NLSR: Named-data Link State Routing Protocol

A K M Mahmudul Hoque
ahoque1@memphis.edu
University of Memphis

Syed Obaid Amin
soamin@memphis.edu
University of Memphis

Adam Alyyan
aalyyan@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

Lan Wang
lanwang@memphis.edu
University of Memphis

ABSTRACT

This paper presents the design of the Named-data Link State Routing protocol (NLSR), a routing protocol for Named Data Networking (NDN). Since NDN uses names to identify and retrieve data, NLSR propagates reachability to name prefixes instead of IP prefixes. Moreover, NLSR differs from IP-based link-state routing protocols in two fundamental ways. First, NLSR uses Interest/Data packets to disseminate routing updates, directly benefiting from NDN's data authenticity. Second, NLSR produces a list of ranked forwarding options for each name prefix to facilitate NDN's adaptive forwarding strategies. In this paper we discuss NLSR's main design choices on (1) a hierarchical naming scheme for routers, keys, and routing updates, (2) a hierarchical trust model for routing within a single administrative domain, (3) a hop-by-hop synchronization protocol to replace the traditional network-wide flooding for routing update dissemination, and (4) a simple way to rank multiple forwarding options. Compared with IP-based link state routing, NLSR offers more efficient update dissemination, built-in update authentication, and native support of multipath forwarding.

Categories and Subject Descriptors

C.2.2 [**COMPUTER-COMMUNICATION NETWORKS**]: Network Protocols—*Routing Protocols*

General Terms

Design, Security

Keywords

Routing, NDN, Trust Model

1. INTRODUCTION

The Named Data Networking (NDN) [3, 4] architecture is a fundamental paradigm shift from the current IP-based Internet architecture. Instead of carrying the destination IP address 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, and the response is a *Data* packet containing the name, the data, and a signature by the original data producer. By explicitly naming and signing data, NDN enables features such as in-network caching, multipath forwarding, multicast data delivery and data authenticity.

For NDN to work well over wide-area networks, a routing protocol is needed to compute and install proper forwarding entries into an NDN node's forwarding table (FIB). Each FIB entry contains a name prefix and one or multiple next-hops, and is used to forward Interest packets whose names match the name prefix of the entry. While IP has to either use a single best next-hop or limit 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 prevention in the forwarding process. Thus an NDN network needs a routing protocol that can support name-based multipath routing.

We previously developed OSPFN [8], an extension to OSPF (Open Shortest Path First) for routing in NDN, and deployed it on the NDN testbed. OSPFN defines a new type of opaque link state announcement (LSA) to carry name prefixes in routing messages. It installs the best next-hop to each name prefix in the FIB; operators may manually configure a list of alternative next-hops for OSPFN to install in the FIB in addition to the best one. Although OSPFN can build a FIB with name prefixes, it has significant limitations. As in conventional IP-based routing protocols, OSPFN still uses IP addresses as router IDs, relies on GRE tunnels to cross legacy networks, and computes only a single best next-hop for each name prefix. Our experience from OSPFN deployment shows that managing IP addresses and tunnels are major operational problems, and the inadequate multipath support limits NDN's effectiveness.

In this paper we present the design of Named-data Link State Routing protocol (NLSR), which runs on top of NDN. In other words, NLSR uses NDN's Interest/Data packets to exchange routing messages. It is a link-state protocol as OSPF – link state advertisements are propagated throughout the entire network and each router builds a complete network topology. However, the route computation no longer produces just a single shortest path. It now *ranks* all policy-compliant next-hops and installs them into the FIB in order, essentially providing a name-based multipath routing table as input to NDN's forwarding strategy [9].

NLSR uses names instead of IP addresses to identify routers and links, therefore it can use any underlay communication channels (e.g., Ethernet, IP tunnels, TCP/UDP tunnels) for routing message exchanges. NLSR directly benefits from NDN's built-in data authenticity: since a routing update is carried in an NDN data packet and every NDN data packet carries 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 during dissemination.

