IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

SPECIAL SECTION ON FUTURE NETWORKS: ARCHITECTURES, PROTOCOLS, AND APPLICATIONS

Received January 29, 2017, accepted February 14, 2017, date of publication January 4, 2018, date of current version March 15, 2018.

Digital Object Identifier 10.1109/ACCESS.2017.2789330

A Secure Link State Routing Protocol for NDN

LAN WANG[1], (Senior Member, IEEE), VINCE LEHMAN[2], A. K. M. MAHMUDUL HOQUE[3], BEICHUAN ZHANG[4], YINGDI YU[5], AND LIXIA ZHANG[6], (Life Fellow, IEEE)

[1]The University of Memphis, Memphis, TN 38152, USA
[2]ReTrans, Memphis, TN 38120, USA
[3]Amazon, Toronto, ON M5J 0A1, Canada
[4]The University of Arizona, Tucson, AZ 85721, USA
[5]Facebook, Menlo Park, CA 94025, USA
[6]University of California, Los Angeles, CA 90095, USA

Corresponding author: Lan Wang (lanwang@memphis.edu)

This work was supported by NSF under Grant 1040036, Grant 1039615, Grant 1040868, Grant 1344495, Grant 1345142, Grant 1345318, Grant 1629009, Grant 1629769, and Grant 1629922.

ABSTRACT The Named-data Link State Routing protocol (NLSR) is a protocol for intra-domain routing in Named Data Networking (NDN). It is an application level protocol similar to many IP routing protocols, but NLSR uses NDN's interest/data packets to disseminate routing updates, directly benefiting from NDN's built-in data authenticity. The NLSR design, which was first developed in 2013 and deployed on the NDN test bed in August 2014, has undergone significant changes. Following an application-driven design approach, NLSR's development helped drive the development of the trust/security functionality of NDN libraries as well as a number of features in NDN's forwarding daemon and ChronoSync. In this paper, we describe the current design and implementation of NLSR, with emphasis on those features that differentiate it from an IP-based link state routing protocol: 1) naming: a hierarchical naming scheme for routers, keys, and routing updates; 2) security: a hierarchical trust model for routing within a single administrative domain; 3) routing information dissemination: using ChronoSync to disseminate routing updates; and 4) multipath routing: a simple way to calculate and rank multiple forwarding options. Although NLSR is designed in the context of a single domain, its design patterns may offer a useful reference for future development of inter-domain routing protocols.

INDEX TERMS Routing protocols, next generation networking, computer security.

I. INTRODUCTION

Named Data Networking (NDN) [1]–[3] is an information-centric Internet architecture addressing network usage and unsolved problems in the existing TCP/IP Internet. NDN changes the network service model from "delivering packets from one endpoint to another" to "fetching named data". By explicitly naming and signing data and by maintaining a stateful forwarding plane [4], NDN provides several desirable functions that are difficult to realize in IP, such as in-network caching, multicast delivery, multi-path forwarding, and data provenance.

An NDN routing protocol propagates routing updates and computes routes to *name prefixes*. Because NDN names are hierarchically structured and NDN's forwarding semantic is a *superset* of IP's, NDN's best route computation can use any of the *routing algorithms* that work for IP, e.g., link-state or distance-vector. However, an NDN routing protocol must be able to offer multiple next hops for packet forwarding to support NDN's multipath forwarding, where the multiple paths can be toward either one data producer or multiple producers of the same data.[1] Moreover, the protocol needs to propagate the reachability of name prefixes instead of IP address prefixes, and most importantly, use NDN's *Interest* and *Data* packets to exchange routing information instead of IP packets. Doing so also allows the routing protocol to benefit from NDN's built-in data authentication capability.

This paper describes the design and implementation of the Named-data Link State Routing protocol (NLSR), an intra-domain routing protocol for NDN. Since we sketched out the initial design of NLSR in 2013 [5], the design has gone through substantial revisions over the last four years as we gain deeper understandings of both NDN and NDN application development approaches through real implementations

[1]There is a difference between the two architectures' *semantic* in prefix reachability that needs to be taken into account in the routing protocol design (see [1, Sec. 4.1] and Section II-C of this paper).

and experimentation. Our goal in this paper is both to demonstrate the feasibility and benefits of building a routing protocol using NDN and to share our experience and observations with the community at large.

NLSR differs from IP-based link-state routing protocols in the following three major ways.

A. NAMING

NLSR uses hierarchically structured names to identify routers, routing processes, routing data, and keys as the relationship among them is inherently hierarchical. This design approach not only facilitates routing security (see below), but also allows NLSR to use all types of communication channels, e.g., Ethernet, IP, TCP/UDP tunnels, in the same way, as it has no dependency on specific types of addresses. In contrast, IP-based link-state protocols such as OSPF [6] typically use one of the addresses of a router, e.g., the smallest IP address, as its identifier. This approach has several drawbacks. First, it introduces a dependency on a particular type of addresses that are often embedded in router configuration and other management functions. Second, the identifier is unstable, i.e., it changes as the address changes. Third, the identifier lacks context information, which is important for security and management. Furthermore, these protocols do not explicitly name their routing processes, routing data, and keys. Without meaningful names identifying these entities and their relationships, it is much more difficult to monitor the routing system and diagnose its problems.

B. SECURITY

Since every NLSR routing message is carried in an NDN data packet containing a signature, a router can verify the signature of each routing message to ensure that it was generated by the claimed origin router and was not tampered with during dissemination. We devised a hierarchical trust model for routing within a single domain, based on common management structures and operational practices in a domain, to verify the keys used to sign the routing messages. The names in NLSR reflect the relationship among routing entities, enabling the use of NDN trust schemas [7] to automatically derive signing key names and verify received routing updates, which is implemented in NLSR.

C. MULTI-PATH FORWARDING

While IP either uses a single best next-hop to forward packets or limits its forwarding to multiple equal-cost paths in order to avoid forwarding loops, NDN can utilize multiple paths freely because it has built-in loop detection in the forwarding process. NLSR builds FIB entries with multiple next hops for each name prefix, even if the name prefix is originated by a single router.

In the remainder of this paper, we first introduce basic NDN concepts and discuss related work in Section II. We then articulate the rationales behind our design choices on naming, trust, LSA dissemination, and multipath routing calculation (Section III). We also present implementation details (Section IV) and experiment results (Section V), as well as share the lessons from our development and deployment (Section VI). Finally, Section VII concludes our work.

II. BACKGROUND AND RELATED WORK

A. NAMED DATA NETWORKING (NDN)

Communication in NDN is driven by receivers, i.e., data consumers, through the exchange of two types of packets: *Interest* and *Data*. A consumer puts the name of a desired piece of data into an *Interest* packet and sends it to the network. When a router receives the *Interest*, the router first checks the Content Store (CS), which contains previously received Data, for any matching data. If the matching *Data* packet is found, it is sent back on the incoming interface on which the *Interest* was received. Otherwise, the router examines the Pending Interest Table (PIT). If there exists an entry with the same name as the newly received *Interest*, the new incoming interface is added to the interface list so that a copy of the matching *Data* packet can be sent on all interfaces from which the *Interest* packets arrived. Finally, if the *Interest* does not have a matching PIT entry, it is forwarded to the next hop(s). Once the *Interest* reaches a node that has the requested data, the *Data* packet is returned. With the help of the PIT entries, this *Data* packet follows the reverse path of each pending *Interest* back to the requesting consumer.

