NDN, Technical Report NDN-0037, 2016. http://named-data.net/techreports.html
Revision 1: January 26, 2016

Obsoletes: A. Hoque, S. O. Amin, A. Alyyan, B. Zhang, L. Zhang, and L. Wang, NLSR: Named Data Link State routing protocol, in Proceedings of the 3rd ACM SIGCOMM Workshop on Information-Centric Networking, 2013.

A Secure Link State Routing Protocol for NDN

Vince Lehman
vslehman@memphis.edu
University of Memphis

A K M Mahmudul Hoque
akmhoque@gmail.com
Amazon

Yingdi Yu
yingdi@cs.ucla.edu
University of California, Los Angeles

Lan Wang
lanwang@memphis.edu
University of Memphis

Beichuan Zhang
bzhang@cs.arizona.edu
University of Arizona

Lixia Zhang
lixia@cs.ucla.edu
University of California, Los Angeles

***Abstract*—The Named-data Link State Routing protocol (NLSR) is an intra-domain routing protocol for 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 initial NLSR design, which was developed in 2013, has undergone significant changes. The new NLSR has been deployed on the NDN testbed since August 2014; its development helped drive the development of the trust/security functionality of NDN libraries as well as a number of features in NFD 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.**

I. Introduction

Named Data Networking (NDN) ([1]–[3]) is an information-centric Internet architecture inspired by years of empirical research on 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". Instead of carrying source and destination IP addresses in each packet, NDN puts a data name in each packet. A data consumer sends out an *Interest* packet whose name identifies the desired data; routers forward Interest packets based on their names; and a *Data* packet with the matching name is returned to the requesting consumer. The *Data* packet contains the name, the data, and a signature by the original data producer. By explicitly naming and signing data and by maintaining a stateful forwarding plane [4], NDN provides several desired network functions that are difficult to realize in IP, such as in-network caching, multicast delivery, and multi-path forwarding.

An NDN routing protocol propagates routing updates and computes routes to *name prefixes* that are hierarchically structured. NDN's best route computation can use any of the *routing algorithms* that work for IP, e.g., link-state or distance-vector, although 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. However, NDN does require a fundamentally different *routing protocol* design. Besides the need to propagate the reachability of name prefixes instead of IP address prefixes, a native NDN routing protocol must 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 routing information authentication.

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

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.

b) Security: Since each 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 practice in a domain, to verify the keys used to sign the routing messages. The names in NLSR reflect the relationship among routing entities, enabling automatic derivation of signing key names and the use of NDN trust schema [6] to automatically verify received routing updates.

c) Multi-path forwarding: While IP uses either 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 [7] – 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 a FIB entry, a module called "forwarding strategy" decides which next hop(s) will be used in forwarding Interests 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 [7].

B. Evolution of NLSR

The NLSR design has evolved significantly over the last two years since its first design sketch in 2013 [5]. The earlier design used the Sync mechanism provided by CCNx [8] to distribute Link State Advertisements (LSAs) between routers. However, during extensive testing using an internal testbed of 12 nodes, 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 NLSR on the NDN testbed.

As a new NDN platform [9] 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 [10] 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-F for more information). The new NLSR implementation [11] was released and deployed on the inter-continental NDN testbed in August 2014.

C. Related Work

The routing protocol proposed by Dai et al. [12] 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/NDN network; instead, it uses IP packets to deliver routing updates and has no support for 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. Third, its multi-path forwarding function is limited to contents announced by multiple producers only, 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, most notably [13] and [14], were developed after our initial NLSR design [5] was published. The Link State Content Routing (LSCR) protocol [13] 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. The Distance-based Content Routing (DCR) protocol [14] 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 potential problem in the above two protocols is that *selective forwarding of routing advertisements may cause some data to be unreachable* due to the different forwarding semantics in NDN and IP. 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. 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 announced by different nodes to ensure data retrieval (see Section 4.1 of [1]). Furthermore, although [13] and [14] compared favorably with our original NLSR design [5], which advertises each name prefix in a separate LSA, it is questionable whether those results may still hold true when one compares those designs against the current NLSR, which advertises all the prefixes originated by a router in a single LSA.