As the first distributed routing protocol on NDN, NLSR's design needs to answer the following important questions that are unique to applications running over NDN:

- **Naming**: how to name routers, links and routing updates.
- **Trust**: how to distribute routers' cryptographic keys and how to derive trust in these keys.
- **Information Dissemination**: how to disseminate routing updates in the network. While IP-based routing protocols *push* updates to other routers, NLSR routers need to *pull* the updates.
- **Multipath**: how to produce and rank the next-hops to facilitate multipath forwarding.

In this paper we describe our design choices and articulate the rationales behind these choices. Our goal is not to invent a new routing scheme as NLSR is essentially another link-state protocol, but rather, to demonstrate the feasibility and benefits of building a routing protocol using NDN.

Since NDN's adaptive, multipath forwarding is able to handle many packet delivery problems at the forwarding plane, the requirements on the routing plane is relaxed. For example, ensuring loop-freedom at the routing plane is no longer critical. Thus routing protocol designs that previously do not work in IP networks may now work in NDN networks, and new types of routing designs may also become possible. Exploring other types of routing designs would be our future work.

The remainder of the paper is organized as follows. The next section briefly introduces a few basic functions provided in an NDN network. Section 3 presents the design details. Section 4 provides the evaluation results. Section 5 discusses related work, and finally Section 6 concludes our work.

2. NDN PRIMER

NDN has three main components in its forwarding plane [3]: (1) *Forwarding Information Base (FIB)*: it stores the forwarding entries that direct Interest packets toward potential source(s) of matching Data. Unlike IP, it allows for a list of outgoing faces (next-hops) rather than a single one for each name prefix. The FIB is populated manually and/or by an NDN routing protocol in the control plane; (2) *Pending Interest Table (PIT)*: it stores the unsatisfied Interest packets and the faces on which they were received, so that Data packets can be routed back to the nodes interested in the data; and (3) *Content Store (CS)*: it is used for caching data.

When an Interest arrives at an NDN router, the CS is checked first for any matching data. If the Interest can be fulfilled by the CS, a Data packet is sent back on the face on which the Interest was received. Otherwise, it is added to the PIT. If there exists an entry with the same name, the new face number is added to the face list, so that a copy of the matching Data packet can be sent on all faces from which the Interest arrived. Finally, if the Interest does not have a matching PIT entry, the Interest is forwarded to the next-hop(s) based on the FIB. If multiple next-hops exist in a FIB entry, a module called "forwarding strategy" determines how the multiple routes will be used in forwarding Interests.

3. DESIGN

As a link-state protocol, NLSR disseminates Link State Advertisements (LSAs) to both build a network topology and distribute name prefix reachability. 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 LSA to the entire network. Moreover, it advertises name prefixes from both static configuration and dynamic registration by local content producers. Whenever any name prefix is added or deleted, it also disseminates a new LSA. The latest version of the LSAs are stored in a Link State Database (LSDB) at each node.

Such topology and reachability dissemination may at first appear to be straight-forward as similar functions have been implemented in IP routing protocols. However, because we implement NLSR using NDN Interest and Data packets, the design must shift away from the familiar concepts of IP addresses and IP data pushing (i.e., any node can simply send any packet to any other node). Instead, we have to think in terms of data names and data retrieval. More specifically, we need a systematic naming scheme for routers and routing updates (Section 3.1). 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 3.3).

In terms of routing functionality, NLSR distinguishes itself from all previous link-state routing protocols in two aspects: (a) providing multiple routes to each name prefix, instead of a single shortest path; and (b) signing and verification of all LSAs to ensure that each router can originate only its own prefix and connectivity information. We present our route calculation algorithm in Section 3.4 and our trust model in Section 3.6.

As a preliminary step in developing NDN-based routing protocols, our initial design of NLSR is in the context of a single routing domain with a single authority that our trust model is built upon. We are in the process of deploying NLSR on the NDN testbed. We believe that this initial design and deployment experience can offer us a concrete stepping stone toward developing an NDN-based inter-domain routing protocol that incorporates routing policies and an inter-domain trust model.