Compared to IP routing, NDN's stateful forwarding plane changes the basic relationship between routing and forwarding [8] – forwarding decisions are made based on not only the routes and route ranking produced by the routing protocol but also a few other factors. More specifically, by maintaining a PIT, the forwarding plane can measure the performance (e.g., RTT) of each next hop in retrieving data. When multiple next hops exist in the FIB entry for an Interest, a ''forwarding strategy'' matching the Interest's name decides which next hop(s) will be used in forwarding the Interest based on routing ranking, forwarding plane measurements, and local policies. Note that the routing protocol's ranking of available next hops is still important in forwarding the initial *Interest* to a name prefix before measurement results are collected and for exploring alternative routes when the route in use fails to retrieve data [8].

B. EVOLUTION OF NLSR

The NLSR design has evolved significantly over the last few years since its first design sketch in 2013 [5]. The earlier design used the Sync mechanism provided by CCNx [9] to distribute Link State Advertisements (LSAs) between routers. However, during extensive testing on an 12-node testbed, we identified several problems with the CCNx sync/repo implementation including high memory consumption, inability to delete information from the repo, and failure to notify NLSR of routing changes when the update rate is high. These problems prevented us from deploying that version of NLSR on the NDN testbed in 2013.

As a new NDN platform [10] with a new forwarder and libraries was developed in 2014, we redesigned and reimplemented NLSR to work on the new platform. Several major design changes have significantly improved the performance of NLSR including using ChronoSync [11] to distribute LSAs, advertising *all* the name prefixes originated by a router in *one* name LSA, and detecting link failures using forwarding plane notifications (see Section III-G for more information). The new NLSR implementation [12] was released and deployed on the inter-continental NDN testbed in August 2014.

C. RELATED WORK

The routing protocol proposed by Dai *et al.* [13] looks similar to NLSR on the surface, but it differs from NLSR in the following important aspects. First, it is not designed to run in an ICN (Information Centric Networking)/NDN network; instead, it uses IP packets to deliver routing updates and did not address routing security. Second, and related, it uses OSPF as is to collect the topology and compute shortest path while NLSR uses ChronoSync to disseminate LSAs. Thus it inherits OSPF's problems (e.g., lack of naming for management and security, and computing only equal-cost multiple paths to each destination). In addition, the ChronoSync approach is receiver-driven, meaning that a router will request for LSA content when it has cycles. Thus it is less likely a router will be overwhelmed by a flurry of updates. Third, the multi-path forwarding function in Dai *et al.*'s protocol is limited to contents announced by multiple producers only, e.g., supporting anycast among a number of server replicas, while NLSR can forward *Interest* packets along multiple paths toward either the same producer or multiple producers of the same data.

Several NDN/ICN routing protocols were developed after our initial NLSR design [5] was published. Link State Content Routing (LSCR [14]) and Distance-based Content Routing (DCR [15]) aim to avoid permanent routing loops and to outperform traditional routing protocols when a name prefix is replicated at multiple sites in the network. LSCR disseminates *adjacency* information in the same way as IP link-state routing does but propagates *prefix* information selectively – among multiple instances of a prefix, a router will propagate only the nearest instance to its neighbors. DCR provides name prefix reachability without routers knowing the complete network topology. DCR uses distance information to calculate paths to prefixes, and similar to LSCR, does not propagate information about all the name prefix replicas in the network.

One common problem in LSCR and DCR is that *selective forwarding of routing advertisements may cause some data to be unreachable*. In NDN, announcing a name prefix to the network simply means that the announcer possesses some data under this name space but not necessarily all the data under it; NDN's adaptive forwarding plane can try alternative paths to retrieve all desired data. For example, an NDN router advertising the `/com/cnn` prefix may be able to obtain data under `/com/cnn/sports` and `/com/cnn/entertainment`, but not all the data with the `/com/cnn` prefix. In contrast, an IP router advertising an address prefix means that it can reach all the nodes under that address prefix. This semantic difference means that *an NDN router needs to propagate advertisements for the same name prefix originated by different nodes to ensure data retrieval* (see [1, Sec. 4.1]). Furthermore, the selective prefix advertisement saves bandwidth and CPU only when a router advertises each prefix in a separate *Data* packet. Although [14] and [15] compared favorably with our original NLSR design [5], which advertises each name prefix in a separate LSA, those results are no longer relevant as the current NLSR advertises all the prefixes originated by a router in a single LSA. The original design was based on the concern of having to fragment LSA data when the list of prefixes gets large. However, since NLSR is designed for intra-domain routing, the list of prefixes originated by each router should be usually small enough to fit into a single Data packet (8800 bytes by default in the current NDN platform implementation) in most cases. Therefore, we changed the design to reduce the number of LSA data packets.

III. DESIGN

As a link-state protocol, NLSR's basic functionality is to discover adjacencies and disseminate both connectivity and name prefix information. Such functionality may appear to be straight-forward to design and implement. However, because NLSR uses NDN's *Interest* and *Data* packets to propagate routing updates, the design must shift away from the familiar concept of pushing packets to given IP addresses (i.e., any node can send any packet to any other node). Instead, one must think in terms of data names and data retrieval.

More specifically, we need a systematic naming scheme for routers and routing updates (Section III-A). We also need to retrieve routing updates promptly without a priori knowledge of when an update may be generated, since a topology or name prefix change can happen any time (Section III-C). In terms of routing functionality, NLSR distinguishes itself from previous link-state routing protocols in two aspects: (a) providing multiple next hops for each name prefix instead of a single one; and (b) signing and verifying all LSAs to ensure that each router can originate only its own prefix and connectivity information. We present our trust model in Section III-D and route calculation algorithm in Section III-E.

A. HIERARCHICAL NAMING SCHEME

Perhaps the most important piece in our design is a proper naming scheme for each element in the routing system and its corresponding public key, which supports a variety of functionality such as automatic routing data validation, system configuration, and problem diagnosis. Based on the current network structures and operational practices, a hierarchical naming scheme best captures the relationship among various components in the system, thus making it easy to identify routers belonging to the same network, messages generated by a given routing process, and keys for each entity.

In our design, each router is named according to the network it resides in, the specific site it belongs to, as well as an assigned router identifier, i.e., /⟨network⟩/⟨site⟩/⟨router⟩. The ⟨router⟩ component contains two parts: a router tag and a router label, e.g., `%C1.Router/router3` (%C1 is a command marker indicating this is a special name component). For example, an ATT router in a PoP (point of presence) in Atlanta may be named `/ATT/Atlanta/%C1.Router/router3`. This way, we know that if two routers share the same /⟨network⟩ prefix, they belong to the same network; and if they share the same /⟨network⟩/⟨site⟩ prefix, they belong to the same site. This naming scheme makes it easy to filter out erroneous routing messages. The NLSR process on a router has the router's name as its prefix, i.e., /⟨network⟩/⟨site⟩/⟨router⟩/NLSR. This name is used in periodic *hello* messages between adjacent NLSR routers to detect the failure of either links or routing processes themselves (Section III-F).

OSPF and other traditional routing protocols also need to identify routers and neighbors, but they do not have such a systematic naming scheme to identify each entity in the routing system and their relationships. Moreover, their router identifier, e.g., the Router ID in OSPF, is usually selected from the addresses owned by each router, which lacks contextual and semantic information, changes whenever the router's address changes, and makes the routing protocol's operations dependent on a particular address type.

B. NAMING AND FORMAT OF LSAs

As in other link-state routing protocols, every router in NLSR collects connectivity and reachability information through Link State Advertisements (LSA). More specifically, a router advertises its links to neighboring NLSR routers in an *Adjacency LSA* and its name prefixes in a *Name LSA*. Note that the reachability here refers to name prefixes (not address prefixes) that a router or its directly connected nodes can reach. In other words, they either produce or host content with names that fall under the advertised name prefixes.

