NLSR Developer's Guide

Vince Lehman, Muktdir Chowdhury, Nicholas Gordon[1]

[1]University of Memphis

January 24, 2017

Abstract

The Named Data Link-State Routing Protocol (NLSR) is a Named Data Networking (NDN) routing protocol that populates NDN Forwarding Daemon's (NFD) Routing Information Base (RIB). The main design goal of NLSR is to provide a routing protocol to populate NFD's RIB and Forwarding Information Base (FIB). NLSR calculates the routing table using link-state or hyperbolic routing and produces multiple faces for each reachable name prefix in a single authoritative domain. NLSR will continue to evolve alongside the NDN protocol, NFD, and ndn-cxx. This document is meant to explain the design of NLSR including major module and data structures descriptions and the interactions between those components.

Contents

1   Introduction

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. 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.

NLSR supports name-based routing in NDN, computes routing ranks for all policy-compliant next-hops which provides a name-based multi-path routing table for NDN's forwarding strategy, and ensures that routers can originate only their own routing updates using a hierarchical trust model.

1.1   NLSR Modules and Data Structures

NLSR contains multiple modules that each contribute to the total realization of the protocol. Many of the modules interact with one another to trigger some behavior or to modify information in data structures. NLSR uses the following modules:

- **Hello Protocol** (Section 2) - determines the status of neighboring routers using periodic Hello Interests and notifies other modules when neighbors' statuses change.

- **NSync** - provides LSDB synchronization by extending the ChronoSync protocol [1].

- **Sync Logic Handler** (Section 3) - handles sync update notifications from NSync by retrieving updated LSAs.

- **LSAs** (Section 4) - represent routing information published by the router.

- **LSDB** (Section 5) - stores the LSA information distributed by other routers in the network.

- **Routing Table** (Section 6) - calculates and maintains a list of next hops for each router in the network.

- **Name Prefix Table** (Section 7) - stores all advertised name prefixes and their next hops.

- **FIB** (Section 8) - maintains a shadow FIB which represents the intended state of NFD's FIB [2].

- **Prefix Update Processor** - listens for dynamic prefix announcements to advertise or withdraw name prefixes.

1.2   Protocol Overview

NLSR is designed to accomplish three main tasks: (1) discover adjacent neighbors; (2) disseminate and synchronize topology, name prefix, and hyperbolic routing information; and (3) calculate a consistent routing table and populate NFD's FIB. The entire protocol is described in detail in the NLSR paper [3].

1.2.1   Discovering Neighbors

NLSR determines the adjacency status of neighboring routers using the Hello Protocol module (Section 2). When the Hello Protocol detects a status change for a neighbor, it will ask the LSDB module (Section 5) to update the router's advertised adjacency information.

1.2.2   Disseminating Routing Information

When one of a router's LSAs (Section 4) changes, the information of this change should be distributed to every other router in the network. The Sync Logic Handler module (Section 3) is used to notify the synchronization protocol of changes to the router's own LSAs as well as to learn of LSA changes from other routers in the network; the Sync Logic Handler module interfaces with NSync to perform the two tasks.

When the Sync Logic Handler module learns of a new LSA, it will inform the LSDB module. The LSDB module will attempt to fetch the new LSA and will store it in the LSDB module's database if it can be retrieved. If the newly fetched LSA informs the router of previously unknown routing information, the LSDB module will inform other modules depending on the type of routing information:

- **Change in network topology** - the LSDB module will inform the Routing Table module (Section 6), so the Routing Table module can calculate an up-to-date routing table.

- **Change in name prefix advertisement** - the LSDB module will inform the Name Prefix Table module (Section 7), which will in turn notify the FIB module (Section 8) in order to add or remove the changed name prefixes.

