Name Space Analysis: Verification of Named Data Network Data Planes

Mohammad Jahanian
University of California, Riverside
Riverside, CA, USA
mjaha001@ucr.edu

K. K. Ramakrishnan
University of California, Riverside
Riverside, CA, USA
kk@cs.ucr.edu

ABSTRACT

Named Data Networking (NDN) has a number of forwarding behaviors, strategies, and protocols proposed by researchers and incorporated into the codebase, to enable exploiting the full flexibility and functionality that NDN offers. 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 particular 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, and allowing for name-based properties such as content reachability and name leakage-freedom. These important additions reflect the behavior and requirements of NDN, requiring modeling and verification foundations fundamentally different from those of traditional host-centric networks. For example, in name-based networks (NDN), host-to-content reachability is required, whereas the focus in host-centric networks (IP) is limited to host-to-host reachability. 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 pertaining to content reachability, loops, and name leakage, has good performance, and is scalable.

CCS CONCEPTS

• **Networks** → *Network reliability*; • **Software and its engineering** → *Formal methods*;

KEYWORDS

Named Data Networks, Network Verification

ACM Reference Format:
Mohammad Jahanian and K. K. Ramakrishnan. 2019. Name Space Analysis: Verification of Named Data Network Data Planes. In *6th ACM Conference on Information-Centric Networking (ICN '19), September 24–26, 2019, Macao, China.* ACM, Macao, China, 11 pages. https://doi.org/10.1145/3357150.3357406

ICN '19, September 24–26, 2019, Macao, China

ACM ISBN 978-1-4503-6970-1/19/09…$15.00
https://doi.org/10.1145/3357150.3357406

1 INTRODUCTION

Named Data Networking (NDN) [11, 34] 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 [34].

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 [4], rate-based forwarding [2], and hyperbolic forwarding [19] have been adopted and incorporated into the standard NDN Forwarding Daemon (NFD) [3]. 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 [24], Interest anonymization [17, 31], name resolution [1], cache-aware forwarding [18], *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 [7, 9, 13, 14, 16, 20, 21, 30, 33] 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 a network snapshot, 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) [14]. 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 [25]), 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 also mention a number of practical problems that NSA can be used to address, *e.g.*, name space conflict and content censorship (§5). In addition to a single snapshot, NSA can also be successively applied to a number of (finite, limited) snapshots, to help verify state changes of the data plane and identify selected data and control plane problems. We implemented NSA [12], 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 (§6).

The contributions of this paper are as follows: 1) We provide a framework for verification of NDN data planes, based on Header Space Analysis concepts. NSA targets the nature of an NDN, rather than the previously developed tools for host-centric architectures, and provides the flexibility and support for analysis of NDN-specific properties. 2) We introduce modeling name spaces and name space functions to analyze names, as they are the main assets required to access content in NDN. 3) We specify essential NDN-specific properties and approaches to analyze them. These include content reachability test (to check unwanted and unsolicited names, as defined in §5.1), name-based loop detection, and name (space) leakage detection. 4) We implemented NSA [12]; we describe important implementation descriptions and optimizations. Our results from various test cases show the scalability and efficiency of NSA. 5) Applied to a real-world NDN testbed [27] snapshot, NSA managed to find many reachability and loop errors.

2 BACKGROUND AND RELATED WORK

This section provides a brief overview of Header Space Analysis (HSA), network verification and NDN analysis tools.

2.1 Overview of Header Space Analysis

Header Space Analysis (HSA) [14] 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.

2.2 Network Verification and NDN Diagnostics

Network verification aims at analyzing large, complicated networks in order to find corner case errors and investigate essential properties. There have been efforts to build models to describe and verify networks. For the purpose of building verification frameworks, some works focus on analyzing control plane (to analyze all data planes caused by configurations) and some on data plane (to analyze the current state of the network). Computational feasibility and full verification coverage are challenges of control plane verification [7, 28]. We focus on data plane verification in this paper. Some of the more notable data plane verification tools are Anteater [21], HSA [14], VeriFlow [16], and NetPlumber [13]. These methods typically consist of snapshot-based static checking. Anteater [21] models the data plane as a set of boolean expressions and runs a SAT solver to verify invariants. HSA [14] 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 [13]. 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 [7, 9] propose data plane equivalence checks. While we focus on single snapshot verification in this paper, *i.e.*, checking a data plane against a predefined set of properties, our work can be extended to checking equivalence between two (or any limited number of) data plane snapshots using methods of [7, 9].