Each LSA has the name /localhop/⟨network⟩/NLSR/LSA/⟨site⟩/⟨router⟩/⟨lsa-type⟩/⟨version⟩. The first component contains a name used for scope control – the *localhop* scope limits an LSA *Interest*/*Data* packet to be forwarded to the immediate nexthops only (no further propagation). Since *every* node sends an *Interest* to retrieve each LSA, there is no need to propagate any LSA *Interest* beyond a node's immediate neighbors. The ⟨lsa-type⟩ component can be `name` or `adjacency`. The ⟨router⟩ component identifies the router that originates the LSA. The ⟨version⟩ component of an LSA is increased by 1 whenever a router creates a new version of the LSA.

The LSA format is shown in Figure 1. A *Name LSA* contains all the name prefixes registered locally with NLSR and those injected by connected end hosts. An *Adjacency LSA* contains all the *active* links of a router, each associated with a neighboring router's name and a link cost. It is created at router startup time and whenever there is any status change in a router's links, as detected by periodic "*hello*" messages (Section III-F).

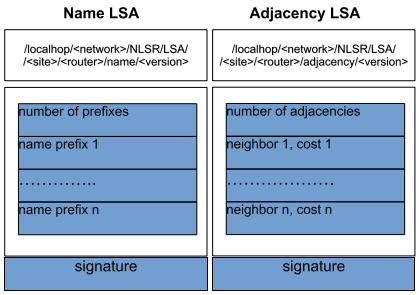


FIGURE 1. LSA format.

C. DISSEMINATION OF LSAs

Whenever a router establishes or removes an adjacency with a neighboring router, it disseminates a new version of its *Adjacency LSA* to the entire network. Moreover, it advertises name prefixes from both static configuration and dynamic registrations. Whenever any name prefix is added or deleted, the router also disseminates a new *Name LSA*. The latest versions of the LSAs are stored in a Link State Database (LSDB) at each node. Upon receiving any new LSA, each router recalculates its routes and updates the FIB (Section III-E).

We consider the LSA dissemination problem as data synchronization of the LSDBs. NLSR uses the *ChronoSync* protocol [11] to synchronize changes in the routers' LSDBs. ChronoSync maintains all the latest LSA names in each LSDB as a name set and uses a hash of the name set as a compact expression of the set. Routers running ChronoSync use the hashes of their LSA name sets to detect the difference in the sets. If a new LSA name is detected, ChronoSync notifies NLSR to retrieve the corresponding LSA. Compared to flooding a new LSA to the entire network in other link-state protocols, this approach enables the separation between the detection of new data names and NLSR's data retrieval, meaning that a router can request LSAs when it has CPU cycles. Thus, it is less likely a router will be overwhelmed by a flurry of updates.

IP-based link-state routing protocols such as OSPF and IS-IS [16] also have mechanisms to compare the LSDBs of two neighboring routers. Typically, the routers send summaries containing identifiers of the LSAs in their LSDBs to each other. If a router detects a difference, it will either request the different LSA from the neighbor or send its own LSA. ChronoSync is much more efficient in this regard – only one hash, instead of all the LSA names, is exchanged among the nodes. More importantly, because ChronoSync, or Sync in general, already provides the functionality to compare two data sets, NLSR does not have to reinvent the wheel, which makes the routing protocol design much simpler.

Figure 2 shows how an LSA is disseminated in the network. To synchronize the digest tree representing the LSAs in the

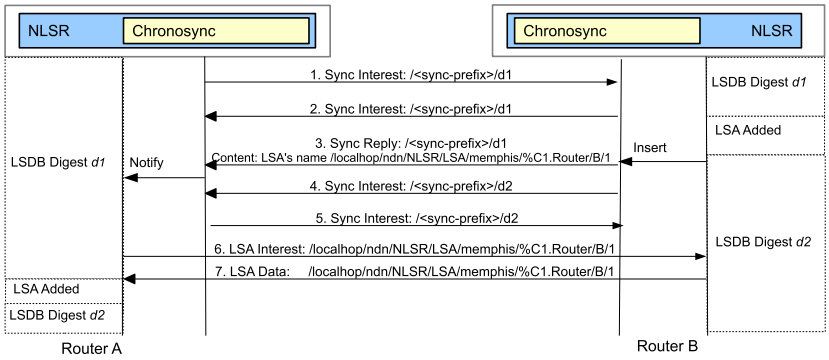


FIGURE 2. LSA dissemination via ChronoSync.

LSDB, the ChronoSync protocol on each node periodically sends *Sync Interests* with the hash of all the LSA names in its LSDB to all the other nodes (step 1 and 2). Note that NDN aggregates Interests with the same name into one PIT entry and forwards only one of them, so there is at most one Sync Interest pending on each link in each direction when all the nodes are synchronized. When an LSA is added to *B*'s LSDB, the LSA name is updated in *B*'s LSA name set. ChronoSync responds to the Sync Interest from *A* with a *Sync Data* packet containing the new LSA name (step 3). *A*'s ChronoSync receives the *Sync Data*, notifies NLSR of the new LSA name, and updates its LSA name set. Both *B* and *A* compute a new hash for the set and send a new *Sync Interest* with the new hash (step 4 and 5). Since the NLSR process on *A* has been notified of the new LSA name, NLSR sends an *LSA Interest* to retrieve this LSA (step 6). *B* responds to this *Interest* with the requested LSA data (step 7). When *A*'s NLSR receives the *LSA data*, it inserts the LSA into its LSDB. Now both routers' LSDBs are synchronized.[2]

Because link-state routing relies on LSAs to bootstrap routing calculation, it has no way to calculate routes for the LSA interests. Therefore, the name prefix of LSAs (/localhop/⟨network⟩/NLSR/LSA) is configured with a multicast forwarding strategy that allows the *Interest* for an LSA to be forwarded to all the neighbors of a node. If any of the neighbors have a copy of the LSA in its Content Store or NLSR process, the neighbor will return it. Otherwise, the *Interest* is discarded due to the *localhop* scope restriction.

In order to remove obsolete LSAs caused by router crashes, every router periodically refreshes each of its own LSAs by generating a newer version. When the new version of an LSA is received, any earlier version is removed from the LSDB. Moreover, every LSA has a lifetime associated with it and will be removed from the LSDB when the lifetime expires, which means that if a router crashes, its LSAs will not persist in other routers' LSDBs. Note that after the router crashes, its neighbors will update the status of their LSAs so traffic will not be directed over those links connected to the crashed router. Therefore, correct routing does not rely on the periodic refreshes and LSA lifetime to remove the obsolete LSAs. However, having an LSDB mostly free of obsolete LSAs can still help management and problem diagnosis. As such, the LSA lifetime and refresh period should be set to a relatively long interval, e.g., on the order of hours or even days, to reduce the associated message and processing overhead.

[2]Please see [11] for details about ChronoSync (e.g., hash calculation and difference resolution).

D. SECURITY

Every NDN *Data* packet is digitally signed and the signature is part of the *Data* packet. The signature covers the name, the content, and the metadata for signature verification. One piece of the metadata is the key locator [17], which indicates the name of the public key used to sign the packet. The data consumer can fetch the key to verify the signature [3]. It also needs to verify that the key is trusted to sign the LSA, which requires a trust model for key authentication. NDN uses public keys, certificates, and local trust anchors extensively for data and key authentication, following a distributed security model proposed in SDSI [18].

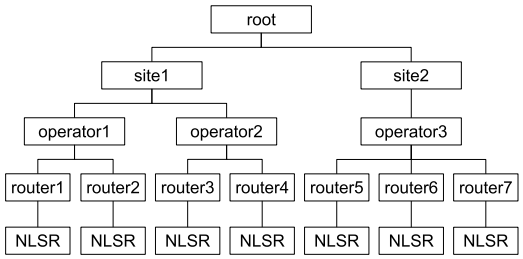


FIGURE 3. NLSR trust hierarchy.