- **Change in hyperbolic coordinates** - the LSDB module will inform the Routing Table module (Section 6), so the Routing Table module can calculate an up-to-date routing table.
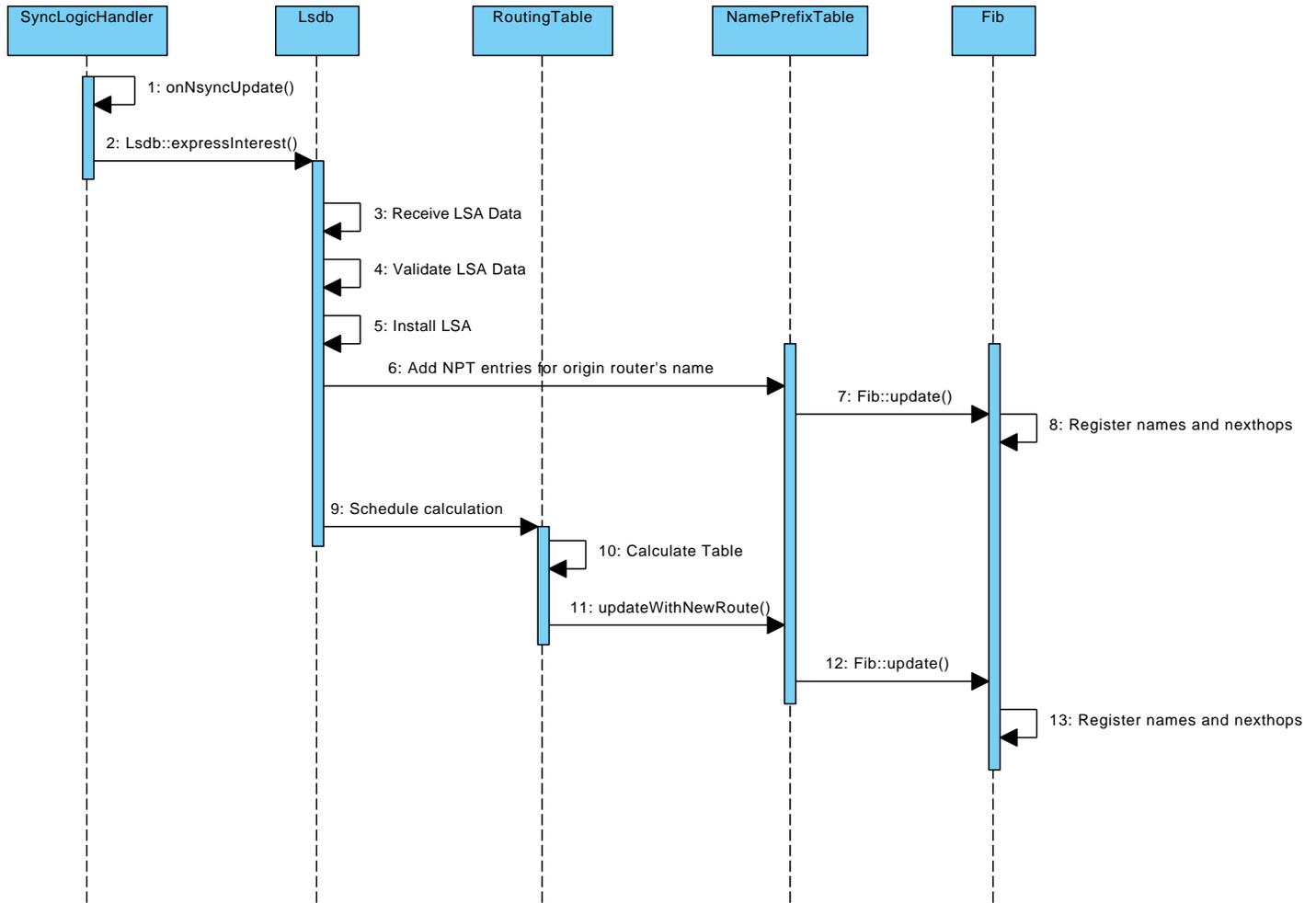
Figure 1: Simplified Diagram of the Actions of NLSR's Modules

1.2.3   Calculating the Routing Table and Populating NFD's FIB

When the routing table is calculated by the Routing Table module, the computed next hops are passed to the Name Prefix Table module. The Name Prefix Table module will then further pass the next hops to the FIB module to update NLSR's expected state of NFD's FIB. The FIB module will then perform the registrations or unregistrations with NFD's FIB.

A simplified diagram of NLSR's actions when receiving new routing information is shown in Figure 1. The remainder of this documentation will describe the purpose of and interaction between each module in more detail.
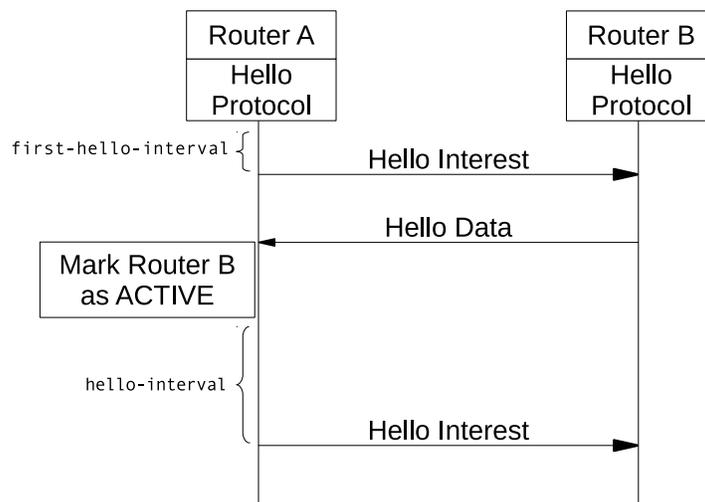
Figure 2: Router A determines the initial status of Router B

2 Hello Protocol

The Hello Protocol module periodically sends Hello Interests to learn the activity status of the router's neighbors. Hello Interests' names are constructed in the form: `/<neighbor's-router-prefix>/NLSR/INFO/<this-router's-prefix>`. If a neighbor responds to a Hello Interest, the neighbor is considered to be up and `ACTIVE`. A Hello Data's name is constructed using the following convention: `/<neighbor's-router-prefix>/NLSR/INFO/<this-router's-prefix>/<version>`. If a neighbor fails to respond to a configurable number of Hello Interests (`hello-retries`), the neighbors is considered to be down and `INACTIVE`. The Hello Protocol continues to send these periodic Hello Interests to each of its neighbors every `hello-interval` seconds. If the Hello Protocol detects a change in a neighbors status (i.e. a router that was previously `ACTIVE` is not responding to Hello Interests or a router that was previously `INACTIVE` responds to a Hello Interest), it will notify the LSDB (Section 5) to schedule a new Adjacency LSA build to include the updated neighbor information.

2.1 Determining Neighbor's Status

The Hello Protocol begins by scheduling Hello Interests to be sent to each neighbor of the router after `first-hello-interval` seconds. When the scheduled event is triggered, the Hello Protocol iterates through the list of neighbors first checking if there is already a Face to the neighbor. If there is a Face that has already been created, the Hello Protocol will construct and send a Hello Interest to the neighbor. If a Face has not been created for the neighbor, the Hello Protocol will attempt to create a Face to the neighbor and register the neighbor's router prefix. If the Face is created successfully, the Hello Protocol registers the Sync prefix, LSA prefix, and Key prefix using the Face ID returned by the Face creation command and sends out the Hello Interest. If the Face cannot be created, the Hello Protocol considers the failure as a Hello Interest timeout.