Our work presents an NDN-specific verification framework. Diagnostic tools such as Ping [22] and Traceroute [5, 15, 23, 29] have been proposed and developed for NDN. While these tools are very helpful for performance measurements and small-scale connectivity checks, a formal approach gives us a higher level of flexibility for property checking. The coverage of a larger problem space enables a more holistic verification of a network.

3 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 combination of 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 [25], 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. 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, that need to be addressed. 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*. Details of NSA elements are provided in §4.

NSA, just like HSA and other data plane verification approaches, focuses on a single snapshot, and models how the state of packets change with regards to a given network state, but not how the

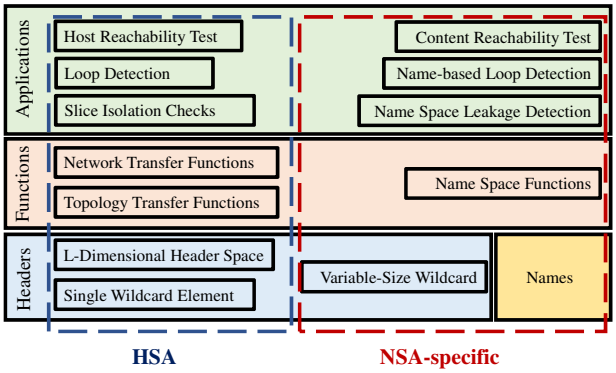


Figure 1: Overall framework of NSA

state of the network itself changes. In other words, NSA reads and writes to state at a packet-level, as explained in §5 (Fig. 2(b)), and only reads from state at the network-level. Thus, it focuses on a set of properties that are dependent on the current state of the network, and not on how other packets may change it. The properties cover all packets and their paths in the existing network state. However, this is still a huge improvement in terms of coverage of analysis compared to existing solutions [5, 15, 22, 23, 29], and allows for important classes of properties such as reachability and loop-freedom [14, 21]. 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, 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. The first snapshot would be taken while the producer is connected at its initial point of attachment, and the second (final) snapshot is taken at the new point of attachment, after the network's state converges (*e.g.*, FIBs are updated). NSA can check if the received headers at the producer are same across those snapshots. However, a full-fledged dynamic verification of network properties across data plane states needs to take into account efficiency and overhead considerations, which we plan to work on in the near future.

4 NSA DESIGN

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

4.1 Modeling NDN Header Space

4.1.1 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 [25]. 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 [26]. 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.

4.1.2 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 $[?] = a_1 \cup a_2 \cup \cdots \cup a_n$, 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,

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

Note that the "[∗]" wildcard is not currently part of the NDN architecture; we use it as part of NSA headers for the model's representation and verification efficiency, to be used in a *symbolic execution* fashion, which we explain in §5.

4.1.3 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 $a \cap a = a$, $a \cap [?] = a$. For two unequal atom values, $a_1 \cap a_2 = [z]$. 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:

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

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

4.2 Modeling NDN Nodes

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

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

where a function T maps header h_0 coming to face f_0, to all headers h_1, h_2, *etc.*, going out of faces f_1, f_2, *etc.* of the node respectively. NSA's transfer functions are at the level of a face, rather than being port-level as in HSA. While this does not change the algebraic operations on the transfer functions, in practice it does enable one to write such functions using faces, regardless of the underlying strategies associated with those faces. Domain and range of NSA transfer functions are of the same type (both Interest or both Data headers). Transitioning from Interest to Data is not a part of NSA verification as it requires changing the state of the data plane. Depending on the functionality being modeled, function T may or may not change h_0, and may or may not depend on the incoming face f_0. Any NDN packet processing, including an NDN forwarding behavior, can be modeled using (a set of) transfer functions.

