Formal Verification of Interoperability Between Future Network Architectures Using Alloy

Mohammad Jahanian[1(✉)], Jiachen Chen[2], and K. K. Ramakrishnan[1]

[1] University of California, Riverside, CA, USA
mjaha001@ucr.edu, kk@cs.ucr.edu
[2] WINLAB, Rutgers University, North Brunswick, NJ, USA
jiachen@winlab.rutgers.edu

Abstract. The Internet is composed of many interconnected, interoperating networks. With the recent advances in Future Internet design, multiple new network architectures, especially Information-Centric Networks (ICN) have emerged. Given the ubiquity of networks based on the Internet Protocol (IP), it is likely that we will have a number of different interconnecting network domains with different architectures, including ICNs. Their interoperability is important, but at the same time difficult to prove. A formal tool can be helpful for such analysis. ICNs have a number of unique characteristics, warranting formal analysis, establishing properties that go beyond, and are different from, what have been used in the state-of-the-art because ICN operates at the level of content names rather than node addresses. We need to focus on node-to-content reachability, rather than node-to-node reachability. In this paper, we present a formal approach to model and analyze information-centric interoperability (ICI). We use Alloy Analyzer's model finding approach to verify properties expressed as invariants for information-centric services (both pull and push-based models) including content reachability and returnability. We extend our use of Alloy to model counting, to quantitatively analyze failure and mobility properties. We present a formally-verified ICI framework that allows for seamless interoperation among a multitude of network architectures. We also report on the impact of domain types, routing policies, and binding techniques on the probability of content reachability and returnability, under failures and mobility.

1 Introduction

Today's computer networks, the Internet being a dominant example, are heavily used to fulfill users' *information-centric* needs: users primarily seek information over the network without necessarily wanting to focus on its location or the underlying mechanisms used to retrieve it [9]. However, the current way of using "location-based" access in IP networks results in a less convenient and less efficient means for information retrieval and dissemination. *Information-Centric Networks* (ICNs) address this content-oriented networking paradigm by separating content identity from its location [9]. ICN enables access to content based on

A. Raschke et al. (Eds.): ABZ 2020, LNCS 12071, pp. 44–60, 2020.
https://doi.org/10.1007/978-3-030-48077-6_4

its name, from wherever it resides, supporting mobility as well as accessing the named content from the best, any, or all source(s). It also allows for network-wide caching to reduce access latency. There are a variety of ICN architectures which have been proposed in the past decade. Two of the most notable ones, which we primarily focus on in this paper, are Named Data Networks (NDN) [20], and MobilityFirst [16], which have been considered for *Future Internet* designs [3].

Currently, there are two main factors that make the discussion of network interoperability important: 1) Today, IP is ubiquitous and used on a majority of network devices, despite the legacy of end-point address-oriented communication, especially considering new services and demands on today's networks [15]. 2) Research on designing new network architectures radically different from IP, is ongoing, and in many cases has already led to implemented systems; our focus in this paper is on an important class of such architectures, namely ICN. It is anticipated that we may have a number of interconnected networks (domains) using different architectures [15]. To go beyond the *interconnection* (*i.e.*, physical connections between different domains) towards *interoperation* between them (*i.e.*, being able to use a service, or content, provided by one domain in another domain), we need network interoperability. In the past decade, several designs have been proposed for interoperation between an ICN architecture (either NDN or MF) with IP [3]. However, such designs and their requirements were presented informally, describing the primitives and operations. It has been observed that network interoperability is complex [19]; thus, a formal structure for analysis of *information-centric interoperability* (ICI) can be very helpful, as it can provide proofs or expose errors early on, before the universal deployment of ICI frameworks for Future Internet.

Formal methods have been extensively used for designing and analyzing computer networks and protocols (surveyed in [14]). As for interoperability, work in [19] proposed a formal model to analyze interoperation of legacy networks. However, it only deals with *host-centric interoperability* (HCI), and only uses classic *model finding* [17] reasoning techniques. We extend that to support ICI as well as modeling failure and mobility with *model counting* [7] techniques. Network verification tools have also been proposed to analyze network data and control planes. Recently, work in [10] proposed a tool to verify ICN data planes, analyzing properties such as reachability. However, it only deals with a single domain, while our goal here is to cover multiple domains with different architectures coexisting with each other. Also, the symbolic execution nature of works such as [10] is computationally too expensive when expanded across multiple domains, each having its own data plane.

We present an Alloy [8]-based formalization of ICI, to analyze interoperability correctness. We cover both pull-based (request/response) and push-based (publish/subscribe) [6] content retrieval services, and their most essential properties such as content reachability and returnability. To analyze content-oriented services, we distinguish between static and dynamic content, justifying their differences, and specifying no-conflict properties, especially for dynamic content retrieval. For verification of these properties, we use Alloy Analyzer's built-in SAT solver-based model finding engine [2]. We also consider failure and mobility; to

analyze them, mere model finding is not sufficient, as failure and mobility, when severe, can cause any network protocol to become "incorrect" (and raise counterexamples). Thus, for such analysis, we resort to model counting (to count and compare the number of satisfying instances and counterexamples) to assess "how well" a particular domain or architecture is doing under failure and mobility.

The major contributions of this paper are: 1) a model finding method to analyze basic properties (mainly reachability and returnability) of information-centric interoperability (ICI); 2) a formally-verified ICI framework; and 3) a model counting method to analyze gateway failure and mobility.

2 Background and Related Work

2.1 Information-Centric Networking (ICN) and Interoperability

ICN enables access to content independent of its location, focusing on the fact that what matters to users is *what* the content is rather than *where* that content is located [9]. An ICN network layer recognizes and makes its forwarding decisions based on content *names* (or IDs) instead of addresses (unlike host-centric networks, as in today's IP networks), achieving efficiency and scalability.

