Revised name prefix table
Redmine issue #2863

Nicholas Gordon

August 8, 2016

Abstract

This document describes the design of and motivation for the change to the name prefix table module of NLSR.

1 Introduction

The name prefix table (NPT) in NLSR (Named-data Link-state routing) is responsible for managing the name prefixes served by various routers throughout the network. Including code mergers that are projected to complete before this patch, these table entries as of August 8, 2016 are objects with the following properties:

- An ndn::Name attribute which contains the name prefix being served.
- A sorted set of Nexthops that are the most efficient way for the current router to reach that name prefix from its position in the topology.
- A list of RoutingTableEntry objects that represent the routers that serve that prefix. These RoutingTableEntry objects have the following properties:
  - An ndn::Name attribute which contains the name prefix of the router.
  - A list of Nexthops that represent the most efficient way for the current router to reach that router from its position in the topology. An empty set indicates an unreachable prefix.

Note that those RoutingTableEntries are not references; they are full copies of the object from the routing table. This is the main motivation for this improvement. It is typical in an NDN network that the number of advertised name prefixes will be vastly greater than the number of physical routers. While inefficient in any case, as the number of prefixes grows relative to the number of physical routers, this inefficiency becomes more significant, and represents a major memory problem for very large topologies.

2 Solution

The solution that was implemented made the following structural changes:

- A hash map attribute was added to the NPT that maps router prefixes to a shared pointer (shared_ptr) to the routing information for that router.
- A RoutingTablePoolEntry object was defined, which is a simple extension of the RoutingTableEntry object. The only addition was a uint64_t attribute to keep track of the number of NPT entries using it.
- The NPT entry list attribute was changed to be a list of shared pointers to RoutingTablePoolEntry objects.
- Finally, several overloaded `<<` and `==` operators were written as infrastructure.

Naturally some behavioral changes were necessary to accomodate the structural ones. They were:

- The two addEntry methods in name-prefix-table.cpp were merged; the functionality did not seem logically distinct enough to warrant two functions.
- When a destination is added to a prefix, the NPT will first look through its hash map to locate the needed information. This has multiple advantages:
  - This reduces the dependence of the NPT on the routing table, isolating the module from a cascading failure.
  - It simplifies reasoning about the NPT and its entries, because all the needed information is contained within the class.
  - Finally, it may improve performance, as hash maps have very fast access times compared to lists which must be iterated over.
- The single addEntry method logic is now like this:
  1. The local RoutingTablePoolEntry (RTPE) pool is searched for an entry matching the name given for the destination parameter.
  2. If a match is not found, a new RTPE is made, and an attempt to find a corresponding "raw" RoutingTableEntry is made. The information from a match is copied into the new RTPE.
  3. If no matching RoutingTableEntry is found, the RTPE is instantiated with no next hops.
  4. The RTPE is added to the local pool.
  5. The NPT entry table is searched to find a matching one based on the one specified by the prefix parameter.
  6. If one is not found, a new one is instantiated.

    7. The RTPE is added to the NPT entry, and the entry's next hop list is calculated. The RTPE's useCount is incremented.
    8. The forwarding information base (FIB) is notified of the change.

- Two methods were written for adding and removing RTPEs from the pool. They are passed the shared_ptr directly. This is less efficient than passing the pointer by reference, as the called method becomes an owner of the object, but the overhead is small and not expected to be an issue.

In addition to those specified changes, this change decouples the modules somewhat and makes the system less interdependent; the NPT could now be tested without a routing table by seeding the pool with the necessary entries. In addition, this isolation means it will be possible to make updating the entries more efficient, owing to the centrally-located storage location.

The API was also improved, in that the only interfaces the NPT has with the LSDB are at addEntry and removeEntry, which can be thought of as black boxes that change the state of the NPT, and its only interface with the routing table is at updateWithNewRoute, which as mentioned above can be reworked into a (better) black box.