III. Design

As a link-state protocol, NLSR's basic functionality is to discover adjacencies and disseminate both topology 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 concepts 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-B). 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-C and route calculation algorithm in Section III-D.

Although NLSR is designed to run in a single routing domain with a single authority, we believe that our design and deployment experience offers a concrete stepping stone toward developing an NDN-based inter-domain routing protocol and an inter-domain trust model.

A. Hierarchical Naming Scheme

Based on the current network structures and operational practices, a hierarchical naming scheme seems best to capture the relationship among various components in the system, thus making it easy to identify routers belonging to the same network as well as messages generated by a given routing process. It also helps associate keys with their owners.

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`. For example, an ATT router in a PoP (point of presence) in Atlanta may be named `/ATT/Atlanta/PoP1/%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 *info* messages between adjacent NLSR routers to detect the failure of either links or routing processes themselves (Section III-E).

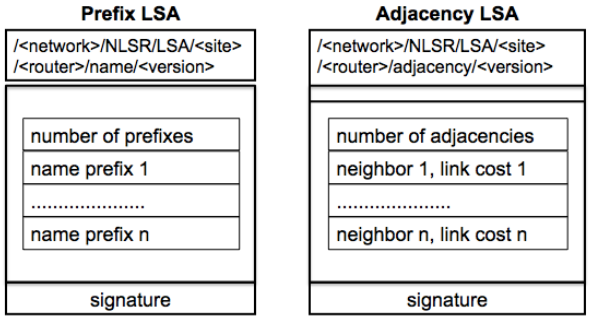


Fig. 1. LSA Format

B. Format and Dissemination of LSAs

An NLSR router establishes and maintains adjacency relations with neighbor routers. Whenever it detects the failure or recovery of any of its links or neighbor processes, it disseminates a new *Adjacency LSA* to the entire network. Moreover, it advertises name prefixes from both static configuration and dynamic registration by local data producers. Whenever any name prefix is added or deleted, it also disseminates a new *Prefix LSA*. The latest versions of the LSAs are stored in a Link State Database (LSDB) at each node.

The LSA format is shown in Figure 1. Each LSA has the name /⟨network⟩/NLSR/LSA/⟨site⟩/⟨router⟩/⟨lsa-type⟩/⟨version⟩, where ⟨lsa-type⟩ 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.

We consider the LSA dissemination problem as data synchronization of the LSDBs. NLSR uses the *ChronoSync* protocol [10] to synchronize changes in the routers' LSDBs. ChronoSync maintains all the 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. This synchronization approach avoids unnecessary flooding to the network – when the network is stable, only one hash, instead of all the LSA names, is exchanged among the nodes. Moreover, it separates ChronoSync's detection of new data names from 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.

Figure 2 shows how an LSA is disseminated in the network. To synchronize the digest tree representing the LSAs in the LSDB, the ChronoSync protocol on each node periodically sends periodic *Sync Interests* with the hash of its LSA name set 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

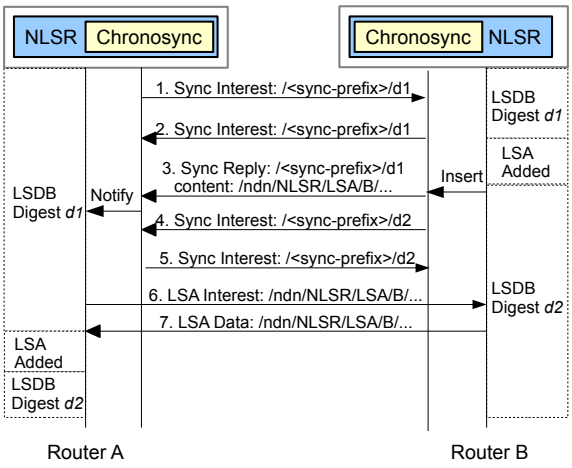


Fig. 2. LSA Dissemination via ChronoSync

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 the new LSA name (step 3). A's ChronoSync receives the *Sync Data* with the new LSA name, notifies NLSR of the missing data, and updates its LSA name set. Both A and B 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 missing LSA data, NLSR sends an *LSA Interest* to retrieve this missing 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.[1]

