Enabling Named Data Networking Forwarder to Work Out-of-the-box at Edge Networks

Teng Liang*, Ju Pan*, Md Ashiqur Rahman*, Junxiao Shi†, Davide Pesavento†,
Alexander Afanasyev‡, Beichuan Zhang*
*The University of Arizona
{philoliang, pjokk722, marahman, bzhang}@cs.arizona.edu
†National Institute of Standards and Technology (NIST)
{junxiao.shi, davide.pesavento}@nist.gov
‡Florida International University
aa@cs.fiu.edu

***Abstract*—Named Data Networking (NDN) changes the network communication paradigm from address-centric in TCP/IP to data-centric. This architectural change extends capabilities of the network's forwarding plane, enabling more efficient and adaptive data retrieval. However, it is unclear how to utilize forwarding behaviors, and reduce the deployment complexity of NDN at edge networks. In fact, to run NDN applications between just two nodes involves decent knowledge of the software forwarder (NFD) and non-trivial network configuration. This paper designs self-configured adaptive forwarding plane for NDN at edge networks, to improve data retrieval efficiency and minimize deployment complexity. The designed functionalities are implemented in NFD and tested on real devices. Results show that NFD is able to run out-of-the-box with different applications. More specifically, the new forwarding strategy is able to learn and utilize multiple faces and routes, and to better handle NACK and link failure, achieving good performance.**

I. Introduction

In discussions about the deployment of Named Data Networking (NDN) [1], one widespread argument is that NDN should be first deployed at edge, such as home networks, enterprise networks, campus networks, sensor networks. These edge networks have various challenging characteristics, such as involving heterogeneous communication stacks, lack of infrastructure routing support and delay tolerance, which pose tremendous challenges to TCP/IP architecture, but can potentially be more easily tackled by NDN. Another reason is that edge networks are normally autonomous, making deployment easier with less dependency requirements.

Based on existing NDN work at edge networks, we make two observations. First, to run NDN applications between just two nodes involves decent knowledge of the software forwarder (e.g., NFD[1]) and non-trivial network configuration, i.e., to manually set up face and route. Second, the current forwarding behaviors are too simple to assist applications with different behaviors and requirements, not fully utilizing adaptive capabilities of forwarding plane. These two observations motivate this work to design self-configured adaptive forwarding plane of NDN at edge networks.

[1]NDN Forwarding Daemon, the most commonly used network forwarder that implements NDN protocol

This work considers any autonomous network supporting application communications within the network as an edge network. An edge network has three important characteristics. First, various communication technologies can exist, e.g., it is not uncommon for a home network to have Zigbee, Bluetooth, Wi-Fi, Ethernet, etc (Figure 1). An application expects data from no matter which communication channel. Second, running and maintaining traditional routing protocols can bring tremendous costs to an edge network deployment. Last, the network environment may be unstable, e.g., packets can be lost, links can go up and down, the network topology can change dynamically, Note that networks with high mobility, such as vehicle networks, are not considered in this work.

To tackle the aforementioned challenges, this work uses self-learning [2] as the basic idea. Self-learning broadcasts an Interest to all potential faces, if no route exists. Broadcasting is an effective mechanism in NDN, because NDN's forwarding plane has built-in loop detection mechanism. The broadcast Interest can discovery and bring back Data. The Data will piggyback a *Prefix Announcement*, that is able to announce a route along the path. After the first round Interest-Data exchange, subsequent Interests can be forwarded along the built path.

The original self-learning has several limitations. First, it only learns one path, even if multiple paths or data sources exist. Second, it is unable to learn and choose among different faces, but faces perform differently, e.g., multicast performs significantly worse than unicast in Wi-Fi. Third, it does not consider delay-tolerant scenarios where face may not exist. In addition, it is unclear how to implement self-learning design in NDN forwarding plane.

