**1. Introduction**

Networking paradigm is shifting from communication between hosts to communication for data. Newly proposed architecture is evolving for communication for name data call Named Data Networking (NDN). We will contribute to evolve NDN by designing a dynamic routing protocol for NDN.

Existing communication in Internet is between hosts and routing decisions are made on looking up IP address. Routing in NDN will be based on name prefixes of content/data. NDN router has to calculate routing table for name prefixes and generate Forwarding Information Base (FIB) for routing Named Data in the Internet. Dynamic routing protocol for NDN will calculate routing based on name prefixes and update FIB with the topology change.

NDN router will collect information from its neighbors about any topology update it receives from other neighbors. Thus router will build topology information and calculate routing table based on network topology. After routing table calculation it will updates the FIB for routable name prefixes.

**2. Motivation**

Van-etal in [1], proposed Link-state Intra-domain Routing technique and also discussed about using both IS-IS and OSPF for nodes to discover and describe their local connectivity and to establish adjacencies. The paper also discussed about construction of FIB. The proposed idea is built on the top of IP networks.

Today’s network is mainly based on IP where in routing is done based on the location of the content and not the name of the content. Routing the user requests based on content naming makes it interesting as it is predicted to eliminate a lot of overhead issues faced in the present IP networks and the related security issues. NDN right now is in evolving state and researcher over the world is conducting research how NDN can be deployed at large scale. For large scale deployment NDN needs a routing protocol for constructing routing table and forwarding data to shortest possible paths, which is built solely on top of NDN. No routing protocol on top of NDN has been developed yet to do routing based on name prefixes. Our project contribution will be designing a routing protocol for NDN.

To implement dynamic routing capability in NDN, we design a link state protocol built on the concept of OSPF to distribute name prefixes and calculate routes to name prefixes. This document describes our design (Sections 1- 4) and implementation (section 5).

**3. Approach**

We are adapting the idea of IP OSPF protocol and develop it over NDN. We call our proposed solution Named-data Link State Routing (NLSR). NDN router will advertise their adjacency and name prefix of the content in the network. Neighbors hear this advertisement and build network topology from the adjacency advertisement and calculate the routing table from the topology. When routing table calculation is complete and router has the information of name prefixes from name prefix advertisement. From this information Forwarding Information Base of NDN router will be formulated. CCND then will route interest based on the FIB entries.

In our design when NLSR boots up read configuration file for router-id, neighbors, name to be advertised, and other configuration parameter from the router. NLSR connects to local CCND and tells CCND to forward every interest he can serves. From the configuration and info interest reply NLSR build two kind of LSAs: Name and Adjacency and advertise them to neighbors NLSR. When updating process is done NLSR calculate routing table by building network topology from the adjacency LSAs and running Dijsktra’s shortest path algorithm. Now router has routing table and name information in Name LSAs. By combining this information NLSR builds routing FIB table and also update CCND with the FIB entries generated by router. Details of the process in described in details design section.

**4. Design Details:**

Design is described below as part by part

**4.1 Configuration Commands**

Each NDN router requires a configuration file for initial configuration. The configuration file contains the router ID, neighbor’s id and link costs of the router’s NDN neighbor nodes as well as the node’s content to be advertised to the network. The commands used in the configuration file are shown below:

|  |  |
| --- | --- |
| ccnneighbor */router/name/ faceX* | |
| /router/name | Variable length char value |
| faceX | Connecting Face for neighbors. X is of integer Value |

|  |  |
| --- | --- |
| router-name */router/id* | |
| /router/id | Variable length char value |

|  |  |
| --- | --- |
| ccnname */name/prefix* | |
| name\_prefix | Name prefix in format determined by name\_type |

|  |  |
| --- | --- |
| lsdb-synch-interval *secs* | |
| secs | Integer Value. Default value is 300. It’s the amount of time interval a neighbor can request LSDB |

|  |  |
| --- | --- |
| Interest-retry *num* | |
| num | Integer Value. Default value is 4. It’s the number of retry in case interest timed out. |

|  |  |
| --- | --- |
| Interest-resend-time *secs* | |
| secs | Integer Value. Default value is 15. It’s the amount of time after which an interest will timed out. |

|  |  |
| --- | --- |
| lsa-refresh-time *secs* | |
| secs | Integer Value. Default value is 1800. It’s the amount of time interval a NLSR will refresh own LSA. |

|  |  |
| --- | --- |
| router-dead-interval *secs* | |
| secs | Integer Value. Default value is 3600. It’s the amount of time interval after which a neighbor will be considered dead if NLSR does not hear anything from it during that period |

|  |  |
| --- | --- |
| multi-path-face-num *num* | |
| num | Integer Value. Default value is 0. It’s the number of face will be added by NLSR |

|  |  |
| --- | --- |
| logdir *path/to/log/dir* | |
| path/to/log/dir | Character value. Path where NLSR will write log file |

**Figure 1: Configuration Command Format**

**4.2 Message format:**

For Link State advertisement NDN OSPF router will use two type of LSA: Name LSA and Adjacency LSA. Below is the header format for both the LSAs:

|  |  |
| --- | --- |
| Adjacency LSA Header | |
| Origination router | /router/name in name prefix format for the origin router. Variable length |
| Origin Router Length | Integer Value. Length of Origination Router |
| LS Type | Two kinds of LSA Type 8 bit unsigned integer value |
| Origination Time | LSA Origination timestamp in microseconds format from epoch time. Used for differentiating the LSAs. |

