Design Summary of Remote Prefix Registration

Yanbiao Li

1. MODULE OVERVIEW

The Remote Registrator is managed by the RIB Manager to process remote prefix registration, which enables local applications to register their prefixes to the gateway router (remote hub) to receive interests on their data.

Figure 1 describs the class diagram of the `RemoteRegistrator` moudle. It keeps the refference to the NFD controller, the key-chain and the RIB, enabling easy communication with them. Besides, there are two connections connect `registerPrefix` and `unregisterPrefix` to `Rib::afterInsertEntry` and `Rib::afterEraseEntry` respectively. Some parameters, that could be shared by all commands, are stored in `m_controlParameters` and `m_commandOptions`. The `m_refreshInterval` defines the interval between refreshing and is configurable. At last, `RemoteRegistrator` employs a list of `RemoteRegisteredEntry` to maintain all registered entries.

Figure 2 presnets the overview sequence diagram of remote prefix registration (the process of remote unregistration is similar), the `RemoteRegistrator` works in close cooperation with `Rib`, `RibManageer` and `Controller`. When the RIB Manager loads configurations, the `rib.remote-register` section is loaded by the Remote Registrator. It will then be enabled if proper parameters are loaded. Whenever the RIB is updated by inserting a new entry (or erasing an existing entry), and there is a active connectivity to the gateway router, remote registration (or remote unregistration) will be triggered. And the `RemoteRegistrator::m_nfdController` will generate and express the command interests with proper parameters.

More specifically, on one hand, whenever the Remote Registrator is being enabled, `registerPrefix` and `unregisterPrefix` will subscribe two signals `Rib:afterInsertEntry` and `Rib:afterEraseEntry` respectively, which will be emitted after a new entry is inserted into the RIB and an existing entry is erased from the RIB respectively. On the other hand, when the local NFD gets connectivity to the remote NFD, a route to the gateway router is registered to the local RIB with the `link local nfd prefix (/localhop/nfd)`. Thus, once such route exists, the remote registration commands, in format of `/localhop/nfd/rib/[register|unregister]/...`, can be forwarded to the gateway router.

2. IMPLEMENTATION DETAILS

2.0.1 Remotely Registered Entries

On the gateway router, each remote registration is kept as a ‘soft state‘. While one the end host, `RemoteRegistrator::m_regEntries` maintains all registered entreis, and the registering prefix is used as the key to retrive each entry. A `RemoteRegisteredEntry` consists of its signing identity, a refresh event, a retry event and the registration status. There are four available registration status.

- `RemoteRegistrationStatus::NONE` is the init state.
- `RemoteRegistrationStatus::IN_PROGRESS` indicates that this entry is being processed.
- `RemoteRegistrationStatus::SUCCESSFUL` means this entry has been successfully registered to the remote RIB.
- `RemoteRegistrationStatus::FAILED` denotes that this entry has failed in registration.

For registration, if a prefix has been successfully registered remotely, an event is scheduled to refresh this registation periodically. And the interval before refreshing is configurable. By contrast, if the registration falis, another event is scheduled to retry this registration, based on the exponential back off strategy. Besides, even if there is no connectivity, satisfied entries (should be registered but can not now) are also maintained as potential registrations for later use. While for unregistration, no matter whether the remote operation succeeds or not, and even if there is no connectivity, unnecessary entries are removed from the registered list immediately.

Eventually, in addition to preventing redundant registrations and unnecessary unregistrations, such a registered list can also be used to deal with connectivity changes. Here, connectivity change means that current