Among many different ICN architectures proposed recently, we focus on the two most popular ones, namely Named Data Networks (NDN) [20] and MobilityFirst (MF) [16]. Both allow users to retrieve content using content names, through pull-based request/response or push-based publish/subscribe methods [6]. In-network content caching in routers is an important feature of ICN, allowing for requests to be satisfied from an intermediate cache on the path to the server/repository [9]. An in-network *namespace* is generally a graphical structure that captures the content names and their relationships in an ICN's content space [12]. Despite both being ICNs, NDN and MF have important differences [16,20]: NDN uses human-readable hierarchically-structured names, with Longest Prefix Matching-based forwarding. NDN content requests (called *Interests*) leave "breadcrumb" state in the routers on their path, which the associated response (called *Data* packets) then follow back, via *Reverse Path Forwarding (RPF)*. MF, on the other hand, uses flat IDs (called *GUIDs*) to identify content. Response packets contain the consumer's ID and do not need to follow the same path as the request. Also, MF inherently supports mobility by *late binding*, which re-directs in-flight packets towards a mobile content repository. Early binding assigns names to locations strictly at the original client, while late binding allows such assignment to be updated on its way in the network [16].

There have been several proposals for interoperability frameworks for ICNs (surveyed in [3]). These frameworks typically consist of *interoperation gateways* between *domains* of different network architectures, performing *translations* between them. All of these proposals allow interoperation of just two domains, IP and one ICN (either NDN or MF), and often require addition of new protocols or modification of existing ones. We generalize these solutions in our model to an interoperability framework of multiple ($\geq$2) domain types (we allow IP, NDN and MF to coexist simultaneously), and do not change any domain-specific protocols.

2.2 Alloy

Alloy is a declarative language based on relations and first order logic [8]. Alloy models a system, M, through the declaration of *signatures* (objects and their relations) and *facts* (constraints and axioms). A *predicate* is defined as a logical formula. An *Assertion* is a logical formula (which can be a combination of predicates) that are required to be always true (*i.e.*, as *invariants*) in the system. Alloy Analyzer [2] allows the automatic analysis of models and their properties through utilizing off-the-shelf SAT solvers. The tool translates Alloy descriptions into *Conjunctive Normal Form (CNF)* expressions. It uses an enumeration of instances, also called *model finding*, within a *bound* (*scope*), to prove whether or not a predicate P *ever* holds (by SAT-solving $M \wedge P$), or an assertion A *always* holds as an invariant (by SAT-solving $M \wedge \neg A$, to look for *counterexamples*).

Alloy has been used in modeling and analysis of many systems, including network protocols and architectures [8]. In the particular case of network interoperability, Zave [19] used Alloy to formally analyze host-centric interoperability for legacy networks, with domains of the Public Switched Telephone Network (PSTN), BoxOS and the Session Initiation Protocol (SIP). We extend the approach to model and analyze interoperability of information-centric services and architectures, since we are dealing with radically different network designs (name-based networking *vs.* address-based [9]) and required properties (node-to-content reachability *vs.* node-to-node reachability [10]). Additionally, we extend the classic Alloy-based model finding approach, such as in [19], to a model counting one, to quantitatively analyze the impacts of failure and mobility. An important feature of Alloy is its strength in efficiently handling graph structures and properties [18], a feature that we benefit from, in two ways: 1) the composite network topology, and 2) a graph-based information namespace. Further, Alloy helps provide proofs for properties with a reasonably large scope [18].

3 Modeling Information-Centric Interoperability

We now describe the basics of our formal model[1]. First and foremost, let us define information-centric interoperability (ICI):

Definition 1. *A sequence of interconnected domains in a network are information-centrically interoperable if and only if any client in any of the domains can access information-centric services provided in any other domain.*

Throughout this paper, we use the term "network" to mean "a composition of multiple network domains", each domain being a different type of standalone architecture (*e.g.*, IP, NDN, or MF). An interoperability framework (such as [3]) is a set of protocols and architectural components that allow interconnected networks of different types to interoperate. Information-centric services are broadly sub-categorized as: 1) requesting for and retrieving content (pull-based), and 2)

[1] Full source files are available in [1].

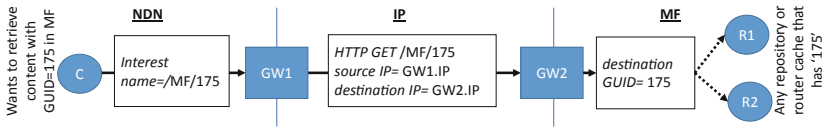

Fig. 1. Information-centric interoperability (ICI): request for content

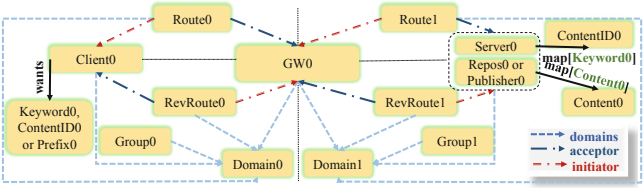

Fig. 2. Example (partial) instance for ICI Alloy model (objects and relations)

subscribing to and receiving content (push-based). Both of these may be based on namespaces defined by content producers. An example 3-domain ICI scenario is depicted in Fig. 1. As shown, ICI accesses content by name, rather than an address. Also, requests can be satisfied at any cache node, not just the original server. As for formal analysis, in ICI, the main property we care about is *node-to-content reachability* [10], while in traditional host-centric interoperability (HCI) analysis [19], the focus is on *node-to-node reachability*.

We model our networked environment using Alloy's relational and logical atoms. We have *Domains* (as abstract signatures), each of which can be an IP, NDN, or MF type (extended signatures) (Listing 3.1). A *Node* is at least in one *Domain* and has at least one *NodeID*. A *Node* can be either a *Client*, *Repos* (repository/server), or *GW* (gateway). A gateway is associated with exactly two *Domains* (constrained using facts), that it is stitching together (Listing 3.2.)

