# On Broadcast-based Self-Learning in Named Data Networking

Junxiao Shi
The University of Arizona
shijunxiao@email.arizona.edu

Eric Newberry
The University of Arizona
enewberry@cs.arizona.edu

Beichuan Zhang
The University of Arizona
bzhang@cs.arizona.edu

*Abstract*—**In local area networks and mobile ad-hoc networks, broadcast-based self-learning is a common mechanism to find packet delivery paths. Self-learning broadcasts the first packet, observes where the returning packet comes from, then creates the corresponding forwarding table entry so that future packets will only need unicast. The main benefits of this mechanism are its simplicity, adaptability, and support of mobility. While the high-level idea of broadcast-based self-learning is straightforward, making the scheme efficient and secure, especially in a data-centric network architecture like Named Data Networking (NDN), requires careful examination. In this paper, we study how broadcast-based self-learning may be applied to NDN networks, point out two major issues: the name-prefix granularity problem and the trust problem, and propose corresponding solutions. We also apply self-learning to switched Ethernet as an example to develop a specific design that can build forwarding tables without any control protocol, recover quickly from link failures, and make use of off-path caches. Simulations are conducted using both real and synthetic traffic to evaluate the performance of the design.**

## I. Introduction

Broadcast-based self-learning is a common mechanism in network designs. Self-learning broadcasts the first packet with an unknown path across the network. When, and if, a response packet returns, a forwarding table entry is created toward the destination, so that future packets will only need unicast. Forms of broadcast-based self-learning have been implemented in both switched Ethernet and Ad hoc On-Demand Distance Vector (AODV) Routing [1].

Named Data Networking (NDN) [2], [3] is a data-centric Internet architecture, in which the basic network service semantic is changed from "packet delivery" to "content retrieval." NDN packets carry content names, which are used by NDN routers and hosts to match user requests (called *Interests*) with proper content objects (called *Data*). Communication is receiver-driven: a client (called a *consumer*) sends an Interest, then network forwarders find a path to a server (called a *producer*) or an in-network cache using the content name, and finally the Data packet is returned in reverse direction along the same path.

Broadcast-based self-learning allows networks to adapt to changing environments and allows for producer mobility, without using any routing or other control protocols. These benefits are particularly notable in mobile ad hoc and wireless network environments, where no fixed infrastructure has been established and periodic routing announcements would cost undue computation time and energy. This mechanism also fits the NDN architecture well because it does not require prior knowledge of the location of data. *Listen First, Broadcast Later* (LFBL) [4], *sCDN* [5], and *Reactive Optimistic Name-based Routing* (RONR) [6] are early works that apply self-learning to NDN.

ISBN 978-3-901882-94-4 © 2017 IFIP

The high-level idea of broadcast-based self-learning is straightforward, illustrated in Figure 1: Node C sends the first Interest, which is broadcasted. When the Interest reaches the producer node B, a Data is returned on the reverse path of the Interest. Network forwarders create FIB entries leading to B, so that future Interests will be sent via unicast. Nevertheless, making the scheme efficient and secure requires addressing a number of technical issues: (1) NDN uses hierarchical names and FIB entries are associated with name prefixes instead of host addresses. Given the names of the broadcast Interest and the returned Data, what prefix granularity should be used in the FIB entry for a learned path? (2) Broadcast-based self-learning lacks authentication, enabling attacks such as ARP spoofing. Can NDN's data-centric security prevent such attacks?

After examining these issues common to all networks, we developed **NDN self-learning**, a forwarding scheme applying self-learning to local area networks and switched Ethernet in particular that (1) builds forwarding tables in the data plane with low overhead; (2) recovers quickly from link failures and other path problems; (3) makes use of off-path caches for Internet contents.

We conducted evaluations using both real and synthetic traffic. Compared to RONR and Ethernet, NDN self-learning can: (1) more accurately learn FIB prefixes from producers, leading to less packet flooding; (2) recover from link failures faster, without waiting for control plane convergence; and (3) divert Interests requesting Internet contents to off-path caches in order to reduce the load on the Internet connection.

