Having your cake and eating it too: Domain Name Data Networking (DNDN)

Eric Osterweil Craig Murray Aziz Mohaisen

Abstract

Named Data Networking (NDN) is a new architecture that has been proposed for the Internet. Among its insights is the realization that we predominantly use the Internet to access *data*, and that the endpoints of communications are generally not our focus. Indeed, the Internet Protocol and the use of IP addresses is just a means for delivery of content that could be directly addressed and distributed in a more efficient manner. However, the NDN architecture has several design and deployment obstacles. First, it relies heavily on caching among a dense population of NDN nodes, and yet there will be very few nodes deployed initially. Second, it uses an unstructured namespace in which names can be chosen by content providers. This makes it challenging to disambiguate *which* provider is the actual authority for a name. Third, NDN ensures security through cryptographic signatures on all data packets, but NDN does not have an architecture for securely discovering and verifying the authenticity of keys. These are just some of the barriers to widespread adoption of NDN. We suggest that all of these problems (and more) can be solved by implementing NDN on *top* of the Domain Name System (DNS). We propose Domain Name Data Networking (DNDN) as a possible way to facilitate use of NDN within the existing infrastructure. In this model, DNS becomes a substrate from which NDN can leverage essential components. We show the feasibility of DNDN, we evaluate the benefits if it were deployed, and we show a simulated sample application that uses it. DNS is one of the Internet's most operationally well understood, ubiquitous, and resilient protocols. Why not use it to gain the benefits of NDN's new design features and keep the stability of the Internet's existing infrastructure?

1. INTRODUCTION

Information Centric Network (ICN) has recently been advocated as a new networking paradigm to address various shortcomings in today's Internet, including security and performance. Two prominent architectures that use this paradigm are Content Centric Networking (CCN) [15] and Named Data Network (NDN) [27]. Both architectures are built on the premise that content is fetched by *names,* rather than by location, and that pervasive caching is deployed to optimize performance and reduce latency.

For example, when a user is interested in content previously served to another user on the same network path, the interest is fulfilled from a near-by cache. This design choice is considered a great advantage in reducing overall content retrieval latency [15, 27]. Names in NDN (and CCN; collectively related to as ICN) are structured hierarchically. Security is facilitated by per-packet signing at the data producers and these signatures are verified by the consumers (c.f. §2).

However, to realize the NDN architecture in a future Internet, several issues must be addressed: i) name management, ii) operational caching, iii) routing, and iv) deployable security and scalability [5]. These issues are tangible obstacles to seeing broad deployment of NDN. To this end, we propose to jump-start NDN deployment by leveraging the existing Domain Name System (DNS) [19] infrastructure with an architecture we call *Domain Name Data Networking (DNDN)*. DNS was designed as a distributed database [19], but has not (as of yet) been used to transmit actual data content. However, we argue that it can and should. In this way, we imagine that the old Internet architecture can become the substrate for a new Internet architecture. Our intuition is that i) the DNS already gives collision-free namespace, ii) DNS caches are already richly deployed (in numbers and topological diversity), [2, 23] iii) DNS' namespace is delegated along authoritative boundaries to canonical resource holders, and iv) the DNS Security Extensions (DNSSEC) [6] already give operational secure key learning, source authenticity, integrity protection, and secure denial-of-existence of DNS data. [22] With this approach, we gain the operational stability and knowledge of more than 30 years, *and* the novel new faciliities needed for the next 30 years to operate DNDN [1].

Contributions: In this work, we introduce the case for DNDN, a realization of the NDN paradigm on top of the DNS. We advocate various design aspects provided in today's DNS that can enable various functional properties advocated in NDN, and can provide suitable bootstrapping opportunities for the architecture. We discuss challenges towards realizing such design aspects, and describe a potential ways of addressing them, including a preliminary evaluation.

[1] In the rest of this paper, we use NDN to exemplify architectures that have such features. We note, however, that our contribution may not require NDN and should apply to ICN generally.

