Title

Name Space Analysis: Verification of Named Data Network Data Planes

Permalink

https://escholarship.org/uc/item/9w6426j2

Journal

IEEE/ACM Transactions on Networking, 29(2)

ISSN

1063-6692

Authors

Jahanian, Mohammad
Ramakrishnan, KK

Publication Date

2021-04-01

DOI

10.1109/tnet.2021.3050769

Name Space Analysis: Verification of Named Data Network Data Planes

Mohammad Jahanian and K. K. Ramakrishnan, *Fellow, IEEE*

***Abstract*—Named Data Networking (NDN) has many forwarding behaviors, strategies, and protocols to enable the benefits of Information-Centric Networking. This additional functionality introduces complexity, motivating the need for a tool to help reason about and verify that basic properties of an NDN data plane are guaranteed. This paper proposes Name Space Analysis (NSA), a network verification framework to model and analyze NDN data planes. NSA can take as input one or more snapshots, each representing a state of the data plane. It then provides the verification result against specified properties. NSA builds on the theory of Header Space Analysis, and extends it in a number of ways, *e.g.*, supporting variable-sized headers with flexible formats, introduction of name space functions, allowing for name-based properties such as content reachability and name leakage-freedom, and multi-snapshot verification such as equivalence checks. These important additions reflect the behavior and requirements of NDN, requiring modeling and verification foundations fundamentally different from those of traditional host-centric networks. As a case study, we show how NSA can detect name space conflicts in NDN, which can be often hard to catch. Leveraging the learning from this study, we outline a conflict detection and resolution protocol and a name space registry to avoid such conflicts. We have implemented NSA and identified a number of optimizations to enhance the efficiency of verification. Results from our evaluations, using snapshots from various synthetic test cases and the real-world NDN testbed, show how NSA is effective, in finding errors, has good performance, and is scalable.**

***Index Terms*—Named Data Networks, Network Verification**

I. Introduction

NAMED DATA NETWORKING (NDN) [1], [2] provides a content-aware network layer where information is accessed over the network without necessarily focusing on its location or the underlying mechanisms used to retrieve that information. To enable this location-independence, NDN supports name-based forwarding, and in-network caching, thereby improving performance and availability. NDN routers primarily rely on a Forwarding Information Base (FIB), Content Store (CS) and Pending Interest Table (PIT) with reverse path forwarding to deliver Data associated with an Interest [2].

The flexible structure of NDN supports a wide variety of network functions and applications. On top of basic PIT, CS, and FIB checks, additional packet processing such as forwarding hint processing [3], rate-based forwarding [4], and hyperbolic forwarding [5] have been adopted and incorporated into the standard NDN Forwarding Daemon (NFD) [6]. Additionally, a number of useful extensions to the core NDN packet processing have been proposed in the literature to potentially be part of any NDN network, such as path switching [7], Interest anonymization [8], [9], name resolution [10], cache-aware forwarding [11], *etc.* While these network functions, whether deployed in separate middleboxes or softwarized into a basic ICN router, make NDN powerful, they may make the network's data (forwarding) plane, more complex. As a result, it is very useful to ensure that the data plane, *i.e.*, the forwarding and processing rules for packets, is correct. To tackle this, an automated framework to model and verify NDN network data planes would be highly desirable.

Network verification [12]–[21] is an active research area, useful in analyzing large, complicated networks in order to ensure a network is free of bugs and corner-case errors, investigating essential properties such as reachability and loop-freedom. Data plane verification focuses on analyzing a particular (*e.g.*, the current) forwarding state, *i.e.*, data plane, of the network. These tools normally rely on a formal foundation that covers a large space of possibilities. They can be automated and applied to network snapshots, representing the data plane. While these tools have focused on IP networks and are powerful in verifying host-centric properties, they can be extended and integrated for use in an ICN-based environment such as NDN.

We propose Name Space Analysis (NSA), a framework for modeling and verification of NDN data planes. NSA is based on the theory of Header Space Analysis (HSA) [13]. HSA uses a geometric view of packet headers, where each packet header is generally modeled as a point in a space and network functions transform that point to another one within that space. Additionally, the ability to analyze a "space" rather than a "single point", makes this an efficient analysis approach. This flexibility and efficiency make it a good formalism for integration in the analysis of NDN. We add another geometric space in NSA, namely the *name space*, and a new function, *name space function*, that transforms a point in the header space domain to a (collection of) point(s) in the name space domain. We extend HSA by enabling flexible atoms and variable-size wildcards to model headers (to support NDN-specific packet formats [22]), and adding name spaces as an essential part of the analysis. Analyzing name spaces in NDN is necessary and very useful as they are key to accessing content. We propose NDN-specific properties that can be checked by NSA; *e.g.*, in NDN we are interested in verifying host-to-content reachability, rather than the host-to-host reachability requirement expected of traditional host-centric IP networks. NSA has a number of verification applications (to prove key properties), namely content reachability test, name-based loop detection, and name leakage detection. We additionally support verification applications that go across multiple snapshots to analyze

M. Jahanian and K. K. Ramakrishnan are with the Department of Computer Science and Engineering, University of California, Riverside, CA, 92521 USA. E-mails: mjaha001@ucr.edu and kk@cs.ucr.edu

This work was supported by the US Department of Commerce, National Institute of Standards and Technology (award 70NANB17H188) and US National Science Foundation grant CNS-1818971.

changes between multiple states (*e.g.*, consistent producer mobility check) and report on their equivalence. The complex structure of names in NDN may cause issues, *e.g.*, interfering prefix announcements by two non-coordinated data producers, which can potentially lead to blackholes. We call this *name space conflicts* and show how NSA can identify them (§VIII-A). The importance of having a name management method in NDN has been identified in [23], [24]. Using the concepts of NSA, we propose a name registration method that can catch and resolve such conflicts in the data plane (§VIII-B). We implemented NSA [25], including all the essential components of our design: name atoms, set operations, transfer functions, state space generation, and verification applications. We also identified a number of optimizations, and evaluating our implementation on synthetic snapshots and real-world NDN testbed snapshots shows that NSA is effective, efficient and scalable (§VII).

Overall, the contributions of this paper are: 1) a framework for verification of NDN data planes, focusing on the nature of NDN, rather than the previous tools for host-centric architectures; 2) modeling name spaces and name space functions, as they are the main assets required to access content in NDN; 3) specifying essential NDN-specific properties and approaches to analyze them (content reachability test, name-based loop detection, name (space) leakage detection, and cross-snapshot equivalence checks); 4) studying the practical issue of name space conflicts in NDN and guidelines for a conflict detection and resolution engine; 5) an implementation of NSA [25] with its optimizations; and 6) demonstrating NSA's applicability to the real-world NDN testbed [26].

II. Background and Related Work

A. Overview of Header Space Analysis

Header Space Analysis (HSA) [13] is a network data plane verification tool used to model nodes and verify essential properties. A network node is any packet processor that performs in-network processing on a packet on its path. The most important primitives in HSA are *Header Space*, *Network Transfer Function*, and *Topology Transfer Function*.

Based on a *geometric model*, a Header Space H is an L-dimensional space of packet headers, with L being the upper bound on header length, in bits. One header is one point in this L-dimensional space, consisting of 0's and 1's. A special wildcard bit 'x' can be used to form a header space that constrains only certain bits. HSA defines primitive set operations (union, intersection, *etc.*) to manipulate header spaces.

Using these operations and conditionals, we can define Transfer Functions. A Network Transfer Function T models the packet processing done by a network node. Function $T(h, p)$ takes as input a header space and incoming port, and produces a new (h', p') pair denoting what header space will be produced as output, and which port it has to go out of.