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

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

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

In general, a typical Interest processing transfer function can be modeled as $T_I(.) = T_{I.fwd}(T_{I.CS}(T_{I.PIT}(.)))$. What elements we put into a transfer function depends on our architecture and the purpose of the analysis. For example, if we have the assumption of the CS and PIT being empty upon the arrival of an Interest, then we can simply have $T_I(.) = T_{I.fwd}(.)$. Additional functions can be added to the pipeline as well, including those that modify the incoming header space, *e.g.*, function $T_{HopLimit}$ that decrements the *HopLimit* field in the Interest [25] if it is above zero and passes

it to the subsequent transfer function in the pipeline, and drops it otherwise:

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

Using similar patterns, we can model Data forwarding or any additional Interest forwarding transfer functions such as the full forwarding pipelines in the NFD specification[3] or link object processing[4], complicated forwarding strategies and Nonce checks, *etc.* A packet processing pipeline can be modeled as a cascade of functions, *i.e.*, $T_n(T_{n-1}(\ldots T_1(.)))$ where each T_i is a specific function (step) in the pipeline. It can also be a named function, performing an operation if the Interest name has a particular prefix; this operation can involve changing the name in the header. *E.g.*, an arbitrary Interest anonymizer, that encrypts the name with key K and encryption algorithm *Enc*, and triggered by the prefix "`/Anon`", will have a transfer function in its Interest process pipeline, as follows:

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

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

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

4.3 Modeling Name Spaces

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

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.

5 USING NSA FOR VERIFICATION

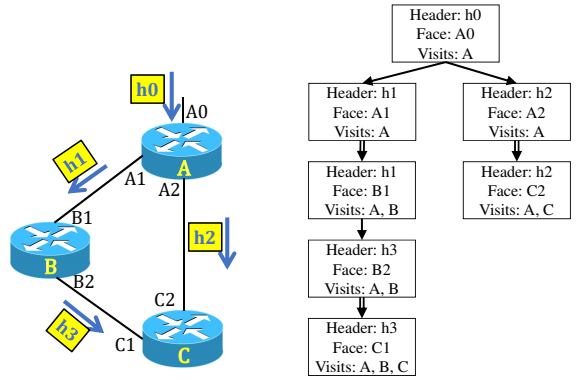
This section explains a number of verification applications that NSA can check. Specifically, we look at the important applications of testing content reachability, loop detection and name leakage detection. NSA provides significant benefits both in terms of the verification results and its efficiency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

An important part of NSA's formal verification approach that facilitates automated checking is the generation and analysis of the state space, or *propagation graph*. The graph represents all possible *paths* any packet can take, rather than a single trace that a simulation-based approach would support. This provides the desired coverage we need for verification. An example is shown in Fig. 2. 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. 2, the propagation graph (Fig. 2(b)) implies that header $h0$ is injected to face $A0$ of node A (as shown in Fig. 2(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. 2(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 [3], 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) of complicated distributed algorithms (*i.e.*, routing protocols). All these motivate more careful attention.

5.1 Content Reachability Test

Reachability analysis in HSA, and other host-based verification solutions, focuses on host (content provider) reachability. We extend



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

(b) Propagation graph (⟶: network transfer function transitions; ⟹: topology transfer function transitions)

Figure 2: Propagation graph example

this to content (name space) reachability in NSA, since this is a main concern in NDN. This analysis generates name spaces that can reach content repositories, *i.e.*, at producers or content stores. To this end, we apply a name space function on the header space received at a content repository:

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

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

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

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

The process is exemplified in Fig. 3, 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.

Algorithm 1 specifies the application for the name space reachability test in a network, denoted by its *Network Space N*, which is the