LSAs have a broadcast name prefix /⟨network⟩/NLSR/LSA 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, the neighbor will return it. Otherwise, the *Interest* is further forwarded. The broadcast is necessary as the NLSR routing protocol uses the LSAs to build a forwarding table, so there is no existing FIB entry for the LSAs. Since each LSA is supposed to be propagated to every NLSR router, this broadcast should not incur extra overhead.

In order to remove obsolete LSAs caused by router crashes, every router periodically refreshes each of its own LSAs by generating a newer version. Every LSA has a lifetime associated with it and will be removed from the LSDB when the lifetime expires. Therefore, if a router crashes, its LSAs will not persist in other routers' LSDBs. Note that route calculation should not be impacted by the obsolete LSAs in NLSR – if a router crashes, its neighbors will update the status of their LSAs so traffic will not be directed over those links.

[1]For details about ChronoSync (e.g., hash calculation, difference resolution, etc.), please see the original ChronoSync paper [10].

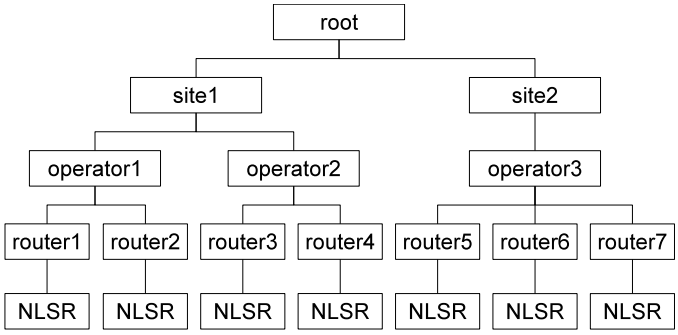


Fig. 3. NLSR Trust Hierarchy

TABLE I
KEY NAMES

Key Owner	Key Name
Network	/<network>/KEY/<key>
Site	/<network>/<site>/KEY/<key>
Operator	/<network>/<site>/<operator>/KEY/<key>
Router	/<network>/<site>/<router>/KEY/<key>
NLSR	/<network>/<site>/<router>/NLSR/KEY/<key>

As such, these refreshes should be sent at a relatively long interval, e.g., on the order of days.

C. 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 [15], which indicates the name of the key used to sign the packet, thus the receiver can fetch the key to verify the signature. In addition, the receiver needs to verify that the key is trusted to sign the LSA, which requires a trust model for key authentication.

1) Trust Model: NLSR models the trust management as a five-level hierarchy reflecting the administrative structure of an intra-domain routing protocol, as shown in Figure 3. At the top level, there is a single authority that is responsible for the whole *network*. The network consists of multiple *sites*, which can be departments in an organization or PoPs in an ISP. Each site has one or more *operators* who collectively manage a number of *routers* belonging to a site. Each router can create an *NLSR process* that produces LSAs. With this hierarchical trust model, one can establish a chain of keys to authenticate LSAs. Specifically, 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 network authority using the network key, which we call *trust anchor*.