3.1 Naming

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. 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 associating keys with their corresponding 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 name, i.e., `/<network>/<site>/<router>`. For example, an ATT router in a PoP (point of presence) in Atlanta may be named `/ATT/AtlantaPoP1/router3`. This way, we know that if two routers share the same name prefix `/<network>`, they belong to the same network; and if they share the same prefix `/<network>/<site>`, they belong to the same site. This naming scheme makes it easy to filter out erroneous routing messages.

The NLSR process on a router is named after the router name: the router name is used as its prefix, followed by the process name NLSR, i.e., `/<network>/<site>/<router>/NLSR`. This NLSR routing demon name is used in periodic *info* messages between adjacent NLSR routers to detect the failure of either links or routing processes themselves (see Section 3.5).

Ideally, any routing updates originated by an NLSR process should have the process name as its prefix to easily identify the messages originator. In other words, the name for an LSA should begin with `/<network>/<site>/<router>/NLSR/LSA` to indicate that it is generated by the NLSR process. However, because

Table 1: Contents of an LSA

Type	Content
Adjacency LSA	# Active Links (N), Neighbor 1 Name, Link 1 Cost, ..., Neighbor N Name, Link N Cost
Prefix LSA	isValid, Name Prefix

our implementation uses CCNx Sync [5] and Repo [5] to disseminate LSA data, and CCNx repo imposes a constraint that all the data to be synchronized must share a common name prefix, our current implementation is confined to using a common prefix for the LSAs generated by all the routers. We name each LSA using the common prefix `/<network>/NLSR/LSA` (we call this `<LSA-prefix>`), and append `/<site>/<router>` to this prefix to differentiate LSAs originated by different NLSR routers.

3.2 LSAs

NLSR is designed to propagate two types of LSAs – Adjacency LSA and Prefix LSA. The Adjacency LSA is used to advertise all active links connecting one NDN router to its neighbors. The Prefix LSA, on the other hand, is used to advertise a name prefix that has been registered with the router. Their contents are shown in Table 1.

An Adjacency LSA has the name format `/<LSA-prefix> /<site>/<router>/LsType.1/<version>`, where <router> is the name of the router that originates the LSA and <version> indicates the ordering in the various versions of a particular LSA as it changes over time. It is currently implemented as the LSA origination times in microseconds from epoch time. However, similar to OSPF, sequence numbers can also be used for this purpose. As shown in Table 1, the 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 periodical "*info*" Interest messages (Section 3.5).

A Prefix LSA has the name format `/<LSA-prefix>/<site> /<router>/LsType.2/LsId.<ID>/<version>`. Note that each Prefix LSA advertises one name prefix. Since one router may have multiple name prefixes registered with it, it needs to announce multiple Prefix LSAs, using a unique LSA ID[1] in their name to differentiate them. The rationale for this design decision is that bundling all the name prefixes of a router in a single LSA may make it too large to be transported in one message and also inefficient to update (even if only one prefix is added or removed, all the other prefixes in the same LSA need to be advertised again). Each Prefix LSA contains a flag *isValid* (set to 1 initially) and the name prefix to be advertised (Table 1). When a name prefix is de-registered, NLSR updates the corresponding prefix LSA by setting *isValid* to 0, and disseminates the new LSA to other nodes. An NLSR node receiving this LSA will delete this name prefix from its LSDB and update its FIB accordingly.

In order to remove obsolete LSAs caused by router crashes, every router periodically refreshes each of its advertised 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. Since we do not use the refreshes to handle packet losses or state corruption (CCNx Sync handles it) and the obsolete LSAs do not affect route calculation, these refreshes should be sent at a relatively long interval, e.g., on the order of days.

[1]The LSA ID can be manually configured or calculated based on the name prefix (e.g., a hash of the name prefix).

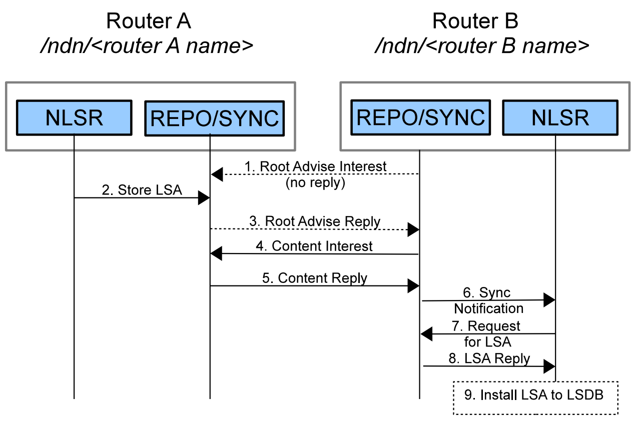

Figure 1: LSA dissemination from router to router via CCNx Sync/repo (dotted line represents periodic messages.)

