Remote Register Entry Machine

Yanbiao Li

Figure 1 presents the complete state machine of the remote registered entry[1]. It describes all possible states during the whole processes of remote registration/unregistration. But for implementation, it can be simplified.

Firstly, all unregister-related states can be eliminated. As depicted, any unregister-related state is driven by the 'rib erase' event or switched from another unregister-related state. Thus, let's see what we could do when an entry is erased from the RIB[2]. If the corresponding registered entry exists, there are 5 cases according to its status:

- `NEW`: In this case, there is no connectivity to the hub. We could delete this entry from the registered list only.
- `REGISTERED` or `REGISTER_FAIL`: In these two cases, we have already get the response from the remote hub. No matter the registration succeeds on the remote hub or not, we could delete this entry from the registered list and then send out unregistration command if there is a connectivity (whether this is required is discussed later).
- `REGISTERING` or `REGISTER_WAIT`: In these two cases, we also delete this entry and then do remote unregistration when feasible. Then, there may be two remote operations working at the same period. If the current unregistration is processed after the previous registration, the situation gets back to the above point. Otherwise, the current unregistration will fail because trying to unregister an non-existing entry. If the previous registration finally fails, the 'unregistering entry' will not exists. However, if the previous registration succeeds at last, it seems to be a problem. But, actually, this entry has already been deleted from the registered list, once we get response of registration of an non-existing entry, we will start unregistration instead of scheduling refresh event or retry event. Thus, it may not be a really problem.

In a word, no matter in which case, once a 'rib erase' event happens, we just delete the corresponding entry from the registered list and do unregistration if there is a connectivity. Will this lead to some potential problem? The answer is NO! Assume we have deleted an entry due to some 'rib erase' event (this means there may not be another 'rib erase' event toward this entry unless it is inserted again.). There may be 5 events related to this entry will trigger further state transition after this 'rib erase' event:

- `rib insert`. **Expect:** Inserting an unregistered entry is the same as inserting a new entry. **Actual:** As the inserting entry can not found in the registered list, it will be processed as a new entry.
- `hub connect`. **Expect:** Unregistered entry should not be re-registered. **Actual:** Only the entry in the registered list will be re-registered.
- `hub disconnect`. **Expect:** Nothing need to do with unregistered entry. **Actual:** Only the events associated with all entries in the registered list will be canceled.
- `register succeed`. This means unregistration of this entry is processed before previous registration. **Expect:** unregistered this entry remotely. **Actual:** remote unregistration is started in response to successful of remote registration of an entry that does not exist in the registered list.
- `register fail`. This also means unregistration of this entry is processed before previous registration. **Expect:** unregistered this entry remotely (the failure may resulted by time out only.). **Actual:** remote unregistration is started in response to failure of remote registration of an entry that does not exist in the registered list.

[1] draw based on http://redmine.named-data.net/attachments/download/315/remoteregister-statemachine_20150430.png

[2] Assume that there is the only rib entry corresponding to a registered entry. Otherwise, if there are two or more, nothing need to do with the registered list unless all of them are erased.

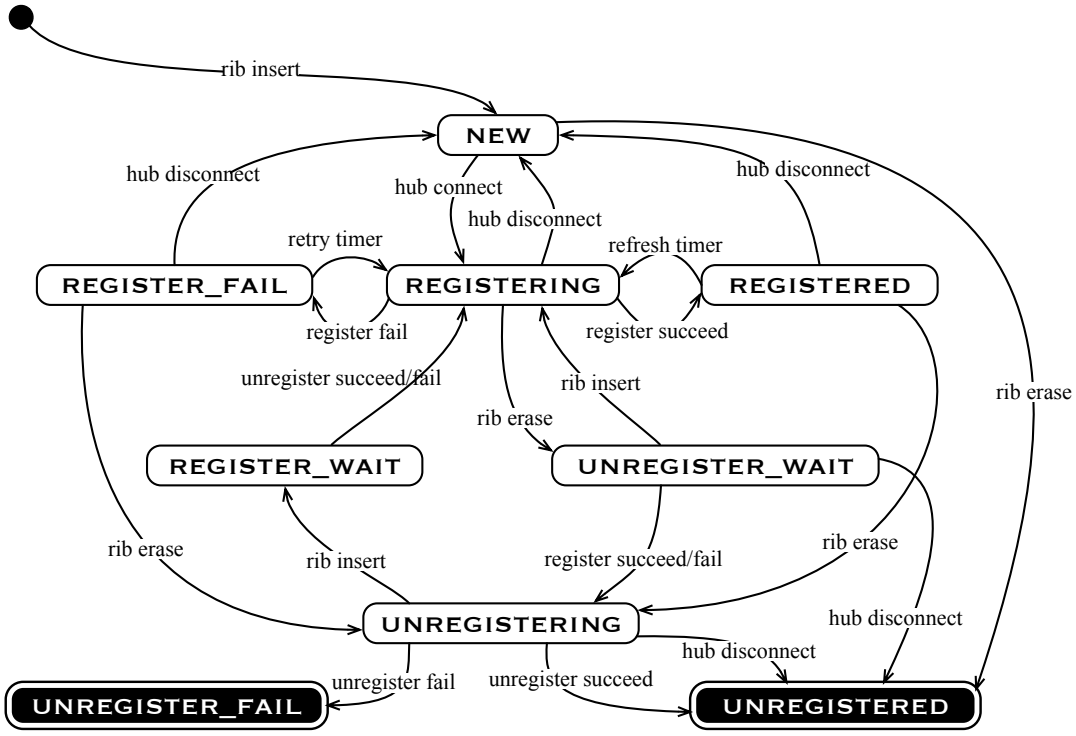

Figure 1: Complete state machine.