The Topology Transfer Function Γ models link behavior. Assuming the link is up and working, this function basically relays the header, unchanged, from the output port of one node to the input port of the next node, assuming the two ports are connected by this link. Using a long-lived snapshot, we can model a topology of fixed, wired links, while a sequence of short-lived snapshots may be used to capture the effects of a mobile, wireless environment.

Using the aforementioned building blocks, HSA provides algorithms to check the following properties in a network configuration: *Reachability Analysis*, *Loop Detection*, and *Slice Isolation*. The analyses typically consist of an initial header space injected to a (set of) network node(s). The higher the coverage of these header spaces, the more thorough the search will be. Usually for a full analysis, initial header spaces of all wildcard bits are injected. Reachability analysis gives all the headers that a node B receives, starting from an initial header space injected at a node A. For loop detection, the history of a header space is checked, to see whether or not (a part of) it has visited a node more than once. Slice Isolation uses header spaces flowing in and out of critical network nodes, to ensure certain traffic stays within a private network slice, *e.g.*, a VLAN, and does not leak to another slice.

B. Network Verification and NDN Diagnostics

Network verification aims at analyzing large, complicated networks in order to find corner case errors and investigate essential properties. There have been efforts to build models to describe and verify networks. For the purpose of building verification frameworks, some works focus on analyzing control plane (to analyze all data planes caused by configurations) and some on data plane (to analyze the current state of the network). Computational feasibility and full verification coverage are challenges of control plane verification [17], [27]. We focus on data plane verification in this paper. Some of the more notable data plane verification tools are Anteater [12], HSA [13], VeriFlow [16], and NetPlumber [14]. These methods typically consist of snapshot-based static checking. Anteater [12] models the data plane as a set of boolean expressions and runs a SAT solver to verify invariants. HSA [13] uses a geometric view of packet headers, not making any presupposition about what each packet header element represents, thus making it a flexible model for integration for new network architectures. Some verification tools additionally support real-time checking of network policies of Software-Defined Networks (SDN) such as VeriFlow [16] and NetPlumber [14]. These methods leverage and rely on control update messages issued by the centralized SDN controller for fast, incremental checking of network data planes. Thus, they can react to changes before those changes are applied to every one of the associated routers. Our proposed model is a generic one, with no assumption on how the network is managed. However, if we have NDN integrated with SDN, real-time verification using control update messages may be leveraged. Work in [17], [20] propose data plane equivalence checks. While they focus on equivalence pertaining to host-centric properties, NSA can check information-centric equivalence, *e.g.*, checking if the same subset of the content namespace of a particular content provider is reachable in two (or multiple) data plane snapshots. This is important since we may need to have multiple snapshots each with the desired differences, and compare them against each other, *i.e.*, cross checks, where the goal is not to conform to an external property, but rather to compare against a complete, separate snapshot in time [20].

An important prerequisite of data plane verification is collecting the current state of the data plane in the form of a snapshot. Based on how the network is managed, different methods can be used for this collection procedure. With traditional non-SDN networks, methods such as SNMP [28], NETCONF [29] or node-specific terminals [30], can be used to collect FIBs and topology information. NDNconf [31] presents an NDN-ized version of NETCONF, to collect NDN-specific FIBs. In addition to the capability of querying, NDNconf allows for a push-based notification of changes in the network state to management servers, which helps with real-time collection of up-to-date snapshots. SDN-controlled networks can support a more efficient snapshot collection by monitoring the forwarding rule updates (insert, modification or deletion) on the southbound interface [32]. Snapshot collection and verification are two logically independent procedures. NSA focuses on the verification component, while leveraging the complementary support of these snapshot collection methods.

Our work presents an NDN-specific verification framework. Diagnostic tools such as Ping [33] and Traceroute [34]–[37] have been proposed and developed for NDN. While these tools are very helpful for performance measurements and small-scale connectivity checks, they are often limited in high-coverage checks across the network in a scalable way, and also use network resources. Thus, a formal approach gives us a higher level of flexibility and coverage for property checking [13].

III. Overview of NSA

Fig. 1 shows the overall functionality provided by NSA, what specific building blocks it proposes, and the ways in which it extends and integrates HSA for NDN. HSA leverages a number of functions, using header primitives to enable verification. Each verification application analyzes a particular network property. As Fig. 1 shows, NSA is designed to be modular, so it can be extended to support additional verification applications.

HSA is most suitable for analysis of protocols with headers having fixed formats, *e.g.*, IP packet headers, where (mandatory) fields have fixed sizes and positions, according to the protocol version. Since this is not the case for NDN packets [22], we develop NSA to support the modeling of NDN-style flexible headers with variable fields. NSA introduces variable-length wildcard elements to enable modeling NDN's header space. Also, we change HSA's bit-based header space modeling to a flexible atom-based one. Atoms can be bytes (octets), fields, or names. This allows us to reasonably model how NDN packets are encoded, at the desired level of abstraction.

While utilizing HSA network and topology functions, NSA also proposes a *Name Space Function*, which enables transforming a header space into a *name space*, which is an essential part of a content-oriented network. Details of NSA elements are provided in §IV. Using this function, a wide variety of name-based network properties can be reasoned about. Almost all of the HSA verification applications can also be used to analyze NDN. However, there are additional properties, specific to an ICN/NDN, that need to be addressed. This necessitates a verification tool that takes NDN specifics and intents into account. To this end, we introduce some applications that use these additional properties in NSA, namely *Content Reachability Test*, *Name-based Loop Detection* and *Name Space Leakage Detection*. These properties focus on the specifics related to the NDN architecture, and how consumers interact with *named content*. Additionally, NSA supports *Cross-Snapshot Equivalence Check*, which takes as input multiple snapshots (states of the data plane), and can be used to check a variety of properties, *e.g.*, consistent NDN producer mobility. Details of NSA's verification applications are provided in §V.

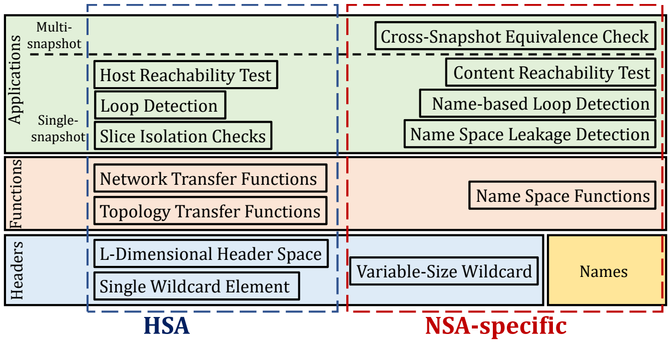


Fig. 1. Overall framework of NSA

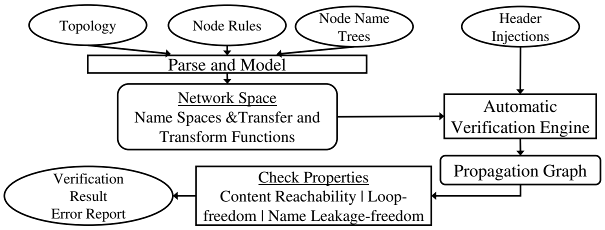

Topology
Node Rules
Node Name Trees
Header Injections
Parse and Model
Network Space
Name Spaces &Transfer and Transform Functions
Automatic Verification Engine
Propagation Graph
Check Properties
Content Reachability | Loop-freedom | Name Leakage-freedom
Verification Result Error Report

Fig. 2. Overall procedure of NSA

NSA, just like HSA and other data plane verification approaches, focuses on a snapshot, and models how the state of packets change with regards to a given network state, but not how the state of the network itself changes. In other words, NSA reads and writes to state at a packet-level, as explained in §V (Fig. 3(b)), and only reads from state at the network-level. Thus, it focuses on a set of properties that are dependent on a particular state of the network, and not on how other packets may change it. The properties cover all packets and their paths in the network state. However, this is still a huge improvement in terms of coverage of analysis compared to existing solutions [33]–[37], and allows for important classes of properties such as reachability and loop-freedom [12], [13]. At any state of the network (except for temporary transient states, perhaps), each content must be correctly reachable from anywhere, and packets must not loop. Modeling the transition of network state from one data plane state to another will require control plane verification approaches as well. It is feasible to analyze a finite, limited number of data plane snapshots (*e.g.*, NSA's cross-snapshot equivalence check), each representing a state of the network, by successively running NSA on those snapshots. An example of multi-snapshot verification is checking if the network's handling of producer mobility is correct (§V-D).

