested record type" (e.g., "/NDNS-R/net/ndnsim/www/TXT"). The response for the caching query is the NDNS RR (or a NACK as a proof of non-existence of the RR) that is encapsulated in the data packet with the name matching the query and signature of the caching resolver. In other words, the encapsulated data packet carries the original signature of the authoritative zone, which can be verified by the consumer (if needed, the required certificates can be fetched through the same recursive process). At the same time, if the consumer has established trust relationship with the caching resolver, it can simply trust responses based on the outer signature of the recursive response.

F. Security

NDNS uses the hierarchical trust model similar to DNSSEC illustrated in Figure 11. In this model all records in zones are signed with one or more "data signing keys" (DSKs) (e.g., "/net/ndnsim/NDNS/KEY/dsk-8/CERT"). DSKs are signed by one or more "key signing keys" (KSKs) of the same zone (e.g., "/net/ndnsim/NDNS/KEY/ksk-7/CERT"). KSKs are either self-signed (e.g., for root "/NDNS" and well-known zones such as "/com/google/NDNS") or signed by the "delegation keys" (D-KEYs) records stored in the corresponding parent zones.

Note that the version number does not shows up in the KeyLocator in order to decouple the content authentication from certificate version [18]. This way, RRs do not have to be re-signed if certificates in the authentication chains are updated to a new version (e.g., when a grand-parent certificate is changed).

G. Zone Update and Synchronization

To update NDNS records in the zone, one either needs to leverage Interests to carry data packets with the new/updated records, or solicit Interest for the updates [19]. As an example, one can use a special "UPDATE" type of iterative query Interest, embedding the update as part of the label part of the query. Once one of the authoritative name servers for the zone receives such a query, it will extract the data packet from it, verify the record, and updates the zone.

To realize redundant authoritative NDNS servers, it is necessary to efficiently synchronize updated records. Such synchronization is a classic problem with several promising NDN-based solutions, including ChronoSync [17]. At the same time, the current design has limitations regarding deployment of redundant authoritative servers. To respond with a legitimate NACK, each NDNS server instance must have access to the zone's DSK. Our ongoing work is to use an NSEC-like mechanism [20] to allow pre-creation of signed denial-of-existence records and encapsulate them in response packets that do not require DSK-based signature.

V. Prototype & Deployment of NDNS

To verify and exercise the proposed design, we have developed several prototypes of NDNS. The now deprecated initial Python-based prototype [21] revealed unnecessary complexity in re-using the DNS message format and led to a clean-slate re-design of the payload of resource records. The next version written in C++ [22] went through several iterations, refining our understanding of the relationship between certificates used to *authenticate* NDNS zones and application certificates *stored* in NDNS zones. In particular, the intention to use NDNS as the storage for application certificates resulted in the conflict of names: the required "NDNS" component in the certificates names was not appropriate for applications that merely use NDNS as a storage. We then tweaked the NDNS naming conventions to use "KEY" instead of "NDNS", e.g., changing "/net/NDNS/www/CERT" to "/net/KEY/www/CERT", but did not get rid of the problem of moving "KEY" component in the certificate name, needed when the NDNS lookup process walks down the name tree. Applications were still forced to name the same key differently depending on the zone in which it is stored. The latest iteration of NDNS and NDN certificate naming (NDN Certificate 2.0 [23]) addressed this problem by decoupling NDNS and application certificates: the former use "NDNS" as part of the their names and are stored in NDNS directly and the latter use application-defined naming and are stored in NDNS in the encapsulated form (i.e., an NDN certificate as a payload in NDNS record).

The current version of the prototype has been deployed on NDN Testbed [24] and is actively used as an application certificate storage for testbed root ("/ndn/NDNS") and several other zones, including "/ndn/edu/ucla/NDNS" and "/ndn/guest/NDNS". NDNS has been also used to facilitate several NDN-based applications, e.g., ChronoChat [25], a distributed group chat tool for NDN, and NDNTube [26], an online video player over NDN. To show the universality and flexibility of NDNS, we also built a demo application called CPUSensor [27]. This application monitor each PC's CPU temperature and stores the measurement data into the designated NDNS zone using a custom "CPU-INFO" resource type.