If the Hello Protocol receives Data in response to the Hello Interest, it will first verify that the Data is signed by the correct entity. If the Data is valid, the corresponding neighbor is set as `ACTIVE` and its timeout count is reset to zero. If the neighbor was previously `INACTIVE`, an Adjacency LSA build is scheduled to include the newly `ACTIVE` neighbor. If the Data is not valid, the packet is dropped.

If the Hello Interest sent to the neighbor times out, the corresponding neighbor's timed-out count is incremented. If the neighbor's timed-out count is less than `hello-retries` in the configuration file, the Hello Protocol will send another Hello Interest after `hello-timeout` seconds. If the neighbor's timed-out count equals the `hello-retries` value and the neighbor is currently marked as `ACTIVE`, the neighbor's status is set to `INACTIVE` and an Adjacency LSA build is scheduled.

2.2 Responding to Hello Interests

If the Hello Protocol receives a Hello Interest from another router, it will first verify that the Hello Interest came from one of its configured neighbors. If so, the Hello Protocol responds to the Interest with Hello Data. To optimize the time to respond to link recoveries, the Hello Protocol will then immediately send a Hello Interest to the neighbor if the neighbor is currently marked as `INACTIVE`.
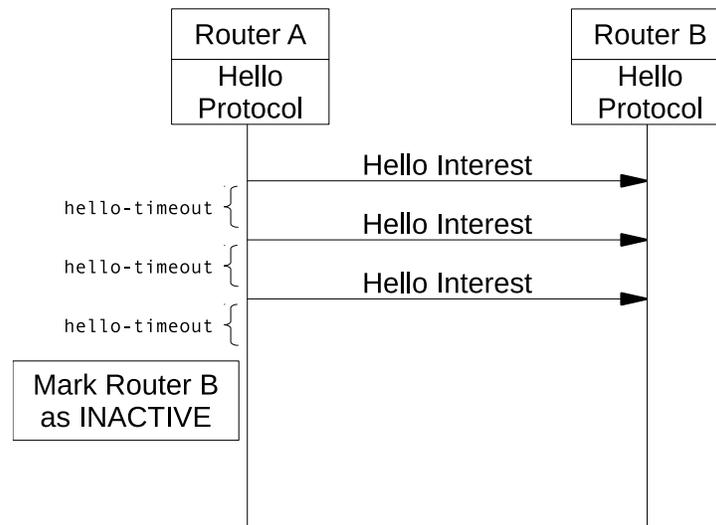
Figure 3: Router A determines that Router B has failed

2.3 Failure and Recovery Detection

The Hello Protocol will consider a neighbor as failed if the neighbor is currently `ACTIVE`, but Hello Interests sent to the neighbor have timed-out `hello-retries` number of times. A failure can also be detected if a FaceEventNotification is received with the information that a Face to the neighbor has been destroyed. The event is handled by the `Nlsr` class, but the triggered events simulate the actions of the Hello Protocol. If the neighbor was currently `ACTIVE`, the neighbor will be set to `INACTIVE`, the neighbor's timed-out count will be set to `hello-retries`, and an Adjacency LSA build will be scheduled.

The Hello Protocol will consider a neighbor as recovered if the neighbor is currently `INACTIVE`, but the Hello Protocol has received valid Data in response to a Hello Interest sent to the neighbor.

3   Sync Logic Handler

The Sync Logic Handler acts as the interface between the synchronization protocol and the NLSR application. The Sync Logic Handler receives notifications from the synchronization protocol when an sync update is detected, and the Sync Logic Handler then determines if the updated LSA should be retrieved. The Sync Logic Handler also notifies the synchronization protocol when the router's LSAs are modified or refreshed, so the synchronization protocol can update its internal digest and synchronize the updated information.

3.1   On Sync Update

When the synchronization protocol receives a sync update, the updated names and sequence numbers will be passed to `SyncLogicHandler::onNsyncUpdate()`. The Sync Logic Handler will process each updated name individually first by verifying that the update is not for one of the router's own LSAs. The Sync Logic Handler will then check the updated sequence number to see which LSAs were updated. If the sequence number for the LS in the update is greater than the sequence number of the existing LSA in the LSDB or there is no LSA for the updated Name, the Sync Logic Handler will use the `Lsdb::expressInterest()` interface to retrieve the LSA. The module will not try to retrive Coordinate LSA if link-state routing is on, or Adjacency LSA is hyperbolic routing is on. The logic used to express an LSA Interest and handle an LSA Data response is handled by the LSDB module (Section 5).

3.2   Publish Routing Update

When any of a router's LSAs are updated or refreshed by the LSDB, the LSDB will use the `SyncLogicHandler::publishRoutingUpdate()` interface to notify the synchronization protocol that the sequence number for the router's LSA prefix has changed. The Sync Logic Handler will also write the updated sequence number to file, so that a restarting router can begin publishing routing updates with sequence numbers larger than the sequence numbers it had published previously.

4   Link-State Advertisements

Link-State Advertisements (LSAs) represent pieces of routing information distributed by routers. NLSR uses three types of LSAs to distribute routing information: Adjacency LSAs which include neighboring node and link information, Coordinate LSAs which include a router's hyperbolic coordinates, and Name LSAs which include advertised name prefixes reachable through the router. All of the LSAs received by the router are maintained by the LSDB module (Section 5).

4.1   LSA Base Class

All three LSA implementations inherit from an LSA Base class, `Lsa`, which maintains information that is included in each LSA. The `LSA` class contains the following member variables:

- **Origin Router** - the router that advertised the LSA.

- **Sequence Number** - a number used to indicate the LSAs version as well as its ordering compared to other LSAs received from the same router.

- **Expiration Time Point** - a time point indicating when the LSA is no longer valid.

4.2   Adjacency LSAs

Adjacency LSAs maintain an `AdjacencyList` which contains all the currently `ACTIVE` neighbors of the origin router.