Listing 1. Schema for LSA Validation

```
security
{
  validator
  {
    rule
    {
      id "NSLR 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><ksk-.*><ID-CERT>$
            k-expand \\1
            h-relation equal
            p-regex ^([^<NLSR><LSA>]*)<NLSR><LSA>(<>*)<><><>$
            p-expand \\1\\2
          }
        }
      }
    }

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

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

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

The name of a key explicitly expresses the role that the key plays in the system, as shown in Table I.[2] The trust relationship between these keys can be expressed using a trust schema [6] 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. In other words, a router can express an *Interest* with a key name to retrieve the key.[3] Because one must be able to retrieve keys to verify routing updates before routes are established, NLSR lets each router retrieve keys from its neighbors. Specifically, a router expresses an interest for a key to all its direct neighbors. Neighbors will lookup the requested keys in their local key storage (and Content Store) and return the key if it is found. In case a neighbor does not have the requested key, the neighbor can further query its neighbors for the key. Note that such key retrieval requests resemble broadcast, but will stop at routers who either own or cache the requested keys. We assign a special name prefix for key retrieval /⟨network⟩/broadcast/KEYS, which is prepended to the requested key name. Each router also listens to this prefix to receive the key request. Their response to the key request is an encapsulated data packet whose content is the original data packet containing the requested key. Since the authenticity of a key depends on the signature of the inner data packet, the signature field of the outer data packet is filled with a SHA-256 hash for integrity checking of the packet.

Once a key is received, it can be cached for future use and for satisfying the key request from other neighbors.

D. 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 destination in this topology. This process is repeated for every adjacent link. Afterwards, it ranks the next hops for each destination based on their costs to reach the destination. Since we know which name prefixes are associated with which routers based on the Prefix 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 controlled when a node has many neighbors. The computational cost, however, 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 algorithms.

E. Hello Protocol

NLSR sends periodic "*info*" *Interests*, at a default interval of 60 seconds, to each neighboring node to detect its status.

[2]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.

[3]Note that the last component of a key name is always the hash of the key, so that the name always matches a specific key.

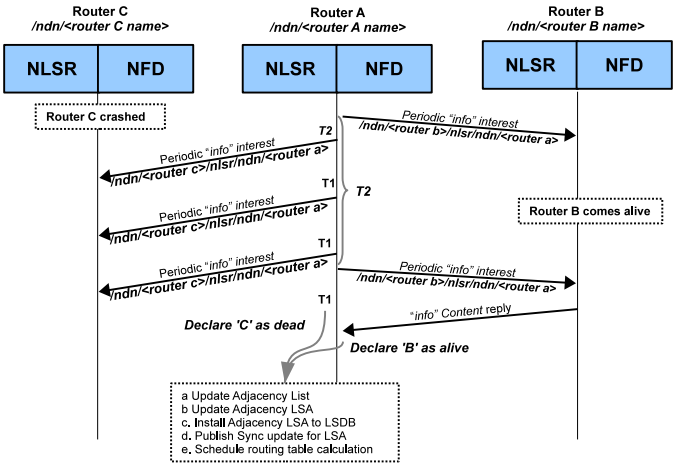


Fig. 4. Adjacency failure and Recovery detection

If the neighbor responds to the *Interest* with *Data* and the *Data* can be validated based on the trust model, the neighbor is considered up, or *ACTIVE*. If an *info* 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 determine whether the remote NLSR process has died or the link has failed. However, this distinction is not important since 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.