1) TRUST MODEL

NLSR uses a five-level hierarchical trust model reflecting the administrative structure of an intra-domain routing protocol, as shown in Figure 3. At the top level, there is a root authority local to the network, called the *trust anchor*, which is responsible for issuing certificates to the *sites*, e.g., departments in an organization or PoPs in an ISP. Each site

```
security
{
  validator
  {
    rule
    {
      id "NLSR Hello Rule"
      for data
      filter
      {
        type name
        regex ^[^<NLSR><INFO>]*<NLSR><INFO><><><>$
      }
      checker
      {
        type customized
        sig-type rsa-sha256
        key-locator
        {
          type name
          hyper-relation
          {
            k-regex ^([^<KEY><NLSR>]*)<NLSR><KEY><>$
            k-expand \\1
            h-relation equal
            p-regex ^([^<NLSR><INFO>]*)<NLSR><INFO><><><>$
            p-expand \\1
          }
        }
      }
    }

    rule
    {
      id "NLSR LSA Rule"
      for data
      filter
      {
        type name
        regex ^[^<NLSR><LSA>]*<NLSR><LSA>
      }
      checker
      {
        type customized
        sig-type rsa-sha256
        key-locator
        {
          type name
          hyper-relation
          {
            k-regex ^([^<KEY><NLSR>]*)<NLSR><KEY><>$
            k-expand \\1
            h-relation equal
            p-regex ^<localhop>([^<NLSR><LSA>]*)<NLSR><LSA><>(<>*)<><><><><>$
            p-expand \\1\\2
          }
        }
      }
    }

    rule
    {
      id "NLSR Hierarchy Exception Rule"
      for data
      filter
      {
        type name
        regex ^[^<KEY><%C1.Router>]*<%C1.Router>[^<KEY><NLSR>]*<KEY><><><><>$
      }
      checker
      {
        type customized
        sig-type rsa-sha256
        key-locator
        {
          type name
          hyper-relation
          {
            k-regex ^([^<KEY><%C1.Operator>]*)<%C1.Operator>[^<KEY>]*<KEY><><>$
            k-expand \\1
            h-relation equal
            p-regex ^([^<KEY><%C1.Router>]*)<%C1.Router>[^<KEY>]*<KEY><><><><>$
            p-expand \\1
          }
        }
      }
    }

    rule
    {
      id "NLSR Hierarchical Rule"
      for data
      filter
      {
        type name
        regex ^[^<KEY>]*<KEY><><><><>$
      }
      checker
      {
        type hierarchical
        sig-type rsa-sha256
      }
    }

    trust-anchor
    {
      type file
      file-name "root.cert"
    }
}
```

Listing 1. Schema for hello and LSA validation.

TABLE 1. Key and data names.

Key/Data	Name
Network Key	/<network>/KEY/<key>
Site Key	/<network>/<site>/KEY/<key>
Operator Key	/<network>/<site>/<operator>/KEY/<key>
Router Key	/<network>/<site>/<router>/KEY/<key>
NLSR Key	/<network>/<site>/<router>/NLSR/KEY/<key>
LSA Data	/localhop/<network>/NLSR/LSA/<site>/<router>/<lsa-type>

has one or more *operators* who collectively manage a number of *routers* belonging to a site. Each router can create an *NLSR routing process* that produces LSAs. This hierarchical trust model enables one to establish a chain of keys to authenticate LSAs – an LSA must be signed by a valid NLSR process running on the same router where the LSA originates. To become a valid NLSR process, the process key must be signed by the corresponding router key, which in turn should be signed by one of the operators of the same site. Each site operator's key must be signed by the site key, which must be certified by the trust anchor using its self-signed root key.

The name of a key explicitly expresses the role that the key plays in the system, as shown in Table 1.[3] The trust relationship between these keys can be expressed using a trust schema [7] as shown in Listing 1, which limits the privilege of each key to a small scope. For example, an operator can only certify routers belonging to his own site – any other keys certified by the operator will be treated as invalid. The restricted privilege limits the impact of key compromise. Moreover, the *multi-level* key hierarchy reduces the use of each key, further mitigating the risk of key exposure.

2) KEY RETRIEVAL

In NDN, a public key is simply another type of data and can be retrieved using Interest/Data exchange similar to how LSAs are retrieved (Section III-C). In other words, a router can express an *Interest* with a key name to retrieve the key.[4] The network will forward the *Interest* towards the data containing the corresponding public key. However, because one must be able to retrieve keys to verify routing updates before routes are established, NLSR requires a key retrieval mechanism that does not rely on FIB entries, in the same way as LSA retrieval is independent of routing. We use the DirectFetch mechanism for this purpose – when a router receives an LSA, it sends an *Interest* for the signing key back to the face from which the LSA is received. Since the neighbor that

[3]Both routers and operators have names under /<network>/<site>/. To differentiate between router keys and operator keys, we use different tags in their names – the <router> component contains a router tag, `%C1.Router`, and a router label, while the <operator> component contains an operator tag, `%C1.Operator`, and an operator label.

[4]Note that the last component of a key name is a KeyID that distinguishes different keys with the same prefix, so that the name always matches a specific key. By default, the KeyID is a hash of the key.

sent the LSA has verified the data, it must have retrieved the key which can satisfy the router's *Interest*. Once the key is received, the router can store it for future use to avoid fetching it again.

The above process is repeated multiple times to retrieve all the keys up to the trust root following the trust model described in Figure 3 in order to verify the LSA signing key (i.e., NLSR process key). To speed up the key fetching and verification process, the router that originated the LSA can package all the keys in a ''Key Bundle'' and other routers can fetch the Key Bundle instead of individual keys.

E. MULTIPATH CALCULATION

Based on the information available in the Adjacency LSAs, each NLSR node builds a network topology. It then runs the following simple extension of Dijkstra's algorithm to produce multiple next hops to reach each node. It first removes all immediately adjacent links except one and uses Dijktra's algorithm to calculate the cost of using that link to reach every node in this topology. This process is repeated for every adjacent link. Afterwards, it ranks the next hops for each node based on their cost to reach the node. Since we know which name prefixes are associated with which nodes based on the Name LSAs, we can obtain a list of next hops to reach each name prefix.

Note that NLSR allows an operator to specify the maximum number of next hops per name prefix to insert into the FIB, so that the FIB size can be limited when a node has many neighbors. However, the computational cost is still controlled by the total number of neighbors, since the algorithm goes through all available next hops to produce the cost for each next hop. We plan to investigate more efficient multipath computation algorithms.

F. ADJACENCY ESTABLISHMENT

Before two NLSR routers can exchange their LSAs, they first need to establish an adjacency between them. NLSR sends periodic *hello Interests* with the name /⟨neighbor-router⟩/NLSR/INFO/⟨this-router⟩, at a default interval of 60 seconds, to each neighboring node to detect its status. If the neighbor responds to the *Interest* with *Data* signed using the neighbor's NLSR process key and the *Data* can be validated based on the trust model, the neighbor is considered up, or *ACTIVE*.[5]

If a *hello Interest* times out, NLSR will try sending the *Interest* a few more times at short intervals in case the *Interest* was lost. If there is no response from the neighbor during this period, the adjacency with the neighbor is considered down, or *INACTIVE*. Note that it is impossible to infer whether the remote NLSR process has died or the link has failed. However, in either case the link should not be used to forward traffic. Whenever the status of an adjacency changes, NLSR rebuilds its Adjacency LSA and distributes it. It also schedules a routing table calculation.

Note that NLSR continues to send periodic *hello Interests* to the INACTIVE neighbor in case the problem is caused by a link failure. When the link recovers, the two NLSR processes will receive the next periodic *hello Interests* and the adjacency will be set up accordingly.