Unlike NDNS [4], which builds a DNS-like system for NDN that implements its naming characteristics, our work in this paper builds NDN in the existing DNS with little changes to the way DNS operates and does naming.

Organization. The rest of this paper is organized as follows. Preliminaries are introduced in §2. The case for DNDN is made in §3. We introduce the design concepts and evaluation of DNDN in §4 and §5. A discussion is provided in §6 and concluding remarks are made in §7.

2. PRELIMINARIES

2.1 Named Data Networks

Named Data Networkign is a growing research area. We direct the interested reader to [27] for further details on the design aspects of NDN. Here we review key components related to DNDN. In NDN, content is fetched by its name [27]. A NDN network consists of routers, where each router has a cache that acts as a *content store*, and edge routers are connected to users and origin servers. An *interest packet* in NDN encapsulates a request for content by its name. An origin server is a server that originates contents to be served in the network, thus fulfilling interests. The contents (data packets) may or may not be cached in the network. In NDN, content is forwarded back to a user on the same path as it is requested by that user, and PIT (*pending interest table*) at each NDN router records which interest is not fulfilled yet. A *face* in NDN is the access point (e.g. a port) at which data is sent or received in a router ([27]; §2.3).

Naming in the future Internet in general is still a topic of a great debate [12], where both flat and structured names are considered. In NDN in particular ([27]; §2.1), names are assumed hierarchically structured. For example, video produced by a producer `prod` and published by that producer might be referred to as `/prod/videos/demo.mpg`, where '/' delimits name components.

NDN assumes a data-centric security approach ([27]; §2.2), contrary to today's IP networks that leave it to devices to ensure security, often using a connection-centric security approach. In particular, each data packet in NDN is signed by a producer's private key and signatures are verified by the potential consumer.

2.2 The Domain Name System

The DNS [19] maps domain name such as `www.example.com` to service identifiers (such as IP addresses, email servers, cryptographic certificates, and more). Virtually every Internet application relies on looking up certain DNS data. In this section we introduce a basic set of DNS terminology which is used throughout the text, including resource records (RRs), resource record sets (RRsets), and zones, followed by an overview of the DNS Security Extensions (DNSSEC).

All DNS data is represented in the same data structure called *Resource Records* (RRs), and each RR has an associated name, class, and type. For example, an IPv4 address for the domain name `www.example.com` is stored in an RR by that name. A host with several IPv4 addresses will have several RRs, each with the same name but a different IPv4 address. The set of *all* resource records associated with the same name is called an *Resource Record Set* (RRset). DNS resolvers query for RRsets. For example, when a browser queries for `www.example.com`, the reply will be the RRset with all the IPv4 addresses for that name. Note that the smallest unit that can be requested in a query is an RRset, and all DNS actions including cryptographic signatures, discussed later, apply to RRsets instead of individual RRs.

The DNS is a distributed database organized in a tree structure. At the top of the tree, the root zone delegates authority to Top Level Domains (TLDs) like `.com` or `.net`. The zone `.com` then delegates authority to create `example.com`, the zone `.net` delegates authority to create `example.net`, and so forth. In the resulting DNS tree structure, each node corresponds to a *zone*. Each zone belongs to a single administrative authority and is served by multiple *authoritative nameservers* to provide name resolution services for all the names in the zone. In DNS, the delegations are hierarchical (from a domain to a proper subdomain), but not necessarily sequential (e.g., they can skip a level). Every RRset in the DNS belongs to a specific zone and stored at the nameservers of that zone. For example, the RRset for `www.example.com` belongs to the `example.com` zone and is stored in the `example.com` nameservers; while the RRset for `www.example.net` belongs to the `example.net` zone and stored in the example.net nameservers.

Additionally, the types of RRs in DNS is an extensible list ranging from IP addresses (`A` RRs) to cryptographic keys (`DNSKEY` RRs) to text records (`TXT` RRs), and is actively added to for application-specific types.