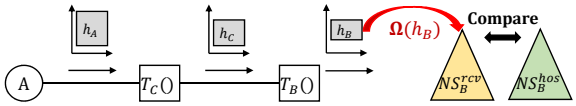


Figure 3: Content reachability test

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

Algorithm 1 Content Reachability Test

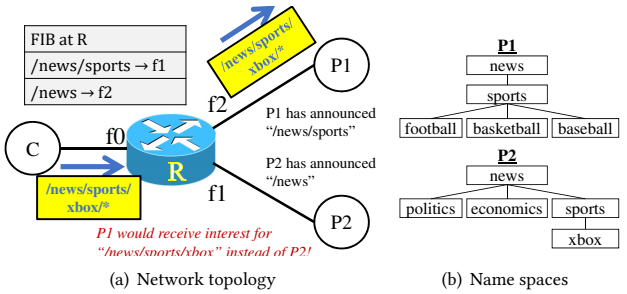
```
1:  procedure ConReach(C, h_0, N)  ▷ Injecting h_0, at C, network space N
2:      Start with h_0 at C                ▷ Typically all wildcard, i.e., [*]
3:      Calculate all h_{P_i}'s            ▷ Headers reached at provider P_i
4:      for all P_i do
5:          NS^{rcv}_{P_i} ← Ω(h_{P_i})
6: 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

The time complexity of an NSA content reachability test for injecting a header to a consumer that leads to a single content provider is $O(dLR^2s)$, where d, L, R, and s are maximum network diameter (number of hops), maximum header length, maximum number of node rules, and maximum number of paths in a trie-based content provider name space. This analysis is based on the *linear fragmentation* assumption in [14], which says that typically very few rules in a node match an incoming packet. Unlike NSA, the complexity of a simulation-based test would be $O(da^LRs)$, where a is the maximum number of values an atom can take; *e.g.*, with byte-based atoms, a would be 256. This shows the huge benefit of NSA over purely simulation-based approaches, for a content reachability analysis with high coverage.

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

- **Route computation outcome correctness.** We can use NSA's content reachability analysis to see if a particular content request reaches the nearest (or all/any) content, in case a content resides at two repositories with the same names. This is very useful to analyze the correctness of the computation outcome (*i.e.*, resulting state in the FIB, and *not* the routing protocol itself) of traditional routing protocols such as NLSR [10] (only focusing on content providers) or nearest replica routing protocols [6] (focusing on both content providers and ICN router content stores).
- **Security infrastructure soundness.** In NDN's content-oriented security design, keys that are used to perform security-related operations (authentication, *etc.*), are just like any other content: they have names, their names/prefixes populate FIBs, and they can/should be retrieved using Interests [32]. The reachability of the correct keys is important for NDN security mechanisms to be sound. As an important case, NSA can check if all public keys (*e.g.*, data with "`/KEY`" prefix) can be reached at appropriately, requested from all appropriate end points in the network.
- **Name space conflict-freedom.** In NDN, different content providers can use and announce the same prefixes, especially when names are topic-based. No content provider has sole ownership or authority to announce a certain prefix. While this allows for democratization of content and better efficiency, it can cause conflicts that can lead to blackholed interests. Fig. 4 shows an example of this: content providers $P1$ and $P2$, with hosted content name spaces shown in Fig. 4(b), send prefix announcements "`/news/sports`" and "`/news`" respectively, leading to two FIB entries, namely "`/news/sports` → f1" and "`/news` → f2" at router R (Fig. 4(a)). Announcing "`/news/sports`" implies that $P1$ claims that he has 'everything' under "`/news/sports`", which is challenged when considering $P2$'s name space, who unlike $P1$'s, has content under "`/news/sports/xbox`"; it may be the case that $P1$ and $P2$ have different views on whether or not 'xbox' is a sub-category of 'sports', which they are allowed to. This conflict causes Interests for "`/news/sports/xbox`" to be misdirected to $P1$ instead of $P2$. NSA's content reachability test can catch these errors. Also, to overcome these conflicts, name registry methods, such as [8] can be used, to have content providers register their prefixes before announcement, and grant them permission only if it is a conflict-free announcement. NSA can be used to check correctness of the outcome of such registration mechanisms as well.