When a neighboring NLSR process or a link recovers, NLSR will receive a response to its "*info*" Interest and change the adjacency status to *ACTIVE*. To quickly set up an adjacency, NLSR will also send an *info* Interest after receiving an *info* Interest from a previously *INACTIVE* node instead of waiting to send the next scheduled periodic *Interest*. Figure 4 illustrates how Node A detects an adjacency failure with Node C and a recovery with Node B.

The Hello Protocol 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 down, 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.

F. Summary of Design Changes

One of the most important changes since NLSR's initial design 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. Moreover, ChronoSync uses a broadcast model to synchronize among all NLSR routers which can propagate new information faster than a hop-by-hop synchronization model used in CCNSync.

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 Hello protocol to use face event notifications so that it can react to link failures much faster than relying on Hello messages alone, which by default are sent every 60 seconds.

IV. Implementation

The current NLSR design is implemented in C++ using the ndn-cxx [16] library to run over NFD [17] (the initial NLSR design was implemented in C using CCNx [5]). It is open source [18]. Below we describe some implementation details.

a) Hello Protocol: There are three parameters in NLSR's configuration file that can be used to modify the behavior of the Hello Protocol. The *Hello Interval* can be changed to reduce or increase the frequency of periodic *info* Interests (default is 60 seconds), the *Hello Retry Amount* specifies the number of times the *info* Interest can be resent before determining the adjacency as down (default is 3 times), and the *Hello Timeout Time* is used to specify the *info* Interest lifetime before the Interest times out (default is 1 second).

b) LSA Version Numbers: The version number for each LSA increases by one after each change. It is represented by a 24-bit value for the name LSA and a 20-bit value for the Adjacency LSA. 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. But, the shorter the delay, the faster the router can build an Adjacency LSA so that the network can use paths through its 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, besides the NLSR process key, is created using the ndn-cxx [16] *ndnsec* tools. A public/private key pair and corresponding certificate are created for each key owner in the trust model hierarchy. The certificates for each public key in the key pairs are signed by the key owner one level higher in the hierarchy; the network key is self-signed.

The NLSR process key is created automatically when NLSR is initialized using the ndn-cxx security API. On initialization, NLSR generates a key pair and gets the certificate for the public key (signed with the router's private key). The NLSR process uses its private key to sign *info* data and LSA data.

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 Prefix 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 signed 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 verify that the command Interest is signed by an operator of the router and perform the specified action on the Prefix LSA. This new Prefix LSA will then be disseminated to the network.