2.3 DNSSEC Overview

Security was not a primary objective when the DNS was designed in mid 1980's and a number of well known vulnerabilities have been identified [9, 7]. DNSSEC [6] provides a fairly straightforward cryptographic solution to the problem. To prove that data in a DNS reply is authentic, each zone creates public/private key pairs and then uses the private key(s) to sign data. Its public keys are stored in a type of RR called `DNSKEY`, and all the signatures are stored in another type of RR called `RRSIG`. In response to a query, an authoritative server returns both the requested data and its associated `RRSIG` RRset. A resolver that has learned the `DNSKEY` of the requested zone can verify the *origin authenticity* and integrity of the reply data. To resist replay attacks, each signature carries a definitive expiration time.

In order to authenticate the `DNSKEY` for a given zone, say `www.example.com`, the resolver needs to construct a *chain of trust* that follows the DNS hierarchy from a trusted root zone key down to the key of the zone in question (this is shown in Figure 1). In the ideal case, the public key of

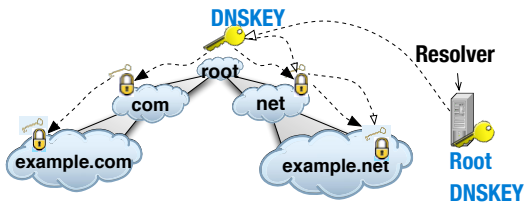

Figure 1: Resolvers preconfigure the root zone's KSK (its public key) as a trust anchor and can then trace a "chain of trust" from that key down the DNSSEC hierarchy to any zone's key that they have encountered.

the DNS root zone would be obtained offline in a secure way and stored at the resolver, so that the resolver can use it to authenticate the public key of `.com`. The public key of `.com` can then be used to authenticate the public key of `example.com`.

3. THE CASE FOR DNDN

As a future Internet architecture, NDN's promise is compelling: native content caching embedded in the network's substrate to reduce latency, communications that focus on named content rather than service endpoints to reduce congestion, and primitives to support intrinsic security of all data [10]. However, the deployment of any new Internet architecture faces significant hurdles, including achieving a critical mass. Without enough nodes supporting the new technology, there will be insufficient benefit to justify expense of time and materials. Simply put, without enough of a network, there is no network effect. However, in the search for a new Internet architecture, we may not have to replace the old one. DNDN is an architecture that *uses* the old architecture as a substrate to enable the new one.

The future Internet will not be built in a vacuum. It must be built considering the context of today's Internet [12, 10, 20]. Not only are there suggestions that we should build on what already works, but market forces will likely mandate that we use legacy systems in any future architecture.

In this proposal, we suggest that we can jump-start NDN deployment by leveraging the existing DNS infrastructure. By using DNS as a substrate for NDN we can project its model of named data into the distributed name-based database that DNS already forms. Based on our observation that DNS *can* hold actual data, we gain the benefit of years of development, rich tool-sets, testing, and operational experience, as well as functional benefits that are analogous to the design principles of NDN. The DNS has evolved over the past 30 years to become a highly salable, resilient, and available global system. DNS caches are richly deployed; large in numbers and broad in topological diversity, as evidenced by the Open Resolver Project [2]. While these servers are often maligned as a blight on the Internet [2], they serve as evidence that ubiquitous DNS caching is not only achievable, but is actually available *today*. NDN is an ideal technology to benefit from this wide deployment and caching capacity, and to take today's Internet into tomorrow.

What is in a name? A rose by any other name could be a thorny mess. While the concept of naming data is appealing, the devil lies in the details. How will the namespace of the future Internet be managed? There will have to be a system in place to prevent name collision and the delegation of naming authority (after all, a rose from a Montague might not be the same as a rose from a Capulet). The DNS today provides a collision-free namespace to build on, and approaches to remediate various layer seven name ambiguities are even well explored[14, 16]. This is further motivation to leverage the DNS namespace infrastructure for use in NDN.