**Figure 2: Adjacency LSA Header**

|  |  |
| --- | --- |
| Name LSA Header | |
| Origination router | /router/name in name prefix format for the origin router. Variable length |
| Origin Router Length | Integer Value. Length of Origination Router |
| LS Type | Two kinds of LSA Type 8 bit unsigned integer value |
| LS Sequence Number | Used to differentiate between continuing LSAs from same router. |
| Origination Time | LSA Origination timestamp in microseconds format from epoch time. Used for differentiating the LSAs. |
| LS Id | Integer Value. LS Identifier for Name LSA |
| isValid | Value 0/1. 1 indicates the LSA is valid and should be installed in LSDB. 0 indicates the LSA is not valid anymore and should be deleted from LSDB if exists |

**Figure 3: Name LSA Header**

|  |  |
| --- | --- |
| Adjacency LSA Body | |
| Number of Neighbors (Unsigned Integer 4 bytes) | |
| Neighbor 1 router name (Variable length /name/prefix) | Neighbor 2 Connecting face (Unsigned Integer 4 bytes) |
| Neighbor 2 router name | Neighbor 2 connecting face |
| … |  |
| Neighbor n router name | Neighbor n connecting face |

**Figure 4: Adjacency LSA Body**

|  |  |
| --- | --- |
| Name LSA Body | |
| Name prefix (Variable length /name/prefix) | Name length (Length of name prefix in unsigned 4 bytes) |

**Figure 5: Name LSA Body**

**4.3 Booting Up Router:**

Router is started/restarted its boots up and read configuration file. By reading the configuration file it set the router-name, builds Adjacency List (ADL), Name prefix List (NPL), and sets other configuration parameter configured. NLSR connects to CCND, register names router/name/nlsr to CCND telling that, if any interest comes to CCND for name router/name/nlsr then forward it to me.

**4.4 Updating Adjacency List**

NLSR sends i*nfo* interest to all the neighbors from ADL. When sending interest, lifetime of the interest is set to “*interest-resend-time”*.If NLSR hears reply for the *info* from neighbors the status of the neighbor is updated to “Active” from “down” state. If *info* interest is timed out, NLSR will try “*interest-resend”* times. For any neighbor, if *info* interest is timed out “*interest-resend”* times, then status remains unchanged for that neighbor. Neighbor’s NLSR hearing “*info”* interest reply with content containing information of it’s LSDB version and info version.

**4.5 LSA Origination:**

Two kinds of LSA origination is carried out by router. Name LSA origination and Adjacency LSA origination. Router reads the name prefixes from list, builds Name LSA and installs it to own LSDB. Adjacency LSA is built by including the active neighbors from the neighbors list by checking the status of the neighbors. If there is any change in neighbor list status then router builds Adjacency LSA by including all the active neighbors and installs the LSA in own LSDB.

**4.6 LSDB Synchronization:**

LSDB synchronization is done in four steps: i. Sending LSDB interest ii. Sending LSDB Summary by Neighbors iii. Sending LSA interest, iv. Sending LSA by neighbors, and v. LSA installation. Each steps is described below in next five subsections. LSDB synchronization is done by NLSR periodically with neighbors.

**4.6.1 Sending LSDB interest**

In the first step of LSDB synchronization, NLSR sends “*lsdb”* interest on name prefix “*neighbor/router/name/nlsr/lsdb”* for all the active neighbors in ADL. When sending “*lsdb”* interest, NLSR also include the last version of LSDB it received from neighbors in exclusion filter. In reply it hears from neighbors then perform the work described in subsection 4.6.3. But if NLSR does not hear any reply from any neighbors, it will try sending “*lsdb”* interest “*interest-resend”* times. If “*lsdb”* interest for any neighbor is timed out for “*interest-resend”* times, then that neighbor is considered down, and NLSR will update its ADL accordingly and will also schedule building of Adjacency LSA.

**4.6.2 Sending LSDB Summary by Neighbors**

Neighbor’s NLSR hearing interest for “*lsdb”* will check the version number. If the version number in exclusion filter is older that the version number of LSDB, then NLSR prepares “LSDB Summary Content” with all the header information of all LSA and reply back to neighbors. On the other hand, if the version in exclusion filter is not older than the version number of LSDB, then NLSR reply with NACK content.

**4.6.3 Sending LSA interest**

In this step, if NLSR get NACK reply content from neighbors then does nothing as it is already synchronized with that neighbor. But if NLSR get “LSDB Summary Content” from neighbors, then for every LSA header in LSDB Summary Content, first it checks own LSDB for existence. Secondly if the LSA does not exist in LSDB then sends “*lsa”* interest to neighbors.

**4.6.4 Sending LSA by neighbors**

Neighbor’s NLSR hearing interest for “*lsa”*, will check its LSDB with header information provided in interest name, prepare a content with LSA information and reply back with the content.

**4.6.5 LSA installation**

NLSR receiving LSA content from neighbor will install it into LSDB. Installation process checks if the LSA is new/newer. If LSA is new then it is added into LSDB. If LSA is newer and its LS Type is Adjacency then delete old LSA and add the new one. But if LSA is newer and Name LSA then checks for isValid field of Name LSA header information. If isValid field is set then delete old LSA and install new one. And if isValid field is not set, then delete the old LSA and discard new LSA. Regardless of LS Type is any LSA is found older, it is discarded.

**4.8 Calculate Routing Table:**

**4.9 Construct FIB entry from Routing Table and Name LSAs**

**4.10 Link Failure Handling**

**XXX. References**

1. "Networking Named Content", V. Jacobson and D. K. Smetters and J. D. Thornton and M. F. Plass and N. H. Briggs and R. L. Braynard.

2. “ACT: Audio Conferencing Tool in NDN”, Zhenkai Zhu, Sen Wang, Xu Yang, Lixia Zhang, Van Jacobson