If the neighboring NLSR process crashes and then recovers, the neighbor will send a *hello Interest* and receive the corresponding *hello Data*. NLSR will also send a *hello Interest* immediately after receiving the message from the *INACTIVE* neighbor, rather than waiting to send the next scheduled *hello Interest*, in order to speed up the adjacency establishment. Figure 4 illustrates how Node A detects an adjacency failure with Node C and a recovery with Node B.

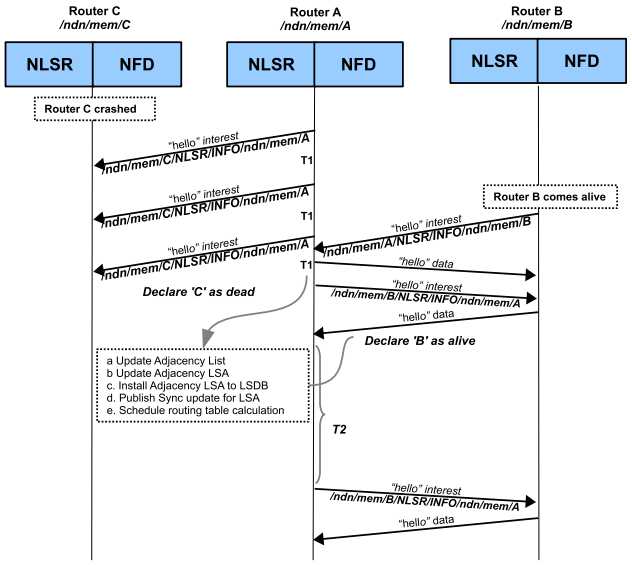


FIGURE 4. Adjacency failure and recovery detection. T1 is the timeout for receiving a *hello Data* packet after which the *hello Interest* is retransmitted. Its default value is 1 second. The *hello Interest* is retransmitted up to two times by default after the first timeout. T2 is the interval between periodic *hello Interests*. Its default value is 60 seconds. For brevity, we did not include the router tag in the *Interest* and *Data* names.

NLSR also uses *face event notifications* from NFD (NDN's Forwarding Daemon) to quickly respond to a link failure. When an interface to an adjacency is destroyed, NFD will send to NLSR a *Face Event Notification* with a Face ID corresponding to the interface. NLSR will use the Face ID to find the adjacency which is reached through this interface, mark the adjacency as *INACTIVE*, rebuild its Adjacency LSA, and schedule a routing table calculation. When a face is created, NLSR will also receive a notification so that it can send a *hello Interest* to set up the adjacency.

G. SUMMARY OF DESIGN CHANGES

NLSR's initial design was published in 2013 [5]. In the past four years, we have made major changes to its design and

[5]The *hello Data* can carry information about the neighbor that allows the node to adjust its operations accordingly. Although we do not currently include such information, we plan to add this feature in the near future.

implementation, but we still find papers that refer to and compare with the original design. A main reason for publishing this paper is to highlight the changes and draw attention to our current design.

One of the most important changes is replacing CCNSync with ChronoSync. CCNSync was bundled with CCNx's Repo, and all the data CCNSync retrieved was stored in the Repo and could not be deleted. This caused a memory problem after running NLSR for some time. In contrast, ChronoSync simply informs NLSR of new data names and NLSR retrieves the data using the names. Since NLSR only cares about the latest version of an LSA, it can discard earlier versions of the LSA which eliminates the memory problems associated with storing all LSAs.

Another major design change is to advertise *all* the name prefixes originated by a router in *one* LSA. This means fewer messages are required to collect the name prefix information. If a router originates many name prefixes, the LSA may exceed the default packet size in NDN. We have implemented LSA segmentation to support large LSAs.

Furthermore, we also augmented the adjacency establishment protocol to use face event notifications so that it can react to link failures and recoveries much faster than relying on *hello* interests alone, which by default are sent every 60 seconds.

Finally, we discovered that the unconstrained multicast propagation of Sync interests and LSA interests caused excessive number of duplicate NACKs as they loop back to the original routers that sent the interests. The NACKs also erased the necessary PIT entries for the Sync interests causing the Sync data packets carrying new LSA names to be delayed. To address these problems, we added the /localhop scope to the Sync and LSA interest names to limit their propagation to the immediate neighbors only, thus eliminating the duplicate NACKs and associated problems.

IV. IMPLEMENTATION

The current NLSR design is implemented in C++ using the ndn-cxx [19] library to run over NFD [20] (the initial NLSR design was implemented in C using CCNx [5]). It is open source [21]. Below we describe some implementation details.

A. ADJACENCY ESTABLISHMENT PROTOCOL

There are three parameters in NLSR's configuration file that can be used to modify the behavior of the Adjacency Establishment Protocol. The *Hello Interest Interval* can be changed to reduce or increase the frequency of periodic *hello* interests (default is 60 seconds). The *Hello Interest Timeout* is used to specify how long to wait for the Hello data packet before retransmitting the interest. The default is 1 second as the round-trip time between two neighboring routers in a network is usually much less than 1 second. The *Hello Interest Retry Amount* specifies the number of times the *hello* interest can be resent before determining the adjacency to be down (the default is 3 times including the first hello Interest).

B. LSA VERSION NUMBERS

The version number for each LSA increases by one after each change. On start up, NLSR must use version numbers that are larger than previously used for each LSA type. Otherwise, other routers in the network will consider the LSAs as obsolete. To solve this problem, NLSR records the current version number for each LSA type and writes them to a file whenever a version number changes. When NLSR is initialized, it reads these version numbers from the file and publishes its first LSAs with version numbers larger than the recorded version numbers.

C. ROUTING OPERATION DELAYS

Two parameters in the NLSR configuration file can be modified to balance performance with overhead. Each of the parameters is used to control the timing of important routing operations. The *Adjacency LSA Build Interval* configures the delay after an Adjacency LSA build has been requested until the LSA is actually built. A longer delay allows for multiple adjacency changes to be aggregated into one Adjacency LSA build, reducing the CPU overhead. On the other hand, the shorter the delay, the faster the router can build an Adjacency LSA so that the network can use paths through its up-to-date adjacencies. The default value is 5 seconds. The *Routing Calculation Interval* is used to specify the delay after a routing table calculation is scheduled until the routing table is built. A longer wait time allows for multiple changes to the LSDB to be aggregated into one calculation, but it also means that the router cannot begin using updated paths until the calculation is performed. The default value is 15 seconds.

D. SECURITY

Each key utilized by NLSR's trust model, except the NLSR process key, is created using the ndn-cxx [19] *ndnsec* tools. A public/private key pair and corresponding certificate are created for each key owner in the trust model hierarchy. The certificate for each public key in the key pairs is signed by the key owner one level higher in the hierarchy.

The NLSR process key is created automatically when NLSR is initialized using the ndn-cxx security API directly. On initialization, NLSR generates a key pair and gets the certificate for the public key signed by the router's private key. The NLSR process uses its private key to sign *hello* data and LSA data. Whenever NLSR is restarted, the new NLSR process will generate a new key pair and create a new certificate.

E. DYNAMIC NAME PREFIX ADVERTISEMENT AND WITHDRAWAL

A network operator can specify a set of name prefixes to be advertised by NLSR in the NLSR configuration file. NLSR builds a Name LSA which includes the set of names and advertises it to the network. To modify the advertised name prefixes while the NLSR process is running, a command Interest can be sent to NLSR to advertise or withdraw a specific name prefix. The command Interest's name contains

the desired action, advertise or withdraw, as well as the name prefix to be advertised or withdrawn. NLSR will construct a new Name LSA accordingly and disseminate it to the network.