3.3 LSDB Synchronization

To simplify our design conceptually, we decided to view the LSDB as a collection of data, and the LSA dissemination problem as a data synchronization problem of the LSDBs maintained by the routers. Routers periodically exchange their hashes of the LSDB to detect inconsistencies and recover from them. This hop-by-hop synchronization approach avoids unnecessary flooding to the network – when the network is stable, only one hash, instead of all the LSAs, is exchanged between neighbors. Moreover, it is also receiver-driven, meaning that a router will request LSAs only when it has CPU cycles. Thus it is less likely a router will be overwhelmed by a flurry of updates.

Our current implementation uses the CCNx synchronization protocol, or *Sync* [5], to disseminate the LSAs to the neighboring routers. Sync is associated with the CCNx repository (or Repo). It allows applications to define collections of named data in a repo, called *slices*, which are kept in sync with identically defined slices in neighboring repos. Sync computes a hash tree over all the data in a slice and exchanges the root hash between neighbors to detect inconsistencies. If the hash values do not agree, two neighboring nodes then exchange the hash values of nodes on the next tree level until they detect the specific leaf nodes (data) causing the problems. They then exchange the data to reach consistency.

Figure 1 shows how an LSA is disseminated in the network. To synchronize the slice containing LSAs, the Sync protocol periodically sends special Interest messages, called Root Advise, with the combined hash value of the slice to the neighboring nodes (step 1). When Router A's NLSR creates an LSA and writes it in the Sync slice (step 2), its hash value becomes different from that of Router B, which causes Router A's Sync to reply to the Root Advise Interest from Router B with the new hash value of its local slice (step 3). Router B's Sync then compares the hashes and recursively requests for the next level hashes that cause the differences. Eventually, Router B's Sync identifies the data that needs to be synchronized (LSAs in the context of NLSR) and retrieves them using Interest messages (step 4 and 5). The Sync on Router B then sends the data *name* to the local NLSR agent (step 6), which fetches the data from the local repo (step 7 and 8) and updates its LSDB (step 9).

Each Root Advise Interest has a lifetime and a new Root Advise is sent when the lifetime expires. Such periodic transmission is designed to handle the loss of Root Advise Interests, and thus reduce the delay in routing convergence caused by the losses. However, if the loss rate is low, frequent transmission of the Root Advise may lead to high overhead without much benefit. Ideally we would like to adjust the frequency of Root Advise Interests based on routing requirements and network characteristics. To support this feature and address other issues with the CCNx Sync implementation, we are working on a newer version of NLSR with its own sync mechanism to achieve the same hop-by-hop distribution of LSAs.

3.4 Multipath Calculation

Based on the information available in the Adjacency LSAs, each NLSR node builds a network topology. It then runs a simple extension of the Dijkstra's algorithm to produce multiple next-hops for each destination node. From the Prefix LSAs, we know the name prefixes associated with each router (destination). Therefore, we can then obtain a list of next-hops to reach each name prefix.

Our multipath calculation works as follows. It removes all immediately adjacent links except one and uses the 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. Note that NLSR allows an operator to specify the maximum number of paths per name prefix to insert into a FIB, so that the FIB size can be controlled when a node has many neighbors. The computational cost, however, still increases as the number of faces increases, because we need to go through all available faces to produce the cost for each possible path. We plan to investigate other multipath algorithms to address this issue.

Unlike in IP, routing information in NDN acts only as a hint to the forwarding plane; the forwarding plane can observe data delivery performance using state maintained in the PIT and thus rank the multiple next-hops of a name prefix using the actual observation as well as the ranking from the routing protocol. However, the ranking information from the routing protocol is still important for forwarding of the initial Interest to a name prefix, and for exploring alternative routes when the current route fails to retrieve data.

