NDN, Technical Report NDN-0045. http://named-data.net/techreports.html
Revision 1: March 18, 2018

NDN Automatic Prefix Propagation

Yanbiao Li[1], Alexander Afanasyev[2], Junxiao Shi[3], Haitao Zhang[1], Zhiyi Zhang[1], Tianxiang Li[1], Edward Lu[1], Beichuan Zhang[3], Lan Wang[4], and Lixia Zhang[1]

[1]University of California, Los Angeles, [lybmath,haitao,zhiyi,tianxiang,edward,lixia]@cs.ucla.edu
[2]Florida International University, aa@cs.fiu.edu
[3]The University of Arizona, [shijunxiao,bzhang]@email.arizona.edu
[4]The University of Memphis, lanwang@memphis.edu

ABSTRACT

In an NDN network, when a producer application wants to publish data, it registers the data's name prefix P with the local NDN forwarding Daemon (NFD) on the same host machine; this registration informs the local NFD where to forward Interests whose name falls under P. To propagate the reachability to P beyond the local NFD requires additional mechanisms, such as running routing protocol. This paper describes *automatic prefix propagation* protocol, an alternative to running a full-featured routing protocol on host machines connected to the NDN network via one or multiple NDN gateway routers. The automatic prefix propagation protocol uses the local configuration of NDN certificates and the inferred presence of gateway router(s) to automatically send remote registration request when necessary.

1. INTRODUCTION

Producer applications in the Named Data Networking (NDN) architecture [10] express the intent to make data available for retrieval by registering data's prefixes with the NDN forwarder(s) [3]. In the current management NDN API, this intent takes the form of a command Interest [1] that is directed towards the routing information base (RIB) manager running in the instance of the NDN forwarder on the same machine as the producer (*local forwarder*) [3]. If the request is authorized, the RIB manager creates corresponding entries in the forwarding information base (FIB) in the local forwarder, allowing it to properly direct Interests with the registered prefix towards the producer application, if Interests cannot be satisfied by other means.

Directing Interests from the remote NDN forwarders is a more complex issue, which is being addressed in a number of ways: manual configuration, dynamic name announcement protocols, and opportunistic data discovery strategies. Different methods have different trade-off between the implementation complexity, usage complexity, and potential overhead in terms of unnecessary forwarded Interests. Manually configuring remote forwarders has trivial implementation cost and, when properly used, can result in minimal overhead. However, dealing with usage complexity is tedious and complicated, unless a forwarding strategy relies on a predefined naming convention to forward Interests in a greedy fashion (e.g., using geo or hyperbolic coordinates embedded in the data names [4,6]). The dynamic name announcements, e.g., using a routing protocol such as NLSR [4], minimizes the usage complexity and overhead, but has significant implementation and deployment costs. Opportunistic data discovery, e.g., using the AccessRouter strategy, is simple to use but has relatively high implementation complexity and network overhead.

This paper describes the *automatic prefix propagation* protocol, which minimizes the complexity and overheads in environments with one or more remote NDN forwarders acting as NDN gateways.[1] The protocol automatically triggers the remote prefix registration when an application registers a prefix locally, provided that (1) there is an active remote NDN gateway, and (2) the forwarder possesses a private key and the corresponding NDN certificate that matches the locally registered namespace. The first condition is satisfied when the "`/localhop/nfd`" prefix is present in the local RIB, e.g., configured by NDN auto-configuration [2]. The second requirement is fulfilled when the user requests and installs an NDN certificate that covers the desired namespace in the local forwarder. Coverage of the NDN certificate is determined using a basic trust schema [9], described in Section 3. Note that depending on which namespace the user-installed NDN certificate corresponds to, the remotely registered prefix can be shorter than the one registered by the application. This allows the forwarder to aggregate multiple local registrations into one remote action, reducing communication and book-

[1]The current version of the protocol assumes a single NDN gateway. In the future, the protocol will be extended to multi-gateway environments.

keeping overheads.