- **Content censorship-freedom.** Censorship leads to content reachability errors; in the example in Fig. 5, censoring node R may drop all interests for "`/democracy`" [17]. This would result in (all or part of) content provider P's name space to be unreachable, injecting headers from C. This is an undesired effect that can easily be detected by NSA. While NSA cannot definitively deduce that such a problem is caused by content censorship, the lack of existence of such errors would imply content censorship-freedom. Furthermore, the effectiveness of a censorship countermeasure mechanisms can be checked using NSA.

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



(a) Network topology (b) Name spaces

Figure 4: Name space conflict example

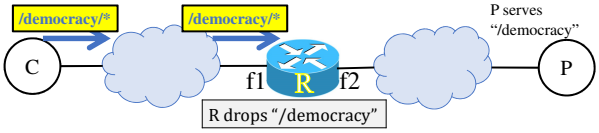


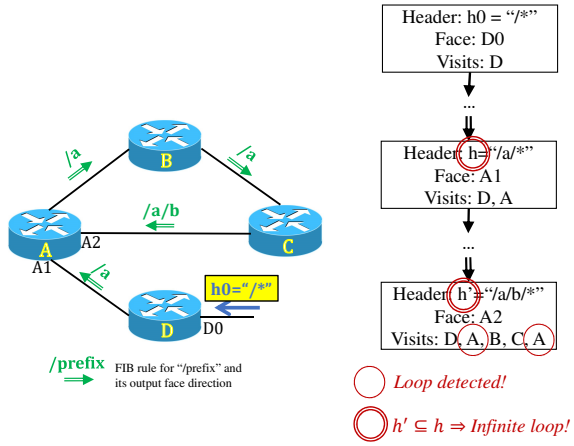
Figure 5: Content censorship example

5.2 Loop Detection

Loop freedom is an important property in networks. For NDN in particular, looping Interests is a widely known issue, which led to the addition of extra processes in the forwarding pipelines, such as a Dead Nonce List [3]. 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 [14]). 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. 6, 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.



(a) Topology, injected header, and FIB rules (b) Propagation graph (partial)

Figure 6: Loop detection example

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.

5.3 Name (Space) Leakage Detection

What if a consumer issuing an Interest for a particular name, wishes (parts of) the name, *e.g.*, his ID or a particular content name, to not be visible in the network except for certain authorized nodes, *e.g.*, those in his home network? This can be a desirable property for a variety of reasons. Works such as [31] 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. 7: Consumer C issues Interests with header h_0, which results in headers h_1, h_2 and h_3 leaving the authorized zone of routers, denoted as $Z1$. We define all the headers going out of $Z1$ as $h_{out} = h_1 \cup h_2 \cup h_3$. NSA allows us to define and apply access control rules on names in a number of ways, and check name constraints on h_{out} accordingly, *e.g.*, the following examples:

- Headers of particular form, *e.g.*, containing a particular name component or prefix, should not appear in any packets leaving zone Z1. Then we should have $h_{out} \cap h_{prohibited} = \varnothing$, where the left-hand side of the equation denotes the intersection of all headers leaving $Z1$ with all prohibited headers. Prohibited headers can be built using NSA's atoms and algebra, as described in §4. The "$\varnothing$" on the right-hand side means that we do not want any header in the result of the intersection to leave $Z1$.
- Packets associated with name space NS_0 should not leave $Z1$; then we should have $\Omega(h_{out}) \cap NS_0 = \varnothing$. This way of defining a rule is more efficient for constraints of a larger set of prefix-suffix name relations representing a portion of a name space graph: instead of checking many prefixes one by one, we can check once against name space NS_0 comprising all those prefixes.