This paper is organized as follows: Section II examines the FIB prefix granularity problem. Section III investigates the trust model for prefix announcements. Sections IV and V propose NDN self-learning, our specific design for switched Ethernet. Section VI evaluates the benefits and overhead of our design. We conclude the paper in Section VII.

## II. Prefix Granularity

The NDN forwarding daemon (NFD) [7] determines where to forward an Interest by doing a longest prefix match on

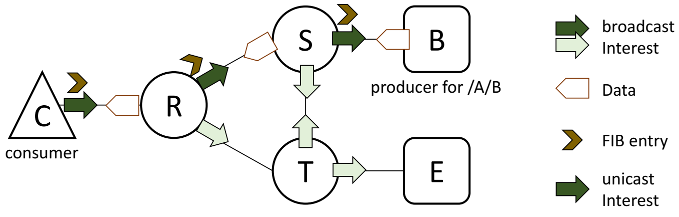

Fig. 1: Broadcast-based self-learning in NDN

the Forwarding Information Base (FIB), a name-based lookup table of forwarding paths. With self-learning, NFD floods Interests that do not match an entry in the FIB, and learns forwarding paths through observation of Data return paths, which are then inserted into the FIB. However, this still leaves unresolved the issue of how to determine the name prefix of an inserted FIB entry from the Data name. Having an accurate FIB prefix is critical for the performance of NDN self-learning.

In Figure 1, node B serves the /A/B prefix. After flooding an Interest /A/B/C/1 and receiving a Data with the same name from B, a network forwarder can see the names of the Interest and Data, but has no knowledge of B's true prefix (/A/B). If the FIB prefix is too specific (or long), such as /A/B/C, a subsequent Interest /A/B/D/1 would be flooded unnecessarily. Conversely, if the FIB prefix is too generic (or short), such as /A, an Interest that cannot be served by the producer (e.g. /A/E/F/1) would be incorrectly forwarded to B.

We have identified three potential solutions to the prefix granularity problem, which are described below.

### A. $k$-shorter Prefix

The simplest solution to this problem is to derive the FIB entry prefix from the Data name, removing the last $k$ components. One issue with this solution is that the number of name components to remove ($k$) is highly dependent on the naming scheme used by the application. For example, *ndnputchunks*, a file transfer application, generates Data names in the format /network-prefix/file-name/version2/segment0, while *NdnCon*, a real-time conferencing application, generates Data names in the format /network-prefix/user-name/cam3/1920x1080/key/frame500/data/segment0 [8]. The correct $k$ value for each application is 2 and 6, respectively.

In a general-purpose network, it is impossible for forwarders to understand the naming scheme of every application and adjust $k$ accordingly. Therefore, there will always be some FIB entries with prefixes that do not match the producer's prefix, negatively impacting network performance. Since unnecessary flooding does less harm than mis-forwarding Interests to the wrong producer, $k$ is often conservatively set to a small value. RONR adopts this solution with $k = 1$.

### B. FIB Aggregation

Another solution is to aggregate related prefixes into more generic prefixes over time. In this solution, NFD initially sets FIB prefixes conservatively, such as the Data name minus one component. Then, if most FIB entries under a common prefix point to the same nexthop, they can be aggregated into a single entry at the shorter common prefix. As an example, if /A/B/C, /A/B/D, /A/B/E, etc all point to the same nexthop, they are aggregated into an /A/B entry pointing to that nexthop.

This mechanism is similar to IP prefix aggregation, but is necessarily different because the NDN namespace is infinite. Even after a nexthop is observed to serve many prefixes under a common prefix, NFD cannot conclude with absolute certainty that the same nexthop is able to serve all content under this prefix, because there are infinitely many sub-prefixes under it. Therefore, aggregations are optimistic, and a deaggregation procedure is necessary in case an aggregation is found to be incorrect.

During our investigation of this method, we found that there is significant computational overhead in implementing the aggregation algorithm, and that the optimal parameters of the algorithm differ with the naming structure of each application.