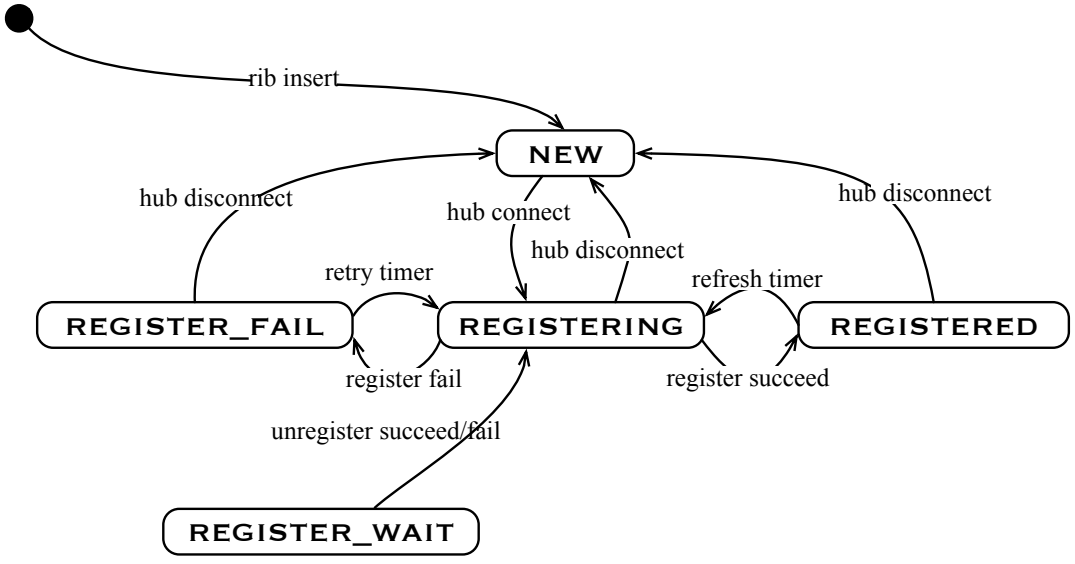

Figure 2: Eliminate unregister-related states.

- `unregister succeed/fail`. **Expect:** Nothing need to do (this is discussed later). **Actual:** Do nothing.

Accordingly, in response to a 'rib erase' event, we can just delete the corresponding entry from the registered list and do remote unregistration if feasible. In another word, there is no other state transitions after this point because the target entry has been removed. Thus, as shown in figure 2, we can eliminate all unregister-related states in figure 1.

Secondly, we can merge `REGISTERING` and `REGISTER_WAIT` into one state. As shown in figure 3, once the remote hub receives the registration command and starts processing it, the status of this entry transits from `REGISTERING` to `REGISTER_WAIT`. This is an inevitable transition and must happen on the remote NFD. While the local NFD will not know exactly when this transition happens. Thus, it's hard for the local NFD to distinguish these two states.

Besides, as shown in figure 2, `REGISTER_WAIT` only connectes to `REGISTERING` directly and the driven event are 'unregister succeed/fail'. As the unregistering entry has already been deleted from the registered list before the remote unregistration succeeds or fails, it will not affect later operation on other entries any more. If the unregistration succeeds finally, nothing need to do. Even if it fails, we can also do nothing because the remotely registered entry on the remote hub will expire automatically without refreshing. Thus, we should do nothing when the unregistration succeeds/fails, and can merge `REGISTERING` and `REGISTER_WAIT` into one state `REGISTERING` (as shown in figure 4), which starts from sending out registration command and ends by receiving command response.

At last, we add a virtual state `RELEASED` as the final state, the transition to which is triggered by the 'rib erase' event (as shown in figure 5). While for any registered entry, there are only four states should be implemented: `NEW`, `REGISTERING`, `REGISTERED` and `REGISTER_FAIL`.

The rest issue is whether we should do remote unregistration when the 'rib erase' event happens in the case that the previous registration of this entry has failed. If we start remote unregistration in this case, the advantage is we can avoid receiving some unexpected interests in a given period probably (the previous registration may succeed, but it may expire automatically without refreshing). But the disadvantage is we should definitely spend extra cost on generating, signing and expressing the unregistration command. So I prefer to do nothing in this case.