V. EVALUATION

This section presents the evaluation results of NLSR in terms of CPU processing time, routing convergence time, and forwarding plane performance. All of our experiments were performed using Mini-NDN [22], an NDN network emulation tool based on Mininet [23]. In Mini-NDN, an entire network topology can be run on a single machine, and each node in the topology is executed in a container with its own resources. The experiments were run on a server with a 2.7Ghz Intel Xeon E5-2680 CPU.

A. SCENARIOS

Each experiment lasts 600 seconds. First, NLSR is started on each router in the network and allowed to converge and stabilize for 300 seconds. The next three events are designed to test and evaluate the performance of NLSR in different situations. At the 300-second mark, routers begin to refresh their LSAs. Note that in order to test the protocol in a short period of time, we set the refresh timer to be 300 seconds instead of on the order of days. At 480 seconds, the most connected node in the topology is brought down and remains failed for 60 seconds. At 540 seconds, the previously failed node is brought back up.

To evaluate forwarding performance under single-path and multipath routing, we run the same scenario, except we run a ping server on each host in the network and after two minutes, perform one ping per second between each host in the network for the remainder of the experiment. We generate ping traffic using the *ndnping* utility [24]. In addition, to evaluate the effects of the routing operation delays on routing and forwarding convergence, we run the experiments with the default delay values and then no delays.

B. TOPOLOGIES

We run our experiments on four different topologies to measure the performance of NLSR as topology size increases. Our first experiment topology is a previous snapshot of the NDN testbed topology with 22 nodes and 50 links (Figure 5). The routing cost of each link is set to the delay between the two neighboring nodes. Our three larger topologies are realistic Internet-like topologies with an increasing number of nodes (N = 41, 58, 78), the upper limit constrained by our computational resources. Since the AS Internet topology is self-similar [25], meaning that its subgraphs retain all the structural properties of the original full topology, we extract subgraphs of the AS Internet topology of differing size N.

C. RESULTS

The first experiment is performed on the NDN testbed topology to determine the CPU impact of key authentication and multpath calculation before scaling to larger topologies.

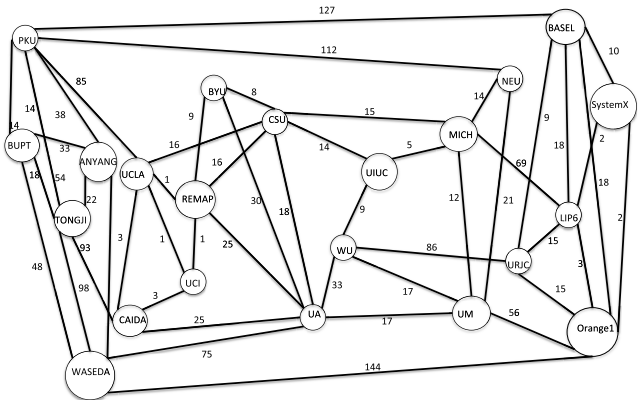


FIGURE 5. NDN testbed topology.

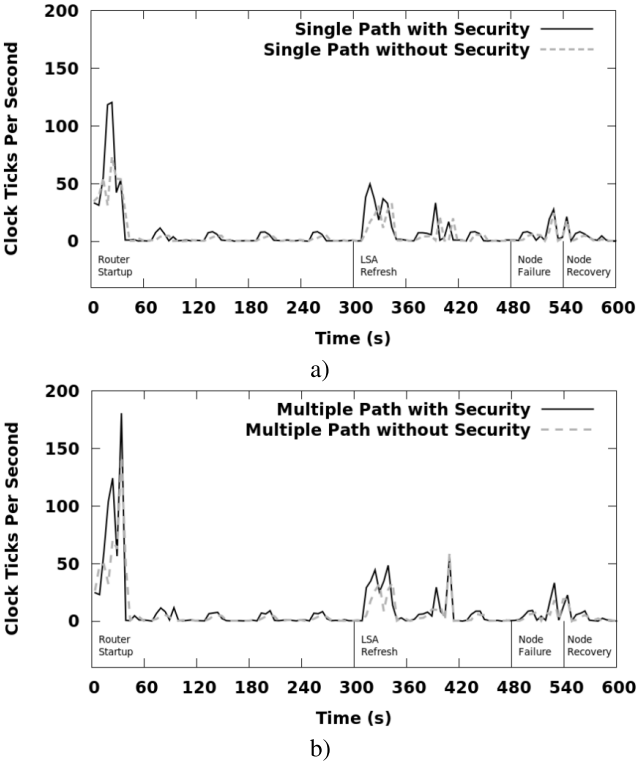


FIGURE 6. Total network CPU utilization for NLSR. a) Single-path Routing. b) Multipath Routing.

The maximum number of next hops per name prefix is set to 4 for all the multipath experiments. Figure 6 shows the CPU overhead of NLSR for all the nodes over time with key authentication disabled and enabled; the first figure is with single-path calculation and the second figure is with multipath calculation.

It is evident from the figure that even with the proposed trust model, which requires verification of multiple levels of keys, NLSR hardly incurs much extra processing cost after router startup. During the startup period, key authentication adds 29% extra CPU overhead in the single path case and 24% in multipath. This is due to the fact that by design NDN signs and verifies all Data packets. The only difference between the two schemes lies in the key verification, where NLSR

with the proposed trust scheme requires more time to fetch multiple keys recursively from the network and verify them; however, as this is done only once per new key, it incurs a very low CPU cost after a key is verified and cached. Figure 6 also shows that with multipath routing, NLSR shows higher CPU usage than single path. Since the CPU cost due to *messaging* is the same in the two schemes, the difference here is mainly due to the higher cost of multipath calculation. Multipath calculation adds 23% and 14% CPU overhead total over the entire experiment without and with security enabled, respectively.

The next experiment shows how delay parameters (Section IV-C) in the routing protocol can be used to achieve a balance between routing convergence time and routing overhead. While similar trade-off also exists in IP routing protocols, one should keep in mind that because NDN can use adaptive multipath forwarding, it does not depend solely on routing to handle topological changes and as such routing convergence is not as important in NDN as in IP. Therefore, an NDN operator can use larger delay parameters to lower routing overhead.

To measure routing convergence time, we track the number of LSDB changes over time – when the number reaches 0, it means the LSDBs have been synchronized and the network has converged. The experiment is run on all four topologies with *the default values for routing operation delays* (Section IV-C) and with *no routing operation delays*. Figure 7 shows the average per-node LSDB changes each second. *Without routing operation delays*, NLSR converges more quickly than with the *default delays* but also has more cumulative LSDB changes. During the startup period, the *default delays* generate 19% less LSDB changes than *no delays* in the NDN testbed topology and 22% less in the 78-node topology. During the LSA refresh and node failure periods, *default delays* and *no delays* generate the same number of LSDB changes in both topologies. After the recovery, the *default delays* generate 1% less LSDB changes than *no delays* in the NDN testbed topology and 27% less in the 78-node topology.

Moreover, Figure 7 shows that the larger topology takes longer to converge and incurs more cumulative LSDB changes per node regardless of whether routing operation delays are used, which is expected. For example, with the default delay, the 78-node topology took 23% longer to converge than the NDN testbed topology during router startup, 19% longer during LSA refresh, 40% longer during failure, and 17% longer during recovery.