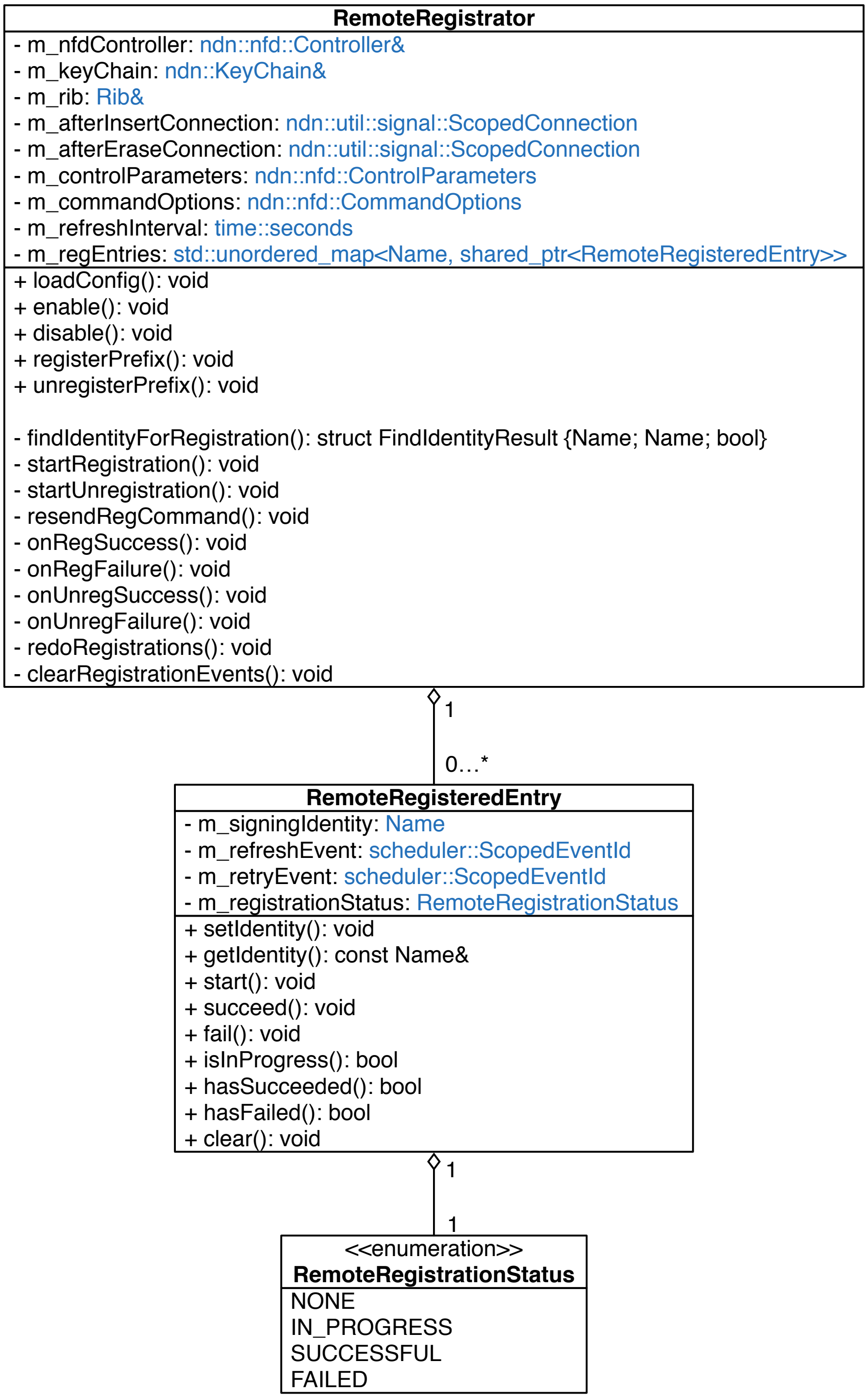
RemoteRegistrator
- m_nfdController: ndn::nfd::Controller&
- m_keyChain: ndn::KeyChain&
- m_rib: Rib&
- m_afterInsertConnection: ndn::util::signal::ScopedConnection
- m_afterEraseConnection: ndn::util::signal::ScopedConnection
- m_controlParameters: ndn::nfd::ControlParameters
- m_commandOptions: ndn::nfd::CommandOptions
- m_refreshInterval: time::seconds
- m_regEntries: std::unordered_map<Name, shared_ptr<RemoteRegisteredEntry>>
+ loadConfig(): void
+ enable(): void
+ disable(): void
+ registerPrefix(): void
+ unregisterPrefix(): void
- findIdentityForRegistration(): struct FindIdentityResult {Name; Name; bool}
- startRegistration(): void
- startUnregistration(): void
- resendRegCommand(): void
- onRegSuccess(): void
- onRegFailure(): void
- onUnregSuccess(): void
- onUnregFailure(): void
- redoRegistrations(): void
- clearRegistrationEvents(): void
1
0…*
RemoteRegisteredEntry
- m_signingIdentity: Name
- m_refreshEvent: scheduler::ScopedEventId
- m_retryEvent: scheduler::ScopedEventId
- m_registrationStatus: RemoteRegistrationStatus
+ setIdentity(): void
+ getIdentity(): const Name&
+ start(): void
+ succeed(): void
+ fail(): void
+ isInProgress(): bool
+ hasSucceeded(): bool
+ hasFailed(): bool
+ clear(): void
1
1
<<enumeration>>
RemoteRegistrationStatus
NONE
IN_PROGRESS
SUCCESSFUL
FAILED