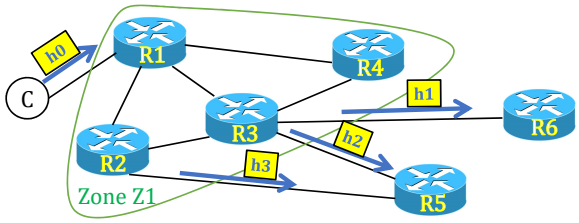


Figure 7: Name leakage detection example

6 EVALUATION

An important part of network formal verification is developing a tool that automatizes the generation of state spaces and verification checks in a reasonably efficient way. We have implemented NSA, including its main components and modules, in Java; the source code is available at [12]. 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 [27]. 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 a number of optimizations that further improves NSA's performance.

6.1 Synthetic Networks

6.1.1 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 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. 8, 9, and 10, in terms of execution times, in milliseconds.

Fig. 8 shows the execution time of content reachability on the grid networks. This verification, as explained in §5.1, checks both unreachable and unsolicited names. Typically, NSA injects all-wildcard headers into all faces, since some node rules may depend on the incoming faces ('All faces' bars in Fig. 8). As seen in Fig. 8, 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. 8 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 (Fig. 8), 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. 11). This way, we preserve all reached header spaces, but not their exact paths from

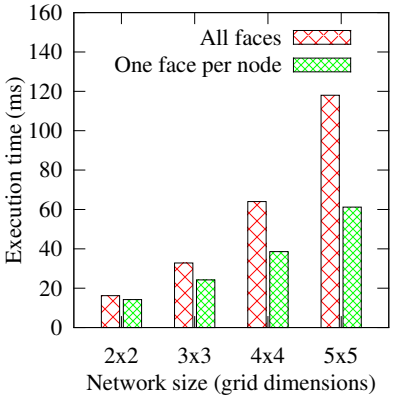


Figure 8: Content reachability

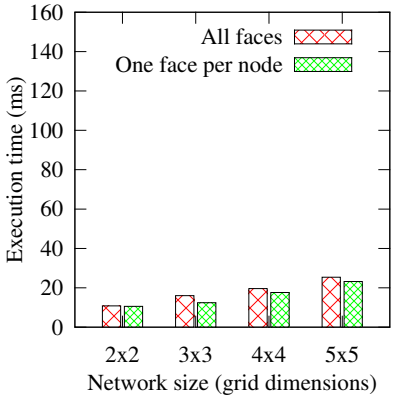


Figure 9: Content reachability with header aggregation

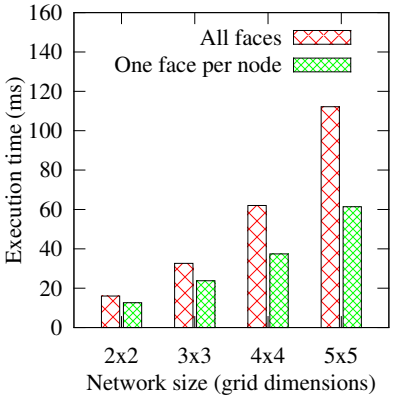


Figure 10: Loop detection

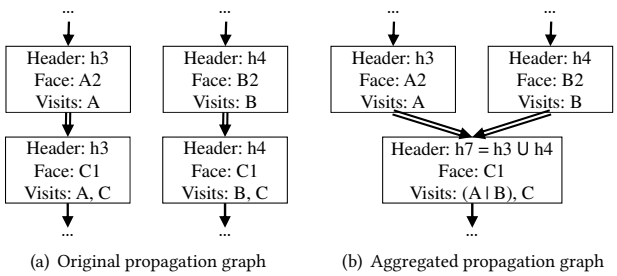


Figure 11: Propagation graph aggregation example

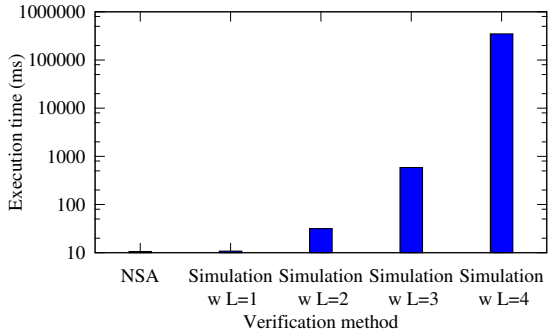


Figure 12: NSA and simulation-based verification

origin in the visited list. Fig. 9 shows the significant performance enhancement of this optimization, compared to full reachability analysis in Fig. 8, 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 §5.1. We show the empirical results for the use of wildcards in Fig. 12. Each 'Simulation' scenario 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. Fig. 12 shows the large benefit, in terms of performance and scalability of NSA compared to these simulation-based verifications.

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.

6.1.2 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. 13, indicating its scalability (showing a linear growth) with the increase in network size.

6.2 NDN Testbed

To evaluate NSA's performance on an operational, practical NDN, we considered the NDN testbed [27]. 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. 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 §4.2. To show this empirically, we picked one particular node from the testbed, the

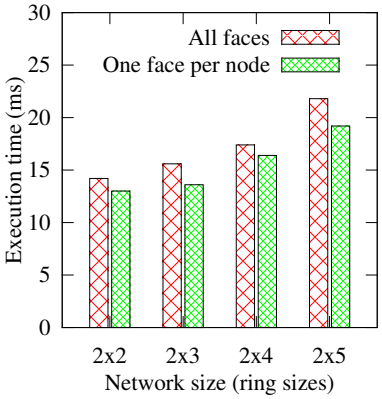


Figure 13: Name leakage

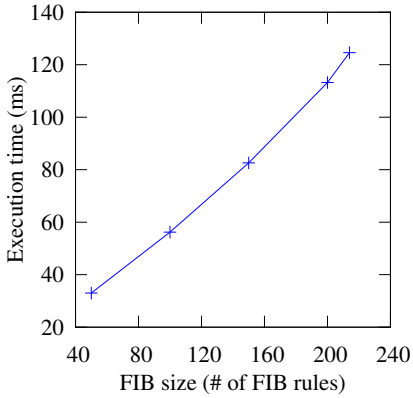


Figure 14: Network transfer function: rule generation ('UCI' node)

Table 1: Execution time (ms) for NDN testbed verification for alternate forwarding strategies

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

'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. 14. 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 (this is less complex than the upper bound shown in §4.2).

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 as well. 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 1, showing that 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 are generated every 10 minutes at the date of this submission) be verified. This would be very helpful for the users of the NDN testbed, and for their research experiments.