Finally, whatever the next Internet architecture will be, it must have a robust system for verifying its data. If we design a system that caches content extensively, we must include mechanisms for ensuring the authenticity of the cached data. DNSSEC is an operational system that gives secure key learning, source authenticity, integrity protection, and secure denial-of-existence of non-existent data. DNS offers the very facilities that NDN needs from a substrate, and DNDN illustrates how the new architecture can be built on one of the Internet's oldest core protocols.

4. ARCHITECTURE

The architecture of DNDN has four general components: the method by which the NDN namespace is mapped into DNS, the transcoding of NDN data into DNS RRsets, the logic used to establish faces (or peer) with other entities in DNDN, and the event logic that manages the mis-match between NDN's persistent queries and DNS' transactional query and response model.

4.1 Namespace

One of NDN's core enhancements has been to lift the Internet's thin waist from IP to names [1]. NDN names offer an open-ended extensibility, while DNS imposes more orthodox limitations on domain names. DNDN's architecture reconciles these using a set of mapping primitives.

The first step in our mapping method is to reverse the order of labels. The most significant labels in an NDN name are the left-most and in DNS domain names the most significant are the right-most. So we reverse them.

Next, DNDN's mapping method reconciles NDN's arbitrarily long names with DNS' domain name length limit by collapsing multiple NDN labels into collision-free hashes. The initial architecture uses truncated SHA256 for that. We collapse multiple labels only from the middle of the name, thereby preserving the least-significant labels for any control labels (such as version information, segmentation information, etc.) We note that in certain circumstances, content authors may wish to use more than one hash label in DNS, in order to manage their zone sizes and response sizes (but we discuss this further in §6.1. Our transcoding scheme allows

DNDN content at these DNS names to be DNSSEC signed and secured by simply including the content in a standard authoritative DNS name server.

At a high-level, NDN names such as:

`/com/cnn/headlines/2015/.../⟨ver⟩/⟨seg⟩`

map to DNS names like:

`⟨seg⟩.⟨ver⟩.⟨hash⟩.headlines.cnn.com`

We discuss these issues further in §6.1

4.2 Data

Data in NDN is broken down into Type Length Values (TLV) data structures. Using TLV, Binary Large OBjects (BLOBs) can be segmented into segments of TLVs for transport. In DNDN, data will be transported in DNS RRsets, and secured with DNSSEC signatures and keys (`RRSIG` and `DNSKEY` records). The limitations in DNS message and RRset sizes suggest that content authors will want to segment data into RRs and RRsets that match the operational needs of their zone sizes and network capacities [22, 3]. For this reason, DNDN collapses TLV data structures into a single BLOB, and then re-segments the BLOB into RRsets. The decision of how to choose segment sizes is discussed further in §3.

4.3 Peer Logic and Routing

The connectivity model of DNDN is slightly simplified from the approach taken by native NDN nodes. When assigning faces to a DNDN node, the primary goal is to offer rich caching. In DNDN, authoritative data is initially located on a DNS zone's authoritative name servers. The DNS cache becomes the content store for DNDN. Once fetched for the first time, DNDN data can persist in *any* standard DNS cache for the lifetime specified by the DNS TTL field. As a result, DNDN nodes are configured with a primary caching DNS recursive resolver, and augmented using other available caching recursive resolvers.[2] When an interest is issued by an NDN application and received by a DNDN node, the cache of that node is checked for the named content. If the content is not in the cache, the list of peer DNDN caches is queried over DNS with the Recursion Desired (RD) bit set to 0. This informs these peer nodes that the query should only be answered if it is already in cache. If none of the caches have the requested data, the primary caching recursive resolver is instructed to fetch it. In each case, the query instructs caches to use DNSSEC checking (via the AD bit), and optionally ask to have all verification information sent to the DNDN node (by setting the CD bit). This allows the local cache to contain data for other peers in the DNDN network in the future, while being able to directly retrieve content after a cache miss. In the event an NDN interest is scoped, the final step of querying the authority can be omitted.