The rest of this report is organized as follows. Section 2 demonstrates our motivation by an example. Section 3 reviews key design issues and proposes our solutions. At last section 5 concludes this report .

2. USE CASE EXAMPLE AND SECURITY ASSUMPTIONS

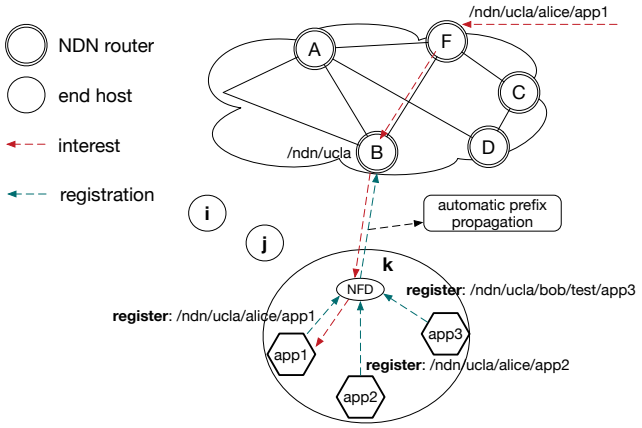

Figure 1: Use case example of prefix propagation

Figure 1 demonstrates how an *Interest* packet, i.e., "`/ndn/ucla/alice/app1`" in this example, gets forwarded to the target producer application, say the *app1* runs on the host *k*.

Whenever NFD on the host *k*, receives prefix registration from a local app, it automatically creates a "remote prefix registration" command directed toward the gateway router *B*. Thus, when receiving the *Interest* "`/ndn/ucla/alice/app1`", the gateway router *B* will know where to forward such an Interest when data is not otherwise available.

2.1 Applications Delegate Prefix Registration Key to Local NFD

In this paper, we assume that local NFD will be given the delegated keys by local applications and that these keys are sufficient to generate valid signatures for prefix registration. The key can either be a higher-level namespace key, e.g., "`/ndn/ucla/alice`"'s key for prefix "`/ndn/ucla/alice/app1`" to be registered, or a specialized key for RIB operation only, e.g., a key with name "`/ndn/ucla/alice/nrd`" can only be used to register/unregister prefix "`/ucla/alice`".

2.2 Hierarchical Trust Model of Remote NFD

NDN builds security into its architecture, and supports different trust models, by using semantically meaningful name-based configurations. In our work, we assume that the remote NFD adopts a hierarchical trust model, i.e., remote NFD accepts a registration command for name "P1" only if the command is signed by an identity whose name "P2" is the prefix of "P1". For example, only if the registration command for "`/ndn/ucla/alice`" is signed by "`/ndn/ucla/alice`", "`/ndn/ucla`", "`/ndn`" or "/", can the remote NFD accept it.

A special case is that when a specialized key for registration is used, the remote NFD should also accept the command. For example, a registration command for "`/ndn/ucla/alice`" that is signed by "`/ndn/ucla/alice/nrd`".

2.3 Certificate Availability for Remote NFD

Another assumption is the certificate availability for remote NFD: when verifying signatures created by the local NFD, all nodes along the certificate chain are accessible to the remote NFD. For example, if a signature is created by the local NFD using the identity "`/ndn/ucla/alice`", and the remote NFD's trust anchor is "`/ndn`", we assume that the remote NFD can fetch certificates for "`/ndn/ucla/alice`", "`/ndn/ucla`" and "`/ndn`".

3. DESIGN ISSUES AND SOLUTIONS

Essentially, prefix propagation is the host registers a prefix, representing reachability to some of its local applications, to the gateway router. In this section, we focus on a series of key design issues and introduce our considerations on tradeoffs and final solutions.

3.1 End Host Makes Propagations

Whenever required and feasible, the end host will make registrations for propagating reachability of its local applications.