3.5 Failure and Recovery Detection

To detect link failures, as well as failures of remote NLSR processes, NLSR sends periodic "*info*" *Interest* to each neighboring node. If an *info* Interest is timed out, NLSR will try sending it a few 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. Afterwards, NLSR continues sending these Interests to detect the recovery of this adjacency, but at a relatively long interval to avoid high message overhead during a long-lasting failure. 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.

When the adjacency recovers, NLSR will receive a response to its "*info*" Interest and change the adjacency status to 'Active'. This change will result in updating the Adjacency LSA, disseminating the LSA in the network, and scheduling a routing table calculation. Figure 2 illustrates how Node A detects an adjacency failure with Node C and a recovery with Node B.

3.6 Security

Every NDN *Data* packet is digitally signed and the generated signature is part of the *Data* packet. The signature covers the name, the content, and a small amount of supporting data useful in signature verification [3]. One piece of the supporting data is the key

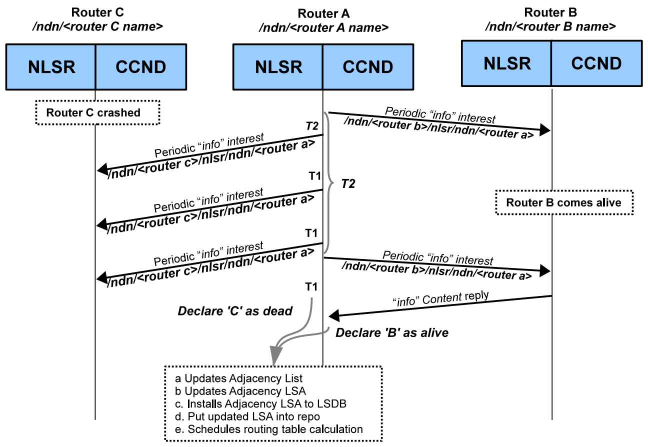

Figure 2: Adjacency failure and Recovery detection

locator [1, 3], which indicates the name of the key used to sign the packet, thus the receiver can fetch the key to verify the signature.

An LSA with a valid signature merely states that the signature is produced using the public key indicated in the key locator field. It does not tell us whether the key belongs to the router that can legitimately originate the LSA. For example, an attacker can sign a Prefix LSA with his/her public key and inject this LSA into the routing system. In order to check the authenticity of the information, we need to verify that this LSA is indeed signed by an authorized NLSR process. In other words, we need to check that the key has the correct name of the corresponding NLSR process. However, the attacker can still forge a key with the same name. We then need a trust model to verify the authenticity of the key.

NLSR is an intra-domain routing protocol. In the context of a single network domain, there is usually a network administrator (trust anchor) that can certify the authenticity of keys in the network. Therefore we use this trust anchor for key signing and verification, which is easy to setup and manage. We could let this trust anchor sign the public key of every router, but this approach presents a greater security risk when one key is used to sign a large number of keys. Instead, we design a hierarchy of five levels rooted at the trust anchor, which limits the signing scope of each key to a smaller size. Table 2 shows the name of each key at every level of the hierarchy. Note that the last component of a key name is always the hash of the key (not shown in the table), so that when someone expresses an *Interest* to a key, the name always matches a specific key. At the top of the hierarchy is a root key, owned by the network domain's administrator. The next level is a set of site keys, each owned by the administrator of a single site[2] in the domain and signed by the root key. Each site key signs a set of operator keys (there may be more than one operator in a site). Each operator key signs a set of router keys, each of which signs the key of the NLSR routing process on that router. Finally, the NLSR key signs the routing data originated by NLSR. Note that we use CCNx sync/repo to disseminate the keys, so all the keys share the common prefix `/<network>/keys`, but they do follow a hierarchical structure. Moreover, we use two tags, %C1.O.N.Start and %C1.O.R.Start, to indicate operator keys and router keys, respectively.