7 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 [14].

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 single 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.

8 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 [12]) includes essential NDN-specific verification applications of content reachability test (to detect name space conflicts, content censorship-freedom, *etc.*), name-based loop detection, and name leakage detection. Applied to the NDN testbed, 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.

9 ACKNOWLEDGEMENTS

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. We thank our shepherd, Craig Partridge, for his support and insightful feedback and the reviewers for their valuable comments.

REFERENCES

[1] Alexander Afanasyev, Xiaoke Jiang, Yingdi Yu, Jiewen Tan, Yumin Xia, Allison Mankin, and Lixia Zhang. 2017. NDNS: A DNS-Like Name Service for NDN. In *Computer Communication and Networks (ICCCN), 2017 26th International Conference on*.

[2] Alexander Afanasyev, Priya Mahadevan, Ilya Moiseenko, Ersin Uzun, and Lixia Zhang. 2013. Interest flooding attack and countermeasures in Named Data Networking. In *IFIP Networking Conference, 2013*.

[3] Alexander Afanasyev, Junxiao Shi, et al. 2018. NFD developer's guide. *Technical report, NDN-0021, NDN* (2018).

[4] Alexander Afanasyev, Cheng Yi, Lan Wang, Beichuan Zhang, and Lixia Zhang. 2015. SNAMP: Secure namespace mapping to scale NDN forwarding. In *Computer Communications Workshops (INFOCOM WKSHPS), 2015 IEEE Conference on*.

[5] Hitoshi Asaeda, Xun Shao, and Thierry Turletti. 2018. Contrace: Traceroute Facility for Content-Centric Network draft-asaeda-icnrg-contrace-04. https://tools.ietf.org/html/draft-asaeda-icnrg-contrace-042.

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

[7] Ryan Beckett, Aarti Gupta, Ratul Mahajan, and David Walker. 2017. A General Approach to Network Configuration Verification. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '17)*.

[8] Alberto Compagno, Xuan Zeng, Luca Muscariello, Giovanna Carofiglio, and Jordan Augé. 2017. Secure producer mobility in information-centric network. In *Proceedings of the 4th ACM Conference on Information-Centric Networking*.

[9] Dragos Dumitrescu, Radu Stoenescu, Matei Popovici, Lorina Negreanu, and Costin Raiciu. 2019. Dataplane equivalence and its applications. In *16th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 19)*.

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

[11] Van Jacobson, Diana K. Smetters, James D. Thornton, Michael F. Plass, Nicholas H. Briggs, and Rebecca L. Braynard. 2009. Networking Named Content. In *Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies (CoNEXT '09)*.

[12] Mohammad Jahanian and K. K. Ramakrishnan. 2019. Name Space Analysis. https://github.com/mjaha/NameSpaceAnalysis.

[13] Peyman Kazemian, Michael Chang, Hongyi Zeng, George Varghese, Nick McKeown, and Scott Whyte. 2013. Real Time Network Policy Checking Using Header Space Analysis. In *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*.

[14] Peyman Kazemian, George Varghese, and Nick McKeown. 2012. Header Space Analysis: Static Checking for Networks. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation (NSDI'12)*.

[15] Siham Khoussi, Davide Pesavento, Lotfi Benmohamed, and Abdella Battou. 2017. NDN-trace: a path tracing utility for named data networking. In *Proceedings of the 4th ACM Conference on Information-Centric Networking*.