Listing 3.1. Domains

```
abstract sig Domain{}
sig IPdomain extends Domain{}
sig NDNdomain extends Domain{}
sig MFdomain extends Domain{}
```

Listing 3.2. Nodes

```
abstract sig Node{domains: some Domain, id:
    some NodeID}
sig Client extends Node{...}{...}
sig Repos extends Node{...}{...}
sig GW extends Node{...}{#domains=2 && ...}
```

Our declarations specify a network *meta-model* [8], which maps to a number of instances (models) each being a network configuration (*i.e.*, with their own topology, content, namespace, *etc.*). An example 2-domain instance is depicted in Fig. 2, as a high-level schematic, showing objects and their inter-relations. The *Client* here wishes to retrieve some *Content* using its *ContentID* or a (set of) *Keyword*(s). Objects of type *Route* and *RevRoute* (reverse route) couple the notion of "a series of links" and "packets carried over them", the packet carrying content request and response, respectively. A *Route* has attributes such as *initiator*, *acceptor*, and a request for *ContentID*. We also extend signatures to

add more fine-grained, domain-specific characteristics. One of *Route*'s extended object types, namely *IPRoute*, inherits its attributes and constraints, and also has additional attributes such as *srcIPaddress* and *destIPaddress*, and constraints saying that source and destination IP addresses must correctly correspond to initiator and acceptor nodes. Gateways perform translation for forwarding requests (over a composition of *Route*s), and retain state information which they use to forward the content back to the client (over composition of *RevRoute*s). We also add a number of additional facts, such as uniqueness of node ID, absence of self-looping routes, and the existence of one-to-one mapping between NDN's forward and reverse routes (to reflect NDN's RPF policy [20]).

We define a global-state relation C that captures routes to/from gateways. To model connectivity, we use the transitive closure of the route-connections relation C where $(r1, r2) \in C$ if and only if there exists a gateway between two domains that connects routes $r1$ and $r2$. *E.g.*, if we have $C = \{(r1, r2), (r2, r3)\}$, then its transitive closure $C^+ = \{(r1, r2), (r2, r3), (r1, r3)\}$ will represent existing paths of any length (*i.e.*, number of routes). We define object type *Connections* (as a singleton) to capture these connections (*i.e.*, relation C); it has attributes being relations themselves, primarily *connected* and *revconnected*, to capture connection relations of *Route*s and *RevRoute*s respectively. Relation *revconnected* has an additional constraint, which says that for two reverse routes $rr1$ and $rr2$ connected at gateway gw, corresponding state information (associated with the *ContentID* or other multiplexing/demultiplexing values in $rr1$ and $rr2$) must be stored on gw, so that the content can be carried over this cascade of reverse routes towards the consumer (Listing 3.3). Additionally, we define a fact (*path_exists*, Listing 3.4) that ensures any two nodes are connected (through one or multiple *Route*s or *RevRoute*s), to reduce our instance space to only the ones with strongly connected topology.

Listing 3.3. Connections: capture the connectivity of routes and groups

```
one sig Connections{connected: Route->Route, revconnected: RevRoute->RevRoute,
     chain: Group->Group, revchain: Group->Group}
fact connectivity{ -- conditions for two (reverse) routes being ‘‘connected’’
   all r1,r2:sRoute, c:Connections |
      (r1->r2) in c.connected <=>
         r1.acceptor = r2.initiator && r1.contentID = r2.contentID &&
               r1.reposdomain = r2.reposdomain -- requests for same content
   -- similar condition for RevRoute paths (with extra criteria: gateway state
        information should match for the two connecting reverse routes) ...
}
```

Listing 3.4. Constraints to ensure that a path exists between any two nodes

```
fact path_exists{
   all co:Connections, disj n1,n2:Node, cid:ContentID, rd: Domain |
      (some r:Repos | rd in r.domains =>
          (some r1,r2:Route | (r1->r2) in ^(co.connected) && r1.initiator = n1
               && r2.acceptor = n2 && r1.contentID = cid && r2.contentID = cid
               && r1.reposdomain = rd && r2.reposdomain = rd))
    -- similar condition for RevRoute paths ...
}
```

While *Route*s represent unicast exchange paths, we define *Group*s to denote multicast groups (one-to-many communication), enabling push-based notification models. Following the principles of ICN, each group is associated with a

content name *Prefix* [6] and can be used for publish/subscribe exchanges regarding that prefix. Each group belongs to one domain. To model a connection of groups across multiple domains, we add relation attributes *chain* and *revchain* to *Connections* (Listing 3.3), to capture connectivity of groups (as a chain) for subscription and publication respectively. To ensure strong connectivity, we add a fact that says any two groups serving the same prefix are chained (Listing 3.5).

Listing 3.5. Constraints to ensure that a chain of connectivity exists between groups

```
fact GroupRules{
   all disj g1,g2:Group, co:Connections | -- group chain conditions
       (g1->g2) in co.chain <=>
           (g1.prefix = g2.prefix &&
           (some gw:GW| g1.domain in gw.domains && g2.domain in gw.domains))
   all disj d1,d2:Domain, co:Connections, p:Prefix | -- chains for each prefix
       some disj g1,g2:Group |
            g1.domain = d1 && g2.domain = d2 && (g1->g2) in ^(co.chain) &&
            g1.prefix = p && g2.prefix = p
   -- similiar conditions for revchain ...
}
```

Content naming is integral in ICI. We define names, *i.e.*, *ContentID* objects for each *Content*. Based on domain type, *ContentID* can be either *URL* (in IP), *NDNName* (in NDN) or *ContentGUID* (in MF) (Listing 3.6). Each *ContentID* is a leaf node under a *Prefix* in the prefix tree (*PTree*). An example prefix tree is shown in Fig. 3, which represents the network's content namespace. *PTree* may contain a number of fragmented sub-trees (*i.e.*, as a forest), each sub-tree representing the namespace of a different (set of) content provider(s) in different domains. To represent the structure of hierarchical prefixes, we use binary relations to model the immediate parent-child relationship between prefixes in *PTree*. In Fig. 3, the relation $P = \{(P1, P2), (P1, P3), (P2, P4), (P2, P5)\}$ represents such relationships, and is captured in the prefix-to-prefix relation *map* in *PTree* (Listing 3.6). We also use its transitive closure to model the ancestor-descendant relationships. We add additional facts to ensure basic constraints on the tree, such as the non-existence of loops.