This work improves the original self-learning in the following aspects. First, it is able to learn all existing paths and utilize multiple paths in network. Second, it is able to learn and choose unicast face after Interest-Data exchange through multicast face on Wi-Fi. Third, to work in the delay-tolerant scenarios, forwarding plane will buffer Interests, and forward them when routes are available. All these features are implemented in NFD and tested on real devices.

The rest of the paper is organized in this way. Design

978-1-7281-7440-2/20/$31.00 ©2020 IEEE

rationales are specified Section II. More design details are explained in Section III. Moreover, the implementation is evaluated in Section IV. Related work and Conclusions are addressed in Section V and Section VI.

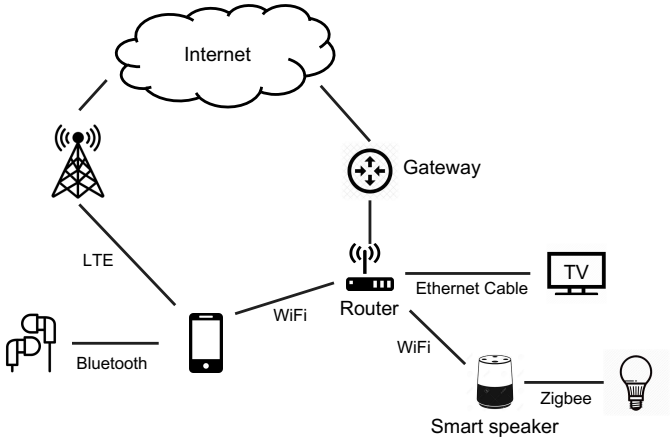


Fig. 1: An example of home network with heterogeneous communication technologies

II. Design Rationale

This paper considers four design aspects to solve the aforementioned challenges.

A. Face and Route Setup

Because of data-centric security, an NDN node can utilize any communication interface (e.g., Ethernet, Bluetooth, UDP, TCP tunnels) to retrieve data. Communication interfaces are called *faces* in NDN. To increase the chances of successful data retrieval, a forwarder should be able to set up as many potential faces as possible.

Then, the network needs to set up routes towards data sources. One solution is to run a routing protocol. However, such approach would incur extra maintenance, configuration, and deployment complexity. In addition, routing protocols ignore off-path caching. To avoid running routing protocols, an edge network can learn routes by itself after broadcasting Interests with successful Data retrieval [2]. This paper improves upon Shi's self-learning work with real implementation and testing.

B. Efficient Data Retrieval

One architectural benefit of NDN is that it is able to detect and drop looped Interests at the forwarding plane, thus an NDN node can safely forward Interests to any face in any topology without worrying about loops. Therefore, broadcasting Interests becomes a simple and effective approach to discover content. A node can simply broadcast Interests to retrieve data. However, Interest broadcasting takes significant network and device resources. To achieve more efficient data retrieval, Interest broadcasting is only used to learn routes. After the initial routes are discovered, most Interests will be sent according to those routes.

Only learning routes on existing faces may not achieve the ideal performance, because different types of faces may perform differently. For instance, in a Wi-Fi network a multicast Ethernet or UDP face performs significantly worse than a unicast face, because the link layer performs better for unicast than multicast. In this scenario, a forwarder should prefer unicast to multicast. However, a unicast face cannot be automatically configured, while a multicast face can be pre-configured with a well-known group address. To avoid manual face configuration, we add unicast face creation along with route creation in self-learning, so two nodes initially communicating through multicast are able to set up unicast faces after the initial broadcasting/learning period.

Another novel mechanism introduced is that the broadcasting of one Interest is able to learn multiple routes. Compared to the original self-learning that learns only one route, this mechanism can potentially utilize multiple paths and resources, and react to application or network failure more quickly.