To understand the benefit of multipath forwarding, which is enabled by NLSR's multipath routing calculation, we measure the RTT of pings in the network during the failure and recovery events. To take advantage of the multiple next hops per name prefix, we use a forwarding strategy called *Adaptive SRTT-based Forwarding (ASF)* that maintains a smoothed RTT for each name prefix through each available next hop [26]. The strategy chooses the highest routing ranked next hop to forward Interests initially and probabilistically probes other next hops periodically to learn RTTs (the probability is proportional to the routing ranking). When a next hop with lower smoothed RTT is found, it switches to that next hop. This capability is important for handling failures and recoveries before routing converges. The experiment is

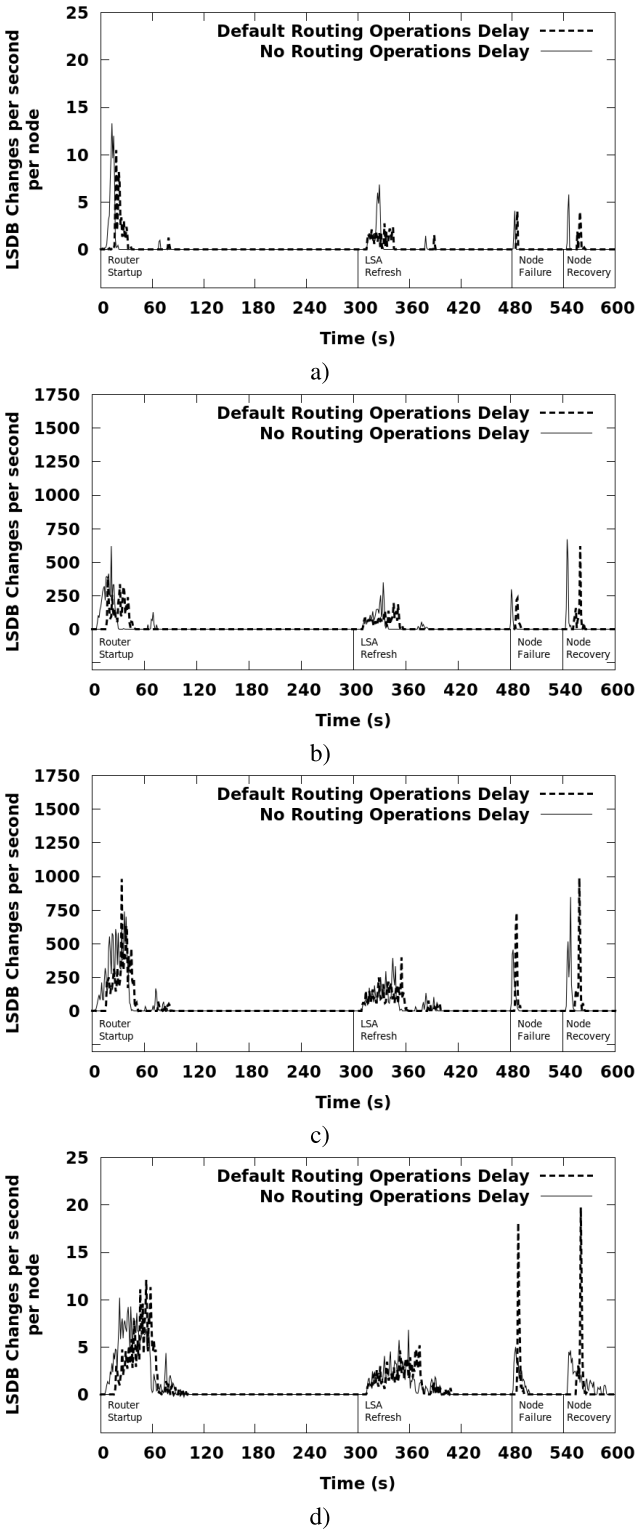


FIGURE 7. NLSR LSDB changes. a) NDN Testbed Topology. b) 41-node Topology. c) 58-node Topology. d) 78-node Topology.

run on the NDN testbed topology with the default values for *routing operations delay*.

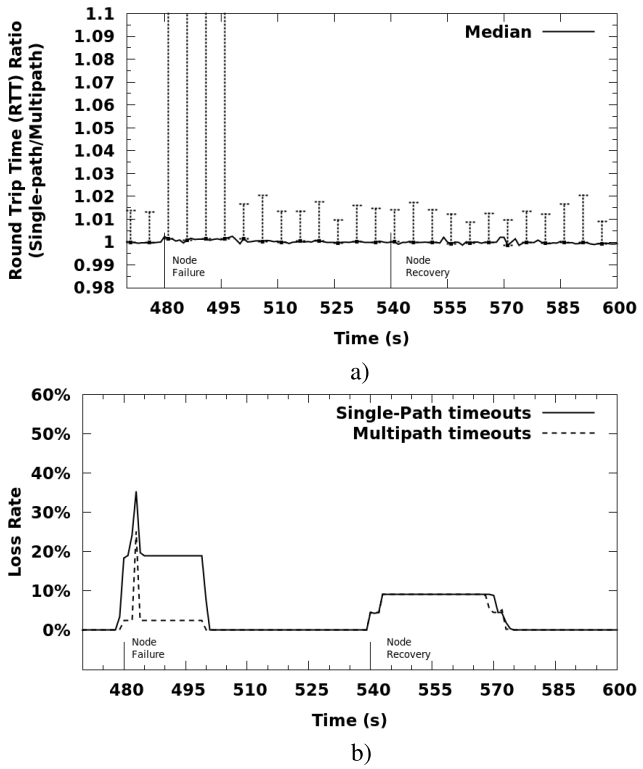


FIGURE 8. Forwarding convergence. a) RTT Ratio between Single Path and MultiPath. b) Ping Loss Rate.

Figure 8(a) shows the RTT ratio between single path routing and multipath routing for each pair of nodes every second. In the case of a timeout, the RTT used in the calculation for the timed-out packet is equal to 971ms, the weight of the longest path in the topology. The median of the ratios is graphed along with the 5th percentile and 95th percentile. During the failure event, the 95th percentile is much higher due to timeouts in the single path case while multipath is able to choose a different next hop for forwarding. Single path is not able to remedy these timeouts until NLSR recalculates the routing table and installs a new next hop. Note that sometimes the ratio is slightly below 1 due to the variations in RTTs caused by queueing and other factors. Also, the graph's maximum Y-value is 1.1, but the 95th percentile extends much higher, approaching a ratio of 5 for the 20 seconds after the node failure.

Figure 8(b) shows the loss rate incurred by both single path and multipath during the failure and recovery events. Single path experiences a loss rate higher than multipath for the reason explained above. We can make one more observation: the higher delay and losses in the single path case happen only in the first 20 seconds after the node failure. This shows that NLSR converges soon after the default operations delay, since once the routing converges, the best next hop in the multipath case is the same as that in the single path case.

VI. LESSONS FROM DEVELOPMENT AND DEPLOYMENT

NLSR has provided a real use case to drive development of several NDN features such as the security and trust schema functionality in the ndn-cxx library, the RIB and prefix management functionality in NFD, and the Sync mechanism in ChronoSync. At the same time, these features greatly simplified our protocol design and implementation. For example, using ChronoSync to disseminate new LSA names meant that we did not need to invent a mechanism to get notifications for new LSAs.

Furthermore, the testbed deployment helped discover potential problems. For example, the key validator verifies that received keys and certificates are not created in the future, but if router clocks are out-of-sync, such situations can arise. Therefore, we added measures to handle slightly out-of-sync clocks. However, if any of the testbed machines has a very different time than others, its LSAs may be rejected by others or vice versa. This means that the network has to be roughly time synchronized for the protocol to work. Another problem is that the sequence number file where NLSR records its LSA version numbers can become corrupted during operation or during reboot, which can cause a router to inject LSAs with older version numbers than the ones already distributed. In this case, the new LSAs will be discarded by other routers, so this router cannot become part of the topology. It is our ongoing work to address this problem.