Listing 3.6. Content IDs and Prefix Tree

```
abstract sig ContentID{prefix: Prefix}
sig URL extends ContentID{} -- if in IP
sig NDNName extends ContentID{} -- if in NDN
sig ContentGUID extends ContentID{} --if in MF
sig Prefix{parent: lone Prefix, domains: some
     Domain} -- each Prefix has exactly one
     parent and is at least in one domain
one sig PTree {map: Prefix set -> set Prefix}
```

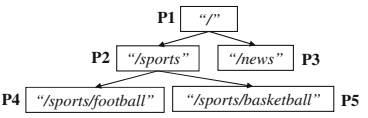

Fig. 3. Prefix tree example

4 Satisfying Information-Centric Service Properties

There are a number of important properties that are required from the framework, to ensure interoperability as defined in Definition 1. We consider properties of two classes of information-centric services here: pull-based (for unicast request/response), and push-based (for multicast publish/subscribe) content retrieval. We further divide the pull-based services into two categories: static

content retrieval (SCR) and dynamic content retrieval (DCR). This distinction is important as the nature, protocol for retrieval, and thus formal properties of the two are different: static content is one that does not change in a long time (*e.g.*, a movie) and can be retrieved from its original producer as well as a cache, while dynamic content is created once on demand (*e.g.*, result of a Google search), and must be retrieved from its original server (not from a cache). Additionally, we assume content requests are assumed to be genuine and correct, *i.e.*, false and bogus content requests are not our focus here.

We study essential invariant properties, guaranteed to hold at all times. These properties are primarily associated with content-oriented reachability and returnability. We formally specify these properties, using Alloy predicates and assertions. For verification, Alloy's built-in model finding engine is used to find satisfying instances and counterexamples. Any counterexample found indicates interoperability violations: *e.g.*, a client cannot generate a request native to its domain, or the gateway does not know what to do with a returned response.

4.1 Pull-Based Retrieval: Request/Response

Static Content Retrieval. In the static content retrieval (SCR) service, the request packets carry content IDs which the client requests, and the response packets produced by repositories (can be content producers or router caches) carry the data associated with that content ID. We describe two of SCR's essential content-oriented properties using Alloy (Listings 4.1 and 4.2).

Property 1.1. SCR Reachability: For every client that wants to retrieve content associated with a content ID and has a direct route to a gateway, there is a repository with content having that ID reachable from that gateway.

Property 1.2. SCR Returnability: For every client that reaches a repository with a request, there is a path back to the client for the response with the content.

Listing 4.1. SCR reachability property

```
pred reach[c:Client, cid:ContentID, re:Repos, gw:GW]{ -- reachability predicate
    all co: Connections | cid in c.want => -- if requested
       (some r:Route, con:Content | r.initiator = c && r.acceptor = gw &&
       r.contentID = cid && (cid->con) in re.map =>
           some r1,r2:Route | (r1->r2) in ^(co.connected) && r1.initiator = gw &&
           r2.acceptor = re && r1.contentID = cid && r2.contentID = cid &&
           r1.reposdomain in re.domains && r2.reposdomain in re.domains)}
assert reach{ -- reachability assertion
    all c:Client, cid:ContentID| some re:Repos, gw:GW | reach[c,cid,re,gw]}
```

Listing 4.2. SCR returnability property

```
pred return[c:Client, cid:ContentID, re:Repos, gw:GW]{ -- returnability
     predicate
    all co:Connections | some gw1:GW | reach[c,cid,re,gw1] => -- if reachable
      (some r,r1,r2:RevRoute | (r1->r2) in ^(co.revconnected) &&
      r1.initiator = re && r2.acceptor = gw &&
      r1.content = re.map[cid] && r2.content = re.map[cid] &&
      r.initiator = gw && r.acceptor = c && r.content = re.map[cid])}
assert return{ -- returnability assertion
    all c:Client, cid:ContentID, re:Repos | some gw:GW | return[c,cid,re,gw]}
```

Dynamic Content Retrieval. In DCR, every request has to be mapped to a unique response, as opposed to SCR. To facilitate this, having a *demux* value (for multiplexing/demultiplexing) is essential for DCR, to provide the correct mapping of responses to requests; since every generated response is specific to not just the request's name, but also its input parameters. To access dynamic content from a server, a client generates a query for which the gateway keeps state as <nodeID, demux> of the requesting side and <demux> for the serving side. Reachability and returnability are still important in DCR (Properties 2.1–2.2). However, if the same SCR protocol is used for DCR, there can be *conflicts* between multiple requests, *e.g.*, a cached content may get sent back to multiple distinct clients. Therefore, we define no-conflict properties for DCR (Property 2.3).

Property 2.1–2.2. DCR Reachability and Returnability: These two properties are similar to those of SCR; with the difference being additional constraints regarding elements of DCR requests, *i.e.*, including generation and verification of the correct *demux* values at gateways (*i.e.*, in addition to contentID, *etc.*).

Property 2.3. No-conflict between distinct requests/clients: For every client that searches for two distinct content items (*no-conflict-A*, Listing 4.3), or a dynamic content requested by two different clients (*no-conflict-B*, Listing 4.4), two distinct, appropriately associated responses, should be received back. In *no-conflict-A*, the focus is on the distinction between two *return*-ed contents, associated with two distinct requests made by a given *Client* for distinct *Keywords* $k1$ and $k2$. On the other hand, *no-conflict-B* focuses on the distinction between two *return*-ed contents, associated with requests for a particular *Keyword* initiated by two distinct *Clients* $c1$ and $c2$.

This property shows the importance of having two separate *demux* values in packets, namely both the request ID (required for Property 2.3.a) and client ID (required for Property 2.3.b), to make each dynamic request globally unique, for correct multiplexing/demultiplexing. If we remove either of those two elements, this property will be violated and counterexamples will arise; *i.e.*, the gateway would not know how to demultiplex incoming response data to serve the correct, corresponding requesting client.