4.4 Event Model

The final component of the DNDN architecture is the event model. The disconnect between NDN's pending interests and DNS' query and response model is an important gap. For example, if an NDN node queries for `/something/DNE` that currently Does Not Exist, and a producer later produces it, data is sent back to the consumer. In DNS, the query for `DNE.something` is simply given an `NXDomain` response, indicating that it does not exist. In DNDN, such cases are handled by silent retries at specific intervals until an ultimate timeout. This is an example of an imperfect alignment between the two models, but we leave it to future work to resolve whether this is a reasonable approximation.

[2]Our Evaluation §5 will illustrate that even without investing in a new deployment, the number of DNDN nodes is already millions.

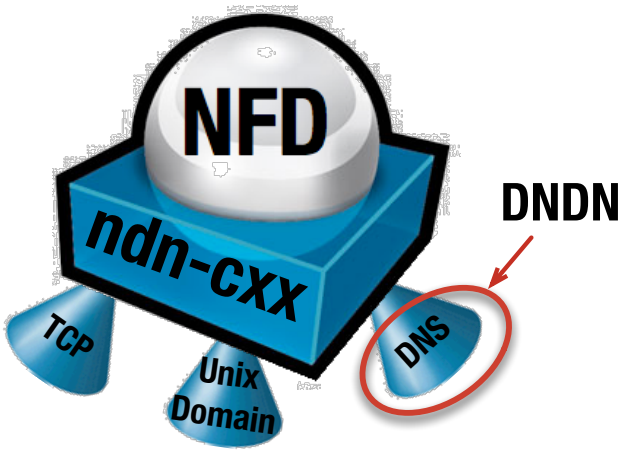

Figure 2: DNDN is implemented as a transport face (DNS) within the current NDN architecture.

5. EVALUATION

To evaluate the feasibility and performance of DNDN, we have implemented a proof of concept within the existing NDN codebase, and conducted simulations of varying preliminary deployment scenarios.

5.1 Proof of Concept

One of the existing implementations of NDN is called the NDN Forwarding Daemon (NFD), which uses an extensible NDN library called ndn-cxx, written in C++. In this framework, various transport protocols are encapsulated so that NDN faces can remain agnostic of how communications are transmitted over different network substrates. Currently, TCP and Unix domain sockets are supported, and we augment this with a DNS transport face.

Figure 2 illustrates where DNDN is implemented within the current framework. We implement DNDN as a transport module so that NDN daemons can maintain their current logic and capitalize on DNDN by simply configuring the right kind of face with a minimal disruption to normal NDN functionality

5.2 Network Simulations

Table 1: Sample DNDN deployments results.

# DNDN Nodes	% of ORRs	# of Peers	% Full Peers	Hit Ratio	Total B/W
601,238	10%	0	N/A	0.0	36.07 TB
601,238	10%	100	95.73%	7.98	4.52 TB

Another critical question about DNDN is how well its simplified caching model works? To evaluate this, we imagine an NDN application, which we call NDNews, that lets news sites create small 30 minute videos (60 MiB in our calculations) of news events and allows NDN clients to watch them. To simulate this, we use the Internet's Autonomous System (AS) topology that is calculated by the Internet Research Lab (IRL) [21], a large scale measurement of Open Recursive Resolvers (ORRs) maintained by the Open Resolver Project [2], and the AS locations of a candidate new source's DNS name servers (CNN.com). Our data set includes 50,485 ASes, and 6,012,387 ORRs.[3]