3.1.1 What Prefix to Propagate

The intention of propagation is to inform the gateway router about reachability of producer applications on end hosts, such that it can be aware of where to forward Interests toward them. The concept is straightforward enough, but considering actual implementation raises several questions, the first being: **What prefix to propagate?**

The simplest approach is to propagate local RIB (routing information base) registraions directly. It works but is definitely unnecessary and will bother the gateway router too much. First of all, given local RIB on the host already records complete reachability information of local applications, the gateway router need only to direct *Interests* to target hosts; it's unnecessary to keep such "specific" reachability information at the gateway router. So prefix aggregation to some degree would be acceptable. Besides, the RIB on the gateway router may be dominated by propagations, especially when there is a large number of hosts or some high-loaded hosts. What's worse, per RIB entry propagation will cause exponentially accumulated affects once the gateway router is allowed to further propagate all its RIB

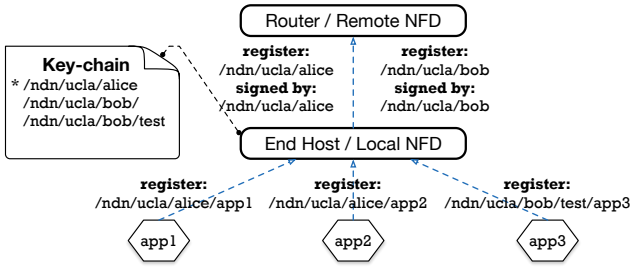

Figure 2: An example of prefix propagation.

entries. Thus, prefix aggregation is required.

Existing prefix aggregation mechanisms [5, 8] do not work here, since the aggregation in this case is made according to certain local RIB registration (section 3.1.2 discusses how local RIB registraions trigger propagations) but should cover as many potential registraions in the future as possible. Such kind of aggregation is challenging; NDN's security credential database offers useful hints, because the same namespace, i.e., the application layer namespace, is used for both forwarding and security. Actually, the namespace defined by a signing identity covers all prefixes (including other identies or application names) it can sign (check section 2 for more detail), which drives us to shift our focus from RIB entry prefixes to identities in the local key-chain[2].

Therefore, the automatic prefix propagation registers prefix represented by the identity of the key in the local-keychain, which can sign the given RIB entry, to the remote gateway for propagation. This identity can also be used to sign the registration command of this propagation. If two or more keys exist that satisfy the above requirement, the identity of the shortest one is selected. Figure 2 shows an example of such key-chain-involved propagation. Three applications register their prefixes, i.e., "`/ndn/ucla/alice/app1`", "`/ndn/ucla/alice/app2`" and "`/ndn/ucla/bob/app3`", with the local NFD. While there are three identities, say "`/ndn/ucla/alice`", "`/ndn/ucla/bob`" and "`/ndn/ucla/bob/test`", in this host's key-chain. On one side, the identity "`/ndn/ucla/alice`" covers prefixes of *app1* and *app2*, and it's the only one, so, according to oru protocol, it's selected not only as the prefix to propagate but also the signing identity of the remote registration command. On the other hand, there are two identies, "`/ndn/ucla/bob`" and "`/ndn/ucla/bob/test`", that covers *app3*'s prefix, so the shorter one, "`/ndn/ucla/bob`", is selected for propagation.

3.1.2 When and How to Propagate

To ensure that the propagation be performed implicitly and efficiently, our protocol triggers propagation automatically as soon as a new entry is inserted into the local RIB, provided that 1) this entry is neither confined to local use nor used to represent the connectivity to the remote gateway, 2) the aggregated prefix (i.e. the identity of the shortest key that can sign this entry) has not been propagated yet, and 3) there is an active connectivity to the remote gateway.

More specifically, all entries whose prefix start with "`/localhost`" are confined to local use only; they should not trigger propagations. Due to aggregation, the propagated prefix may represent two or more existing or potential entries of the local RIB. Accordingly, a new RIB entry being inserted may share a propagable prefix with existing entries. In this case, redundant propagation should be avoided via a careful check of the list of RIB entries. Lastly, we define a **link local nfd prefix** as "`/localhop/nfd`" to denote the connectivity between the host and the remote gateway. When a connectivity between the local NFD and the remote NFD[3] is established[4], an entry with the **link local nfd prefix** will be registered to the local RIB. Therefore, the presence of such an entry is a sign of connectivity to the remote gateway.