Listing 4.3. DCR - No conflict between 2 distinct requests from the same client

```
assert no-conflict-A{ -- Property 2.3.a
    all c:Client, disj k1,k2:Keyword | some s1,s2:Server, gw1,gw2:GW |
        return[c,k1,s1,gw1] && return[c,k2,s2,gw2] => some n1,n2: NodeID,
        d1,d2,d3,d4:Demux |
            (n1->d1->d2) in gw1.state && (n2->d3->d4) in gw2.state &&
            n1 in c.id && d1 in c.demux && d2 in gw1.demux &&
            n2 in c.id && d3 in c.demux && d4 in gw2.demux &&
            !(n1 = n2 && d1 = d3 && d2 = d4) && (some disj r1,r2:RevRoute |
                r1.initiator = gw1 && r1.acceptor = c && r1.contentID = s1.map[k1]
                && r1.demux = d1 && r2.initiator = gw2 && r2.acceptor = c
                && r2.contentID = s2.map[k2] && r2.demux = d3)}
```

Listing 4.4. DCR - No conflict between 2 identical requests from two distinct clients

```
assert no-conflict-B{ -- Property 2.3.b
    all c1,c2:Client, k:Keyword | some s1,s2:Server, gw1,gw2:GW |
        return[c1,k,s1,gw1] && return[c2,k,s2,gw2] => some n1,n2: NodeID,
        d1,d2,d3,d4:Demux |
            (n1->d1->d2) in gw1.state && (n2->d3->d4) in gw2.state &&
            n1 in c1.id && d1 in c1.demux && d2 in gw1.demux &&
            n2 in c2.id && d3 in c2.demux && d4 in gw2.demux &&
            !(n1 = n2 && d1 = d3 && d2 = d4) && (some disj r1,r2:RevRoute |
                r1.initiator = gw1 && r1.acceptor = c1 && r1.contentID = s1.map[k]
                && r1.demux = d1 && r2.initiator = gw2 && r2.acceptor = c2
                && r2.contentID = s2.map[k] && r2.demux = d3)}
```

4.2 Push-Based Retrieval: Publish/Subscribe

In pub/sub, we have domain-specific multicast groups that are associated with prefixes [6]. We want a client to be able to subscribe to and receive all relevant publications in accordance with the prefix tree of the namespace over "chain" of groups across domains. Groups $G1$ and $G2$ form a chain if and only if the publisher of $G1$ can be a subscriber of $G2$, and is then able to relay data received from $G2$ to his subscribers in $G1$.

Property 3.1. Ability to subscribe to any prefix. For every client that wants to retrieve future publications under/associated with an existing prefix and has a direct route to a gateway, if there is some publisher that will publish content under that prefix, then that publisher is accessible through a chain of groups.

Property 3.2. Ability to receive any content published directly associated with the subscribed prefix. For every client who is subscribed to a prefix and can reach the associated publisher, there is a path back to the client to carry any content with a content ID belonging to that prefix. For example, a subscriber of $P2$ in Fig. 3 should receive publications pertaining to $P2$ across domains.

Property 3.3. Ability to receive all content published that is associated with prefixes under the subscribed prefix. This property says that for every client that has subscribed to a prefix and has reached the associated publisher, there is a path back to the client to carry any content with content ID either directly belonging to that prefix or under it in the hierarchy on the prefix tree. For example, a subscriber of $P2$ in Fig. 3 should receive publications pertaining to $P2$ *and also* $P4$ across domains. The assertion *rcvall* in Listing 4.5 depends on how relationships among groups and also between content IDs and prefixes are represented by *Connections* and *PTree*. For a domain with a namespace that does not capture relationships between prefixes, *i.e.*, does not map a prefix to a set of multiple relevant prefixes according to a graph, then *rcvall* would be equivalent to receiving a single content element (Property 3.2). Properties 3.1–3 collectively model and verify properties of a service offering hierarchical pub/sub.

Listing 4.5. Pub/Sub - receiving all relevant publications

```
assert rcvall{ -- all relevant publications in accordance with the prefix tree
    all pub:Publisher, con:Content, cid:ContentID |
        all co:Connections, pt:PTree | (cid->con) in pub.map =>
            ((some c:Client, p:Prefix | (p in c.want || (all p1:Prefix |
            (p1->p) in ^(pt.map) && p1 in c.want)) && cid.prefix = p =>
                (some r1,r2:Route | r1.initiator = pub && r2.acceptor = c &&
                (r1>r2) in ^(co.connected) && some g1,g2:Group |
                    g1.domain = pub.domain && g2.domain = c.domain &&
                    g1.prefix = p && g2.prefix = p && (g1->g2) in
                        ^(co.revchain)))}
```

5 Reasoning About Failure and Mobility

In addition to the basic invariants (Sect. 4), there are other important aspects of formal analysis of networks that warrant a more quantitative analysis; among them are failure and mobility analysis. Failures and mobility of nodes can occur in a network, causing disruption and lack of content availability. To better compare how different network architectural components, *e.g.*, routing, impact the number of success and violation scenarios, we perform model counting [7]. While we can consider the probability for all instances as being equal, we can also calculate each instance's probability by additionally factoring in the real-world probability of individual elements causing failures and mobility, provided as external information (*e.g.*, the probability of a gateway failing when processing a content request, a route disconnecting while carrying a packet, *etc.*). Thus, we can provide a more realistic probabilistic analysis for the effect of failures and mobility using weighted model counting methods [5].

While the Alloy Analyzer (v4.20) [2] allows for a limited, graphical iteration over instances, it does not enable an explicit counting of instances in an efficient manner. To perform model counting, we wrote an application [1] that counts all SAT solutions, using the SAT4J solver [13] (SAT4J can be replaced by any off-the-shelf SAT solver). We feed the Alloy model and properties, in Kodkod format [17], to our application. Predicates and assertions are used for counting instances that satisfy or violate (counterexamples) respectively. Through this counting, we can also look into the details (relations and values) within each instance, and gain insight such as possible cause of violations (in case of counterexamples) and calculate the probability of occurrence of each instance in real-world scenarios. While we do not focus on the performance aspects of model counting in this paper, optimizations of this procedure can be leveraged for enhancing the scalability of our approach in case of very large problem sizes. At a minimum, our approach can provide a rough estimate of failure probabilities. Even if the model counting provided by the SAT solver is through "approximate" model counting (*e.g.*, using repetitive halving procedures) [4] rather than an "exact" one, it still gives us a good enough assessment of the degree of success and violation of properties.