Overall NSA procedure. We outline the procedure in outlined in Fig. 2. It consists of the following:

- The data plane information is fed to NSA as inputs containing *Topology* (*i.e.*, the links' information), *Node Rules* (such as FIB rules or PIT state at routers), and *Name Trees* at nodes. All these elements, combined, constitute the *current state of the network*, inserted as a snapshot, collected through the methods described in §II. If the snapshot does not capture the full global information and is only a subset, NSA focuses its verification on that particular subset.
- After parsing these inputs, NSA models the data plane, called a *Network Space*, by generating the representative *Name Spaces*, and *Transfer and Transform Functions*, to model node operations, links, and mapping between headers and names.
- Another important input for NSA are *Header Injections*, which trigger the verification procedure. These "headers" are symbolic packets which can contain logical elements such as wildcards. For a full test, we typically inject all-wildcard headers at all node faces in the network.
- Based on the header injections and the generated network space, NSA's *Automatic Verification Engine* generates the *Propagation Graph*, which is basically the state space of all packet transitions and paths. The topology and network rules existing in the collected snapshot will dictate these transitions (if we want to consider changes in the network space, we will have to collect new snapshots and process them separately).
- NSA checks network-wide properties, such as loop-freedom, content reachability, and name leakage-freedom, by querying over the generated propagation graph. This way, NSA provides the verification result and error report.

IV. NSA Design

A. Modeling NDN Header Space

1) Atoms and Header Representation: The atoms of analysis in HSA are bits, since some fields can be encoded as single bits in IP. An NDN packet, on the other hand, is a set of nested Type-Length-Value (TLV) codes represented as octets [22]. Thus, the smallest possible atom in NSA is octets (bytes). With byte-based atoms, NSA header representations follow NDN's TLV octet-based encoding. Other atoms could be picked as well: *e.g.*, if checking the correctness of TLV encoding is not important in a particular analysis, atoms can be NDN fields. With field atoms, NSA header representation will be an XML-like structure. If only the name field needs to be checked, atoms can be names. With name atoms, NSA headers are represented as a combination of name components, similar to NDN regular expressions [38]. Unlike HSA's strict use of bit atoms, NSA provides the flexibility of using byte, field and name atoms for header representation. The correct atom depends on the scope of verification and the desired level of abstraction.

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

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

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

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

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

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

Note that the "[∗]" wildcard is not currently part of the NDN architecture [39]; we use it as part of NSA headers for the model's representation and verification efficiency, to be used in a *symbolic execution* fashion, which we explain in §V.

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

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

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

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

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

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

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

B. Modeling NDN Nodes

Packet processing in an NDN node uses Network Transfer Functions, as $T(h, f) : T(h_0, f_0) \rightarrow \{(h_1, f_1), (h_2, f_2), \ldots\}$ where function T maps header h_0 coming to face f_0, to all headers h_1, h_2, *etc.*, going out of faces f_1, f_2, *etc.* of the node respectively. NSA's transfer functions are at the level of a face, rather than being port-level as in HSA. Domain and range of NSA transfer functions are of the same type (both Interest or both Data headers). Transitioning from Interest to Data is not a part of NSA verification as it requires changing the state of the data plane. Depending on the functionality being modeled, function T may or may not change h_0, and may or may not depend on the incoming face f_0. Any NDN packet processing, including an NDN forwarding behavior, can be modeled using (a set of) transfer functions.

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

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

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

In general, a typical Interest processing transfer function can be modeled as $T_I(.) = T_{I.fwd}(T_{I.CS}(T_{I.PIT}(.)))$. What elements we put into a transfer function depends on our architecture and the purpose of the analysis. For example, if we have the assumption of the CS and PIT being empty upon the arrival of an Interest, then we can simply have $T_I(.) = T_{I.fwd}(.)$. Additional functions can be added to the pipeline as well; more generally, as $T_n(T_{n-1}(\ldots T_1(.)))$ where each T_i is a specific function (step) in the pipeline. These functions include those that modify the incoming header space as well. More example transfer functions, *e.g.*, a packet anonymizer transfer function, are provided in [40].

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

C. Modeling Name Spaces

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

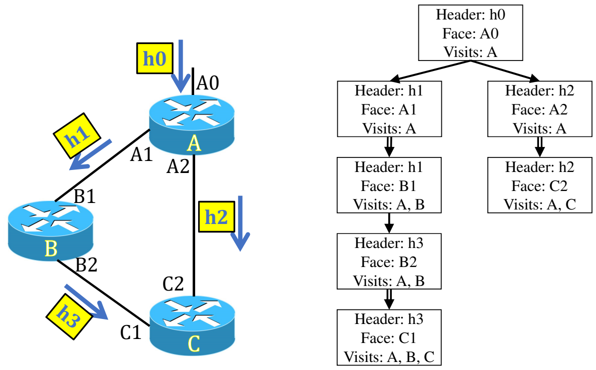
Formally, a name space in NSA represents names and their relations, and is a domain separate from the header space domain. A name space function, transforms a point in the header space domain to a name space domain, *i.e.*, its corresponding name(s). Name space function $\Omega()$ is introduced in NSA. It transforms a (set of) header space(s) (after parsing it to the individual name parts) to a name space. In particular, $\Omega()$ performs the following two steps on an input header space h: 1) extracts the (prefix) names associated with h, 2) provides the reverse construction of the prefix tree from the list of prefixes derived in step 1. This resulting prefix tree is the name space, used in NSA verification applications. It is worth mentioning that in this case, since the name space is generated from an L-bound header space (via $\Omega()$), the height of the name space will be at most L. This does not take away any generality from our analysis: even though a name tree can be potentially infinite, any name component that is beyond the limit of an Interest size cannot be expressed and thus cannot be reached. Thus, they are not visible to the outside world (in other words, we cannot define a "reachability" requirement for them), and can be safely omitted from our verification.

V. Using NSA for Verification

We present a number of verification applications of NSA. Specifically, we look at the important applications of testing content reachability, loop detection and name leakage detection. NSA provides significant benefits both for verification results and its efficiency compared to simulation-based tests.

An important part of NSA's formal verification approach that facilitates automated checking is the generation and analysis of the state space, or *propagation graph*. The graph represents all possible *paths* any packet can take, rather than a single trace that a simulation-based approach would support. This provides the desired coverage we need for verification. An example is shown in Fig. 3. Each node in this propagation graph is a *state*, mainly consisting of a header and a face, denoting the arrival/departure of the header to/from the face. Depending on the specific application, there can be additional state information, such as a visited nodes list, *e.g.*, for loop detection. We record as much information within a state (*e.g.*, list of visited nodes) as needed, so that checks could be done by looking at the state only, so that extra processing on the graph would not be necessary (*e.g.*, checking all ancestors of a state by traversing paths). The initial states, *i.e.*, parent-less nodes in the graph, represent injections to the network. For example, in Fig. 3, the propagation graph (Fig. 3(b)) implies that header $h0$ is injected to face $A0$ of node A (as shown in Fig. 3(a) as well). State transitions in the propagation graph can be through network transfer functions (*i.e.*, processing packets within a node) represented by single arrows in Fig. 3(b), or topology transfer functions (*i.e.*, moving over physical links) shown by double arrows.