C. Best-Effort Forwarding Behaviors

In addition to the aforementioned mechanisms, we define additional forwarding behaviors to assist various applications at the edge to retrieve data in a best-effort fashion. First, a forwarder may not be able to retrieve data after broadcasting Interests, thus no routes can be learned. Instead of dropping the unsatisfied Interest, the forwarder keeps it for some time until a route is learned that matches the Interest's name. This mechanism is useful in scenarios such as Delay Tolerant Networks (DTN). Second, an application may retransmit an Interest before receiving a Data. When a forwarder receives such a retransmission, it picks an unused route to forward the new Interest to, because the previously chosen route(s) did not get any Data back; if all routes have been used, the forwarder picks a route in a round-robin manner for forwarding. Lastly, if a NACK[2] with "no-route" reason is received after sending an Interest via a route, a forwarder will forget that route; then it can either forward the Interest to an unused route (if one exists), wait for other upstream responses (if the Interest was forwarded to multiple upstreams), or return a NACK to the downstream after receiving NACKs from all upstreams.

D. Security Considerations

As this work uses self-learning to learn routes, one concerning question is how to trust the route. To avoid *Fraud Route Attachment*, i.e., an attacker makes a fake route announcement, the route must be announced through an object named *Prefix Announcement*, which contains a cryptographic signature and can be verified. A Prefix Announcement can only be attached to Data packets. The trust anchor and trust rules for verifying a Prefix Announcement must be set up when bootstrapping a forwarder. The second potential attack is *Route Replay Attack*, i.e., an attacker uses an eavesdropped Prefix Announcement to hijack routes. Therefore, only verifying Prefix Announcement

[2]NACK, referring to negative acknowledgment, is a special type of packet sent in reply to an Interest as an explicit signal of a network situation, such as loop and no route.

is not enough, the solution is to verify Data packet, Prefix Announcement, and the name relationship between Data and Prefix Announcement.

III. System Design

We now turn to the design overview and details.

A. Design Overview

The basic idea, built on top of self-learning [2], is straightforward. When a forwarder receives an Interest but has no route to forward it, the forwarder broadcasts the Interest to all potential faces. Ideally, some of the faces bring back Data packets, and the forwarder sets up routes to these faces. After learning routes, the forwarder forwards subsequent Interests according to the routes.

Although the idea is straightforward, it involves several complex design challenges. Some of them have been resolved in [2], such as whether to broadcast an Interest or return a NACK; when receiving a Data packet after Interest broadcasting, what name prefix (route) to learn and how to trust it. However, the original design only learns a single path with Interest broadcasting, making it impossible for forwarding strategies to utilize multiple paths. In addition, more challenges popped up themselves after implementing self-learning on NFD with real-device testing, e.g., multicast face performs poorly in WiFi; Interest will only be forwarded once during its lifetime and will be dropped afterwards, but applications expect forwarder to take more aggressive forwarding behaviors. We summarize all design decisions in the following subsections.

B. Interest Processing

The basic Interest processing [1] involves *Content Store* (CS), *Pending Interest Table* (PIT) and *Forwarding Information Base* (FIB) lookups. As this work did not modify CS and PIT lookups, the related processing is not depicted in Figure 2. Before turning to processing details, we introduce three objects and their usage.

- **Discovery tag**, attached to an Interest, identifying the purpose of the Interest sender. Attaching this tag to Interest indicates that the Interest sender has no routes and expects Prefix Announcement to be attached on Data, while no tag means the Interest expects at least one route exits. Combining this information with local route states, a forwarder is able to decide whether to broadcast the Interest, send the Interest via specific routes without tag, or return a NACK with no-route reason back.

In addition to Interest processing, Data processing also relies on discovery tag. However, the difference is that Data processing looks up the tag existence on both incoming and outgoing Interest (details explained in Section III-C). Therefore, a forwarder has to maintain these states, and *PIT in-record* and *PIT out-record* objects are used.

- **PIT in-record table**, a data structure attached to a PIT entry, recording which faces the Interest was received from and related information. We use PIT in-record to store discovery tag on incoming Interests.
- **PIT out-record table**, the same as PIT in-record, except that it records faces that Interest was sent to and related information. This table not only stores discovery tag, but also helps to forward a retransmitted Interest and process NACK.