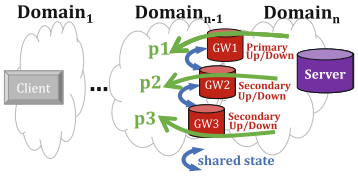

Table 1. Model finding

Domain n constraints	Returnability
Const. 1	✗
Const. 2	✗

Table 2. Model counting

$Domain_n$ constraints	Returnability		
	I	C	R
Const. 1	x_1	y_1	$x_1/(x_1+y_1)$
Const. 2	x_2	y_2	$x_2/(x_2+y_2)$

Fig. 4. Gateway failure scenario

5.1 Failure

Our interoperability framework depends on gateways that retain state information. What would happen to a response packet if that state is lost at the gateway for any reason? For reliability, we consider state sharing between redundant gateways that have the same domains on either side. Figure 4 depicts an example for this. Consider the gateway that received the request and created the state as the *primary* gateway for the request ($GW1$ in the Fig.), and the replicas that have the shared state as the *secondary* gateways ($GW2$ and $GW3$). Formally, we add an extra condition to our reachability and returnability properties such that, for two routes to connect, the gateway attaching them must be up and running at the time the packet is received. Additionally, for returnability, the state information must be present at the gateway. If any gateway goes down, the corresponding potential path going through it ($p1$–3) back for the content cannot be leveraged. If the gateway is neighboring an NDN domain (*e.g.*, in $Domain_n$ or $Domain_{n-1}$), then the gateway has to be the primary only, for correct operation with the NDN reverse-path-forwarding (RPF) policy [20]. For other domain types, a secondary gateway that is active and has the shared state information is adequate to forward the response data back. We model the conditions representing this in Alloy as shown in Listing 5.1.

Listing 5.1. Failure scenario constraints: impact of gateway status on route connectivity

```
all r1,r2: Route, c:Connections | -- forward routes (request) condition
    (r1->r2) in c.connected <=> r1.acceptor = r2.initiator &&
    r1.initiator.status1 in Up && r2.initiator.status1 in Up
all r1,r2: RevRoute, c:Connections | -- reverse routes (response) condition
    (r1->r2) in c.connectedR <=> r1.acceptor = r2.initiator &&
    r1.initiator.status1 in Up && r2.initiator.status1 in Up &&
    ((r1.domain in NDNdomain || r2.domain in NDNdomain) =>
        r1.acceptor.type in Primary) -- NDNdomain enforces RPF policy
```

Gateways can go down due to various reasons such as completely failing or just losing state information due to a software failure. Our method can be used to reason about various scenarios and measure failure probability given an input configuration space, *i.e.*, a set of Alloy facts that set constraints on some objects or variables while relaxing others. As Table 1 shows, a simple model finding analysis does not provide a helpful comparison between different such constraints:it will say that both cases lead to counterexamples raised (*e.g.*, for the

case that all gateways go down). To gain a better assessment of which constraint does better, we resort to model counting (Table 2). Using model counting, we can count (satisfying) instances (I) and counterexamples (C), and calculate (even if approximately [7]) the probability of reliability ($R = I/(I + C)$). This reliability indicates to what degree interoperability is impacted in presence of failure, given certain conditions (*i.e.*, choice of domain policies, *etc.*).

5.2 Mobility

To model and analyze mobility (Fig. 5), we add the notion of "time" to our model. In particular, we associate timeout values to state entries at gateways and *birthTime* and *deathTime* to routes (and similarly for reverse routes). We assume gateways are stationary, but other nodes can move, causing the "death" of their route (*route*1) to/from their closest gateway. A new route to the gateway is "born" (*route*2) after some time, assuming the existence of a domain-specific method to handle mobility. Temporal conditions must be incorporated into reachability/returnability properties. The most critical case is when a mobility event occurs while the packet is in-flight [21]. At high-level, the sum total latency formulated as $firstDeliveryAttempt + recovery + secondDeliveryAttempt$, must be below a certain *expiration* threshold (at every gateway and consumer). $firstDeliveryAttempt$ is the incomplete partial delivery latency via *route*1 and *secondDeliveryAttempt* is the delivery via *route*2 (continuation in MF, and complete retransmission in IP and NDN). The *recovery* delay is the time it takes for the packet to be transmitted back on the new path again; it includes re-registration (MF and IP), FIB re-population (IP and NDN in case of provider mobility) and/or PIT re-population (for NDN in the case of consumer mobility) delays [16,20,21]. Using this formal method, we check properties in the presence of mobility, find appropriate values for a timeout threshold on gateways and investigate the effect of domain-specific mobility handling methods on interoperability. Listing 5.2 generally specifies how the reachability property (to deliver a named request) depends on the condition of mobility (stationary or mobile) and the domain policy on handling mobility (early binding or late binding). Returnability is similarly specified (for content). Predicates *stationary*, *mobileEarlyBinding*, and *mobileLateBinding* specify timing conditions for successful delivery assuming their corresponding conditions (details of the three properties are omitted here due to space but are in [1]). As shown in Fig. 5, we only consider intra-domain mobility here, *i.e.*, the mobile node changes its location and point of attachment, but stays within its domain.