With this setup, we simulated 10% of the OORs acting as DNDN nodes, where each DNDN node attempts to choose 100 local (in the same AS) neighbor caches to act as peered faces. We also used a configuration of 0 peers to be a baseline for content fetching *without* caching (i.e. native behavior of video browsing today). We then re-ran this same setup multiple times during which we assigned random wake-up times to each DNDN node and measured the amount of network bandwidth consumed. The results of these experiments are summarized in Table 1. Of note is **% Full Peers**: not every AS in the ORR set has 100 neighboring IPs, so only some of the DNDN nodes are able to have full peer lists. Nevertheless, the average hit ratio in our simulations shows the substantial gains, even with this simplistic caching model (only at the edge instead of including caching deeper in the network—following the insight in [10]).

According to our simulations, DNDN saves substantial network overhead by reducing content fetching from 36.07 TB to 4.52 TB. To compound this, recent measurements [18] suggest that there has been a relatively constant AS diameter for over 10 years (despite large growth in the Internet). As such, the network savings (as modeled by bandwidth per AS hop) could actually be $\langle \text{Difference in Traffic} \rangle \times \langle \text{Average number of hops} \rangle = (36.07 - 4.52) \times 3.5 = 110.425$ TB.

6. DISCUSSION

The deployment of DNDN does not come without certain design challenges.

6.1 Namespace Management

A main property of DNS is that it provides strong guarantees for collision avoidance. To ensure that same avoidance in DNDN, we argue that by design one should keep the top n level domains reserved for authority and collision avoidance. For example, an NDN name `/com/cnn/headlines/...` would resolve to `headlines.cnn.com` in DNDN.

We note that NDN names are arbitrarily long by nature, potentially too long for DNS. Thus, we suggest compressing multiple NDN labels into a single DNS label or a hash. However, a single hash may make the zone content too flat, and the resource record sets (RRsets) too big, which may affect the operation of DNDN. Thus, figuring out the right number of labels, including various user-specified labels, is needed.

One approach to address this issue is to use multiple hashes, which could also be used to manage delegation depth, ensure that RRset sizes can be managed while also aggregating multiple NDN labels, and to allow high-level user-specified description in names. For example, the above example of converting the NDN name to a DNDN name would include `<hash>.<hash>...` before the record name. Finally, we suggest appending segmentation and versioning labels for explicit requirements (i.e., appending `<seg>.<ver>` to the resulting domain name for the transformation from the NDN name) and to meet application-specific needs.

6.2 Data Management and Transport

As shown in §4, data in DNDN is transported from caches or origin servers hosting the contents to hosts requesting this data in today's DNS RRsets. Furthermore, the security of such data is addressed by DNSSEC: DNDN uses the `RRSIG` to verify contents of RRset.

To cope with the challenge of representing and transporting DNDN data in DNS, we identify two options: using an existing record type or creating a new one with its specification of representation. For the first approach, we could use the `TXT` record type, which is today used for transporting opportunistic encryption material, sender policy framework (SPF) material, and DNS service discovery material. The main advantage of this approach is that it would require no modifications to the existing DNS infrastructure and record types. The other approach for representing data is to introduce a `DNDN` record type, which is specifically used for representing and transporting DNDN data. As active members of the IETF, we advocate creating application-specific record types over using a generic types.

We note that today's DNS responses allow up to 4096 Bytes (4KiB) responses, although in reality there is no reason why responses should not exceed such size. Furthermore, today's DNS deployment is facilitated by a wide use of UDP as a transport protocol, where the fraction of TCP traffic seen at a large top-level domain registry, such as `.com` is just below 1%. However, to enable DNDN with large `TXT` or `DNDN` records without having to deal with issues like fragmentation, we argue that TCP is used as the means of transportation from the servers hosting DNDN data. The rest of DNS need not change from UDP. When enabled, a maximal message size 65535 (64KiB) Byte is allowed. We further note

[3]Not all ORRs discovered by [2] function properly. We have pruned our evaluation to only use well functioning ORRs.

that TCP is already of particular and independent interest in the Internet research community today to allow stateful connections that facilitate security and privacy [28].