V. Evaluation

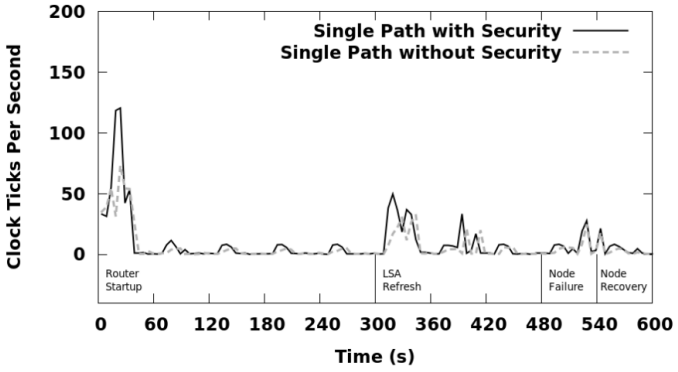
This section presents the evaluation results of NLSR in terms of CPU processing time, routing convergence time, and forwarding plane performance. All of our experiments were performed using Mini-NDN [19], an NDN network emulation tool based on Mininet [20]. In Mini-NDN, an entire network topology runs 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

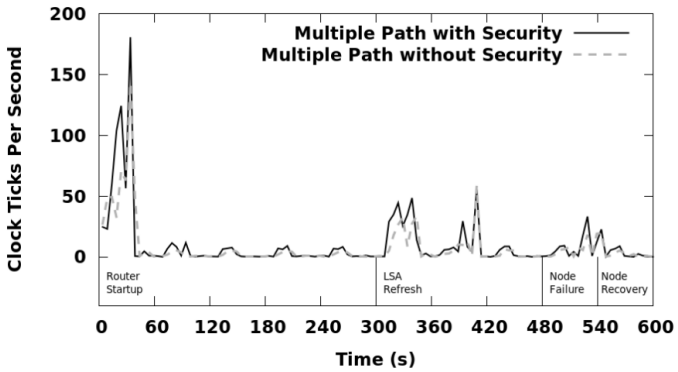
Each experiment lasts 600 seconds. First, NLSR is started on each router in the network. After 300 seconds, routers begin to refresh their LSAs. Note that in order to test the protocol in a short period of time, we set the refresh timer to be 300 seconds instead of on the order of days. At 480 seconds, 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.

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 snapshot of the actual NDN testbed topology with 22 nodes and 50 links. The routing cost of each link is set to the delay between the two neighboring nodes. Our larger three 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 [21], 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.



a) Single-path Routing



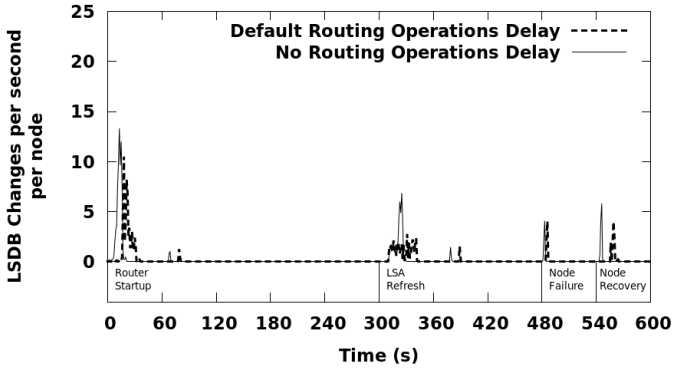
b) Multipath Routing

Fig. 5. Total Network CPU Utilization for NLSR

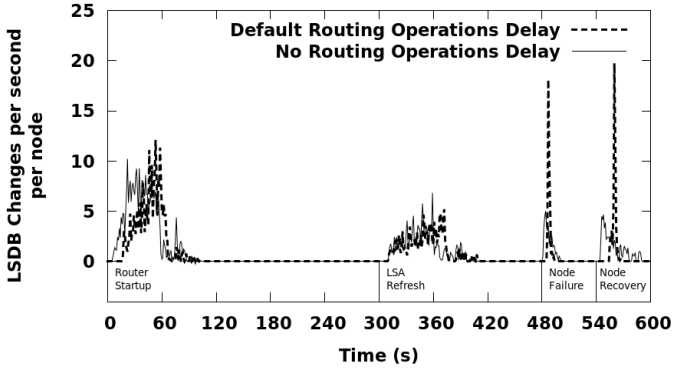
C. Results

The first experiment is performed on the NDN testbed topology to determine the CPU impact of key authentication and multpath calculation. The maximum number of next hops per name prefix is set to 4 for all the multipath experiments. Figure 5 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.

Even with the proposed trust model, which requires verification of multiple levels of keys, NLSR hardly incurs much extra processing cost. During the router 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 only done once per new key,

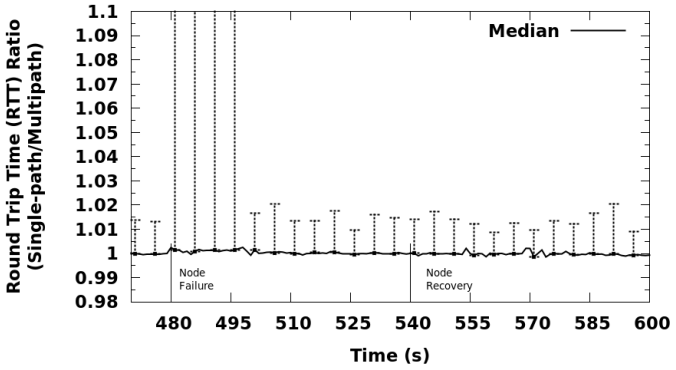


a) NDN Testbed Topology

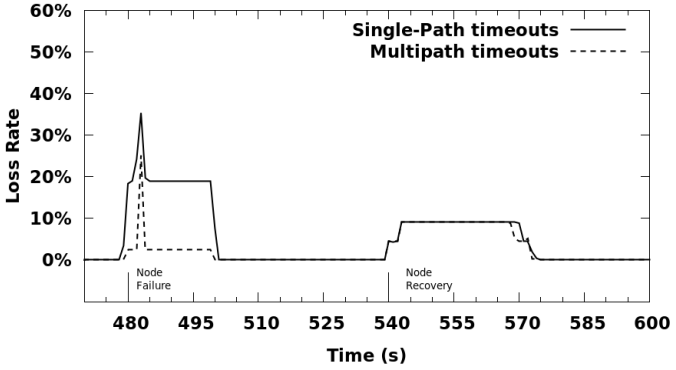


b) 78-node Topology

Fig. 6. NLSR LSDB Changes



a) RTT Ratio between Single Path and MultiPath



b) Ping Loss Rate

Fig. 7. Forwarding Convergence

it incurs a very low CPU cost. Figure 5 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.

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 operations delay* (Section IV-0c) and with *no routing operations delay*. Due to space constraint, we only show the results from the smallest and largest topologies; the other figures are similar. Figure 6 shows the average per-node LSDB changes each second. *Without routing operations delay*, NLSR converges more quickly than with the *default delay* but also has more cumulative LSDB changes. During the startup period, the *default delay* generates 19% less LSDB changes than *no delay* in the NDN testbed topology and 22% less in the 78-node topology. During the LSA refresh and node failure periods, *default delay* and *no delay* generate the same number of LSDB changes in both topologies. After the recovery, the *default delay* generates 1% less LSDB changes than *no delay* in the NDN testbed topology and 27% less in the 78-node topology. Comparing the two figures, the larger topology takes longer to converge and incurs more cumulative LSDB changes per node regardless of whether routing operations delay is 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* that maintains a smoothed RTT for each name prefix through each available next hop. 