NLSR strictly enforces the trust model rooted at the trust anchor. Figure 3 depicts the flow of signing and verification process of each NLSR packet. When an NLSR router sends an LSA to the network, it signs the packet with its NLSR key and puts the key name in *'SignedInfo/KeyLocator/KeyName'* field of the *Data* packet. Upon

[2]A site can be a department in an organization or a PoP in an ISP.

Table 2: Keys Names

Key Owner	Key Name
Root	/<network>/keys
Site	/<network>/keys/<site>
Operator	/<network>/keys/<site>/%C1.O.N.Start/<operator>
Router	/<network>/keys/<site>/%C1.O.R.Start/<router>
NLSR	/<network>/keys/<site>/%C1.O.R.Start/<router>/NLSR

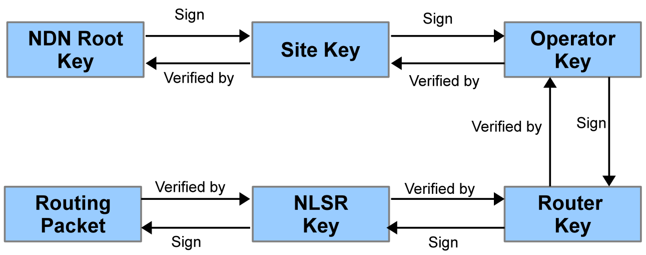

Figure 3: Signing and Verification Chain of Each NLSR packet

receiving an LSA, an NLSR router fetches the key from its local *content store* or *repo* (since the key has been distributed through the key repo) and verifies the content. NLSR also checks whether the key indeed belongs to the origination router's NLSR. This process repeats until NLSR reaches the self-signed key of the trust anchor. If at any step key fetching is unsuccessful, or NLSR finds that the key is signed by an unauthorized key, or the final verification step does not reach the trust anchor, the LSA is considered illegitimate. Note that once a key is verified, we record this information and do not repeat the verification on this key for future packets.

4. EVALUATION

This section evaluates the performance of NLSR in terms of processing time, messaging overhead and convergence time. All tests are conducted on a network consisting of six heterogeneous nodes with different OS's and system specifications. The network topology is shown in Figure 4. Note that in order to test the protocol in a short period of time, we set the refresh timer to be 30 minutes instead of on the order of days.

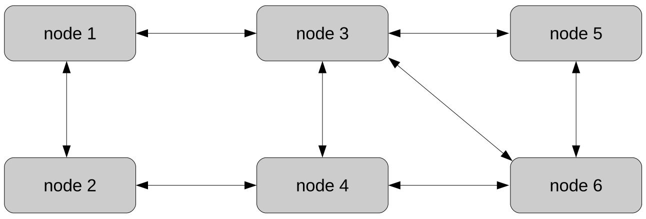

Figure 4: Network Topology

Figure 5 shows the CPU utilization of NLSR at each node. The number in parenthesis following the node's name represents the degree, or the number of neighbors of each node. The nodes with a higher degree of connectivity exhibit higher CPU utilization. In other words the computational cost increases as the number of links increases at a node. This is mainly because of the per link shortest path calculations (Section 3.4), and higher messaging overhead.

Figure 6 shows the processing overhead of NLSR with and without the proposed trust model. Even with the proposed trust model, which requires multiple levels of keys to sign and verify a packet, NLSR hardly incurs any extra processing cost. This is due to the fact that by design NDN signs all outgoing Data packets. The only

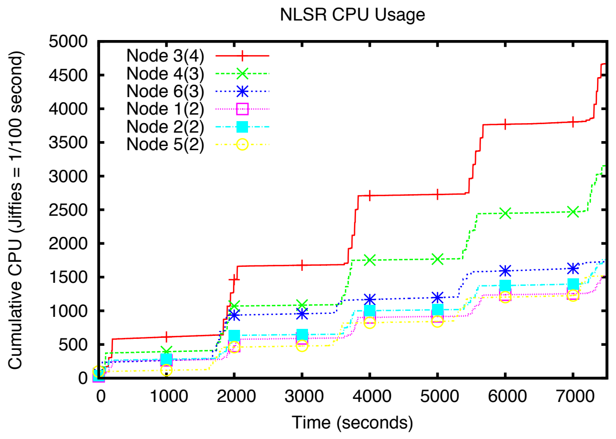

Figure 5: NLSR CPU utilization at each node

difference between the two schemes lies in the verification, where NLSR with the proposed trust scheme requires more time to fetch multiple keys recursively from the repo and verify them; however as this is done locally and only once per new key it incurs a very low CPU cost. Figure 6 also shows that with multipath routing, NLSR shows higher CPU usage than the 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.