6.3 Connectivity Issues

Several general connectivity-related issues in DNDN arise, which may affect the deployment of our architecture. In particular, a simplistic deployment elides the need for a more elaborate routing scheme, while focusing on data resolution and locality of caching. The choice of peer caches is an important one. Issues ranging from locality [5] to liveliness to responsiveness to trustworthiness [11] to robustness of large data sets [24] and more may be critical in deciding which caches to configure as neighboring faces [25], or how and where caching should be performed [10]. While the prior work addressing those issues in the general NDN platform could provide clues for addressing those issues, we leave the specific mechanism and policy around peer choice and maintenance in DNDN to future work.

We note, however, that various well-understood and widely deployed operational practices in DNS could play a significant role in enhancing DNDN's performance. For example, while locality of caching (as we simulated in §5) is a desirable feature to reduce latency in DNDN, the existence of large DNS forwarding caches (which aggregate queries from downstream recursive resolvers and cache for them) could improve cache hit rates and trustworthiness and efficiency. Those caches correspond to proxies [8, 17] (or hidden DNS servers [23]) in many enterprise and open recursive resolver settings. Those and other similar efforts on measures to improve DNS trustworthiness [13] and on resolver selection [26] could guide our future work.

6.4 Hybridizing DNDN and Native NDN

Among the potential security issues in DNDN is interactions between DNDN's collision-free namespace and native NDN nodes that author content under the namespace that exists in DNS. For example, if a native NDN author places content at `/com/example/chat`, which is not affiliated with the DNS zone `chat.example.com`, then there will be a name collision. A DNDN node will reject the data, but a native NDN node (which may not be subject the DNDN's rules) will allow he collision. This issue, of hybridizing the two approaches, is fundamental and suggests that the two types of networks (DNDN and native NDN) not be hybridized. We leave this issue to future work.

7. FUTURE WORK

In this work we have outlined a synergy between the future Internet architecture of NDN and the current Internet architecture, and have proposed a design to capitalize on both of them: Domain Name Data Networking. We implemented a proof-of-concept of DNDN, simulated its performance, and have outlined a number of challenges. Specifically, we have found evidence that reusing the DNS cleanly addresses several specific open challenges faced by NDN, and our simulations show almost a factor of eight reduction in network utilization, over conventional content streaming.

In the future, we plan to investigate the open issues we have outlined above, using a full implementation and deployment of DNDN, and investigate its performance in actual deployment. We believe that the path forward, for the future Internet architectures, may be to strive for greenfield advantages on top of solid operational staples.

8. REFERENCES

[1] Named data networking: Motivation & details. `http://named-data.net/project/archoverview/`.

[2] Open resolver project. `http://openresolverproject.org/`.

[3] Availability problems in the dnssec deployment. In *RIPE 58*, 2009. `http://www.ripe.net/ripe/meetings/ripe-58/content/presentations/dnssec-deployment-problems.pdf`.

[4] A. Afanasyev. *Addressing operational challenges in Named Data Networking through NDNS distributed database*. PhD thesis, University of California Los Angeles, 2013.

[5] A. Afanasyev, C. Yi, L. Wang, B. Zhang, and L. Zhang. Snamp: Secure namespace mapping to scale ndn forwarding. In *Proceedings of 18th IEEE Global Internet Symposium (GI 2015)*, 2015.

[6] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. Protocol Modifications for the DNS Security Extensions. RFC 4035, March 2005.

[7] D. Atkins and D. Austein. Threat Analysis of the Domain Name System (DNS). RFC 3833, August 2004.

[8] R. Bellis. Dns proxy implementation guidelines. RFC 5625, August 2009.

[9] S. M. Bellovin. Using the domain name system for system break-ins. In *Proceedings of the Fifth Usenix Unix Security Symposium*, pages 199–208, 1995.