While NSA can be used for both Interest and Data packets, we focus on Interests in the remainder of this paper. As pointed out in [6], Interest processing is more complicated than Data processing: it has longer, more complicated pipelines, has additional procedures such as forwarding strategy selection, and its forwarding decisions are made through the result (*i.e.*, FIB)



(a) Topology and header spaces (b) Propagation graph

Fig. 3. Propagation graph example

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

A. Content Reachability Test

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

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

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

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

1) If part of NS_B^{rcv} is not in NS_B^{hos} (Case 1: *unsolicited names*), it means B would receive Interests for names the node does not have, *i.e.*, those packets get blackholed.
2) If part of NS_B^{hos} is not in NS_B^{rcv} (Case 2: *unreachable names*), part of B's name space is untouched, *i.e.*, requests for them would never be received. Cases 1 & 2 need not be disjoint.
3) If neither of the cases occur, verification is successful, *i.e.*, $NS_B^{hos} = NS_B^{rcv}$ (Case 3).

The process is exemplified in Fig. 4, where header space h_A injected at host A traverses nodes (*e.g.*, routers) with transfer functions T_C and T_B where the header space h_B gets transformed and compared with the content name space at B. Node B can generally be any node in the network that has the capability of storing and serving content, be it a content publisher or an ICN-capable router with content store.

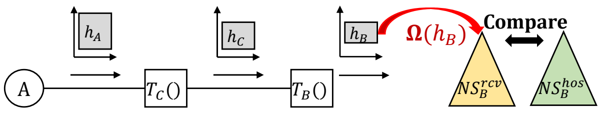


Fig. 4. Content reachability test

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

Algorithm 1 Content Reachability Test

```
procedure ConReach($C, h_0, N$)                ▹ Inject $h_0$, at $C$, netw. space $N$
    Start with $h_0$ at $C$                      ▹ Typically all wildcard, i.e., [*]
    Calculate all $h_{P_i}$'s                    ▹ Headers reached at provider $P_i$
    for all $P_i$ do
        $NS_{P_i}^{rcv} \leftarrow \Omega(h_{P_i})$
        $NS_{P_i}^{UW} \leftarrow NS_{P_i}^{rcv} - NS_{P_i}^{hos}$      ▹ 'Unsolicited' names
        $NS_{P_i}^{UR} \leftarrow NS_{P_i}^{hos} - NS_{P_i}^{rcv}$      ▹ 'Unreachable' names
        if $NS_{P_i}^{UR} \cup NS_{P_i}^{UW} = \varnothing$ then
            $Result_{P_i} \leftarrow$ True                ▹ Success at $P_i$
        else
            $Result_{P_i} \leftarrow$ False               ▹ Failure at $P_i$
    return $\bigwedge_{\text{all}P_i\text{'s}} Result_{P_i}$       ▹ Overall verification result
```

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

- **Content censorship-freedom.** Censorship leads to content reachability errors; *e.g.*, in Fig. 5, censoring node R may drop all interests for "`/democracy`" [8]. This would result in (all or part of) content provider P's name space to be unreachable. This is an undesired effect that can easily be detected by NSA. While NSA cannot definitively deduce that such a problem is caused by content censorship, the lack of existence of such errors would imply content censorship-freedom. Furthermore, the effectiveness of a censorship countermeasure mechanisms can be checked using NSA.
- **Content neutrality.** We define *Content Neutrality* as not favoring a content provider over another (by not discriminating), with regards to same prefixes that they serve. An example of content neutrality violation is shown in Fig. 6, where content provider $P3$'s served name space "`/news`" is not reachable, even with the Interest going through nodes with multicast forwarding strategies enabled (which are supposed

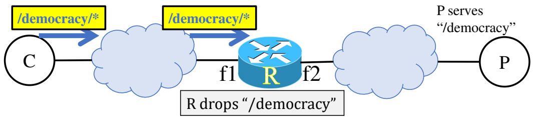


Fig. 5. Content censorship example

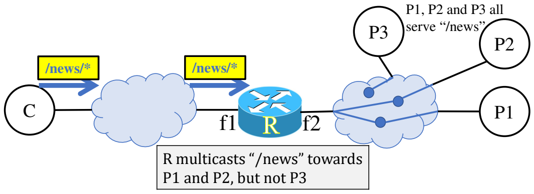
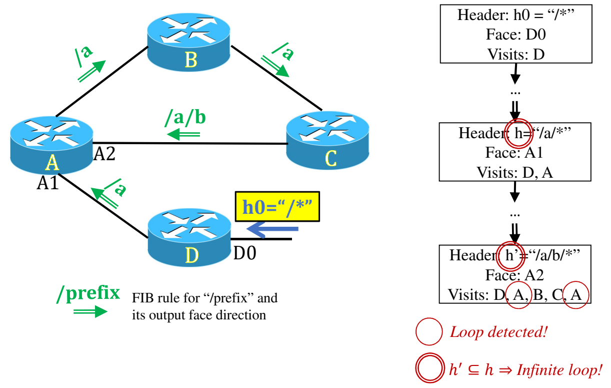

P1, P2 and P3 all serve "/news"
P3
P2
/news/*
/news/*
C
f1
R
f2
P1
R multicasts "/news" towards P1 and P2, but not P3

Fig. 6. Content neutrality violation example



(a) Topology, injection, and FIB rules (b) Propagation graph

Fig. 7. Loop detection example

to send Interests towards the direction of *all* potential content providers.) Here, router R (which can be part of an ISP) is discriminating by favoring $P1$ and $P2$, over $P3$, for Interests requesting name "`/news`" (all three may be news organizations, thus producers). With multicast forwarding strategy at every router for every prefix, NSA can check whether all content providers receive Interests matching their entire name space, for every 'all-wildcard' injection. While NSA cannot detect if a reachability error is caused by discriminatory neutrality violation or benign configuration mistakes, an error-free data plane could be used to show if content neutrality holds.

A number of other use cases of NSA's content reachability test, *e.g.*, checking route computation outcome correctness, and security infrastructure soundness are provided in [40].

B. Loop Detection

Loop freedom is an important property in networks. For NDN in particular, looping Interests is a widely known issue, which led to the addition of extra processes in the forwarding pipelines, such as a Dead Nonce List [6]. While such reactive measures detect looped Interests after they occur, looped Interest would not be prevented and could potentially waste a large amount of network resources. Also, it is very likely that an Interest is looping because it is not satisfied; *i.e.*, did not reach its intended content provider(s) due to errors in the forwarding state of the network. As a result, making a local decision at an NDN router to discard or drop a looping Interest does not solve the problem of unsatisfiability of certain Interests. Thus, it would be highly desirable to detect all potential loops in a data plane, before they occur, with a holistic view of the network data plane.

NSA helps in identifying all Interests that might potentially loop. NSA typically does this by injecting all-wildcard headers and looking for possible loops. Thus, we can track every possible Interest and find all potential loops by following FIB rules established in a given data plane. We therefore achieve a purely *name-based* loop detection, rather than a *nonce-based* detection. NSA models the transition of all packets within a single data plane snapshot, thus enabling a robust loop detection algorithm (as does HSA [13]). As all FIB rules causing the loops are contained in one single snapshot and it is possible to analyze them with transitioning packets (headers), NSA can catch all potential loops.

The loops detected can be potentially infinite or finite. Suppose node A appears twice in a single path in the propagation graph, visiting two header spaces h and h' (in that order); if $h' \subseteq h$, then this would be a potential infinite loop. An example is shown in Fig. 7, where NSA first detects a loop (as node A appears twice in one particular path), and second, it determines the loop to be infinite, checking the header spaces h and h' associated with the visits, where headers with name "`/a/b/`$*$" return back to node A. Having $h' \cap h = \varnothing$ implies a certainly finite, thus non-hazardous, loop which NSA ignores. By adding the history of each state to NSA, *i.e.*, the sequence of headers and faces, NSA can easily detect infinite loops by checking whether a particular header space (subset) has been visited by a node twice or not.