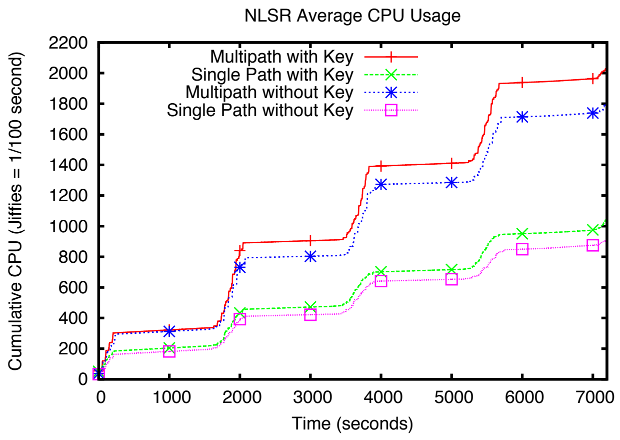

Figure 6: Average CPU utilization

The average per-node message overhead is depicted in Figure 7. We varied the lifetime of Sync's Root Advise (RA) messages from 10 seconds to 30 seconds (the default is 20 seconds). The numbers shown are the average number of Interests sent and the corresponding Data packets received by NLSR and Sync on each node. NLSR only sends *info* Interest messages, which is independent of the Root Advise (RA) interval, so the number of NLSR messages remains the same for different RA intervals. Interests generated by Sync mainly stems from the 'RA' interest, which is sent periodically, and get replied only if there is a disagreement on hashes, therefore the number of RA Interests sent are higher than the number of received RA packets. As expected, a longer RA interval leads to a lower number of RA messages. Sync also transmits Node Fetch (NF) interests to fetch the nodes on Sync Trees. NF does not contribute as much to the messaging overhead as RA, especially when the network is stable. The Interest sent to fetch the actual LSA data is broadcasted to all the faces, but gets replied only from the resource that has the missing LSAs. Therefore the number LSA interests sent is higher than the LSA Data packets received.

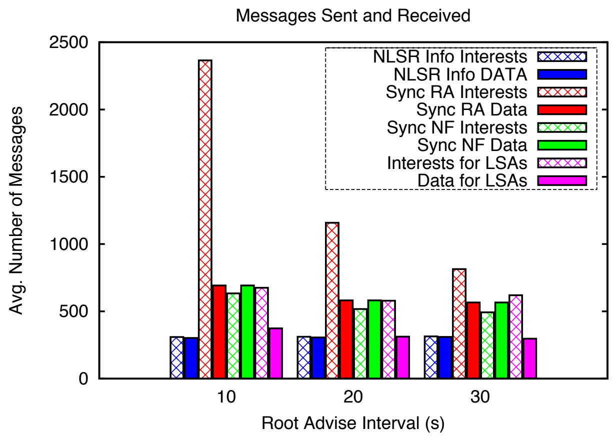

Figure 7: Number of packets sent and received

The same topology is used for the convergence tests. After booting up all the nodes, enough time was provided so that all LSDBs can get synchronized. Once the network is in a converged state, we generated traffic using the *ccnping* utility [7]. The ccnping server is hosted on *node 6*, while *node 2* is used to generate the ccnping Interest (ping) messages, with a default time out value of 4 seconds. After 60 seconds, we brought down *node 4*, which forces *node 2* to change the path to *node 6*. Figure 8 shows the benefits of multipath routing – node 2 did not need to recalculate the path and it moved to an alternate path as soon as it detected a failure. In contrast, NLSR with only single-path calculation took more than a minute to find the alternate route and moved back to the old path once we brought it up back again at 180sec. The convergence time can be controlled by *info* Interest timeout value and the number of retries, which are set to 60 seconds initially and 3 times with a 15-second interval, respectively, before declaring a node or link as down.