The propagation is performed by sending out the registration command of the inferred aggregated prefix; this command is sent out on the face associated to the RIB entry whose prefix is the **link local nfd prefix**. There are three types of entities involved: local applications, the local NFD, and the remote NFD. Generally, local applications, which will publish data as producers, register data prefixes to the local RIB via the local NFD. Then, the local NFD propagates properly aggregated prefixes and propagate them to the remote NFD via registration commands. At last, the remote NFD is responsible to perform those prefix registrations.

3.2 Gateway Router Performs Registrations

Upon receiving remote registraions for propagation, the gateway router should perform all validated registrations to its local RIB.

3.2.1 Secure Propagations

Upon receiving registration commands from remote hosts, the gateway should ensure the authenticity and integrity of these commands. Figure 3 demonstrates the trust model.

First, each individual registration command must be signed by the requester with a proper key, while the receiver holds a trust anchor that directly or recursively signs the signing key of that command.

On the local host, any key in the local key-chain can sign propagation commands. Since the prefix to propagate is an existing identity (see section 3.1.1), it's used to determine the signing key. In this example, *app3* reg-

[2] the key-chain owned by the local NFD

[3] the NFD run on the remote gateway

[4] How to establish such a connectivity is beyond this report 's scope and can be found elsewhere [2].

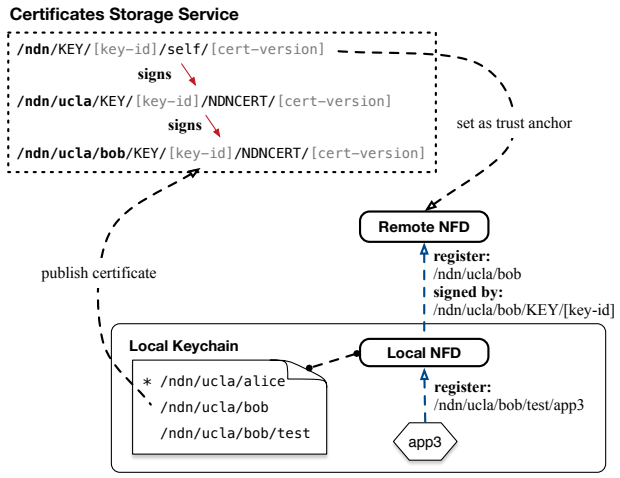

Figure 3: trust model.

isters "/ndn/ucla/bob/test/app3" to the local NFD, which then triggers the propagation of "/ndn/ucla/bob". It's also selected as the signing identity, whose default key signs the propagation command.

While on the remote gateway, the NFD is configured with at least one trust anchor. Later, only those propagations whose signing keys are directly or recursively signed by trust anchors can be authenticated. Meanwhile, along the signing path from one trust anchor to the command, all required certificates must be accessible to complete the verification process. In this example, "/ndn" is set as the trust anchor on the remote NFD. It recursively signs "/ndn/ucla/bob" via "/ndn/ucla". Given their certificates are all available on **DB**, the command to propagate "/ndn/ucla/bob" can be verified via the trust anchor.

3.2.2 *Perform Registrations for Propagation*

The propagation actually aims to brining *Interests* under the propagated prefix back to the host who makes this propagation. So, when performing registraions for the propagation after validating it, the gateway router will pick up its incoming face as the next hop face for the propagated prefix. Besides, the route inheritance **flag** of the propagation must take the default setting (i.e., **CHILD_INHERIT** flag takes effect), no matter which flag is set in the corresponding local RIB entry. This is because the route registered by propagation will never be used if the **CAPTURE** flag is set and some longer prefixes are present in the remote RIB, which is possible given the prefix to propagate is always shorter than actual prefixes due to aggregation.