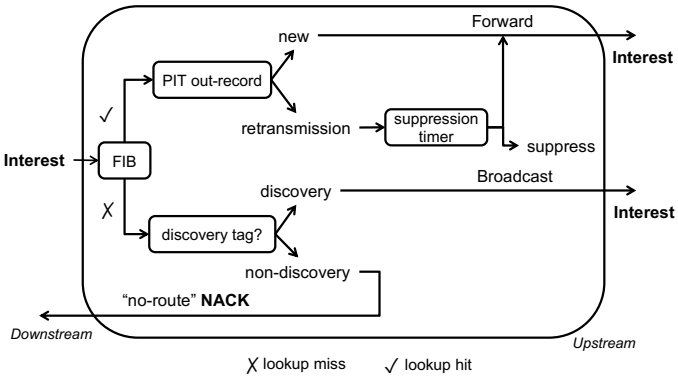


Fig. 2: Interest processing workflow. (CS and PIT lookup are intentionally omitted in the figure, since they are not changed in this work)

Next, we explain Interest processing workflow step by step (Figure 2). Processing steps fit into two major branches after FIB lookup, **no route** or **with routes**. No route is a common issue at edge networks, because of two reasons, either no route has been configured or routes disappear on failures. To distinguish these two cases, the forwarder checks if the incoming Interest has discovery tag. If yes, the forwarder broadcasts the Interest to all potential faces, otherwise the forwarder will send a NACK with no-route reason back.

Interest broadcasting involves two issues. First, a common case is that only multicast face exists at the beginning, but a multicast face over wireless channel is lossy. Therefore, Interest broadcasting to multicast face should be tried multiple times before receiving Data, instead of waiting for application retransmissions. Rebroadcasting Interest is to reduce delay in the event of packet loss. Note that rebroadcasting is only applied to discovery Interest. Second, it is unavoidable that a forwarder receives a burst of discovery Interests to be broadcast, resulting network flooding. To avoid this case, Interest broadcasting is limited by introducing a broadcasting backoff timer, that after an Interest is broadcast, the forwarder waits for the backoff time period to broadcast the next Interest.

For Interest that has routes, the forwarding idea is to forward Interest to the best route, and forward Interest to alternative routes, utilizing multipath. To avoid aggressive Interest retransmission from application, a suppression timer is used on pending Interest. More specifically, if FIB lookup returns routes, the forwarder checks PIT in-record table to identify if the incoming Interest is new or retransmitted. For new Interest, the forwarder picks the route with the lowest cost and forward it. For retransmitted Interest, the forwarder will evaluate if the Interest needs to be forwarded or suppressed by checking suppression timer on PIT entry. Within suppression timer, a retransmitted Interest will be suppressed. Different

algorithms can be applied to suppression timer settings, such as exponential backoff algorithm. The suppression timer is used to avoid aggressive Interest retransmission that wastes network resources. For a retransmitted Interest that can be forwarded, the forwarder will pick an unused route with the lowest cost to forward it. If all routes have been tried, the forwarder will pick routes in round-robin manner among routes.

One network capability an application expects is that a forwarder can cache and forward Interest more aggressively when Data is unavailable, such as in mobile or delayed data producing scenarios. This expectation requires the same mechanism as Interest rebroadcasting. The current NFD caches Interest but do not support **asynchronous Interest forwarding**, i.e., when a new route is added, the forwarder looks up PIT to find if any pending Interests can be forwarded according to the newly added route. We have implemented this mechanism in NFD. To cooperate with this mechanism, a forwarder may periodically broadcast an Interest in networks or send it to a newly added face, to discover Data with routes. Another issue is that PIT size is limited due to forwarder resources. One solution is to cache only a certain amount of Interest that are received first.

C. Data Processing

In this work, Data processing involves different actions, a forwarder may create a face, verify Prefix Announcement and announce route, attach Prefix Announcement to Data, or simply forward the Data. The decisions are made based on answers to two questions, if this forwarder requires a route, and if upstreams require a prefix announcement from this forwarder. The answers can be found with discovery tags in PIT in-record and out-record.