4.3   Coordinate LSAs

Coordinate LSAs maintains the hyperbolic angle and hyperbolic radius of the origin router.

4.4   Name LSAs

Name LSAs maintain a `NamePrefixList` which contains advertised name prefixes that are reachable through the origin router.
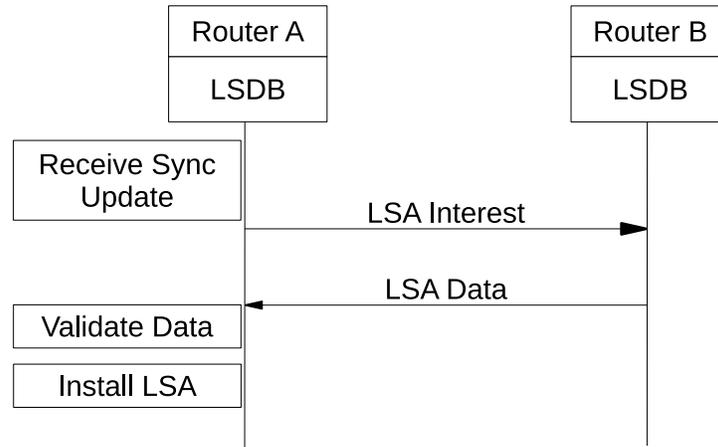
Figure 4: The general LSDB logic for each LSA type

5   Link-State Database

The Link-State Database (LSDB) holds LSA information distributed by other routers in the network. The LSDB stores all three types of LSAs and will trigger necessary events when a new LSA is added, when an LSA is updated, and when an LSA expires. The LSDB also handles LSA retrieval and validation, performs LSA builds, and triggers routing table calculations.

5.1   Retrieving an LSA

The LSDB provides `Lsdb::expressInterest()` as a public interface to retrieve an LSA from the network. If LSA Data is returned, the LSDB will handle the Data validation and perform the necessary LSDB modifications. If the LSA Interest times out, the LSDB will continue to re-attempt to fetch the LSA Data. The LSDB will stop trying these fetch re-attempts once it has tried for as long as the maximum configurable LSA refresh time.

5.2   Building LSAs

The LSDB provides three interfaces to build and install a router's own various LSAs.

- `Lsdb::scheduleAdjLsaBuild()` schedules an Adjacency LSA calculation if one is not already scheduled and if hyperbolic routing is not turned on, i.e. if link-state routing or dry run is on.

- `Lsdb::buildAndInstallOwnAdjLsa()` instantiates a new Adjacency LSA with the router's current adjacency list, incremented sequence number, and number of active neighbors. It also notifies the Sync Logic Handler to publish the routing update(Section 3.2) if link-state routing or dry-run is enabled to let other node in the network know the newly instlled Adjacency LSA.

- `Lsdb::buildAndInstallOwnCoordinateLsa()` instantiates a new Coordinate LSA with the router's hyperbolic coordinates and an incremented sequence number. The newly constructed Coordinate LSA is then installed in the LSDB. It also notifies the Sync Logic Handler to publish the routing update(Section 3.2) if hyperbolic routing or dry run is turned on to let other nodes in the network know the newly installed Coordinate LSA.

- `Lsdb::buildAndInstallOwnNameLsa()` instantiates a new Name LSA with the router's current name prefix list and an incremented sequence number. The newly constructed Name LSA is then installed in the LSDB.

5.3   LSA Installation and Updates

The LSDB provides three internal interfaces for each of the three types of LSAs to add or update LSAs in the LSDB. The general logic for each type of LSA installation is similar (Figure 5.3), but there are slight differences in the events that are triggered by an installation or update.

5.3.1   Lsdb::installAdjLsa()

When an Adjacency LSA is passed to `Lsdb::installAdjLsa()`, the LSDB first attempts to find the LSA in its current database.

   If the Adjacency LSA is not installed in the database, the LSA will first be added to the database. If the installed LSA is advertised by a remote router, the remote router's name will be added to the NPT and the LSA will be scheduled to expire after the expiration time point included in the LSA. Since a new Adjacency LSA introduces a new node and new links in the network, a routing table calculation is scheduled.

   If the Adjacency LSA is already installed in the database, the existing LSA will only be updated if the newly received LSA has a higher sequence number. If an Adjacency LSA with a higher sequence number is being installed, the existing LSA's sequence number and expiration time will be updated. If the newly received LSA contains an adjacency list that is different from the existing LSA, the existing LSA's adjacencies are updated and a routing table calculation is scheduled to account for the network change. Finally, if the updated LSA is advertised by a remote router, the LSA will be scheduled to expire after the expiration time point included in the LSA.

5.3.2   Lsdb::installCoordinateLsa()

When a Coordinate LSA is passed to `Lsdb::installCoordinateLsa()`, the LSDB first attempts to find the LSA in its current database.

   If the Coordinate LSA is not installed in the database, the LSA will first be added to the database. If the installed LSA is advertised by a remote router, the remote router's name will be added to the NPT, and the LSA will be scheduled to expire after the expiration time point included in the LSA. If hyperbolic routing is enabled, a new Coordinate LSA means that the routing table should be re-calculated to include the new destination router.

   If the Coordinate LSA is already installed in the database, the existing LSA will only be updated if the newly received LSA has a higher sequence number. If a Coordinate LSA with a higher sequence number is being installed, the existing LSA's sequence number and expiration time will be updated. If the newly received LSA contains hyperbolic coordinate different from the existing LSA, the existing LSA's coordinates are updated. If hyperbolic routing is enabled, an LSA with new coordinates means the routing table should be re-calculated to account for the different coordinates. Finally, If the updated LSA is advertised by a remote router, the LSA will be scheduled to expire after the expiration time point included in the LSA.