3.3 Propagation Maintenance

On the gateway router, RIB entries produced by propagations are maintained as soft states, which will expire after a specified duration; the end host is responsible to periodically refresh its sent propagations.

While on the end host, all propagated entries are stored locally (in memory), each of which contains not only the propagated prefix as well as the signing identity, but also other useful parameters, such as propagation states (check section 3.3.5 for more detail) and timers, to assist the end host to retransmit propagations and to handle failures and status changes as follows.

3.3.1 *Handle RIB Status Changes*

As mentioned above, a newly inserted entry in the RIB will trigger propagation of an aggregated prefix. Whether or not the remote registration of the propagated prefix succeeds, a new propagated entry should be recorded, even though there is no active connection to send out registration commands. Meanwhile, we should withdraw the corresponding propagation when an existing RIB entry has been erased. If the corresponding aggregated prefix has not been recorded yet, or there are other RIB entries covered by the same aggregated prefix, no action is necessary. Otherwise, the offending entry in the list should be removed immediately, the propagation for which should be withdrawn as well.

3.3.2 *Handle Failures of Remote Operations*

Once there is connectivity to the remote gateway, propagating a prefix will result in remote registration, while withdrawing a propagation will result in unregistration.

If a remote registration fails, the same registration command is re-transmitted according to an exponential back-off strategy. More specifically, the duration between two continuous retransmissions is doubled until it reaches the maximum waiting period. Once a retry succeeds, the time period of waiting for the next retry will be set back to the initial value. Both the initial and maximum waiting periods are configurable.

When a remote unregistration fails, retransmission of the same command is difficult because the corresponding entry has been erased from the list of propagated entries. Thankfully, we do not have to perform such a retransmission because the propagated entry on the remote gateway will expire without periodic refreshing, which will have stopped after erasing the entry from the list.

3.3.3 *Handle Connectivity Changes*

No matter whether the connection between the end host and the remote gateway is established, lost, or recovered, there are two atomic connectivity changes involved from the view of the host: 1) connects to the gateway and 2) disconnects from the gateway. The **link local nfd prefix** will play the role of an 'alarm' for those two connectivity changes.

When the **link local nfd prefix** is inserted into the

RIB, it means there is a connection that was established or recovered. In this case, we should scan the list of propagated entries to resume all propagations suspended due to the lack of connectivity.

Fittingly, the deletion of the **link local nfd prefix** from the RIB signals a loss of current connectivity. In this case, no remote operations could be launched. Any attempt to refresh or retry should be abandoned. However, both propagating a new prefix and withdrawing an existing propagation will only operate on the list of propagated entries.

3.3.4 Handle Key-Chain Changes

As mentioned above, the prefix to propagate is determined by the local key-chain owned by the NFD. Thereby, any changes of the local key-chain may affect the decisions of propagations. For instance, when the identity of the newly added key is a prefix of some propagated prefix, that propagation should be replaced since there is now a better alternative. Similarly, some propagations should be replaced or withdrawn when an existing key is erased from the key-chain.

Problematically, monitoring those changes for propagation will closely intertwine the propagator with security mechanisms, which is bad for the modularity of the whole system. To this point, our protocol deal with those changes in a 'lazy' mode.

For each propagated prefix, the availability and priority of the corresponding identity in the key-chain is checked when that propagation is about to be refreshed, retried or resumed due to connectivity changes. Once either the availability or the priority of that identity is broken, (namely, the key under it does not exist or a key with an identity being its prefix exists) we start looking for another proper prefix for propagation, leaving the previous propagation to expire.

3.3.5 State Machine for Each Propagated Entry

All states of a propagated entry on the end host, as well as transitions between them, can make up a state machine. Figure 4 shows its core parts[5].