C. Name (Space) Leakage Detection

What if a consumer issuing an Interest for a particular name, wishes (parts of) the name, *e.g.*, his ID or a particular content name, to not be visible in the network except for certain authorized nodes, *e.g.*, those in his home network? This can be a desirable property for a variety of reasons. Works such as [9] have identified the need for Interest name privacy.

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

Let us consider the example in Fig. 8: Consumer C issues Interests with header h_0, which results in headers h_1, h_2 and h_3 leaving the authorized zone of routers, denoted as $Z1$. We define all the headers going out of $Z1$ as $h_{out} = h_1 \cup h_2 \cup h_3$. NSA allows us to define and apply access control rules on names in a number of ways, and check name constraints on h_{out} accordingly, *e.g.*, the following examples:

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

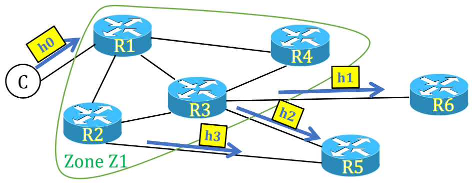
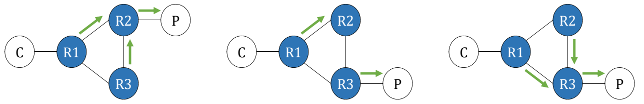


Fig. 8. Name leakage detection example



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

Fig. 9. Data plane state changes due to producer mobility (P serves "/a", green arrows: FIB entries for "/a")

side means that we do not want any header in the result of the intersection to leave Z1.

- Packets associated with name space NS_0 should not leave $Z1$; then we should have $\Omega(h_{out}) \cap NS_0 = \emptyset$. This way of defining a rule is more efficient for a larger set of prefix-suffix name relations representing a portion of a name space graph: instead of checking many prefixes one by one, we can check once against name space NS_0 comprising all those prefixes.

D. Cross-Snapshot Equivalence Check

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

Example use case: Producer Mobility Correctness. Mobility is a major feature of NDN. However, especially when it comes to producer mobility, handling it in a correct way (*i.e.*, making sure the producer's content reachability properties stay the same after the mobility and network re-convergence) can be quite challenging [41]. We can use NSA's Cross-snapshot Equivalence Check to check this correctness property.

Let us consider two snapshots $s1$ and $s2$ of an NDN network, where $s1$ and $s2$ are identical in every way except that the network point of attachment of producer P is different in the two snapshots, as depicted in Fig. 9. In other words, state $s1$ is collected before P's move and $s2$ is collected after P has moved. The state in the network (i.e., FIBs in the routers) has been re-populated and routing convergence, according to the protocol, has been partially or completely achieved.

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

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

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

Fig. 9 shows a simple mobility example, where producer P, which serves name prefix "/a" moves from its initial point of attachment $R2$ (initial snapshot, $s1$, Fig. 9(a)) to $R3$ (finals snapshot, $s2$, Fig. 9(c)). For simplicity, we consider a naive routing-based mobility handling solution that re-populates all FIBs with an updated announcement after the new attachment. An intermediate state ($s1.5$, Fig. 9(b)) shows the state after P's move but before full re-convergence of the network ($R3$ has been notified of the update, but $R1$ and $R2$ have not yet been notified). Using the mobility equivalence property EP defined above, we will have $s1 \equiv_{EP} s2$, but $s1 \not\equiv_{EP} s1.5$ since the range of $CR^{s1}_{C\to P}$ and $CR^{s1.5}_{C\to P}$ do not match; in other words, interests for "/a" from C that reach P in $s1$, do not do so in $s1.5$. Similarly, $s1.5 \not\equiv_{EP} s2$. This example shows that during the transition, the network is temporarily incorrect. The property needs to ultimately hold for the initial and final snapshots. Also, a fast mobility solution, creates erroneous intermediate snapshots that are fewer and last for shorter durations.

Furthermore, checking two different intermediate snapshots of two different mobility solutions can be helpful. *E.g.*, suppose $R2$ in Fig. 9(b) has received an 'invalidation' signal from P once it moves. NSA gives us the full header space leaving $R2$ in the two cases: 'without invalidation message' *vs.* 'with invalidation message' (as two intermediate snapshots). The smaller size of the latter shows that fewer interests will be blackholed by using the invalidation message.

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

Consumer mobility can be checked for correctness in a similar manner. Cross-snapshot equivalence check application can be very useful to check the *outcome* (not the protocol itself, per se) of mobility handling protocols on the data plane, especially with regard to how they re-populate the network FIB or re-direct requests. This may be complimentary to other protocol verification methods used for specific mobility handling protocols [41].

VI. Complexity Analysis

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

Next, we analyze the time complexity of the execution of the verification procedures on prepared network space, starting with content reachability (§V-A). Let us assume we have an injection of a header at *one* consumer, that leads to *one* content provider through a single path. Let us also define d, L, R, and s as maximum network diameter (number of hops), maximum header length, maximum number of node rules, and maximum number of paths in a trie-based content provider name space, respectively. The time complexity of the NSA content reachability test will then be $O(dLR^2 s)$. This analysis is based on the *linear fragmentation* assumption in [13], which says that typically very few rules in a node match an incoming packet. On the other hand, the complexity of a simulation-based test (as well as ping or traceroute) would be $O(da^L Rs)$, where a is the maximum number of values an atom can take; *e.g.*, with byte-based atoms, a would be 256. NDN headers have no specific upper bound; however, it is recommended that a reasonable MTU (which can be thousands of bytes) be conformed to by NDN applications [42]. This will make a^L very large. This way, the simulation approach will reach a very large and exponentially growing complexity. This shows the huge benefit of NSA for a content reachability analysis with high coverage.

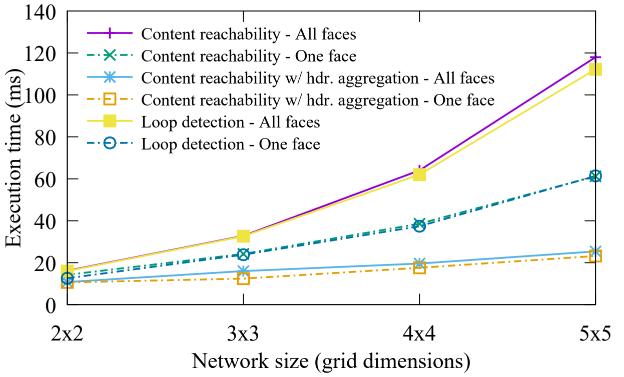


Fig. 10. Results for small grid snapshots

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

VII. Evaluation

We have implemented NSA, including its main components and modules, in Java; the source code is available at [25]. We start by evaluating the performance of NSA using synthetic grid and ring topologies, and then apply it to the NDN testbed topology for evaluating a network that is actively used [26]. All evaluations have been done on a machine with Ubuntu 14.04.6 LTS using Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz dual-socket with 14 cores each with hyper-threading enabled, and 252GB RAM. We do not utilize the whole RAM capacity though; we set the maximum memory heap size of our Java Virtual Machine (JVM) to 10GB only. For each verification application, all wildcard headers, *i.e.*, “/$*$” is injected to all faces or nodes. While reporting our evaluation results, we identify and present optimizations that further improves NSA's performance.