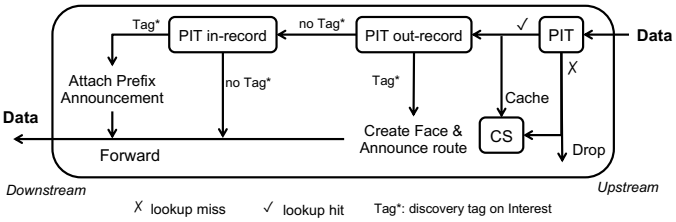


Fig. 3: Data processing workflow

- **Outgoing Interest was discovery**, meaning that the forwarder expects Data to carry Prefix Announcement and learns route. In addition, the forwarder can learn if a better face exists. For the current design, a unicast face is always preferred to a multicast face. Therefore, the forwarder creates a unicast face if necessary, verifies Prefix Announcement, and registers a route towards the created face.
- **Outgoing Interest was non-discovery**. This then depends on the discovery tag of the incoming Interest.
  - **Incoming Interest was discovery**, meaning that this forwarder has a route, thus changing a discovery Interest to non-discovery. Therefore, this router has responsibility to attach a Prefix Announcement to the Data packet before forwarding it.
  - **Incoming Interest was non-discovery**, meaning that both downstream and this forwarder has a route. Data can be simply forwarded. This is the common case for most traffic.

The upcoming question is who generates Prefix Announcement. A bigger question is what is the trust model. In this design, prefix announcement is generated by producer applications. When an application wants to serve data to an NDN network, it has to be issued a certificate first, which will be trusted by any router in this network. When a producer registers a name prefix at a forwarder, it generates a Prefix Announcement with the issued certificate, and the forwarder can verify it. The forwarder stores the Prefix Announcement along with the route, and attaches it to discovery Interest.

The original self-learning design only learns one path with one Interest broadcasting. Although multiple faces may bring back Data packets, only the first Data packet will create a route and clear the PIT entry, and the remaining Data packets will be dropped, since there is no matching PIT entry anymore. To *learn multiple paths* with one Interest broadcasting, instead of clearing PIT entry, a forwarder can keep it for a short time, to allow remaining Data packets triggering face and route creation. This mechanism still follows the characteristic of hop-by-hop flow balance, since the remaining Data packets will not be forwarded to downstreams. The cost is to maintain the PIT entry for a short time longer, which is only applied to discovery Interest.

D. NACK Processing

NACK is designed as a network signal to assist downstream for quicker reaction. There are different reasons for NACK, such as NACK with duplicate reason to indicate loop, and NACK with no-route reason to indicate no route is available. In this work, no-route NACK is only responded to non-discovery Interest when no route exists. On receiving a NACK, the processing workflow is depicted in Figure 4. If the NACK contains no-route reason, the forwarder checks if there is alternative path that has not been tried yet, if so the forwarder will forward the Interest to it. If all faces have been tried, the forwarder will wait until receiving all responses, and either generate a NACK with the least severe reason or return a Data.

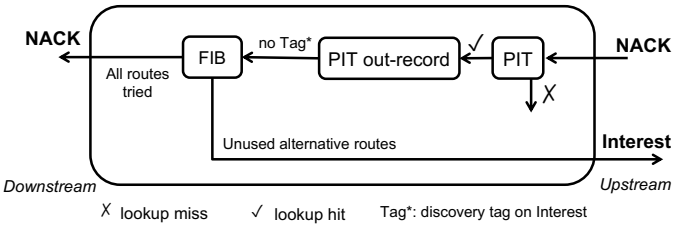


Fig. 4: NACK processing workflow

IV. Evaluation

We implement the described forwarding behaviors in NFD[3], including Interest async forwarding behaviors, the improved self-learning strategy, face and route creation at strategy, and the processing for the retransmitted Interest and NACK. We test our implementation on real devices with three applications: NDN ping, a tool to test the reachability of a data name, NDN chunks, a tool to transfer a file as Data segments[4], and iVisa video streaming service[5]. All numeric results are the average value of ten testing runs.