A propagated entry consists of the propagated prefix (i.e., the signing identity), an event that could be scheduled for either refreshing successful propagations or retrying failed propagations, as well as a **Propagation Status** indicating the current state of this entry. More specifically, a propagated entry will stay in one of the following five states of logic.

- *NEW*, the initial state.
- *PROPAGATING*, the state when the corresponding propagation is being processed but the response is not back yet.
- *PROPAGATED*, the state when the corresponding propagation has succeeded.
- *PROPAGATE_FAIL*, the state when the corresponding propagation has failed.
- *RELEASED*, the state when this entry has been released. It's noteworthy that this state is not required to be explicitly implemented, because it can be easily determined by checking whether an existing entry can still be accessed. Thus, any entry to be released is directly erased from the list of propagated entries.

Given a propagated entry, there are a series of events that can lead to a transition with a state switch from one to another, or some triggered actions, or even both of them. All related input events are listed below.

- *rib insert*, which occurs when the insertion of an RIB entry triggers a necessary propagation.
- *rib erase*, which occurs when the deletion of an RIB entry triggers a necessary revocation.
- *hub connect*, which occurs when the connectivity to a router is established (or recovered).
- *hub disconnect*, which occurs when the connectivity to the router is lost.
- *propagate succeed*, which occurs when the propagation succeeds on the router.
- *propagate fail*, which occurs when a failure is reported in response to the registration command for propagation.
- *revoke succeed*, which occurs when the revocation of some propagation succeeds on the router.
- *revoke fail*, which occurs when a failure is reported in response to the unregistration command for revocation.
- *refresh timer*, which occurs when the timer scheduled to refresh some propagation is fired.
- *retry timer*, which occurs when the timer scheduled to retry some propagation is fired.

4. TYPICAL WORK FLOW AND COOPERATIONS WITH OTHER FUNCTIONS

[5]Invalid transitions, valid transitions without neither state switch nor associated actions, and the actions associated with state switches are not presented.