5.3.3   Lsdb::installNameLsa()

When a Name LSA is passed to `Lsdb::installNameLsa()`, the LSDB first attempts to find the LSA in its current database.

   If the Name LSA is not installed in the database, the LSA will first be added to the database. If the installed LSA is advertised by a remote router, the remote router's name and each advertised name in the LSA will be added to the NPT, and the LSA will be scheduled to expire after the expiration time point included in the LSA.

   If the Name LSA is already installed in the database, the existing LSA will only be updated if the newly received LSA has a higher sequence number. If a Name LSA with a higher sequence number is being installed, the existing LSA's sequence number and expiration time will be updated. A set difference is performed between the new LSA's advertised name prefix list and the existing LSA's advertised name prefix list to determine name prefixes that have been added. If there are added name prefixes, the name prefixes are added to the NPT and to the existing LSA. A set difference is then performed between the new LSA's advertised name prefix list and the existing LSA's advertised name prefix list to determine name prefixes that have been removed. If there are removed name prefixes, the name prefixes are removed from the NPT and from the existing LSA. Finally, If the updated LSA is advertised by a remote router, the LSA will be scheduled to expire after the expiration time point included in the LSA.

5.4   LSA Expiration

LSAs are scheduled to expire after a configurable amount of time in order to allow for LSDB cleanup when a router crashes. After the expiration period, if the LSA belongs to the current router, the LSA is refreshed with an incremented sequence number (Section 5.5). Otherwise, if the LSA belongs to a remote router, the LSA is removed from the LSDB. Removing an LSA from the LSDB triggers different events depending on the type of the LSA.

- **Adjacency LSA** - When an Adjacency LSA is removed, a routing table calculation is scheduled to determine new paths that don't include the associated router.

- **Coordinate LSA** - When a Coordinate LSA is removed and hyperbolic routing is enabled, a routing table calculation is scheduled.

- **Name LSA** - When a Name LSA is removed, the name prefixes advertised by the LSA are removed from the NPT.

In all three cases, the LSA's origin router's name will also be removed from the NPT.

5.5 LSA Refresh

When a router refreshes its own LSA, all three LSA types trigger the same events:

- Increment the LSA's sequence number by one

- Schedule the LSA to expire after the configured `lsa-refresh-time`.

- Publish a routing update in NSync to notify other routers of the change in the LSDB.

6 Routing Table

The Routing Table module performs three main tasks: it performs the routing table calculations using a `RoutingTableCalculator` (Section 6.1), it stores the calculated routing table entries in a table, and notifies the Name Prefix Table module (Section 7) when there are changes to the calculated next hops.

6.1 Routing Table Calculators

The `RoutingTableCalculator` is a base class provides functionality common to both link-state and hyperbolic routing. The Routing Table module uses the implementation class specific to the type of routing currently enabled.

6.1.1 Link-State Routing Table Calculator

The `LinkStateRoutingTableCalculator` class calculates the routing table uses Dijkstra's algorithm to calculate the shortest paths in the network. When `max-faces-per-prefix` is set to one, the `LinkStateRoutingTableCalculator` can simply run Dijkstra's algorithm. When `max-faces-per-prefix` is set to a value indicating multi-path calculation, the `LinkStateRoutingTableCalculator` will iteratively perform Dijkstra's using only a single neighbor link as a next hop. The calculation will be performed using each neighbor in order to learn the path costs for each destination through each next hop.

6.1.2 Hyperbolic Routing Table Calculator

The `HyperbolicRoutingCalculator` class calculates the routing table uses the Coordinate LSAs received from each router in the network to determine the cost from each of its neighbors to every other router in the network. The `HyperbolicRoutingCalculator` iterates through each of the router's neighbors calculating the hyperbolic distance from the neighbor to every other router in the network (excluding itself and the neighbor router). The `HyperbolicRoutingCalculator` then uses these calculated distances to add routing table entries to the destination with the neighbor as the next hop. The `HyperbolicRoutingCalculator` also adds a routing table entry to reach the neighbor itself; a routing table entry using the neighbor as a next hop to the neighbor with a cost of zero is added.

6.2 Notifications for Newly Calculated Next Hops

Once the Routing Table Module has finished calculating the routing table, it will notify the Name Prefix Table module using the `NamePrefixTable::updateWithNewRoute` interface. The Name Prefix Table Module will then update the next hops for each name prefix based on the newly calculated routing table. This process is described in more detail in Section 7.3.

7   Name Prefix Table
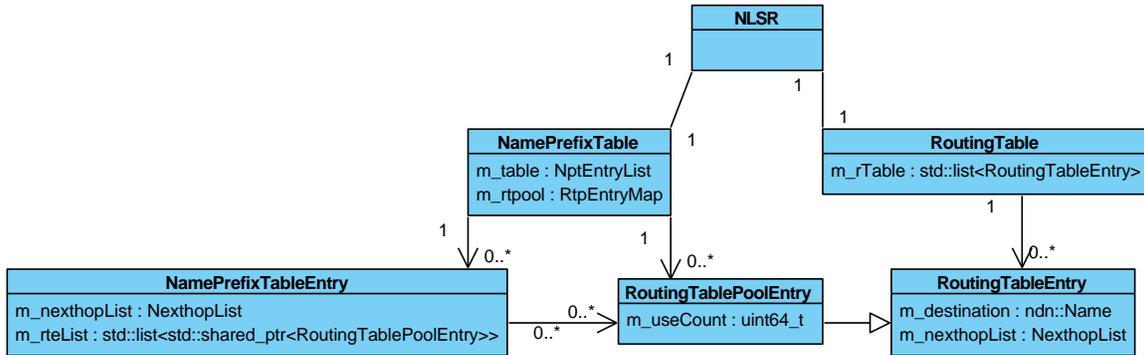
Figure 5: A
diagram of the NPT and Routing Table.
The "wire" arrows represent references (i.e. "x has y"), whereas the "solid" arrows represent inheritance (i.e. "y subclasses x").