Figure 1: Class diagram of Remote Registrator

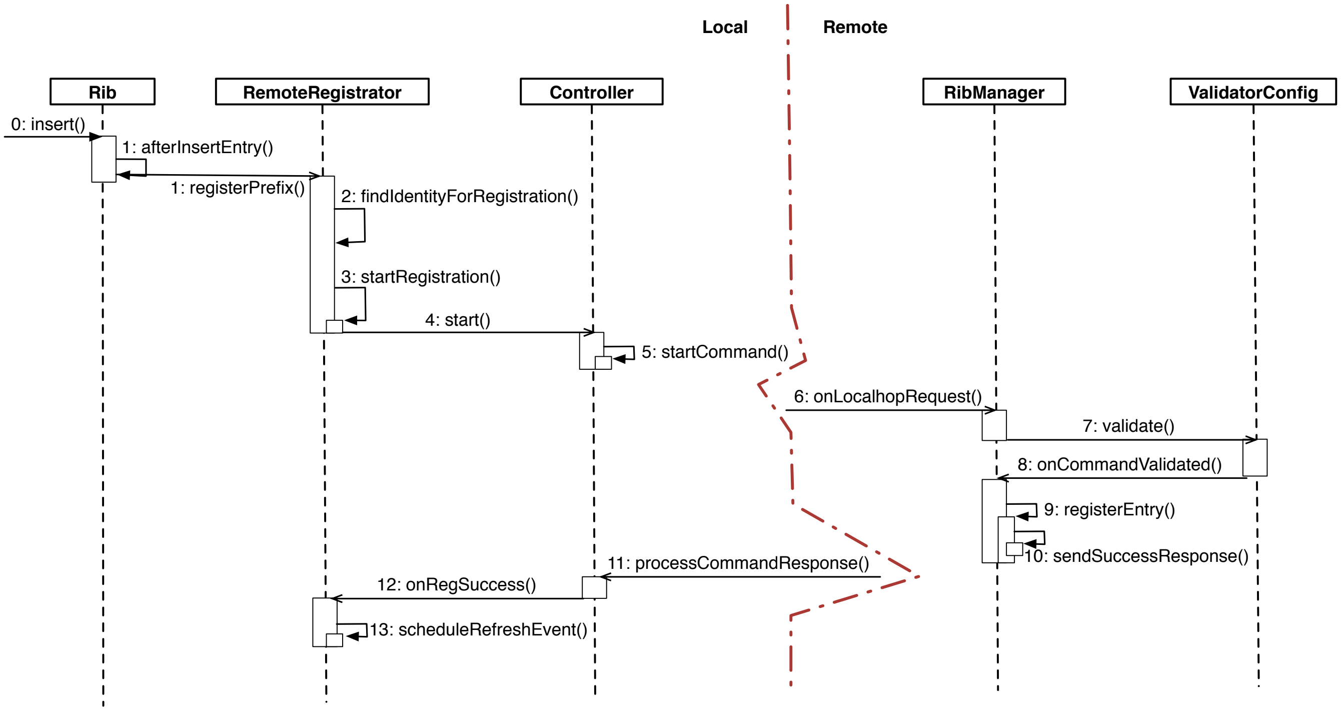

Figure 2: Sequecnece diagram of remote prefix registration

connectivity is switched to another hub or that current connectivity gets lost and recovered afterwards. No matter in which case, there are two types of changes: current connectivity is lost and a connectivity is established. Once current connectivity is lost (i.e., `/localhop/nfd` is erased from the local RIB), the associated events of all entries in the registered list are canceled. While when a connectivity is established (i.e., `/localhop/nfd is inserted into the local RIB`), all maintained entries are then registered to the connected hub.

2.1 Find identity for registration

To reduce the size of remote RIB and the cost of processing remote registrations, the remotely registered prefixes should be aggregated whenever is feasible. Actually, local NFD owns a key-chain consisting of a set of identities, each of which defines a namespace and can cover one or more locally registered prefixes. Given a locally registered prefix, the shortest identity satisfies it (here satisfying it menas being a prefix of it) is selected as the remotely registering prefix and is also used to sign the registration command. Besides, whenever the security check is enabled in `rib.localhop-security` section, a remote registration command can not pass the validation unless its signing identity can be verified by the trust anchor of the gateway router.