### C. Prefix Announcements

The third and final solution is to have the producer explicitly inform the network of the prefix it serves. This solution is implemented by having the producer attach a **prefix announcement** as a link-layer header on a Data packet sent in response to a flooded Interest. The prefix announcement is itself a Data packet, containing the prefix served by the producer. It is signed by the producer, allowing its authenticity to be verified (see Section III-A).

This solution is straightforward and allows FIB prefixes to adapt to applications' differing granularities. One drawback of this approach lies in the bandwidth overhead of transmitting announcements; however, this is compensated for by the reduction in computational overhead compared to FIB aggregation.

There are two variants of this solution: (1) Instead of flooding the first application Interest and attaching the announcement onto the application Data, we can use an ARP-like procedure: the consumer floods an Interest asking "who has /A/B/C/1?", receiving the announcement in response, and then unicasts application Interests. However, this incurs an extra round-trip time before the first Data is retrieved. (2) Instead of letting the producer attach an announcement onto the Data, the producer prefix can be attached onto the Interest by the consumer. This is applied to vehicular networks in *Navigo* [9], but it will not work in general-purpose networks because consumers may not know a producer's prefix granularity.

## III. Trust of Announcements and Data

ARP spoofing is a common attack in IP-based local area networks. In this attack, a malicious node transmits an ARP message with a forged sender IP address, allowing it to capture, manipulate, or even block traffic intended for another host. A similar *producer spoofing* attack can occur with NDN self-learning. To perform this attack, a malicious node responds to flooded Interests with bogus Data, causing subsequent Interests to be forwarded toward itself. This attack is particularly effective when combined with prefix announcements: by announcing a very short prefix, the resulting FIB entry can attract Interests under a large namespace.

Producer spoofing attacks can be mitigated using NDN's mandatory Data authentication, which requires all applications to sign and authenticate every Data packet. This is done by applying two tiers of trust models: (1) a universal, network-wide trust model for authenticating prefix announcements, and (2) application-specific trust models carried in every prefix announcement for validating packets under the announced prefix. Announcement trust and application Data trust are separated because there is not a universal trust model that covers all application Data; instead, each application can define its own trust model. Conversely, a LAN is under the control of one entity, so it is feasible to define a universal trust model for prefix announcements.

### A. Trust Model for Announcements

In a routing-based NDN network, all prefix announcements are authorized by the network administrator. A trust relationship among routers prevents other nodes from announcing arbitrary prefixes [10]. In broadcast-based self-learning, we can apply a similar principle and let the network administrator authorize prefixes announced by producers.

When a content-producing node joins the network, its operator will request an *announcement signer certificate* from the network administrator. Part of the name of this certificate indicates which prefix the producer is allowed to announce. [1] The administrator then signs the certificate with the network's certificate authority. After the certificate has been issued, the producer can sign its prefix announcements with this certificate.

The network administrator configures every switch/router in the network with a trust model for authenticating prefix announcements. Upon receiving a prefix announcement, NFD can verify: 1) that the announcement has a valid signature matching the public key in the announcement signer certificate, 2) that the announced name prefix matches the allowed prefix encoded as part of the certificate name, and 3) that the certificate is issued by the network's certificate authority. If all of these conditions are satisfied, NFD accepts the prefix announcement and inserts a FIB entry.

### B. Trust Schema for Application Data

While an announcement trust model ensures that no one can announce unauthorized prefixes, a malicious node can still collect a prefix announcement from a legitimate producer, and attach it onto a bogus Data packet. This is a form of replay attack, as the malicious node is replaying an old announcement. A standard countermeasure to replay attacks is challenge-response. In this case, a forwarder would generate a random piece of information as the "challenge", and the producer would include this information in the prefix announcement, signed by the announcement signer certificate, which proves that the announcement is fresh (not replayed). However, using challenge-response would require that every flooded Interest be processed at the producer and prevent the use of in-network caches, as they are unable to sign the challenge.