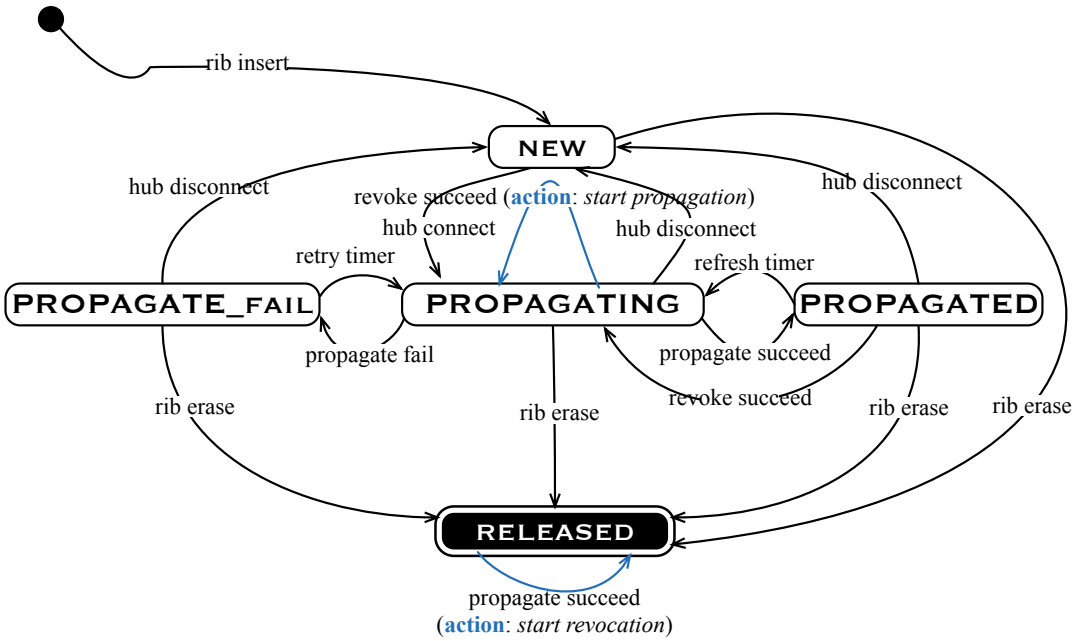

Figure 4: Overview of state machine for each propagated entry

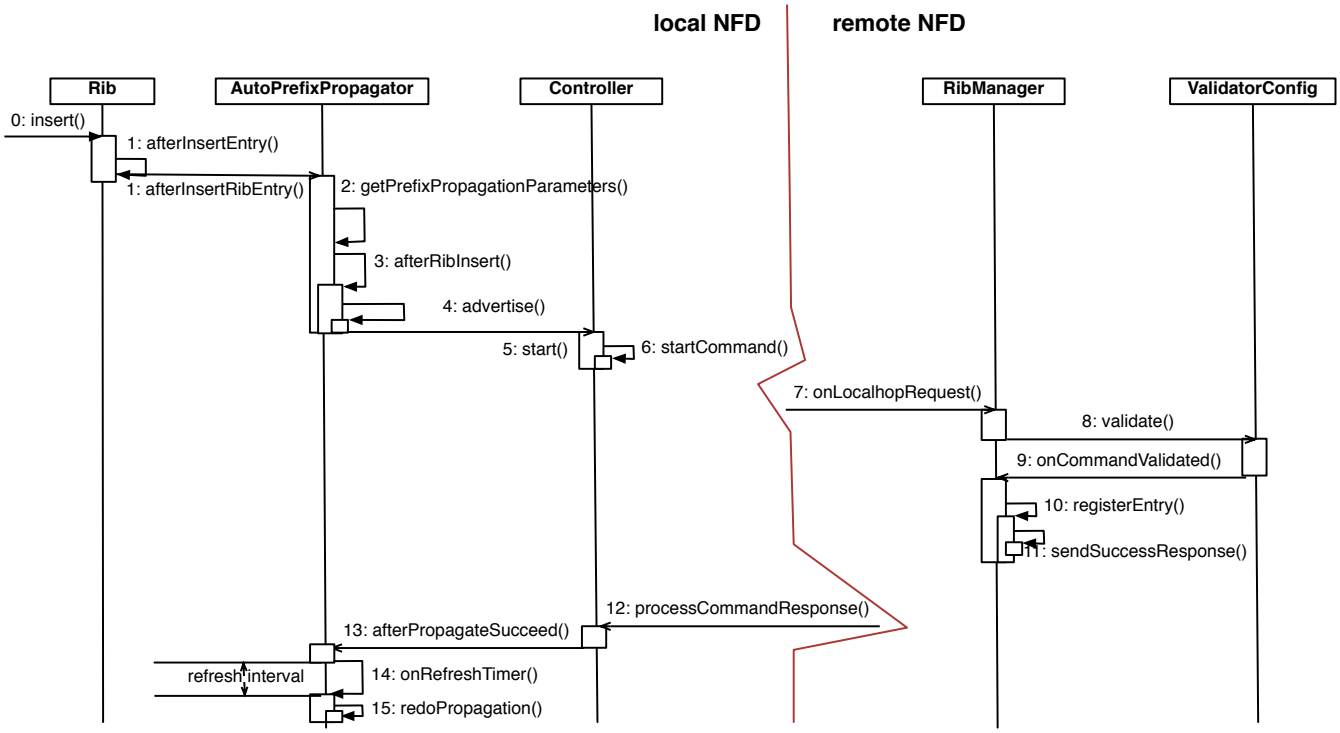

Figure 5: the typical work flow of a propagation

4.1 Work Flow

Figure 5 demonstrates details of the typical work flow of a propagation[6]. First of all, this propagation is triggered by a successful RIB insertion (step 0 and 1) on the end host. Then, after all parameters are ready (step 2) and its necessity is approved (step 3), this propagation's registration command is made (step 4) and sent out (step 5 and 6). On the gateway router, once the propagation request is received (step 7), the corresponding registration is performed to the RIB (step 10) after

[6]the work flows of withdrawing a propagation and the cases wehre there is no connectivity to routers or the remote operation fails at the router are similar but slightly different and simpler than this one.

validation (step 8 and 9). Then, a successful response is sent back to the end host (step 11). Upon receiving this response (step 12), a refresh timer is scheduled (step 13), at whose expiration (step 14) a new request will be sent out to refresh this propagation on the gateway router (step 15).