2.2 Remote Registration

As described in figure 3, not all locally registered prefix that triggers `registerPrefix` will lead to remote registration finally. The prefix scoped for local use (i.e. starts by */localhost*) will terminates the remote registration immediately. While the prefix indicates a newly connected hub (i.e. $/localhop/nfd$) will lead to redoing registrations to the newly connected hub instead of remotely registering that hub prefix.

Given a prefix considered for remote registration, we first find out, through **findIdentityForRegistration**, the exact prefix for remote registration as well as the identity that will sign registration command. If there is an entry corresponding to the registering prefix in the registered list, retrieve that entry as the processing entry. Otherwise, create a new entry as the processing entry with the singing identity and insert it into the registered list. If the processing entry has failed or being processed, nothing else need to do. Otherwise, the registering prefix of this entry should be remotely registered if there is a active connection to some hub.

If the registration succeeds, the processing entry is flagged as SUCCESSFULL and an even is scheduled to periodically refresh this registration. However, if this entry has already been erased from the registered list (due to locally unregistration), the remotely registered prefix should be unregistered.

If the registration fails but there are still remaning retries, just retry this registration immediately. Otherwise, flag the processing entry as FAILED and then schedule an event to periodically retry this registration if it still exists in the registered list.

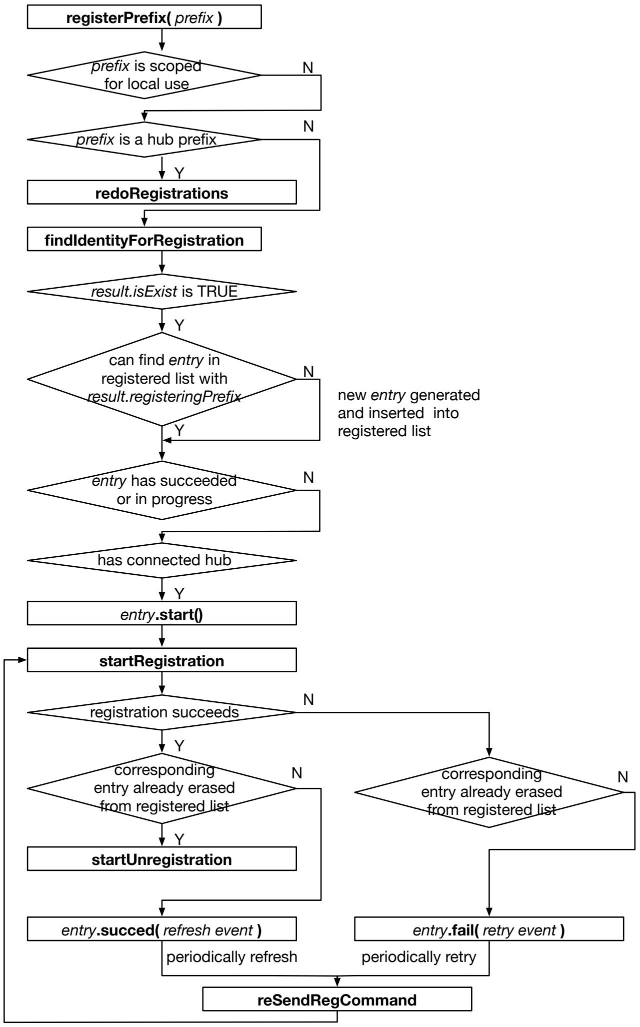
registerPrefix(prefix)
prefix is scoped for local use
N
prefix is a hub prefix
N
Y
redoRegistrations
findIdentityForRegistration
result.isExist is TRUE
Y
can find entry in registered list with result.registeringPrefix
N
new entry generated and inserted into registered list
Y
entry has succeeded or in progress
N
has connected hub
Y
entry.start()
startRegistration
registration succeeds
N
Y
corresponding entry already erased from registered list
N
Y
startUnregistration
corresponding entry already erased from registered list
N
entry.succed(refresh event)
entry.fail(retry event)
periodically refresh
periodically retry
reSendRegCommand

Figure 3: Work flow of registration.