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 run on the NDN testbed topology with the default values for *routing operations delay*. Figure 7(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 7(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 add more statistics collection to facilitate debugging of the protocol in deployment and 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. In the long term, we plan to explore new types of routing designs to scale global 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.

VIII. Acknowledgment

This work was supported by NSF Grants 1040036, 1039615, 1040868, 1344495, 1345142, and 1345318.

References

[1] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," in *Proceedings of ACM CoNEXT*, 2009.

[2] L. Zhang et al., "Named data networking (NDN) project," NDN, Tech. Rep. NDN-0001, October 2010.

[3] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, k. claffy, P. Crowley, C. Papadopoulos, L. Wang, and B. Zhang, "Named Data Networking," *ACM SIGCOMM Computer Communication Review (CCR)*, 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," *Computer Communications*, vol. 36, no. 7, pp. 779–791, 2013, iSSN 0140-3664. [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 *Proceedings of the ACM SIGCOMM Workshop on Information-Centric Networking*, 2013.

[6] Y. Yu, A. Afanasyev, D. Clark, kc Claffy, V. Jacobson, and L. Zhang, "Schematizing and automating trust in named data networking," in *Proceedings of the 2nd ACM ICN Conference*, 2015.

[7] C. Yi, J. Abraham, A. Afanasyev, L. Wang, B. Zhang, and L. Zhang, "On the role of routing in named data networking," in *Proceedings of ACM SIGCOMM ICN Conference*, 2014.

[8] PARC, "CCNx open srouce platform," http://www.ccnx.org.

[9] NDN Project Team, "The NDN platform," http://named-data.net/codebase/platform/.

[10] Z. Zhu and A. Afanasyev, "Let's ChronoSync: Decentralized dataset state synchronization in Named Data Networking," in *Proceedings of IEEE ICNP*, 2013.

[11] N. P. Team, "NLSR 0.1.0," http://named-data.net/doc/NLSR/0.1.0.

[12] H. Dai, J. Lu, Y. Wang, and B. Liu, "A two-layer intra-domain routing scheme for Named Data Networking," *Globecom 2012 - Next Generation Networking and Internet Symposium*, December 2012.

[13] E. Hemmati and J. Garcia-Luna-Aceves, "A new approach to name-based link-state routing for information-centric networks," in *Proceedings of the 2Nd ACM ICN Conference*, ser. ICN '15. New York, NY, USA: ACM, 2015.

[14] J. Garcia-Luna-Aceves, "Routing to multi-instantiated destinations: Principles and applications," in *Network Protocols (ICNP), 2014 IEEE 22nd International Conference on*. IEEE, 2014, pp. 155–166.

[15] "NDN packet format specification," http://named-data.net/doc/ndn-tlv/.

[16] N. P. Team, "ndn-cxx," http://named-data.net/doc/ndn-cxx/.

[17] ——, "NFD - NDN forwarding daemon," http://named-data.net/doc/nfd/.

[18] ——, "Named data link state routing," https://github.com/named-data/NLSR.

[19] "Mini-NDN GitHub," https://github.com/named-data/mini-ndn.

[20] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: Rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, 2010.

[21] M. A. Serrano, D. Krioukov, and M. Boguñá, "Self-Similarity of Complex Networks and Hidden Metric Spaces," *Phys Rev Lett*, vol. 100, p. 78701, 2008.