[16] Ahmed Khurshid, Wenxuan Zhou, Matthew Caesar, and P. Brighten Godfrey. 2012. VeriFlow: Verifying Network-wide Invariants in Real Time. In *Proceedings of the First Workshop on Hot Topics in Software Defined Networks (HotSDN '12)*.

[17] Jun Kurihara, Kenji Yokota, and Atsushi Tagami. 2016. A consumer-driven access control approach to censorship circumvention in content-centric networking. In *Proceedings of the 3rd ACM Conference on Information-Centric Networking*.

[18] Fan Lai, Feng Qiu, Wenjie Bian, Ying Cui, and Edmund Yeh. 2016. Scaled VIP Algorithms for Joint Dynamic Forwarding and Caching in Named Data Networks. In *Proceedings of the 3rd ACM Conference on Information-Centric Networking*.

[19] Vince Lehman, Ashlesh Gawande, Beichuan Zhang, Lixia Zhang, Rodrigo Aldecoa, Dmitri Krioukov, and Lan Wang. 2016. An experimental investigation of hyperbolic routing with a smart forwarding plane in ndn. In *Quality of Service (IWQoS), 2016 IEEE/ACM 24th International Symposium on*.

[20] Nuno P. Lopes, Nikolaj Bjørner, Patrice Godefroid, Karthick Jayaraman, and George Varghese. 2015. Checking Beliefs in Dynamic Networks. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*.

[21] Haohui Mai, Ahmed Khurshid, Rachit Agarwal, Matthew Caesar, P. Brighten Godfrey, and Samuel Talmadge King. 2011. Debugging the Data Plane with Anteater. In *Proceedings of the ACM SIGCOMM 2011 Conference (SIGCOMM '11)*.

[22] Spyridon Mastorakis, Jim Gibson, Ilya Moiseenko, Ralph Droms, and David Oran. 2017. ICN Ping Protocol draft-mastorakis-icnrg-icnping-00. https://tools.ietf.org/html/draft-mastorakis-icnrg-icnping-02.

[23] Spyridon Mastorakis, Jim Gibson, Ilya Moiseenko, Ralph Droms, and David Oran. 2017. ICN Traceroute Protocol Specification draft-mastorakis-icnrg-icntraceroute-01. https://tools.ietf.org/id/draft-mastorakis-icnrg-icntraceroute-01.html.

[24] Ilya Moiseenko and Dave Oran. 2017. Path switching in content centric and named data networks. In *Proceedings of the 4th ACM Conference on Information-Centric Networking*.

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

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

[27] NDN. 2019. NDN Testbed. http://ndndemo.arl.wustl.edu/ndn.html.

[28] Aurojit Panda, Ori Lahav, Katerina Argyraki, Mooly Sagiv, and Scott Shenker. 2017. Verifying reachability in networks with mutable datapaths. In *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*.

[29] Susmit Shannigrahi, Dan Massey, and Christos Papadopoulos. 2017. Traceroute for Named Data Networking. *Technical Report NDN-0055, NDN* (2017).

[30] Radu Stoenescu, Matei Popovici, Lorina Negreanu, and Costin Raiciu. 2016. SymNet: Scalable Symbolic Execution for Modern Networks. In *Proceedings of the 2016 ACM SIGCOMM Conference (SIGCOMM '16)*.

[31] Reza Tourani, Satyajayant Misra, Joerg Kliewer, Scott Ortegel, and Travis Mick. 2015. Catch me if you can: A practical framework to evade censorship in information-centric networks. In *Proceedings of the 2nd ACM Conference on Information-Centric Networking*.

[32] Yingdi Yu, Alexander Afanasyev, David Clark, Van Jacobson, Lixia Zhang, et al. 2015. Schematizing trust in named data networking. In *Proceedings of the 2nd ACM Conference on Information-Centric Networking*.

[33] Hongyi Zeng, Shidong Zhang, Fei Ye, Vimalkumar Jeyakumar, Mickey Ju, Junda Liu, Nick McKeown, and Amin Vahdat. 2014. Libra: Divide and Conquer to Verify Forwarding Tables in Huge Networks. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*.

[34] Lixia Zhang, Alexander Afanasyev, Jeffrey Burke, Van Jacobson, Patrick Crowley, Christos Papadopoulos, Lan Wang, Beichuan Zhang, et al. 2014. Named data networking. *ACM SIGCOMM Computer Communication Review* 44, 3 (2014).