Note that we are currently in process of transitioning NDNS to the latest version of the design described in this paper, as well as in the process of integration of the iterative

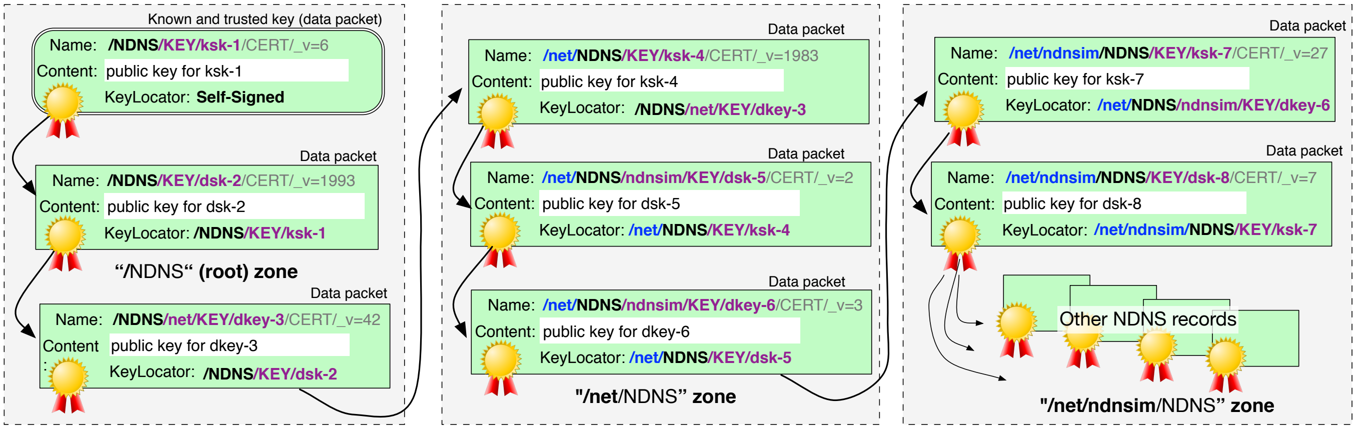
Known and trusted key (data packet)
Name: /NDNS/KEY/ksk-1/CERT/_v=6
Content: public key for ksk-1
KeyLocator: Self-Signed
Data packet
Name: /NDNS/KEY/dsk-2/CERT/_v=1993
Content: public key for dsk-2
KeyLocator: /NDNS/KEY/ksk-1
"/NDNS" (root) zone
Data packet
Name: /NDNS/net/KEY/dkey-3/CERT/_v=42
Content: public key for dkey-3
KeyLocator: /NDNS/KEY/dsk-2
Data packet
Name: /net/NDNS/KEY/ksk-4/CERT/_v=1983
Content: public key for ksk-4
KeyLocator: /NDNS/net/KEY/dkey-3
Data packet
Name: /net/NDNS/ndnsim/KEY/dsk-5/CERT/_v=2
Content: public key for dsk-5
KeyLocator: /net/NDNS/KEY/ksk-4
Data packet
Name: /net/NDNS/ndnsim/KEY/dkey-6/CERT/_v=3
Content: public key for dkey-6
KeyLocator: /net/NDNS/KEY/dsk-5
"/net/NDNS" zone
Data packet
Name: /net/ndnsim/NDNS/KEY/ksk-7/CERT/_v=27
Content: public key for ksk-7
KeyLocator: /net/NDNS/ndnsim/KEY/dkey-6
Data packet
Name: /net/ndnsim/NDNS/KEY/dsk-8/CERT/_v=7
Content: public key for dsk-8
KeyLocator: /net/ndnsim/NDNS/KEY/ksk-7
Other NDNS records
"/net/ndnsim/NDNS" zone

Fig. 11. Hierarchical trust model example. For a zone, DSK is certified by KSK inside a zone; KSK, except for root zone's KSK, is certified by a D-KEY stored at parent zone. Authentication chain can be constructed from trust anchor to any RR.

NDNS query as part of the validation framework of ndn-cxx library [18], one of the popular C++ NDN libraries. In particular, to support applications that store and retrieve their certificates in NDNS, the validation framework needs to be aware of the iterative or recursive NDNS querying process. In other words, when some certificate is needed to verify a signature, the validator may need to use a special certificate fetcher to iteratively or recursively discover zone in which the certificate is stored and then fetch it from the zone.

VI. Discussions

A. The Need for NDNS Caching Resolvers

NDN's in-network caching is shared by all data traffic and can provide opportunistic benefits for NDNS data retrieval but may not be relied on completely. Therefore, it is still beneficial to provide caching resolvers which are dedicated to cache NDNS data. Furthermore, caching resolvers perform the iterative resolution on behalf of end consumers. This can be important for power-constrained devices such as sensors, wearable devices and personal digital assistants (PDAs) that operate on battery power.

