Async-strategy Implementation Introduction

NFD Development

Ju Pan

University of Arizona

July 9, 2019

Contents

Definition

Definition

What is async-strategy?

1. It's an optional forwarding behaviour that the existing forwarding strategies can take advantage of. It's neither a standalone strategy nor a building block. (Reasons are in the next slide)
2. Async strategy is responsible for transmitting pending Interests upon FIB nexthop creation.

Definition / Why not a strategy or a building block?

1. There is no logic other than "retain PIT entry", making a "building block" in a separate class doesn't make sense.
2. Creating a separate strategy doesn't make sense either. Because after FIB nexthop is in place, the required forwarding logic for async-strategy is almost the same as forwarding a new Interest and that differs per strategy. Duplicating an existing strategy increases the code complexity and harms the maintainability.

Behaviour

Behaviour

1. When an Interest arrives but there's no matching FIB nexthop, the PIT entry is still retained.
2. When a new FIB nexthop is inserted, forwarding plane enumerates a portion of the PIT covered by the FIB entry and triggers the strategy. This requires forwarding plane changes.
3. The strategy may forward the Interest to the new FIB nexthop.

Implementation Details

Step 1

Step 1: When an Interest arrives but theres no matching FIB nexthop, the PIT entry is still retained.

1. In *Fib* class, we add an *afterNewNextHop* signal.
2. Create new triggers in *Strategy* base class: *supportNextHop()* and *afterNewNextHop()*
3. Enable user to decide whether to activate "retaining PIT entry" feature by using strategy parameter. (#3868).
4. Add new block in *BestRouteStrategy2::afterReceiveInterest* to check if support "retaining PIT", if so, set the PIT entry expiry timer to the Interest lifetime.

Step 2

Step 2: When a new FIB nexthop is inserted, forwarding planeenumerates a portion of the PIT covered by the FIB entry andtriggers the strategy. This requires forwarding plane changes.

1. In *forwarder* class, we add a *triggerStrategyAfterNewNextHop()* function. It handles the partial enumeration of the affected NameTree entries. *triggerStrategyAfterNewNextHop()* is connected to *afterNewNextHop* signal.
2. For each affected NameTree entry, we lookup strategy choice table to determine the effective strategy for *nte.getName()*, then trigger strategy on PIT entry.

Step 3

Step 3: The strategy may forward the Interest to the new FIB nexthop.

1. In *afterNewNextHop()* trigger, we forward the Interest to nexthops.

Concerns and Solutions

Overhead in Partial Enumeration (1/4)

1. The *NameTree::partialEnumerate* funciton takes *EntrySubTreeSelector* as one of two parameters. *EntrySubTreeSelector* is a function which returns a *<bool, bool>* pair.
   - The first bool indicates whether entry should be accepted;
   - The second bool indicates whether entry's children should be visited.
2. If the current entry's subtree doesn't have any entry support async-strategy behaviour, we can simply set the second bool value to *false* so that the partial enumeration won't visit the subtree at all. (Proposed data structure change is in the next slide).

Overhead in Partial Enumeration (2/4)

Proposing a *`NameTreeEntry`* modification to solve the overhead:
Adding an integer field *`numOfSubtreeSupportAsync`* for *`NameTreeEntry`* class to indicate how many subtrees (root from current node's children) support async-strategy behaviour.

Init: the *`numOfSubtreeSupportAsync`* is set to *`0`* by default.

`numOfSubtreeSupportAsync` is greater than 0 means there are nodes in the subtrees that support async-strategy. So in partial enumeration, we should visit the subtree.

Overhead in Partial Enumeration (3/4)

Case 1: when a new entry is inserted into the NameTree

- If the strategy choice supports async-strategy:
  1. Increase its parent's *numOfSubtreeSupportAsync* by *1*.
  2. If the parent *numOfSubtreeSupportAsync* is set from 0 to 1 (meaning this subtree starts to support async-strategy), we go to parent's parent and repeat the algorithm all the way to the root entry if necessary. Otherwise, stop here.
- If the strategy choice doesn't support async-strategy, we do nothing.

Overhead in Partial Enumeration (4/4)

Case2: when an existing entry's strategy choice is changed.

- SC is changed from supporting async-strategy to not supporting async-strategy:
  1. If *numOfSubtreeSupportAsync* is 0, then go to its parent and decrease *numOfSubtreeSupportAsync* by 1
  2. If parent's *numOfSubtreeSupportAsync* becomes 0 and it doesn't support async-strategy, go to parent's parent and repeat the algorithm.
- SC is changed from not supporting async-strategy to supporting async-strategy:
  1. If its *numOfSubtreeSupportAsync* is 0, repeat the first bullet point in case 1;
  2. Otherwise, do nothing.

Asynchronous Enumeration or Bounded Enumeration

TBD

Paced/Bounded Outgoing Interests to Prevent Congestion

TBD