A. Synthetic Networks

1) Content Reachability Analysis and Loop Detection: To evaluate NSA's content reachability analysis and loop detection we use customized $n{\times}n$ grid topologies (to allow many branches

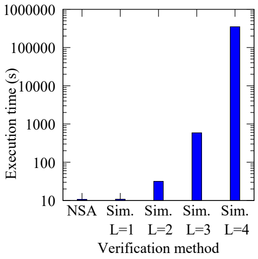


Fig. 11. NSA & simulation-based check

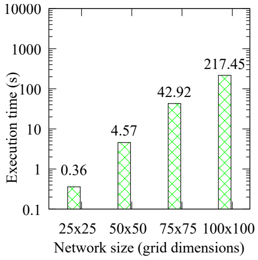


Fig. 12. Large grids, vary network size

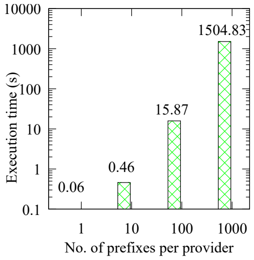
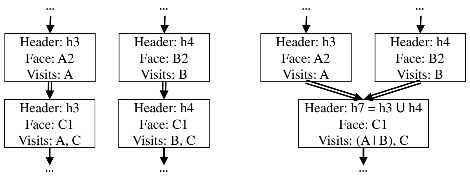


Fig. 13. Large grids, vary prefix count



(a) Original propagation graph (b) Aggregated propagation graph

Fig. 14. Propagation graph aggregation example

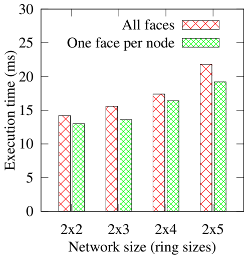


Fig. 15. Name leakage

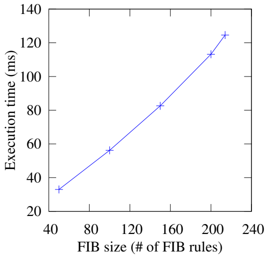


Fig. 16. Rule generation (‘UCI’ node)

TABLE I
EXECUTION TIME (MS) FOR NDN TESTBED VERIFICATION

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

in the propagation graph), with n publishers in each case, each serving one distinct prefix; these prefixes are advertised and populated in every node's FIB in the snapshot being verified. Verification performance results for these grid networks are presented in Fig. 10, in terms of execution times, in milliseconds.

First, Fig. 10 shows the execution time of content reachability on the grid networks. This verification, as explained in §V-A, checks both unreachable and unsolicited names. Typically, NSA injects all-wildcard headers into all faces, since some node rules may depend on the incoming faces (‘All faces’ in Fig. 10. As seen in the Fig., the growth of execution time for ‘All faces’ injection mode is linear with respect to the input network size growth (note the input growth on x-axis is n^2). Since we are only dealing with FIB rules that do not depend on the incoming face, we can limit our injection to ‘One face per node’ injection only. This would not change the outcome of the verification results. Fig. 10 shows that this optimization significantly improves the performance of NSA, which is due to the fact that its fewer number of injections leads to smaller propagation graph.

For the full reachability check, we need to go through a separate propagation graph fragment, built and checked for each injection, to check both unsolicited and unreachable names. If our goal is to only check unsolicited names (and not unreachable names), we can make all injections at once into a single propagation graph fragment, aggregating the headers (Fig. 14). This way, we preserve all reached header spaces, but not their exact paths from origin in the visited list. Fig. 10 also shows the significant performance enhancement of this optimization, compared to full reachability analysis, if our goal is only to detect unsolicited names.

The use of wildcards is an important benefit of NSA (and HSA), compared to simulation-based methods (which have to generate all possible packets within a range), as shown asymptotically in §VI. We show the empirical results for the use of wildcards in Fig. 11. Each simulation scenario (‘Sim’) is a typical simulation-based content reachability analysis (using the aggregation optimization with the sole purpose of detecting misdirected packets) that injects Interests with L name components, each being a single alphabetical letter. Using diagnostic tests through ICN/NDN ping and traceroute tools has the same theoretical complexity as the simulation-based approach, with the additional disadvantage of using too much network resources (as every test packet injected will have to actually traverse the network). Fig. 11 shows the large benefit, in terms of performance and scalability, of NSA compared to these simulation-based verifications.

To demonstrate NSA's performance and scalability, we examine its utility with larger test cases, with $n \times n$ grids. The results are shown in Figs. 12 and 13. Each node's FIB is populated with entries (rules) for all prefixes, with random outgoing faces. We only show the results for the reachability test with aggregation optimizations, with all-wildcard injections at all nodes, one face per node. Fig. 12 shows the execution times (shown in seconds in log scale) when increasing the grid dimensions. The largest case (100×100) has a total of 100×100×100×1=1 million rules in the network, which is similar with the largest test case considered in HSA's performance evaluation [13]. Thus, NSA's performance is in the same order of that of HSA, even though NSA adds the significant feature of checking name-based (information-centric) reachability. Further, note that our grid topology yields much higher number of paths compared to HSA's simpler backbone topology [13]. The growth rates in Fig. 12 is reasonable: note that along the x-axis, each scenario exponentially increases the number of paths, where an $n \times n$ grid has $O(n!)$ paths between two nodes, leading to much higher length and number of paths in the propagation graph. Next, we pick a 10×10 grid (with 10 providers), and gradually increase the per-provider number of prefixes. The largest case (*e.g.*, “1000”) leads to a total of 10×10×10×1000=1 million rules in the whole network. Again, the performance and scalability of NSA with these large test cases is reasonable (both compared to the absolute values for HSA, and comparing the relative growth in execution time). This is especially compelling, considering the high complexity of these test cases and how they affect the runtime, as explained in §VI.

We also evaluated the performance of NSA's loop detection on the same grid networks, injecting all-wildcard headers. Fig. 10 shows the results for both cases of 'All faces' and 'One face per node' injection. The complexities, growth rates and optimization benefit of face selection in loop detection are similar to those of full content reachability analysis. Also, compared to NDN's built-in loop detection mechanisms, NSA can prevent all possible loops caused by forwarding rules, without using network resources, and allowing for hints on how to resolve the loop errors.

2) Name Leakage Detection: To verify name leakage-freedom, we use two-ring topologies; where two rings of size *n* are connected by one node, *i.e.*, is a gateway between the rings. Each ring is considered its own zone, and has one publisher serving (and advertising) two prefixes, one prefix visible to everyone, and one prefix visible only to the nodes within the local zone. Thus, each NDN node has rules for the three prefixes (that are visible to it): two prefixes of its own zone, and one prefix that is public from the other zone. In each of its rounds, NSA's name leakage detection application injects all-wildcard headers to the faces/nodes of one zone, generates headers that reach the other zones, and checks whether or not they violate each zone's name privacy requirements. The performance of name leakage on the two-ring topologies are shown in Fig. 15, indicating its scalability (showing a linear growth) with the increase in network size.

B. NDN Testbed

To evaluate NSA's performance on an operational, practical NDN, we considered the NDN testbed [26]. This is the largest real-world NDN with publicly available forwarding state, with relatively large forwarding tables (of the order of hundreds of entries per node). We captured a snapshot of the testbed on 2019/03/09 14:43:16 CST. We use the globally available topology. Each node in the testbed provides its (near-)real-time local status (FIBs, *etc.*) through a separate webpage. We collected the full network status by crawling these individual local status pages. Some nodes were offline or unresponsive and we removed them from our analysis.

An important pre-processing step for NSA verification is generation of the transfer functions. We have implemented these components in NSA. The topology transfer function generation is trivial. For network transfer function generation for LPM-based forwarding rules in NDN nodes, additional processing needs to be done: for each FIB entry, all other rules (*i.e.*, FIB entries) have to be visited, as explained and analyzed for asymptotically completing the generation of the network transfer function in §VI. To show this empirically, we picked one particular node from the testbed, the 'UCI' node. It has 214 FIB entries in our selected snapshot. We randomly pick 50, 100, 150 and 200 FIB rules from it and perform the network transfer function generation. The execution time (including the case with all 214 entries) is shown in Fig. 16. These results show that NSA's transfer function generation for this particular real-world case is reasonably efficient and scales well with number of FIB rules.