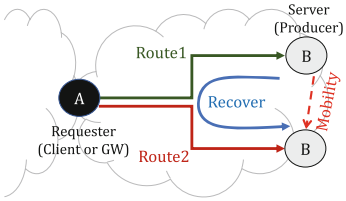

Fig. 5. Mobility scenario example: Route 2 established after B moves and changes its point of attachment

Table 3. Verif. scopes for properties of ICI services

Property	Client	GW	Repos/ Server/ Publisher	Domain	ContentID/ Keyword	Prefix	PTree	Content	Connections	Route	RevRoute	NodeID	Port	NDNreqID	MFreqID	Group	GroupID	Verif. Result
1.1	1	2	1	3	1			1	1	12	12	6						✓
1.2	1	2	1	3	1			1	1	12	12	6						✓
2.1	1	2	1	3	1			1	1	12	12	6	6	3	3			✓
2.2	1	2	1	3	1			1	1	12	12	6	6	3	3			✓
2.3.a	1	1	1	2	2			2	1	8	8	4	4	4	4			✓
2.3.b	2	1	1	2	1			1	1	8	8	5	4	4	4			✓
3.1	1	2	1	3	3	3	1	3	1	12	12					9	9	✓
3.2	1	2	1	3	3	3	1	3	1	12	12					9	9	✓
3.3	1	2	1	3	3	3	1	3	1	12	12					9	9	✓

Listing 5.2. Reachability in presence of mobility

```
pred reach[c:Client, p:Producer, cid:ContentID]{ -- a client and content producer
   (stationary[c,p,cid] && p.mobility in Stationary) -- producer p stationary
   || (mobileLateBinding[c,p,cid] && p.mobility in Mobility
       && Domain.binding in LateBinding) -- p mobile, domain does late binding
   || (mobileEarlyBinding[c,p,cid] && p.mobility in Mobility
       && Domain.binding in EarlyBinding)} -- p mobile, domain does early binding
```

6 Implementation and Results

We implemented the ICI framework discussed in our model in Sect. 3, with gateways for interoperation among IP, NDN, and MF (Fig. 1 as an example) in a software testbed (implementation details in [11]). This section provides the description and results of our analysis of the ICI framework (our Alloy source code is approximately 800 lines of code in total [1]).

To check for correctness, we performed verification (supported by Alloy Analyzer's model finding engine) of our ICI framework model, against the information-centric services properties (as specified in Sect. 4). In order to reach convincing proofs (as advised in [18]), we pick the scopes for verification in Alloy that are large enough to contain all necessary cases (*i.e.*, minimum number of actors and objects for each service), and small enough so that we do not encounter model explosion. The scopes, *i.e.*, upper bounds on the number of key objects, are provided in Table 3. For most properties, we consider 1 *Client*, 1 *Server*, 1 *Content*, and 1 *ContentID*. That is, different <client, request> pairs are considered independent of each other. However, for Properties 2.3.a/b, such a dependency matters, and we want to show lack of conflicts. For Property 2.3.a, we set 1 *Client* and 2 *Content*s (to generate scenarios where *one* client makes *two* separate request for *two* different contents), and for Property 2.3.b, we set 2 *Client*s and 1 *Content* (to look for conflicts between request for *one* content but by *two* clients). We use 3 *Domain*s for most properties, as it contains all cases with 1, 2, or 3 domains of any type, *i.e.*, IP, NDN, or MF. Also, with upper

Table 4. Failure analysis results

Cases	Reachability			Returnability		
	I	C	R	I	C	R
No domain constraints	290	0	1.00	56	210	0.21
One NDN domain	176	0	1.00	8	168	0.04

Table 5. Mobility analysis results

Cases	Stationary			Mobile					
				Late binding			Early binding		
DL range	I	C	R	I	C	R	I	C	R
[0, 20]	100	8	0.92	72	24	0.75	92	64	0.58
[0, 18]	96	0	1.00	72	8	0.90	92	48	0.65
[0, 15]	84	0	1.00	64	0	1.00	92	24	0.79
[0, 10]	64	0	1.00	44	0	1.00	84	0	1.00

bound n on the total number of *Node*s, *i.e.*, sum of *Client*s, *Server*s, and *GW*s, we specify the upper bound on the number of *Route*s (as well as *RevRoute*s) to be $n(n-1)$, enabling the existence of any possible (uni-directional) route. For pub/sub services (*i.e.*, Properties 3.1–3), we set 3 *Prefix*es, *ContentID*s, and *Content*s, to capture inter-relationship of content IDs in a large enough namespace. Additionally, with the upper bound on *Domain*s and *ContentID*s both set at 3, we set the upper bound on total number of *Group*s (and *GroupID*s) to be $3 \times 3 = 9$, so as to contain cases with one group per content ID per domain. The blank cells in Table 3 indicate either "N/A" or "no particular upper bound set", in which case Alloy picks a default value. Within this scope, our verification passes successfully for each property, showing that the stated properties are *invariants* of our ICI framework. In other words, the framework design ensures that *any sequence of interconnected IP, NDN, and MF domains are information-centrically interoperable.*

We use our proposed model counting approach to analyze scenarios with the failure of one or multiple gateways. The most important factor affecting returnability in scenarios with the possibility of failure, is domain-specific routing policies, in particular, whether or not it allows for a secondary (backup) gateway to relay the returning response content. Different domains have different policies; MF and IP decouple the forward (request) and return (response) paths, and they can be delivered through different gateways, while NDN strictly requires the two paths to be the same, due to RPF policy. To investigate the impact of that policy, we considered a scenario of two domains, with two gateways between them (one primary and one secondary), sharing state. Both gateways are *Up* (working) when the request is forwarded, and either *may* go *Down* (failing) when the response is one its way back. Table 4 shows different scenarios for reachability and returnability, with different domain constraints (with different routing policies). In particular, the two domain constraints we consider are the following: 1) no constraint on what any of the domains are; and 2) one domain is definitely NDN. The table shows the values of I (instances), C (counterexample), and R (reliability) for each scenario, as defined in Sect. 5. Our results for R in Table 4 prove that having an NDN domain on one side dramatically reduces the returnability reliability ratio, since basic NDN forwarding strictly forbids data coming back on a different path than the original path taken by the request.