We first test UDP unicast and multicast face performance over Wi-Fi. In wireless networks, multicast is known to perform significantly worse than unicast, due to the lack of L2 mechanisms to handle packet loss caused by RF interference and collisions. This is confirmed in our testing using two Apple MacBook Pro laptops connected to a commodity home router. In one test with NETGEAR C3000, the statistics given by NDN chunks show that the unicast face can achieve $30.55Mbps$, while the multicast face only achieves $214.28Kbps$. Similar results have been reproduced on different commodity devices (both hosts and routers). Given that wireless communication technologies are common in edge networks, and the current L2 support for multicast is limited, unicast face creation is added in self-learning along with route creation.

Next, we test self-learning behaviors and performance in Wi-Fi with one and two producers. The expectation is that, in a network with multiple producers, the improved self-learning is able to learn multiple faces and routes, and to utilize multiple paths on Interest retransmission. Again, NDN chunks with the default setting is used for the performance measurement. The forwarding behaviors are depicted in Figs 5a, 5b, and 5c, initial Interest is broadcast, the reply Data triggers face and route creation, then subsequent Interests are forwarded to the best route. First, only one producer is put in the network, and the goodput is $30.55Mbps$. Then, two producers are put in the network as shown in the figure. Both producers are serving the same file of $500MB$ with the same name (including version number). To verify that NFD works as expected, we checked NFD logs and its face and route status after running the application. The goodput for two producers is $37.89Mbps$, about 24% better than a single producer. This is because NDN chunks consumer keeps increasing Interest window size until a threshold when Data is not retrieved in time, then consumer retransmits Interests. The retransmitted Interests will be forwarded to the alternative producer, and retrieves Data. The results show that Interest retransmission handling mechanisms can improve performance by utilizing multiple paths.

After that we test *failure handover* mechanisms, which allow a network to clear a nonworking path due to down interface or terminated application, and utilize an alternative path. To test it, one more step is added to the previous scenario, that is to kill the application running on the active serving producer (Figure 5d). As the face between the terminated application and its local NFD is down, the route on the NFD is cleared. When receiving a non-discovery Interest from the consumer, the producer will then return a NACK, which triggers route clearing on consumer and traffic shifting to the alternative route. These behaviors are verified by NFD logs, face and route status. In this testing, we ran iVisa, an NDN adaptive streaming service. Thanks to failure handover, even if the active serving producer is down during video playing, the network smoothly uses the alternative route, and users do not notice.

Last, we test the scenario consisting of an edge network connecting to a gateway. Initially, the edge network have no routes registered, but the gateway has route towards outside network. After a consumer connects to one forwarder in the edge network, it expects nodes inside the edge network to learn routes from the gateway with self-learning. To simulate the scenario, we use three laptops serving as a consumer, a middle router, and a gateway, and only the gateway has routes. The result is that the implemented self-learning is able to handle this case that involves multiple hops. To make a gateway and a self-learning based edge network working together, the gateway only needs to add functionalities to attach Prefix Announcement Data that satisfy discovery Interests, as described in Section III-C.

V. Related Work

This work is not claiming a solution to all edge networks, instead it first collects issues observed in the past few years when people running NDN at edge networks, including home networks [3], [4], smart building [5], [6], disruptive ad hoc networks [7], and campus networks [8].