The Name Prefix Table (NPT) is used by NLSR to maintain a list of all known name prefixes advertised by other routers, including router names. The NPT maintains a collection of NPT entries, where each entry represents a name prefix and all of its associated routing table entries. Additionally, to optimize the storage and association of the routing table entries, the NPT also maintains a collection of duplicated routing table entries, called routing table pool entries, which have an additional use count attribute. The NPT entries keep shared pointers to the appropriate routing table pool entries. If a name prefix is advertised by multiple routers, the name prefix will be represented by only one Name Prefix Table Entry, but will have multiple routing table pool entries which correspond with each origin router.

If a Name LSA exists with an advertised name prefix, that advertised name prefix must be represented in the NPT. Thus, if two routers advertise the same name prefix and one Name LSA expires, the NPT entry must not be removed due to the existence of the Name LSA from the other router.

If an any type of LSA for a remote router exists in the LSDB, the remote router's name prefix must be present in the NPT. An NPT Entry for a router name can be safely removed when there are no more LSAs in the LSDB from the origin router or there are no routing table entries for the origin router.

7.1   Adding an NPT Entry

The `NamePrefixTable::addEntry()` method is the public interface for name prefixes to be added to the NPT. The name prefix as well as the router's prefix which originates the name prefix are passed as parameters to the interface. The NPT will look through its collection of routing table pool entries first to see if it already has the appropriate route. If the NPT locates a local entry, it will increment that entry's use count. If none can be found, if will check in the Routing Table to see if a "raw" routing table entry matching the origin of the prefix is available there. If one is found, it will create a routing table pool entry with the same information, and a use count of one. This new routing table pool entry will then be used. If one is not found, a routing table pool entry with no routing information and a use count of one is made, and the NPT entry will retain this "dummy" routing information until the Routing Table performs a calculation and updates it.

The rest of the method determines if this NPT entry already exists in the NPT itself. If it has not been added, i.e. it is new, a new NPT entry will be made and installed in the NPT. Then the routing table pool entry with the routing information to the origin router's prefix will be added to this new NPT entry. If the entry already exists in the NPT, then the existing entry will be updated in the same way as the new one. If the updated NPT entry has next hops, the NPT will update the FIB with the prefix and its next hops. If there are no next hops, then this means that there is no way for the router to reach the origin. The FIB entry will be removed, but the NPT entry will be retained, as the routing table may calculate a new route.

7.2   Removing an NPT Entry

The `NamePrefixTable::removeEntry()` method is the public interface for name prefixes to be removed from the NPT. The name prefix as well as the router's prefix which originates the name prefix are passed as parameters to the interface. The NPT will use the origin router's name to look through its local collection of routing table pool entries. Because this is the NPT's centralized, total pool of routing information, if the entry does not exist here it cannot exist in any NPT entries. In

that case, the method ends. Else, the method will look through the collection of NPT entries for the entry matching the name prefix in the parameters. If it does not find it, there is nothing to be done, and the method ends. If it does find a matching entry, it will remove from that entry the routing table pool entry with the origin router prefix matching the one given in the parameters. The next hop information of the entry is rebuilt, and then the method checks if the entry has any next hops. If it does, the method will inform the FIB of the change with the name prefix and the next hops. If the entry no longer has any next hops, this means that there is no routing information to any router advertising that prefix. In this case, the NPT will remove that entry, as it is no longer useful.

7.3 Updating an NPT Entry with New Routing Table Entries

When the Routing Table module has finished calculating, it will notify the NPT using the `NamePrefixTable::updateWithNewRoute()` interface. The NPT will then iterate over each of its entries and each entry's routing table pool entry list. For each routing entry in the list, the NPT will attempt to fetch the new next hop list from the Routing Table, and set the entry's next hop list to that. If no list is available, it clears the routing table pool entry's next hop list. Then, `NamePrefixTable::addEntry()` is called with the current NPT entry and routing table pool entry pair.

7.4 Adding routing table pool entries to the pool

Whenever a routing table pool entry (RTPE) needs to be constructed, the method `NamePrefixTable::addRtpeToPool()` is called. It constructs and inserts into the hash map a pair object whose first element is the origin router prefix, and whose second element is a shared pointer to the actual routing table pool entry object. As stated in the `NamePrefixTable::addEntry()` method, the RTPE will have a use count of one.

7.5 Removing routing table pool entries from the pool

An RTPE will be deleted when, after removing an RTPE from an NPT entry, `NamePrefixTable::removeEntry()` determines that the just-removed RTPE now has a use count of 0. It will then call `NamePrefixTable::deleteRtpeFromPool()`, passing the RTPE as a parameter. The method will erase the element from the hash map that they are contained in.

8 FIB Interaction

The FIB module interacts directly with NFD to perform registrations and unregistrations of name prefixes. The FIB module is notified of additions, removals, or updates to the Name Prefix Table and will use the updated Name Prefix Table to perform the necessary registrations or unregistrations. The Name Prefix Table notifies the FIB module using the `Fib::update()` method which accepts a name prefix and next hops for that name prefix as parameters. The FIB module maintains a shadow FIB which represents its expectations of NFD's FIB. The FIB module uses the shadow FIB to determine which registrations and unregistrations are necessary.