When a content producer (server) moves while a content request is in-flight (Fig. 5), the domain's handling of mobility recovery determines the reachability probability. NDN and IP use early binding with retransmissions, while MF supports late binding with rerouting. We compare the impact of these mechanisms and techniques using our model counting method, with results shown in Table 5. Our modeled scenario consists of two nodes in a domain, one requester (client or gateway) and one server (producer) with a route established among them. The 'Stationary' columns in the table show reachability results in the stationary server case. With 'Mobile', the route dies due to a server mobility event (at time $t = 10$), leading to the birth of the second route. We set the re-registration and re-population delays to 1 each. Also, a retransmission is initiated 1 time unit after the mobility event. Different binding techniques for mobility, *i.e.*, late and early binding, are also shown in Table 5. We compare cases with different ranges for *Delivery Latency* (DL), which is time approximately needed for a packet to travel from requester to server. For a delivery latency range of $[0, 20]$, we see a higher R for stationary *vs.* mobility cases. The reason is that when the server does not move, the original route stays active, thus providing a higher chance for requests to reach the server. Comparing the two binding techniques, late binding leads to higher chance of reachability compared to early binding, as it allows for packets to be re-routed on the newly-born route, rather than retransmitting from the original requester. These results serve as proof that under similar scenarios, late binding outperforms early binding in ICI. Also, changing the delivery latency ranges, we can find out at what points, reachability is an invariant (if ever) under mobility conditions. As the table shows, with ranges within $[0, 18]$, $[0, 15]$, and $[0, 10]$ (rows in Table 5 labeled in first column accordingly), reachability becomes an invariant in cases of Stationary, Late Binding, and Early Binding, respectively; as zero counterexamples are raised. With a small enough delivery latency ranges, namely $[0, 10]$, reachability becomes an invariant, no matter the mobility conditions or binding techniques. Our approach can be used to find such points of invariance, comparing different techniques, and prove them.

7 Conclusion

This paper presented an Alloy-based formal analysis model for information-centric interoperability (ICI) for Future Internet environments. We showed how model finding can be used to analyze basic (reachability and returnability) properties of ICI. Additionally, our proposed model counting approach analyzes failure and mobility scenarios, which we used to prove the negative impact of certain routing policies (particularly, reverse path forwarding), and the helpfulness of certain mobility-handling mechanisms (particularly, late binding), providing necessary confidence and guidelines for Future Internet interoperability.

Acknowledgements. This work was supported by the US Department of Commerce, National Institute of Standards and Technology (award 70NANB17H188) and US National Science Foundation grants CNS-1455815 and CNS-1818971.

References

1. https://www.cs.ucr.edu/~mjaha001/ICI.zip
2. Alloy: A Language and Tool for Relational Models. http://alloy.mit.edu/alloy/
3. Carofiglio, G., et al.: Enabling ICN in the internet protocol: analysis and evaluation of the hybrid-ICN architecture. In: ACM ICN (2019). https://doi.org/10.1145/3357150.3357394
4. Chakraborty, S., Meel, K.S., Vardi, M.Y.: A scalable approximate model counter. In: Schulte, C. (ed.) CP 2013. LNCS, vol. 8124, pp. 200–216. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40627-0_18
5. Chavira, M., Darwiche, A.: On probabilistic inference by weighted model counting. AI **172**(6–7), 772–799 (2008). https://doi.org/10.1016/j.artint.2007.11.002
6. Chen, J., et al.: COPSS: an efficient content oriented publish/subscribe system. In: ACM/IEEE ANCS (2011). https://doi.org/10.1109/ANCS.2011.27
7. Gomes, C.P., Sabharwal, A., Selman, B.: Model counting: a new strategy for obtaining good bounds. In: AAAI (2006)
8. Jackson, D.: Alloy: a lightweight object modelling notation. TOSEM **11**(2), 256–290 (2002)
9. Jacobson, V., et al.: Networking named content. In: CONEXT (2009)
10. Jahanian, M., Ramakrishnan, K.K.: Name space analysis: verification of named data network data planes. In: ACM ICN (2019). https://doi.org/10.1145/3357150.3357406
11. Jahanian, M., et al.: Managing the evolution to future internet architectures and seamless interoperation. In: Proceedings of the 29th International Conference on Computer Communication and Networks (ICCCN) (2020)
12. Jahanian, M., et al.: Graph-based namespaces and load sharing for efficient information dissemination in disasters. In: ICNP (2019). https://doi.org/10.1109/ICNP.2019.8888047
13. Le Berre, D., Parrain, A.: The SAT4J library, release 2.2, system description. J. Satisf. Boolean Model. Comput. **7**, 59–64 (2010)
14. Li, Y., et al.: A survey on network verification and testing with formal methods: approaches and challenges. IEEE Commun. Surv. Tutor. **21**(1), 940–969 (2019)
15. McCauley, J., et al.: Enabling a permanent revolution in internet architecture. In: ACM SIGCOMM (2019). https://doi.org/10.1145/3341302.3342075
16. Raychaudhuri, D., et al.: MobilityFirst: a robust and trustworthy mobility-centric architecture for the future internet. ACM SIGMOBILE MCCR **16**(3), 2–13 (2012)
17. Torlak, E., Jackson, D.: Kodkod: a relational model finder. In: Grumberg, O., Huth, M. (eds.) TACAS 2007. LNCS, vol. 4424, pp. 632–647. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-71209-1_49
18. Zave, P.: A practical comparison of alloy and spin. Formal Aspects Comput. **27**(2), 239–253 (2015). https://doi.org/10.1007/s00165-014-0302-2
19. Zave, P.: A formal model of addressing for interoperating networks. In: Fitzgerald, J., Hayes, I.J., Tarlecki, A. (eds.) FM 2005. LNCS, vol. 3582, pp. 318–333. Springer, Heidelberg (2005). https://doi.org/10.1007/11526841_22
20. Zhang, L., et al.: Named data networking. ACM SIGCOMM CCR **44**(3), 66–73 (2014)
21. Zhang, Y., et al.: KITE: producer mobility support in named data networking. In: ACM ICN (2018). https://doi.org/10.1145/3267955.3267959