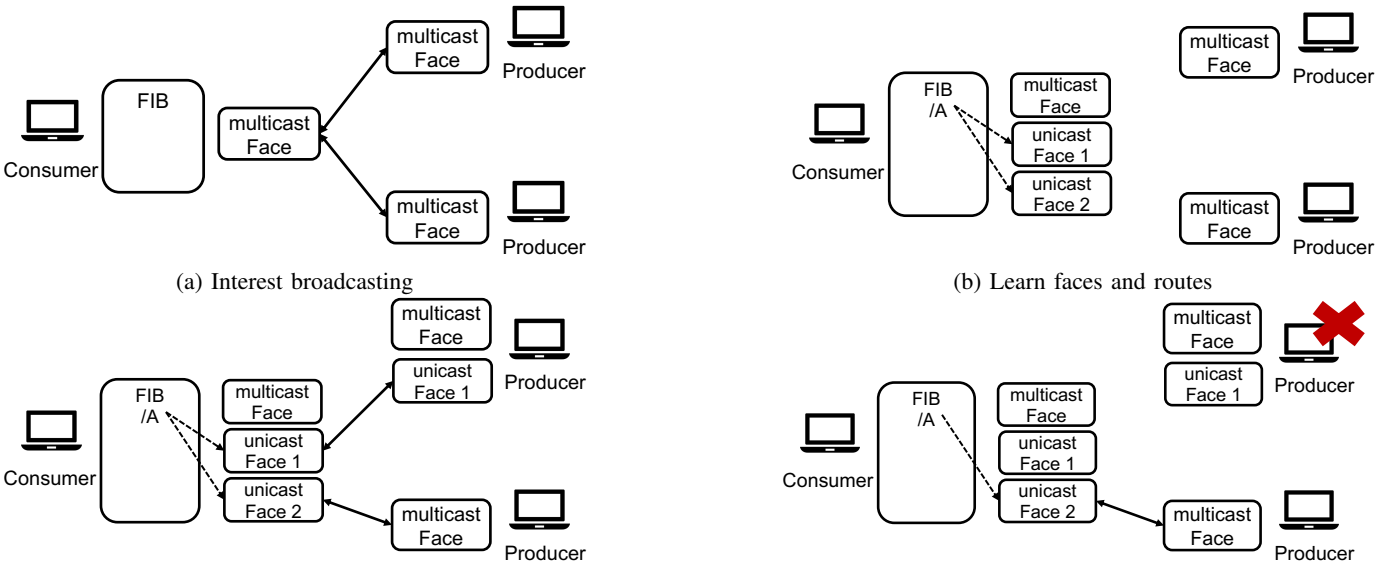
Next, this work summarizes challenges and requirements to NDN forwarding plane. First, heterogeneous communication technologies exist at edge networks, and NDN provides a common data-centric network layer to these technologies, including NDN over Ethernet, UDP, and TCP tunnels [9], NDN over zigbee [10], and NDN over bluetooth [11]. In addition, NDN's adaptive forwarding plane [12] is able to satisfy various application, such as to serve as a packet mule, and to more aggressively retrieve data. Different forwarding strategies are designed in different scenarios, e.g., NDN global Testbed [13] uses Adaptive Smoothed RTT-based Forwarding (ASF) strategy [14], and [15] proposes forwarding strategies to select network interfaces for mobile devices. However, it is unclear what forwarding behaviors including strategies should be applied to edge networks.

In terms of NDN deployment, one common issue is the discovery and connectivity among NDN nodes. One solution is to utilize a global rendezvous service to discover and connect two NDN neighbor nodes over IP networks [16]. However, this approach replies on both IP support and infrastructure connectivity, which may not exist at edge networks. An alternative is to use broadcast based self-learning [2] to discover and connect

[3]https://github.com/philoL/NDN-Self-Learning

[4]Version 0.7, https://github.com/named-data/ndn-tools

[5]https://github.com/chavoosh/ivisa-libraries



(a) Interest broadcasting

(b) Learn faces and routes

(c) Forward new Interest to the first learned route, and retransmitted Interest to the alternative route

(d) Delete the failed route and use the alternative

Fig. 5: Producer failure handover: one consumer and two producers connected to the same Wi-Fi

NDN nodes. This work is built on top of this idea with broader considerations, NFD implementations and real-device testing. In addition, existing applications can be migrated to NDN edge networks with off-the-grid capabilities [17].