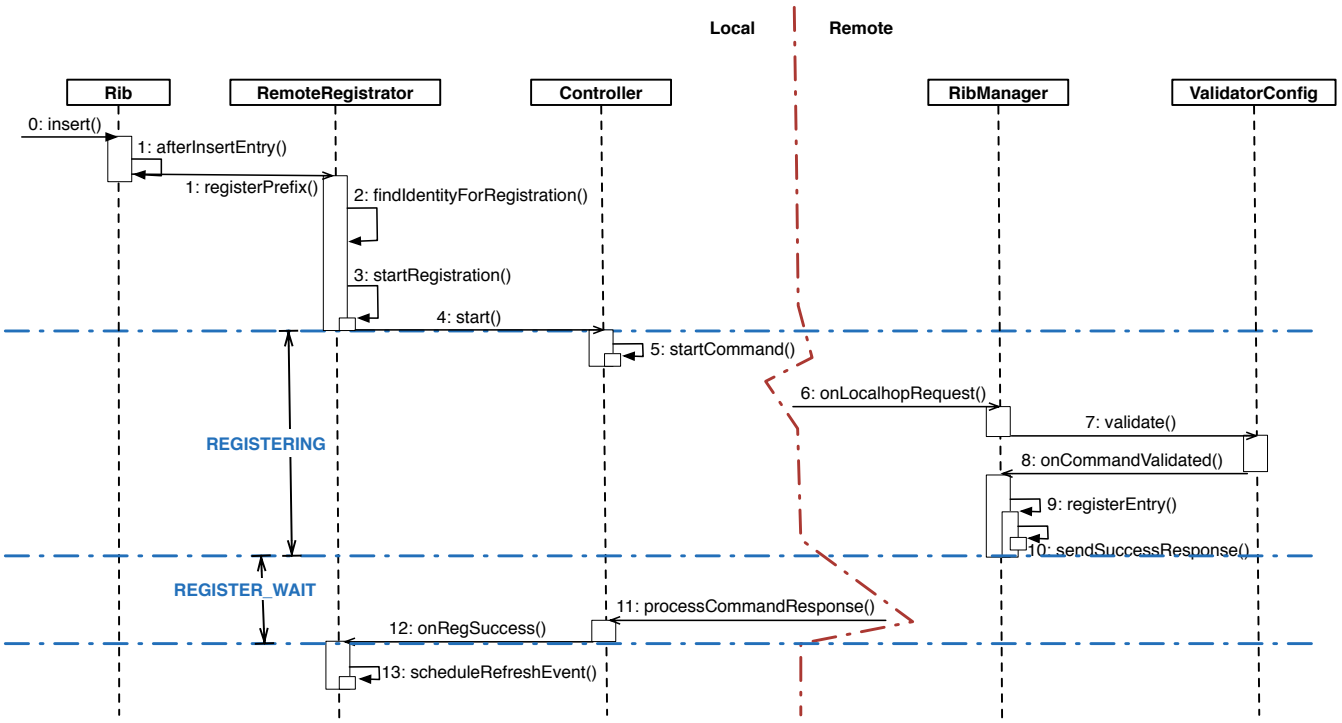

Figure 3: Sequence diagram of remote prefix registration.

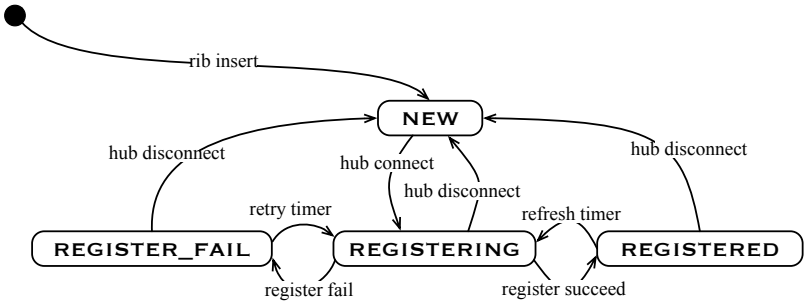

Figure 4: Eliminate the state of REGISTER_WAIT.

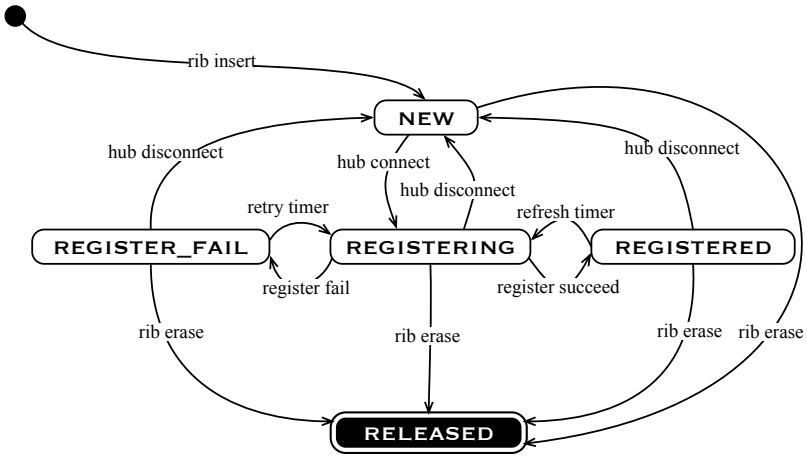

Figure 5: The final state machine.