2.3 Remote Unregistration

Figure 4 presents the detail of **unregisterPrefix**. Unregistration of local-use-only prefixes are also not considered for any remote operations. While unregistration of the hub prefix (i.e. $/localhop/nfd$) indicates that all connected hubs get disconnected. In this case, all events for registration, including both refresh events and retry events, should be canceled to avoid useless registration requests.

Then, if a locally unregistered prefix does require remote unregistration, the first step is also to find out the corresponding remotely registering prefix and the signing identity through **findIdentityForRegistration**. Only if that identity exists and there is an entry (referred to as processing entry) in the registered list corresponding to the remotely registering prefix, can the remote unregistration be promoted.

However, if there is another locally registered prefix can be represented by the processing entry, this entry must be kept without any remote operations. Otherwise, erase this entry from the registered list so that the related event (refresh event or retry event). Thus, even without successful remote unregistration, the corresponding registered entry on remote hub may expire after some time.

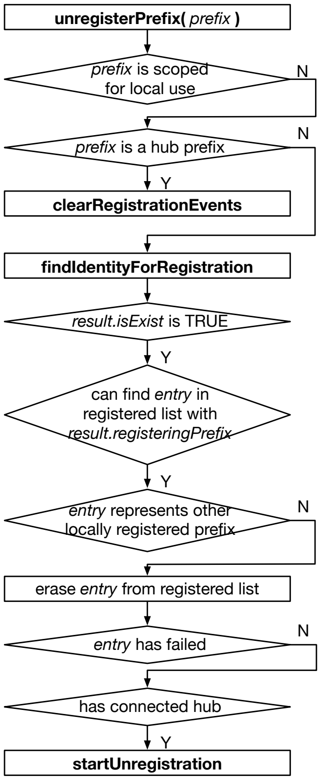
unregisterPrefix(prefix)
prefix is scoped for local use
N
prefix is a hub prefix
N
Y
clearRegistrationEvents
findIdentityForRegistration
result.isExist is TRUE
Y
can find entry in registered list with result.registeringPrefix
Y
entry represents other locally registered prefix
N
erase entry from registered list
entry has failed
N
has connected hub
Y
startUnregistration

Figure 4: Work flow of unregistration.

Then, if the processing has already been flagged as FAILED, nothing need to do. Otherwise, start unregistration if there is a connected hub. If the remote unregistration fails, immediately retry this unregistration with remaning retries decreased by one, until there is no remaning retries.

2.4 Resend Registration Command

No matter for refreshing registrations, retrying registrations or redoing registrations, **reSendRegCommand** of *Remote Registrator* plays the most important role. As presented in figure 5, processing the requestes for resending registration command should be very careful. If the processing entry has already been erased from the registered list (due to local unregistration) and this resending request is just for retry, the registering prefix must be unregistered remotely then. Even the processing entry still exists, we should also check the signing identity carefully. If that identity also exists, just start remote registration. Otherwise, we must erase the processing entry (as the signing identity is no longer avail-

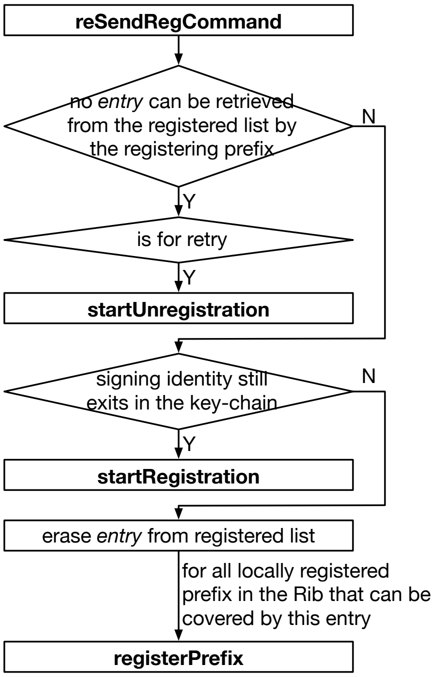
reSendRegCommand
no *entry* can be retrieved from the registered list by the registering prefix
N
Y
is for retry
Y
startUnregistration
signing identity still exits in the key-chain
N
Y
startRegistration
erase *entry* from registered list
for all locally registered prefix in the Rib that can be covered by this entry
registerPrefix

Figure 5: Work flow of resend registration command.

able), find out all locally registered prefix in the RIB that can be covered by this entry, and re call **registerPrefix** for those prefixes.