VI. Conclusions

This work intends to solve two issues of NDN deployment in edge networks. First, running NDN applications between just two local nodes requires decent knowledge of NFD and non-trivial network configuration to manually set up faces and routes. Second, although NDN's adaptive forwarding plane can potentially tackle the challenges of edge networks, forwarding behaviors in edge networks are not specified. These issues motivate our work to design self-configured adaptive forwarding behaviors for NFD at edge networks. Based on existing work, observations, and discussions during the past few years, this work wraps up the design of asynchronous Interest behavior, self-learning for both face and route creation, multipath adaptive forwarding, and NACK handling. Moreover, these enhancements have been implemented in NFD, and have been tested on real devices and networks in simple but common scenarios. This work is one step to ease NDN deployment and better utilize NDN's adaptive forwarding plane in edge networks.

References

[1] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, K. Claffy, P. Crowley, C. Papadopoulos, L. Wang, and B. Zhang, "Named data networking," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 66–73, 2014.

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

[3] A. Bannis and J. A. Burke, "Creating a secure, integrated home network of things with named data networking," *NDN Technical Report NDN-0035*, 2015.

[4] M. Amadeo, C. Campolo, A. Iera, and A. Molinaro, "Information centric networking in iot scenarios: The case of a smart home," in *2015 IEEE international conference on communications (ICC)*. IEEE, 2015, pp. 648–653.

[5] W. Shang, Q. Ding, A. Marianantoni, J. Burke, and L. Zhang, "Securing building management systems using named data networking," *IEEE Network*, vol. 28, no. 3, pp. 50–56, 2014.

[6] J. Burke, P. Gasti, N. Nathan, and G. Tsudik, "Securing instrumented environments over content-centric networking: the case of lighting control and ndn," in *2013 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2013, pp. 394–398.

[7] T. Li, Z. Kong, S. Mastorakis, and L. Zhang, "Distributed dataset synchronization in disruptive networks."

[8] E. Baccelli, C. Mehlis, O. Hahm, T. C. Schmidt, and M. Wählisch, "Information centric networking in the iot: Experiments with ndn in the wild," in *Proceedings of the 1st ACM Conference on Information-Centric Networking*. ACM, 2014, pp. 77–86.

[9] A. Afanasyev, J. Shi, B. Zhang, L. Zhang, I. Moiseenko, Y. Yu, W. Shang, Y. Huang, J. P. Abraham, S. DiBenedetto *et al.*, "Nfd developers guide," *Dept. Comput. Sci., Univ. California, Los Angeles, Los Angeles, CA, USA, Tech. Rep. NDN-0021*, 2014.

[10] A. Abane, M. Daoui, S. Bouzefrane, and P. Muhlethaler, "Ndn-over-zigbee: A zigbee support for named data networking," *Future Generation Computer Systems*, vol. 93, pp. 792–798, 2019.

[11] A. Attam and I. Moiseenko, "Ndnblue: Ndn over bluetooth," *NDN Technical Report NDN-0015*, 2013.

[12] C. Yi, A. Afanasyev, L. Wang, B. Zhang, and L. Zhang, "Adaptive forwarding in named data networking," *ACM SIGCOMM computer communication review*, vol. 42, no. 3, pp. 62–67, 2012.

[13] "Ndn testbed." [Online]. Available: https://named-data.net/ndn-testbed/

[14] V. Lehman, A. Gawande, B. Zhang, L. Zhang, R. Aldecoa, D. Krioukov, and L. Wang, "An experimental investigation of hyperbolic routing with a smart forwarding plane in ndn," in *2016 IEEE/ACM 24th International Symposium on Quality of Service (IWQoS)*. IEEE, 2016, pp. 1–10.

[15] K. M. Schneider and U. R. Krieger, "Beyond network selection: Exploiting access network heterogeneity with named data networking," in *Proceedings of the 2nd International Conference on Information-Centric Networking*. ACM, 2015, pp. 137–146.

[16] A. Padmanabhan, L. Wang, and L. Zhang, "Automated tunneling over ip land: run ndn anywhere." in *ICN*, 2018, pp. 188–189.

[17] T. Liang, J. Pan, and B. Zhang, "Ndnizing existing applications: research issues and experiences." in *ICN*, 2018, pp. 172–183.