8.1 Updating the FIB

When the Name Prefix Table performs an update on the FIB module, the FIB module will first sort the passed next hops with the next hop's costs in increasing order. The FIB module will next determine the number of next hops that should be installed for the name prefix using the `max-faces-per-prefix` parameter as a maximum. If there are more passed next hops than the `max-faces-per-prefix` parameter allows, the FIB module will only use the first `max-faces-per-prefix` number of next hops from the sorted list. If there are less passed next hops than the `max-faces-per-prefix` parameter, the FIB module will use all of the passed next hops.

   The FIB module next determines if there is already a FIB entry in the shadow FIB for the passed name prefix. If the name prefix will create a new FIB entry and the number of passed next hops is greater than zero, a new FIB entry will be created, the next hops will be registered for the name prefix in NFD's FIB, and the FIB entry will be set to expire in two times the `lsa-refresh-time` in order to clean up orphaned entries in NFD's FIB. If there is already an existing FIB entry for the name prefix and the number of passed next hops is greater than zero, any of the passed next hops that weren't previously registered for the FIB entry are registered. Then, any currently registered hops that are not in the passed next hops are removed from NFD's FIB, and the entry's expiration is refreshed. If there is already an existing FIB entry for the name prefix and the number of passed next hops is equal to zero, the routing table was unable to find a path to this name prefix and so the name prefix should be removed from NFD's FIB.
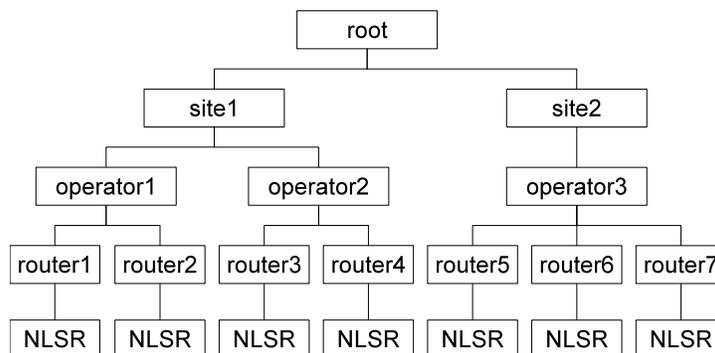
Figure 6: NLSR Trust Hierarchy

Table 1: 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>

9   Security

The trust model of NLSR is semi-hierarchical. An example certificate signing hierarchy is show in Figure 6. In this hierarchy, each entity's name and corresponding certificate name follow the convention described in Table 1.

9.1   Creating Keys and Certificates

The process to create keys and certificates for this hierarchy can be performed using the `ndnsec` [4] tools included with `ndn-cxx` [5]. The steps to create the keys and certificates is outlined below:

1. Create keys for Root:
   `ndnsec-keygen $ROOT_NAME > $ROOT_KEY_OUTPUT_FILE`

2. Create certificate for Root:
   `ndnsec-certgen -N $ROOT_NAME -p $ROOT_NAME $ROOT_KEY_OUTPUT_FILE > $ROOT_CERT_OUTPUT_FILE`

3. For each site, create keys and certificates:

   (a) On the Site machine, generate keys for the Site:
       `ndnsec-keygen $SITE_NAME > $SITE_KEY_OUTPUT_FILE`

   (b) Copy `$SITE_KEY_OUTPUT_FILE` over to the machine where the Root certificate was created.

   (c) Generate a certificate for the Site on the Root machine:
       `ndnsec-certgen -N $SITE_NAME -p $SITE_NAME -s $ROOT_NAME $SITE_KEY_OUTPUT_FILE > $SITE_CERT_OUTPUT_FILE`

   (d) Copy `$SITE_CERT_OUTPUT_FILE` over to the Site machine.

   (e) Install the certificate on the Site machine:
       `ndnsec-cert-install -f $SITE_CERT_OUTPUT_FILE`

   (f) On the Site machine, create the Operator keys:
       `ndnsec-keygen $OP_NAME > $OP_KEY_OUTPUT_FILE`

   (g) On the Site machine, create the Operator certificate:
       `ndnsec-certgen -N $OP_NAME -p $OP_NAME -s $SITE_NAME $OP_KEY_OUTPUT_FILE > $OP_CERT_OUTPUT_FILE`

   (h) On the Site machine, create the Router keys:
       `ndnsec-keygen $ROUTER_NAME > $ROUTER_KEY_OUTPUT_FILE`

   (i) On the Site machine, create the Router certificate:
       `ndnsec-certgen -N $ROUTER_NAME -p $ROUTER_NAME -s $OP_NAME $ROUTER_KEY_OUTPUT_FILE > $ROUTER_CERT_OUTPUT_FILE`

4. When NLSR starts, it will automatically create the NLSR keys and certificates for the router.

9.2   Certificate Publishing

In a network, every router should have the root certificate configured as a trust anchor for the `validator` in the configuration file. For each site, at least one router should publish the site certificate, and at least one router should publish the certificate of the site operator. Each router should publish its own certificate. All this information must be explicitly specified in the configuration file.

For example, the following configuration file indicates that NLSR should publish the site certificate and the router certificate:

```
...
security
{
  validator
  {
    ...
  }
  cert-to-publish "$SITE_CERT_OUTPUT_FILE"   ; name of the site certificate file
  cert-to-publish "$SITE_CERT_OUTPUT_FILE"   ; name of the router certificate file
  ...
}
```

10   Configuration File

NLSR's configuration file contains numerous parameters to control the behavior and performance of NLSR. The configuration file also includes the trust schema used by NLSR to verify LSA Data, Hello Data, and prefix update command Interests. The configuration file is divided into six sections each with parameters that mainly affect a specific module.

10.1   General Section

The `general` section in the configuration file includes parameters which deal with the general setup of the router, the behavior of the LSDB, and logging configuration.

There are three parameters used to configure the router prefix of the router. The router prefix is the name that other routers in the network know this router by.