B. Root-Less NDNS

The Internet is a large distributed interconnection of many networks owned by different parties. How to manage DNS root zone has been a subject of discussion and debate over years. Yet due to its top-down lookup process, the root zone is essential to DNS operations, because caching resolvers use the hard-coded addresses of the root zone servers. Not having the NDS root zone implies that each resolver must be configured with the IP addresses of the authoritative name servers of all the TLDs (around 300 TLDs with each have at most 13 name servers). Whenever a TLD authoritative server change its address, all the resolvers must update their configuration.

NDNS is designed to support multiple independent trust anchors at the top, in the absence of a root. In other words, if the routing system knows how to forward Interests towards the top-level NDNS zone services, NDNS can easily function without the root zone. In this case, iterative query process can start by fetching data from TLD servers to determine whether the second-level zone has been delegated. Moreover, if a popular zone, such as "`/com/google/NDNS`", is known to be directly reachable, the iterative query process for this zone can also skip TLD. Note however, in addition to the routing system, the resolvers must be able to authenticate the responses. As discussed in Section IV-F, they either they need to explicitly trust the corresponding ZSKs or still fetch D-KEYs from root or TLD zones.

C. Denial-of-Service & NSEC

NDNS generates NACK and signs it to answer the requests for non-existing information. Attackers can send a massive number of requests with random names as a form of denial-of-service (DoS) attack on NDNS or data producer. The same vulnerability exists in DNS as well, and is addressed by NSEC [20] and its enhancement, e.g., NSEC3[28]. Our ongoing work is to adopt NSEC-like approach by embedding the NSEC records into the returned NACK without re-signing the whole packet again. The caching resolver can return negative answer to end clients directly on behave of name servers, since it receives the final questions and can understand the application semantic. However, it is still an open architectural question whether it is possible to leverage NDN's in-network caches to process such NSEC records and if possible, which security implication it would entail.

An alternative way to look at DoS and DDoS problem is considering general data retrieval mechanism of NDN. (D)DoS'ing a producer by sending Interests with random names is a vulnerability in NDN that can be addressed by leveraging the stateful and symmetric Interest/Data forwarding in NDN [29], [30].

VII. Conclusion

In this paper, we described the design of NDNS which provides an always-on lookup service in an NDN network. By providing a service to store and lookup forwarding hints, NDNS facilitates retrieval of named data, without having all data producers to announce their name prefixes into the global routing system. At the same time, flexible design of NDNS enables other uses, such as persistent storage for users' and applications' certificates.

The design of NDNS is among our first attempts to convert existing IP-based applications to run over NDN networks. It taught us to appreciate the impacts of the different network architectures on application designs. Although NDNS inherits most of the major design concepts from DNS, its operations differ from that of DNS in fundamental ways. Our lessons from converting IP-based DNS to an NDN service and to leverage NDN's built-in support for better application designs, can be summarized as follows.

a) Effects of Shared Namespace between Application and Network Layers: The design of the namespace is of first and foremost importance in all NDN application developments. The differences between the DNS and NDNS designs reflect the impact of having application and network sharing the same namespace. NDN's direct use of application data names for network-layer data retrieval brings the gains of utilizing NDN's built-in multicast delivery, in-network caching, and network forwarding for automatic server selection. At the same time, this shared use of the same namespace also introduces new challenges.

In DNS, because a query packet carries *both* the name to be looked up and the IP address of a server to answer the query, this enables a DNS query to carry exactly the same DNS name no matter where it goes (to a caching resolver or an authoritative server), or what kind of response it may retrieve (whether a referral or the final answer). The flip side is that the resolver must pick an IP address for one of the servers, without the knowledge of network connectivity. In NDNS, one must explicitly name the information to be retrieved, and queries intended for caching resolvers must carry different names from those going to authoritative servers, but can leave to the network to figure out the best place to retrieve an answer.

b) Network-Level Data-Centric Security: NDN's built-in security primitive, signing and verifying *every* data packet, provides a solid foundation to build secure systems and applications. NDNS effectively has the DNSSEC functionality built in, and can leverage a defined trust schema [16], which defines the relationship between data and signing key names, to realize more flexible trust models than the strict hierarchical trust model of DNSSEC.

The process of NDNS development preceded the conceptualization of NDN trust schemas, during which we stumbled many times over the questions of how to enforce different security policies, or how to automate the signing as well as the signature verification processes. Majority of these questions have since been addressed through the use of trust schemas. However, we believe that how to do security right will continue to be a challenge in future NDN application developments.

c) "The test of all knowledge is experiment": This a famous Feynman quote [31], which offers a concise description of the biggest lesson we learned from our four-year trial-and-error NDNS development process (e.g. see our experience described in Section V), a rather rewarding experience.