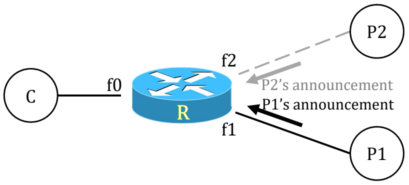
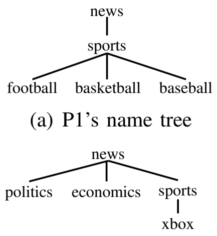


Fig. 17. Name space conflict example (case I: only P1 is present; case II: both P1 and P2 are present)



(a) P1's name tree

(b) P2's name tree

Fig. 18. Name trees of the producers in Fig. 17

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

From a practical standpoint, our experiments and results show that it is feasible to have NSA integrated into the NDN testbed (in one of its nodes), and periodically check for data plane errors, and checking various states of the data plane. Given that these checks only take seconds in total, including transfer function generation and the analysis, it would be quite reasonable to have new NDN snapshots (which can be generated every few minutes or seconds) be verified. The network administrator can run NSA on one of the nodes (a controller or any router) on the testbed, periodically collect snapshots at that node (using methods such as NDNconf [31]) and provide continuous verification results of the network. This would be very helpful for the users of the NDN testbed, and for their research experiments.

VIII. Case Study: Name Space Conflict Detection/Resolution

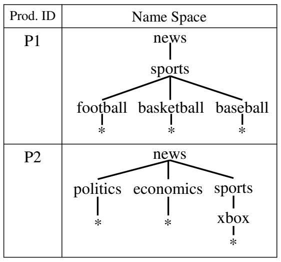
As an important case study, in this section, we explain name space conflicts, and how NSA can detect and help resolve it.

A. Name Space Conflicts

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

TABLE II
Name Space Conflict Scenarios for Case II

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



(a) Global name space

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

(b) Global routable prefixes

Fig. 19. Databases at name registry

same prefixes. This can be true especially when the names are topic-based. No content provider has sole ownership or authority to announce a certain prefix. While this allows for the democratization of content and better efficiency, it can cause conflicts that can lead to blackholed interests. We illustrate this using an example. In addition to checking with NSA, we also ran these scenarios in ndnSIM [43] and observed that the blackhole effect in question indeed does occur.

Fig. 17 shows a simple network topology. Suppose in the beginning, there exists only one producer $P1$ (case (I)), with the name tree depicted in the Fig. 18(a) and creates a name announcement for "/news/sports". The announcement is used to populate the router R's FIB in accordance with NDN policies. Announcing "/news/sports" implies that $P1$ claims that it has 'everything' under "/news/sports", which is correct from the network layer's perspective. In reality, a producer may not have 'everything' under a particular name prefix it announces, *i.e.*, there may be a possible suffix not covered in this announced 'everything' set. However, since there is no other producer to 'challenge' $P1$'s claim, $P1$'s announcement stating that requests for anything under "/news/sports" will be available at $P1$ does not cause any conflict.

Now let us assume the same network but with two producers $P1$ and $P2$, as shown in Fig. 17 (case (II)), with $P2$'s name tree shown in Fig. 18(b). The subset of $P2$'s name tree that is interesting for this discussion is "/news/sports/xbox". Note that there is no malicious intent on $P2$'s part; evidently, $P1$ does not recognize video games as 'sport'; however $P2$ does (with the sport being 'xbox'). $P2$, unaware of this conflict, announces his prefix also at the coarsest granularity, announcing "/news" (Scenario II-1 in Table II). Using NSA's content reachability test, we can show that this scenario is erroneous: requests for "/news/sports/xbox/1" reach $P1$ instead of $P2$. $P2$ is this Interest's (most) relevant provider. Formally, header space of form "/news/sports/xbox/∗" reaching R at $f0$, has a non-empty intersection with R's FIB rule at $f1$ ("/news/sports/∗") rather than that of $f2$ ("/news/∗"), assuming a best-route forwarding strategy at R. Thus, "/news/sports/xbox/" will be a name reaching $P1$ and not $P2$:

$$\text{“/news/sports/xbox/”} \in Range(CR_{C\to P1})$$
$$\text{“/news/sports/xbox/”} \notin Range(CR_{C\to P2})$$

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

This case study shows that achieving correctness has a cost. It may be important to find the most efficient refinement to the name space announcement so as to keep the FIB size

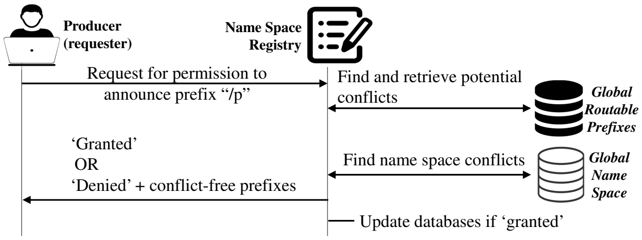

Producer (requester)
Name Space Registry
Request for permission to announce prefix "/p"
Find and retrieve potential conflicts
Global Routable Prefixes
'Granted' OR 'Denied' + conflict-free prefixes
Find name space conflicts
Global Name Space
Update databases if 'granted'

Fig. 20. Name registration procedure

manageable. There is a need for an approach to detect and perhaps resolve conflicts, before they happen. We provide some guidelines for the design of such an approach next.

B. Name Space Registry Guidelines

The name space conflict observed in the case study in §VIII-A may be quite common. While NSA is useful in finding conflicts, an automatic approach, or protocol, for conflict detection/resolution can be very beneficial in NDN. To prevent content provider name space conflicts, a *Name Registration* protocol may be such a solution. The idea of name registration in NDN has been suggested in previous works, such as in [23], mainly to prevent prefix hijacking. Our case study however shows it is important for non-malicious Scenarios as well.

A name space registry can be implemented as a distributed or centralized engine to be contacted by producers whenever they want to announce a prefix. The producer sends his requested announcement prefix and (a pointer to) his content name tree. The response data from name registry would be a signed packet containing the prefix announcement permission result, *i.e.*, 'granted' or 'denied', plus possibly new prefix announcements suggested by the name registry that are conflict-free. The name registry's decision is based on an analysis of name spaces across different providers, which are in its database, called a *global name space* (Fig. 19(a)). The name registry may have another database of *global routable prefixes* (Fig. 19(b)). Both databases are indexed by *Producer ID* which can be a real ID or a locally generated (but unique) ID. If permission is granted, the new prefix announced will be added to the global routable prefix list, and the producer's name space will be added to the global name space database. If the request is denied, the requester needs to pick another high-level name for the prefix announcement (just as in today's IP network, domain names have to be tried one after another, until one is available) or use suggestions provided by the name registry. The suggestions for the prefix announcement can be provided at different granularity levels to help manage the growth of the FIBs. While we describe it as a logically separate entity, the name registry can be integrated with possible existing name resolvers, such as NDNS [10]. An overview of the name registration procedure is depicted in Fig. 20. Upon receiving a request for permission to announce a prefix "`/p`" from a producer, the name registry performs the following steps:

1) Examine the global routable prefixes list to find *potential conflicts*. Two announcements can cause potential conflict if one is the "prefix" of the other (*i.e.*, their intersection is non-empty). An example for potential conflicts between names is $P2$'s announcement "`/news`" that is a prefix of $P1$'s announcement "`/news/sports`" in Scenario II-1 in Table II. The name registry returns the indices, which are producer IDs.