4.2 Cooperations with other functions

Prefix propagation is to announce the reachability of local applications, more precisely their producer ends, to the gateway router within one hop, assisting it to forward *Interests* more accurately. Essentially, it's one-hop prefix advertisement. Then the gateway router will, if required and feasible, make further propagations by readvertising those prefixes to other routers with the routing protocol, e.g., NLSR [4].

On the other hand, the prefix propagation is the producer advertising reachability to the gateway router; the propagation will not reach potentail consumers. Once one consumer, especially when it has connectivities to multiple routers, wants to fetch data, the self-learning strategy [7] can help it figure out which router can reach its target producer to avoid broadcasting to all routers.

5. CONCLUSION

In this report , we introduced Automatic Prefix Propagation, which enables the end host to propagate the reachability of local applications to the gateway router. In this way, the gateway router can forward Interests to end hosts more accurately. Meanwhile, locally-registered prefixes are aggregated for propagation according to the local security configuration. Propagations are maintained as soft states which require periodical refresh to avoid expiration on the gateway router. On the local host, propagated entries are maintained according to a specified state machine, handling propagation failures and connectivity changes efficiently.

Acknowledgment

This work is partially supported by the National Science Foundation under awards CNS-1629922, and CNS-1719403.

6. REFERENCES

[1] Command Interests. `http://redmine.named-data.net/projects/nfd/wiki/Command_Interests`.

[2] ndn-autoconfig. `http://named-data.net/doc/NFD/current/manpages/ndn-autoconfig.html`.

[3] A. Afanasyev, J. Shi, B. Zhang, L. Zhang, I. Moiseenko, Y. Yu, W. Shang, Y. Huang, J. P. Abraham, S. DiBenedetto, et al. NFD Developer's Guide. Technical report, Technical Report NDN-0021, NDN, 2014.

[4] A. Hoque, S. O. Amin, A. Alyyan, B. Zhang, L. Zhang, and L. Wang. Nlsr: named-data link state routing protocol. In *Proceedings of the 3rd ACM SIGCOMM workshop on Information-centric networking*, pages 15–20. ACM, 2013.

[5] S. Luo, H. Yu, et al. Fast incremental flow table aggregation in sdn. In *Computer Communication and Networks (ICCCN), 2014 23rd International Conference on*, pages 1–8. IEEE, 2014.

[6] F. Papadopoulos, D. Krioukov, M. Bogua, and A. Vahdat. Greedy forwarding in dynamic scale-free networks embedded in hyperbolic metric spaces. In *2010 Proceedings IEEE INFOCOM*.

[7] J. Shi, E. Newberry, and B. Zhang. On broadcast-based self-learning in named data networking. In *IFIP Networking*, 2017.

[8] Z. A. Uzmi, M. Nebel, A. Tariq, S. Jawad, R. Chen, A. Shaikh, J. Wang, and P. Francis. Smalta: practical and near-optimal fib aggregation. In *Proceedings of the Seventh COnference on emerging Networking EXperiments and Technologies*, page 29. ACM, 2011.

[9] Y. Yu, A. Afanasyev, D. Clark, V. Jacobson, and L. Zhang. Schematizing and automating trust in named data networking.

[10] L. Zhang, D. Estrin, J. Burke, V. Jacobson, J. D. Thornton, D. K. Smetters, B. Zhang, G. Tsudik, D. Massey, C. Papadopoulos, et al. Named data networking (ndn) project. *Relatório Técnico NDN-0001, Xerox Palo Alto Research Center-PARC*, 2010.