[10] S. K. Fayazbakhsh, Y. Lin, A. Tootoonchian, A. Ghodsi, T. Koponen, B. Maggs, K. Ng, V. Sekar, and S. Shenker. Less pain, most of the gain: Incrementally deployable icn. In *ACM SIGCOMM Computer Communication Review*, volume 43, pages 147–158. ACM, 2013.

[11] C. Ghali, G. Tsudik, and E. Uzun. Network-layer trust in named-data networking. *Computer Communication Review*, 44(5):12–19, 2014.

[12] A. Ghodsi, S. Shenker, T. Koponen, A. Singla, B. Raghavan, and J. Wilcox. Information-centric networking: seeing the forest for the trees. In *Proceedings of ACM HotNets*, 2011.

[13] A. Hubert and R. Van Mook. Measures for making dns more resilient against forged answers. RFC 5452, 2009.

[14] ICANN. Guide to name collision identificationand mitigation for it professionals. `https://www.icann.org/en/system/files/files/name-collision-mitigation-01aug14-en.pdf`, August 2014.

[15] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. Braynard. Networking named content. In *Proceedings of ACM CoNEXT*, pages 1–12, 2009.

[16] B. Kaliski. Uncontrolled interruption? dozens of "blocked" domains in new gtlds actually delegated. Between the Dots `http://blogs.verisigninc.com/blog/entry/uncontrolled_interruption_dozens_of_blocked`, February 2014.

[17] D. Kaminsky. Explorations in namespace: White-hat hacking across the domain name system. *Commun. ACM*, 49(6):62–69, June 2006.

[18] Mirjam Kuhne. Update on as path lengths over time. `https://labs.ripe.net/Members/mirjam/update-on-as-path-lengths-over-time`.

[19] P. Mockapetris and K. J. Dunlap. Development of the domain name system. In *SIGCOMM '88*, pages 123–133, 1988.

[20] A. Mohaisen, H. Mekky, X. Zhang, H. Xie, and Y. Kim. Timing attacks on access privacy in information centric networks and

countermeasures. *IEEE TDSC*, 2015.

[21] R. V. Oliveira, B. Zhang, and L. Zhang. Observing the evolution of internet as topology. In *SIGCOMM*, pages 313–324, 2007.

[22] E. Osterweil, M. Ryan, D. Massey, and L. Zhang. Quantifying the operational status of the dnssec deployment. In *IMC '08*, 2008.

[23] K. Schomp, T. Callahan, M. Rabinovich, and M. Allman. On measuring the client-side dns infrastructure. In *Proceedings of the 2013 conference on Internet measurement conference*, pages 77–90. ACM, 2013.

[24] C. Wang, J. Li, F. Ye, and Y. Yang. Netwrap: An ndn based real-timewireless recharging framework for wireless sensor networks. *Mobile Computing, IEEE Transactions on*, 13(6):1283–1297, 2014.

[25] M. Xie, I. Widjaja, and H. Wang. Enhancing cache robustness for content-centric networking. In *INFOCOM, 2012 Proceedings IEEE*, pages 2426–2434. IEEE, 2012.

[26] Y. Yu, D. Wessels, M. Larson, and L. Zhang. Authority server selection in dns caching resolvers. *ACM SIGCOMM Computer Communication Review*, 42(2):80–86, 2012.

[27] L. Zhang, D. Estrin, J. Burke, V. Jacobson, J. Thornton, D. Smetters, B. Zhang, G. Tsudik, D. Massey, C. Papadopoulos, et al. Named data networking (ndn) project. Technical report, PARC, 2010.

[28] L. Zhu, Z. Hu, J. Heidemann, D. Wessels, A. Mankin, and N. Somaiya. Connection-oriented DNS to improve privacy and security. In *Proceedings of the 36th IEEE Symposium on Security and Privacy*, San Jose, Californa, USA, May 2015. IEEE.