References

[1] V. Jacobson, D. Smetters, J. Thornton *et al.*, "Networking named content," in *Proc. of ACM Conference on emerging Networking EXperiments and Technologies (CoNEXT)*, 2009.

[2] L. Zhang *et al.*, "Named data networking (NDN) project," NDN, Technical Report NDN-0001, 2010.

[3] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson *et al.*, "Named Data Networking," *ACM Comp. Comm. Review*, 2014.

[4] A. Afanasyev, C. Yi, L. Wang, B. Zhang, and L. Zhang, "SNAMP: Secure namespace mapping to scale NDN forwarding," in *Proc. of IEEE Global Internet Symposium (GI)*, 2015.

[5] Y. Zhang, A. Afanasyev, J. Burke, and L. Zhang, "A survey of mobility support in Named Data Networking," in *Proc. of INFOCOM Workshop on Name-Oriented Mobility (NOM)*, 2016.

[6] C. Bian, Z. Zhu, A. Afanasyev, E. Uzun, and L. Zhang, "Deploying key management on NDN testbed," NDN, Technical Report NDN-0009, Revision 2, 2013.

[7] P. Mockapetris, "Domain names, implementation and specification," RFC 1035, 1987.

[8] ——, "Domain names: concepts and facilities," RFC 1034, 1987.

[9] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose, "Resource records for the DNS security extensions," RFC 4034, 2005.

[10] ——, "Protocol modifications for the DNS security extensions," RFC 4035, 2005.

[11] ——, "DNS security introduction and requirements," RFC 4033, 2005.

[12] A. Afanasyev, J. Shi, B. Zhang, L. Zhang *et al.*, "NFD developer's guide," NDN, Technical Report NDN-0021, Revision 7, 2016.

[13] NDN Project, "NDN packet format specification," http://named-data.net/doc/ndn-tlv/.

[14] IANA, "Domain name system (DNS) parameters," http://www.iana.org/assignments/dns-parameters/dns-parameters.xhtml.

[15] Y. Yu, D. Wessels, M. Larson, and L. Zhang, "Authority server selection in DNS caching resolvers," *ACM Comp. Comm. Review*, 2012.

[16] Y. Yu, A. Afanasyev, D. Clark, V. Jacobson, L. Zhang *et al.*, "Schematizing trust in named data networking," in *Proc. of ACM Conference on Information-Centric Networking (ICN)*, 2015.

[17] Z. Zhu and A. Afanasyev, "Let's ChronoSync: Decentralized dataset state synchronization in Named Data Networking," in *Proc. of IEEE International Conference on Network Protocols (ICNP)*, 2013.

[18] A. Afanasyev, Y. Yu, J. Shi *et al.*, "ndn-cxx: NDN C++ library with eXperimental eXtensions," http://named-data.net/doc/ndn-cxx/current/.

[19] I. Moiseenko, M. Stapp, and D. Oran, "Communication patterns for Web interaction in Named Data Networking," in *Proc. of ACM Conference on Information-Centric Networking (ICN)*, 2014.

[20] S. Weiler and J. Ihren, "Minimally covering NSEC records and DNSSEC on-line signing," RFC 4470, 2006.

[21] A. Afanasyev, "NDNS: DNS service for Named Data Networking (py-ndns)," https://github.com/cawka/py-ndns, 2013.

[22] X. Jiang, A. Afanasyev, Y. Xia, E. Newberry, and J. Tan, "NDNS: Domain name service for Named Data Networking," https://github.com/named-data/ndns, 2017.

[23] "NDN certificate format version 2.0," http://named-data.net/doc/ndn-cxx/current/specs/certificate-format.html.

[24] "NDN Testbed," http://named-data.net/ndn-testbed/.

[25] "ChronoChat," https://github.com/named-data/ChronoChat.git.

[26] L. Wang, I. Moiseenko, and L. Zhang, "NDNLive and NDNTube: Live and prerecorded video streaming over NDN," NDN, Technical Report NDN-0031, 2015.

[27] X. Jiang, "CPU Sensor," https://github.com/shockjiang/cpu-sensor, 2015.

[28] B. Laurie, G. Sisson, R. Arends, D. Blacka *et al.*, "DNS security (DNSSEC) hashed authenticated denial of existence," RFC 5155, 2008.

[29] 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*, May 2013.

[30] A. Compagno, M. Conti, P. Gasti, and G. Tsudik, "Poseidon: Mitigating interest flooding DDoS attacks in Named-Data Networking," in *Proc. of IEEE Conference on Local Computer Networks (LCN)*, 2013.

[31] R. Feynman, "The Feynman Lectures on Physics," http://www.feynmanlectures.caltech.edu/, 1963.