VII. CONCLUSION

So far, designing NLSR has served as a great learning experience in the following aspects: (1) design of the naming scheme to reflect the relationship among various entities in a routing system, (2) development of a trust model for key verification of a routing protocol, and (3) mental adjustment to NDN's new design patterns of using Interest/Data exchanges to propagate routing update messages. Furthermore, the use of named data for communication enables the concept of Sync, which facilitates robust dataset synchronization in distributed systems, making NLSR more resilient to losses and conceptually simpler.

In the near future, we plan to use ChronoSync to distribute keys similarly to how LSAs are distributed. If keys are proactively distributed in a Sync approach, nodes can immediately learn new keys after a key rollover which prevents certain attacks, such as key replay attacks. For a global routing solution in the long term, we are exploring new types of routing designs, such as hyperbolic routing [26], to scale routing in NDN. Since NDN's adaptive multipath forwarding can handle various packet delivery problems at the forwarding plane, the convergence delay requirements on the routing plane are relaxed. This opens the door to new types of routing designs that have fewer routing updates by trading off convergence speed.

ACKNOWLEDGMENT

The authors would like to thank Muktadir Chowdhury, Ashlesh Gawande and Nicholas Gordon for their work on NLSR and help in revising this paper.

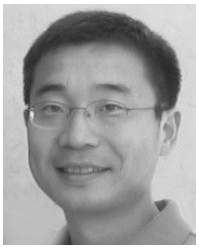
REFERENCES

[1] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," in *Proc. ACM CoNEXT*, 2009, pp. 1–12.
[2] L. Zhang *et al.*, "Named data networking (NDN) project," Named Data Networking Project, Tech. Rep. NDN-0001, Oct. 2010.
[3] L. Zhang *et al.*, "Named data networking," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 66–73, Jul. 2014.
[4] C. Yi, A. Afanasyev, I. Moiseenko, L. Wang, B. Zhang, and L. Zhang, "A case for stateful forwarding plane," *Comput. Commun.*, vol. 36, no. 7, pp. 779–791, 2013. [Online]. Available: http://dx.doi.org/10.1016/j.comcom.2013.01.005
[5] A. M. Hoque, S. O. Amin, A. Alyyan, B. Zhang, L. Zhang, and L. Wang, "NLSR: Named-data link state routing protocol," in *Proc. ACM SIGCOMM Workshop Inf.-Centric Netw.*, 2013, pp. 15–20.
[6] J. Moy, *OSPF Version 2*, document RFC 2328, SRI Network Information Center, Sep. 1998.
[7] Y. Yu, A. Afanasyev, D. Clark, K. Claffy, V. Jacobson, and L. Zhang, "Schematizing and automating trust in named data networking," in *Proc. 2nd ACM ICN Conf.*, 2015, pp. 1–10.
[8] C. Yi, J. Abraham, A. Afanasyev, L. Wang, B. Zhang, and L. Zhang, "On the role of routing in named data networking," in *Proc. ACM SIGCOMM ICN Conf.*, 2014, pp. 27–36.
[9] PARC. *CCNx Open Srouce Platform*. [Online]. Available: http://www.ccnx.org
[10] NDN Project Team. *The NDN Platform*. [Online]. Available: http://named-data.net/codebase/platform/
[11] Z. Zhu and A. Afanasyev, "Let's ChronoSync: Decentralized dataset state synchronization in Named Data Networking," in *Proc. IEEE ICNP*, Oct. 2013, pp. 1–10.
[12] N. P. Team. *NLSR 0.1.0*. [Online]. Available: http://named-data.net/doc/NLSR/0.1.0
[13] H. Dai, J. Lu, Y. Wang, and B. Liu, "A two-layer intra-domain routing scheme for named data networking," in *Proc. Next Generat. Netw. Internet Symp. GLOBECOM*, Dec. 2012, pp. 2815–2820.
[14] E. Hemmati and J. Garcia-Luna-Aceves, "A new approach to name-based link-state routing for information-centric networks," in *Proc. 2nd ACM ICN Conf. (ICN)*, 2015, pp. 29–38.
[15] J. J. Garcia-Luna-Aceves, "Routing to multi-instantiated destinations: Principles and applications," in *Proc. IEEE 22nd Int. Conf. Netw. Protocols (ICNP)*, Oct. 2014, pp. 155–166.
[16] *Intermediate System to Intermediate System Intra-Domain Routeing Information Exchange Protocol for Use in Conjunction With the Protocol for Providing the Connectionless-Mode Network Service (ISO 8473)*, Int. Standard 10589:2002, ISO, 2nd ed., 2002.
[17] *NDN Packet Format Specification*. [Online]. Available: http://named-data.net/doc/ndn-tlv/
[18] R. L. Rivest and B. Lampson, "SDSI—A simple distributed security infrastructure," MIT, Cambridge, MA, USA, Tech. Rep., 1996.
[19] N. P. Team. *NDN-CXX*. [Online]. Available: http://named-data.net/doc/ndn-cxx/
[20] N. P. Team. *NFD—NDN Forwarding Daemon*. [Online]. Available: http://named-data.net/doc/nfd/
[21] N. P. Team. *Named Data Link State Routing*. [Online]. Available: https://github.com/named-data/NLSR
[22] N. P. Team. *NFD—NDN Forwarding Daemon*. [Online]. Available: http://named-data.net/doc/nfd/
[23] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: Rapid prototyping for software-defined networks," in *Proc. 9th ACM SIGCOMM Workshop Hot Topics Netw.*, 2010, Art. no. 19.
[24] *NDN-Tools GitHub*. [Online]. Available: https://github.com/named-data/ndn-tools
[25] M. Á. Serrano, D. Krioukov, and M. Boguñá, "Self-similarity of complex networks and hidden metric spaces," *Phys. Rev. Lett.*, vol. 100, no. 7, p. 78701, 2008.
[26] V. Lehman *et al.*, "An experimental investigation of hyperbolic routing with a smart forwarding plane in NDN," in *Proc. IEEE IWQoS*, Jun. 2016, pp. 1–10.

LAN WANG (SM'10) received the B.S. degree in computer science from Peking University, China, in 1997, and the Ph.D. degree in computer science from UCLA in 2004. She is currently a Professor and the Chair with the Department of Computer Science, The University of Memphis. Her research interests include future Internet architecture, Internet routing, network security, network performance measurement, and sensor networks.

VINCE LEHMAN received the B.S. degree in computer science from The University of Memphis. He is currently a Software Engineer with ReTrans.

A. K. M. MAHMUDUL HOQUE received the M.S. degree in computer science from The University of Memphis. He is currently a Software Engineer with Amazon.



BEICHUAN ZHANG received the B.S. degree from Peking University and the Ph.D. degree from UCLA. He is currently an Associate Professor with the Department of Computer Science, The University of Arizona. He has been involved in named data networking, green networking, and inter-domain routing. His research interest is in Internet routing architectures and protocols. He received the Applied Networking Research Prize in 2011 from ISOC and IRTF, and the Best Paper Award from the IEEE ICDCS in 2005 and IWQoS in 2014.

YINGDI YU received the B.S. and M.S. degrees in electrical engineering from Shanghai Jiao Tong University in 2007 and 2010, respectively, and the Ph.D. degree in computer science from the University of California at Los Angeles (UCLA), Los Angeles, in 2016. He is currently a Research Scientist with Facebook. His research interests were focused on security of the named data networking at UCLA.

LIXIA ZHANG (F'06–LF'17) received the Ph.D. degree in computer science from MIT. She was a Member of the Research Staff with Xerox PARC. She is currently a Professor with the Computer Science Department, University of California at Los Angeles (UCLA), where she also holds the UCLA Postel Chair in computer science. Since 2010, she has been leading the effort on the design and development of named data networking, a new Internet protocol architecture. She is a fellow of the ACM. She was a recipient of the IEEE Internet Award.