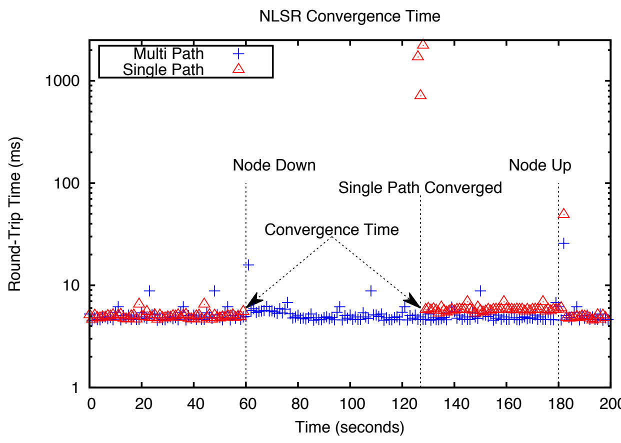

Figure 8: Convergence time with & without multipath support

5. RELATED WORK

To the best of our knowledge, very limited work has been done in the routing area of NDN. The routing protocol proposed by Dai et al. [2] is similar to NLSR on the surface, but it differs from NLSR in the following aspects. First, it uses OSPF to collect the topology and compute shortest path. We use SYNC to disseminate LSAs, not flooding. Second, their routing messages are not sent as Interests/Data and therefore cannot enjoy the benefit of signed updates, i.e., security. Third, their multipath forwarding is limited to contents served by multiple producers, e.g., anycast among a number of server replica. While we support that scenario, we also support multiple paths to the same producer.

In [6], authors proposed a Controller-based Routing Scheme (CRoS) for NDN. The controllers store the network topology, calculate the routes, and store named data locations, so that they can install route for any named data in the network. Although the idea of having decentralized Controllers is interesting, the network needs to be flooded with specially formatted Interest message to search for controllers, which can result in high overhead.

6. CONCLUSION

Although the design of link-state routing protocols for IP-based networks is a well understood subject, our design of NLSR so far has served as a great learning experience. To meet NDN's routing needs, NLSR departs from the conventional routing protocol's single path forwarding and instead provides multiple forwarding options for each name prefix.

However our major gains from this experience came from NLSR being a specific case of developing a new application on top of NDN, which requires: (1) careful considerations on the name space design, (2) the development of a trust model for key verification, and (3) a mental adjustment to NDN's new design patterns of using Interest-Data exchanges for routing update messages. Furthermore, the use of named data for communication enables the concept of Sync, which facilitates dataset synchronization in distributed systems. Our NLSR design utilizes Sync to make the routing protocol more robust and conceptually simpler.

The results so far represent a preliminary step toward the development of an NDN-based routing system. Our ongoing efforts include real-world deployment and operation, exploring new routing schemes, and extending into inter-domain routing.

7. REFERENCES

[1] C. Bian, Z. Zhu, E. Uzun, and L. Zhang. Deploying key management on NDN testbed. Technical Report NDN-0009, Febryary 2013.

[2] 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.

[3] 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.

[4] L. Zhang et al. Named data networking (NDN) project. Technical Report NDN-0001, PARC, October 2010.

[5] PARC. CCNx open srouce platform. `http://www.ccnx.org`.

[6] J. Torres, L. Ferraz, and O. Duarte. Controller-based routing scheme for Named Data Network. Technical report, Electrical Engineering Program, COPPE/UFRJ, December 2012.

[7] University Of Arizona. ccnping. `https://github.com/NDN-Routing/ccnping`.

[8] L. Wang, A. M. Hoque, C. Yi, A. Alyyan, and B. Zhang. OSPFN: An OSPF based routing protocol for Named Data Networking. Technical Report NDN-0003, July 2012.

[9] C. Yi, A. Afanasyev, L. Wang, B. Zhang, and L. Zhang. Adaptive forwarding in named data networking. *SIGCOMM Comput. Commun. Rev.*, 42(3):62–67, June 2012.