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

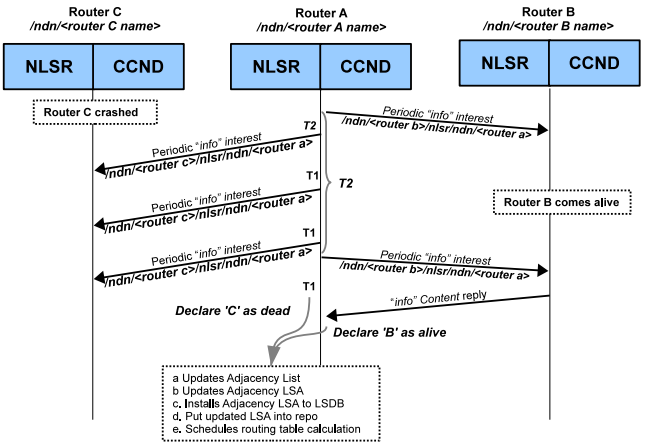

Figure 2: Adjacency failure and Recovery detection

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

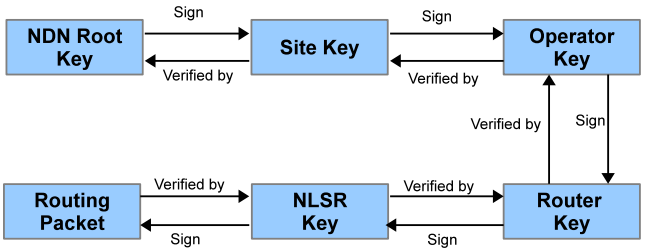

Figure 3: Signing and Verification Chain of Each NLSR packet

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.

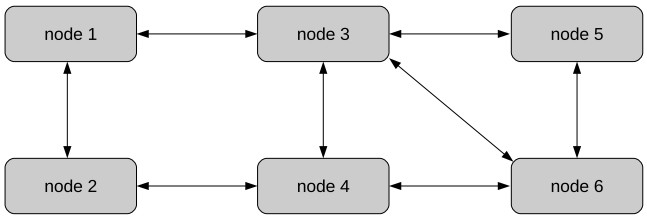

Figure 4: Network Topology

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

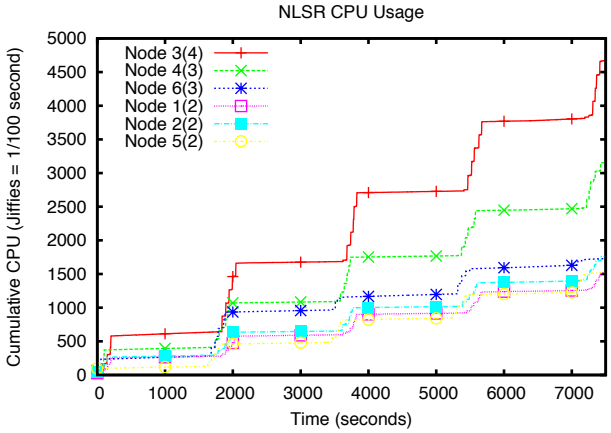

Figure 5: NLSR CPU utilization at each node

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.

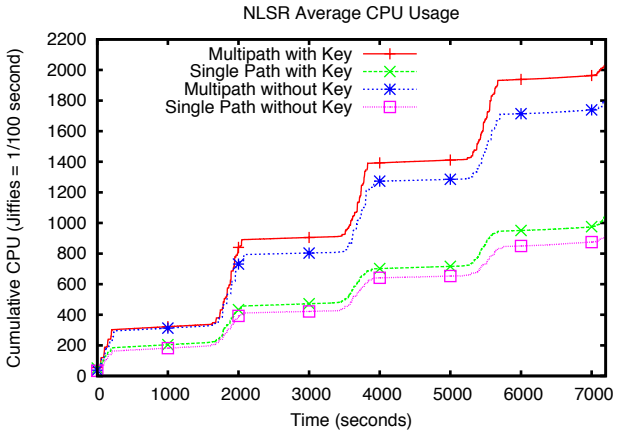

Figure 6: Average CPU utilization

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.

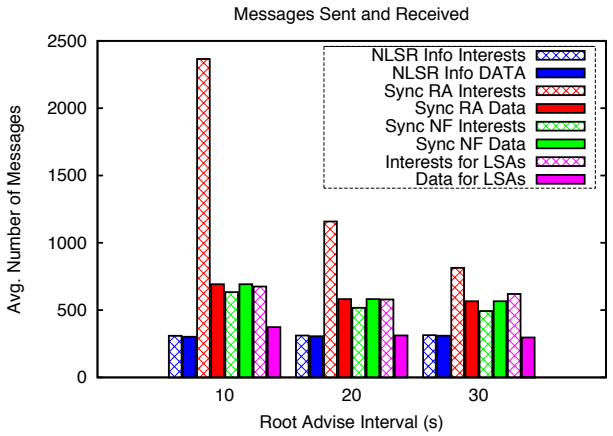

Figure 7: Number of packets sent and received

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.

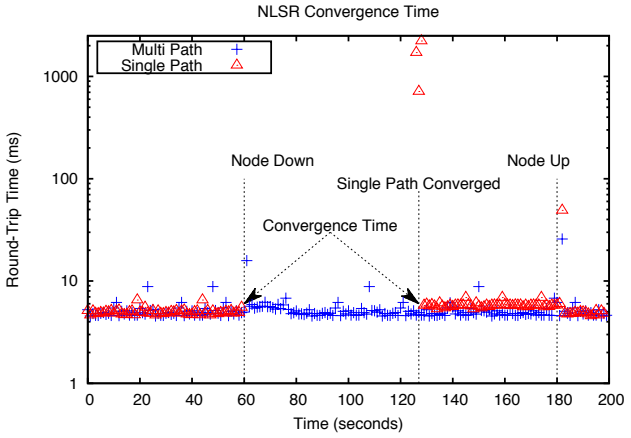

Figure 8: Convergence time with & without multipath support

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.