- `network` - the name of the network to which the router belongs; e.g., `/ndn`.

- `site` - the name of the site to which the router belongs; e.g., `/edu/memphis`.

- `router` - the name to identify the router; e.g., `/%C1.Router/cs/pollux`.

The router prefix is constructed by combining the three parameters following the format: `/<network>/<site>/<router>`.

There are three parameters which affect the behavior of the LSDB.

- `lsa-refresh-time` - the time in seconds the router will wait before refreshing its LSAs (Default value: 1800; Valid values: 240 - 7200).

- `router-dead-interval` - the time in seconds after which an inactive router's LSAs are removed. The configured value for this parameter must be greater than `lsa-refresh-time`. (Default value: two times the value configured in `lsa-refresh-time`).

- `lsa-interest-lifetime` - the interest lifetime used for LSA Interests (Default value: 4; Valid values: 1 - 60).

The `log-level` parameter configures the verbosity of NLSR's logging. The possible `log-level` values are listed in increasing verbosity:

- `NONE` - no messages

- `ERROR` - error messages

- `WARN` - warning messages

- `INFO` - informational messages (default)

- `DEBUG` - debugging messages

- `TRACE` - trace messages (most verbose)

- `ALL` - all messages

Note that all debugging levels listed above the selected value will also be enabled.

The `general` configuration section also includes parameters to choose where the NLSR log file and the NLSR sequence number file are stored. The `log-dir` parameter is an absolute path to the directory where the NLSR log file should be written, and `seq-dir` is an absolute path to the directory where the NLSR sequence number should be written.

10.2   Neighbors Section

The `neighbors` section in the configuration file contains parameters that define the behavior of the Hello Protocol and the neighboring routers of the router.

- `hello-retries` - the number of times to retry a Hello Interest before deciding the neighbor is down (Default value: 3; Valid values: 1 - 10).

- `hello-timeout` - the interest lifetime for Hello Interests in seconds (Default value: 1, Valid values: 1 - 15).

- `hello-interval` - the time in seconds between sending each Hello Interest to a neighbor. (Default value: 60; Valid values: 30 - 90).

- `first-hello-interval` - the time to wait in seconds before sending the first Hello Interests (Default value: 10; Valid values: 0 - 10).

- `adj-lsa-build-interval` - when the Hello Protocol triggers an Adjacency LSA build, the LSDB will wait this amount of time in seconds before performing the Adjacency LSA build. This parameter is intended to allow for Adjacency LSA build requests to be aggregated and the build can then be performed once. (Default value: 5; Valid values: 0 - 5)

The `neighbors` section also includes multiple `neighbor` subsections, each of which configures a neighbor of the router. The `neighbor` subsection includes:

- `name` - the router prefix of the neighboring router

- `face-uri` - the face that should be used to connect to the neighboring router

- `link-cost` - the cost metric for the link connecting this router to the neighbor router.

10.3   Hyperbolic Section

The `hyperbolic` section in the configuration file is used to enable/disable hyperbolic routing and to specify the hyperbolic coordinates of the router.

The `state` parameter indicates whether or not hyperbolic routing should be enabled. There are three possible values for this parameter: `on`, `off`, and `dry-run`. `on` enables hyperbolic routing; `off` disables hyperbolic routing (link-state routing is used); `dry-run` uses link-state routing to populate NFD's FIB, but will also perform the hyperbolic routing calculations and write them to the log file for debugging purposes.

The `radius` parameter defines the router's radius in the hyperbolic coordinate system and `angle` defines the router's angle in the hyperbolic coordinate system.

10.4   FIB Section

The `fib` section in the configuration file contains two parameters: one to limit the number of next hops registered for each name prefix, and the amount of time to wait before calculating the routing table after a request is made.

`max-faces-per-prefix` defines the maximum number of next hops that can be registered for a name prefix. This value is intended to reduce the FIB size for routers with a large number of neighbors. The default value for `max-faces-per-prefix` is `0` which indicates that all available next hops may be added to each name prefix. `max-faces-per-prefix` allows values between 0 and 60.

`routing-calc-interval` is the time to wait in seconds after a routing table calculation is requested before actually performing the routing table calculation. This parameter is intended to limit the number of routing table calculations, which may be performance intensive on some systems. The default value for `max-faces-per-prefix` is 15 seconds and can be configured to be in the range of 0 to 15 seconds.

10.5   Advertising Section

The `advertising` section includes a list of name prefixes that the router should advertise as reachable through itself. Each name prefix that should be advertised should be in the following format: `prefix /name/to/advertise`. This section allows for static configuration of the advertised prefixes, but prefixes can be dynamically advertised and withdrawn using the Prefix Update Processor.

10.6   Security Section

The `security` section of the configuration file includes the configuration for NLSR's validators and the locations of certificates that should be published by the router.

The `validator` subsection includes the configuration for the validator used by NLSR to verify the signatures of Hello Data and LSA Data.

The `prefix-update-validator` configures the validator used by the Prefix Update Processor to verify that prefix update command Interests are signed by the operator of the router.

The `security` section also allows configuration of which certificates should be published by the router using the `cert-to-publish` keyword. If the router should publish a certificate, the absolute path for the certificate file can be configured as `cert-to-publish` value.

References

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

[2] NDN Project Team, "NFD - NDN forwarding daemon," http://named-data.net/doc/nfd/.

[3] V. Lehman, A. M. Hoque, Y. Yu, L. Wang, B. Zhang, and L. Zhang, "A secure link state routing protocol for NDN."

[4] NDN Project Team, "ndnsec," https://github.com/named-data/ndn-cxx/tree/master/tools/ndnsec.

[5] ——, "ndn-cxx," http://named-data.net/doc/ndn-cxx/.