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

$$\begin{gathered}(pr1 \text{ is a prefix of } pr2) \bigwedge (pr2 \subseteq NS_{P1}) \\ \Longrightarrow \forall \text{ non-wildcard sequence of name components } X : \\ \text{``}/pr2/X\text{''} \subseteq NS_{P1} \iff \text{``}/pr2/X\text{''} \subseteq NS_{P2}\end{gathered}$$

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

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

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

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

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

IX. Limitations

While NSA answers several important questions about the network, it has its limitations. These limitations of NSA are quite similar to those of other notable data plane verification systems, such as HSA [13]. Regarding the discovery and reporting of errors, while NSA can give us hints about the details associated with errors, it cannot definitively assert why such error occurred or how it can be resolved. Additional external information, as well as refinement procedures, are required to achieve this. NSA is not well-suited for network-wide dynamic analysis that involves "churning" in the network's forwarding state. This is due to the fact that NSA is of the class of data plane verification tools, "mainly" optimized for static checking (*i.e.*, checking a data plane with regards to operations and properties that do not change the state of the network). Having said that, it is still feasible to check multiple states of the network, represented by multiple snapshots, by successively running NSA on them. However, this feature is limited for NSA and would only work if errors of a dynamic-nature stay longer than the "sampling period", *i.e.*, the interval between collecting two snapshots. Nonetheless, we believe NSA is a valuable tool for verifying key NDN-specific data plane properties.

X. Conclusion

We proposed Name Space Analysis (NSA), a data plane verification framework for NDN, based on the theory of Header Space Analysis. NSA (available at [25]) includes essential NDN-specific verification applications of content reachability test (to detect name space conflicts, content censorship-freedom, *etc.*), name-based loop detection, and name leakage detection. We also design a name registry method to detect and resolve name space conflicts in the data plane. Applied to the NDN testbed, we found a number of data plane errors through NSA's automatized verification. Our evaluation results on various test cases show the effectiveness, efficiency, and scalability of NSA.

References

[1] V. Jacobson *et al.*, "Networking named content," in *CoNEXT*, 2009.

[2] L. Zhang *et al.*, "Named data networking," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, 2014.

[3] A. Afanasyev *et al.*, "Snamp: Secure namespace mapping to scale ndn forwarding," in *INFOCOM Workshops*, 2015.

[4] ——, "Interest flooding attack and countermeasures in named data networking," in *IFIP Networking Conference, 2013*, 2013.

[5] V. Lehman *et al.*, "An experimental investigation of hyperbolic routing with a smart forwarding plane in ndn," in *IWQoS*, 2016.

[6] A. Afanasyev *et al.*, "Nfd developer's guide," *Technical report, NDN-0021, NDN*, 2016.

[7] I. Moiseenko and D. Oran, "Path switching in content centric and named data networks," in *ICN*, 2017.

[8] J. Kurihara *et al.*, "A consumer-driven access control approach to censorship circumvention in content-centric networking," in *ICN*, 2016.

[9] R. Tourani *et al.*, "Catch me if you can: A practical framework to evade censorship in information-centric networks," in *ICN*, 2015.

[10] A. Afanasyev *et al.*, "Ndns: A dns-like name service for ndn," in *ICCCN*, 2017.

[11] F. Lai *et al.*, "Scaled vip algorithms for joint dynamic forwarding and caching in named data networks," in *ICN*, 2016.

[12] H. Mai *et al.*, "Debugging the data plane with anteater," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, 2011.

[13] P. Kazemian *et al.*, "Header space analysis: Static checking for networks," in *NSDI*, 2012.

[14] ——, "Real time network policy checking using header space analysis," in *NSDI*, 2013.

[15] H. Zeng *et al.*, "Libra: Divide and conquer to verify forwarding tables in huge networks," in *NSDI*, 2014.

[16] A. Khurshid *et al.*, "Veriflow: Verifying network-wide invariants in real time," in *NSDI*, 2013.

[17] R. Beckett *et al.*, "A general approach to network configuration verification," in *SIGCOMM*, 2017.

[18] N. P. Lopes *et al.*, "Checking beliefs in dynamic networks," in *NSDI*, 2015.

[19] R. Stoenescu *et al.*, "Symnet: Scalable symbolic execution for modern networks," in *SIGCOMM*, 2016.

[20] D. Dumitrescu *et al.*, "Dataplane equivalence and its applications," in *NSDI*, 2019.

[21] K. Jayaraman *et al.*, "Validating datacenters at scale," in *SIGCOMM*, 2019.

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

[23] A. Compagno *et al.*, "Secure producer mobility in information-centric network," in *ICN*, 2017.

[24] P. F. Tehrani *et al.*, "The missing piece: On namespace management in ndn and how dnssec might help," in *ICN*, 2019.

[25] M. Jahanian and K. K. Ramakrishnan, "Name Space Analysis," https://github.com/mjaha/NameSpaceAnalysis, 2020.

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

[27] A. Panda *et al.*, "Verifying reachability in networks with mutable datapaths," in *NSDI*, 2017.

[28] D. Mauro and K. Schmidt, *Essential SNMP: Help for System and Network Administrators*. O'Reilly Media, Inc., 2005.

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

[30] H. Zeng *et al.*, "Automatic test packet generation," in *CONEXT*, 2012.

[31] A. Afanasyev and S. K. Ramani, "Ndnconf: Network management framework for named data networking," in *ICC Workshops*, 2020.

[32] Y. Li *et al.*, "A survey on network verification and testing with formal methods: Approaches and challenges," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 1, 2018.

[33] S. Mastorakis *et al.*, "Icn ping protocol draft-mastorakis-icnrg-icnping-00," https://tools.ietf.org/html/draft-mastorakis-icnrg-icnping-02, 2017.

[34] H. Asaeda *et al.*, "Contrace: Traceroute facility for content-centric network," https://tools.ietf.org/html/draft-asaeda-icnrg-contrace-042, 2018.

[35] S. Mastorakis *et al.*, "Icn traceroute," https://tools.ietf.org/id/draft-mastorakis-icnrg-icntraceroute-01.html, 2017.

[36] S. Shannigrahi *et al.*, "Traceroute for named data networking," https://named-data.net/publications/techreports/ndn-0055-2-trace, 2017.

[37] S. Khoussi *et al.*, "Ndn-trace: a path tracing utility for named data networking," in *ICN*, 2017.

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

[39] J. Thompson *et al.*, "Ndn-cnl: A hierarchical namespace api for named data networking," in *ICN*, 2019.

[40] M. Jahanian and K. K. Ramakrishnan, "Name space analysis: Verification of named data network data planes," in *ICN*, 2019.

[41] Y. Zhang *et al.*, "A survey of mobility support in named data networking," in *INFOCOM Workshops*, 2016.

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

[43] NDN, "ndnSIM," http://ndnsim.net, 2019.

Mohammad Jahanian received his B.S. degree from University of Tehran in 2012, and the M.S. degree from Sharif University of Technology in 2014. He is currently a Ph.D. candidate at University of California, Riverside. His current research interests include information-centric networks, formal verification, and distributed systems.

K. K. Ramakrishnan is Professor of Computer Science and Engineering at the University of California, Riverside. Previously, he was a Distinguished Member of Technical Staff at AT&T Labs-Research. Prior to 1994, he was a Technical Director and Consulting Engineer in Networking at Digital Equipment Corporation. Between 2000 and 2002, he was at TeraOptic Networks, Inc., as Founder and Vice President. K. K. is an ACM Fellow, an IEEE Fellow and an AT&T Fellow, recognized for his fundamental contributions on communication networks, including his work on congestion control, traffic management and VPN services. He has published nearly 300 papers and has 183 patents issued in his name. K. K. received his MTech from the Indian Institute of Science (1978), MS (1981) and Ph.D. (1983) in Computer Science from the University of Maryland, College Park, USA.