To prevent producer spoofing, while keeping the benefits of NDN's in-network caching, we propose a different solution: Every prefix announcement may carry the trust model for Data packets under their announced prefix. This trust model is encoded as a *trust schema* [11], which contains a set of name-binding rules that dictate which certificates are authorized to sign each Data or sub-certificate, as well as a set of top level certificates as trust anchors. With the trust schema, anyone can verify received Data packets against the trust model in an automated and consistent way. In-network caches are able to respond to flooded Interests with legitimate Data, but malicious nodes cannot poison the network with bogus Data packets, as their Data packets would not comply with the trust schema.

We normally do not require NFD to verify application Data because it would be prohibitively expensive. Instead, if a consumer detects a bogus Data during its validation, it can send a report to the upstream network [12], which then triggers NFD to verify the application Data. If the Data is found to be bogus, NFD deletes the FIB nexthop toward the malicious node and, as a penalty, may stop flooding Interests toward it for a period of time.

## IV. Self-Learning on Switched Ethernet

We developed a specific design that applies NDN self-learning to switched Ethernet in wired LAN environments. In this section, we introduce how self-learning builds forwarding tables in the data plane without any other control protocols and with low overhead, as well as how it can quickly recover from link failures and other path problems. This design primarily targets wired networks with hundreds of nodes, such as those found in an office building or university department. It is assumed that there is no congestion, as most wired LANs are over-provisioned.

### A. Building Forwarding Tables in the Data Plane

NDN self-learning determines the location of content by flooding the first Interest, and observing the return path of the corresponding Data packet. Forwarders use the prefix announcements carried on Data to build the forwarding tables (FIB). This happens entirely in the data plane, and does not require a routing protocol or any other control plane protocols.

This process is similar to the flood-and-learn process in traditional address-based Ethernet. If an Ethernet switch does not know the location of a packet's destination address, it floods the packet. Afterward, it remembers a mapping between the source address and the switch port, allowing subsequent packets to that address to be sent unicast. However, Ethernet flood-and-learn suffers from bridge loops: If the topology contains cycles, a flooded packet will loop along them. To prevent bridge loops, Ethernet employs the Fast Spanning Tree

[1] To produce content under multiple prefixes, each prefix needs a separate announcement signer certificate.

Protocol (FSTP), which trims the topology to a spanning tree by disabling particular links. However, FSTP severely reduces the available bandwidth in the network and negatively impacts the performance of Ethernet.

Conversely, NDN has built-in loop freedom. This is enabled through the use of nonces. The *nonce* is a random number carried in every Interest packet, generated by the consumer. When NFD receives an Interest, it checks its $\langle name, nonce \rangle$ tuple against recently seen $\langle name, nonce \rangle$ tuples. If the same tuple has been seen recently, either this Interest is looping, or it was received on multiple paths and the current incoming Interest is not on the fastest path (for example, in Figure 1, node S would drop the Interest from T during the initial flooding since it arrives later than the Interest from R). In both cases, NFD will drop the Interest, preventing Interests from looping. Loops are not possible for Data packets, as they always follow the reverse path of an Interest. This mechanism prevents bridge loops in NDN, without requiring a protocol like FSTP. Not only does this eliminate a control plane protocol, but it also enables NDN to utilize the bandwidth on all available links, as well as ensuring that flooding will always learn the fastest path. This is because, when two copies of a flooded Interest arrive at the same node via two different paths, the copy received later is dropped, so that the Data reply is only returned via the fastest path traversed by the Interest at the time of flooding.

After a successful Interest flooding, NFD inserts a FIB entry for the prefix listed in the returned Data's prefix announcement. The nexthop for this entry is the adjacent node that the Data was received from. This allows subsequent Interests under this producer's prefix to be forwarded along a known path via unicast, without requiring the use of flooding.

The number of dynamic FIB entries created using this mechanism is subject to a capacity limit: when the FIB is full, the least-recently-used (LRU) entry is erased.

### B. Minimizing Flooding Overhead

Interest flooding consumes network bandwidth, and incurs CPU processing overhead at end hosts that receive the flooded Interests but do not have the content. The key to minimizing overhead is to flood less often and in smaller regions.

In keeping with this goal, only the consumer can initiate Interest flooding. Flooding should be initiated for any Interest with an unknown path or where the previously learned path has failed (see Section IV-D). Whether an Interest is being flooded or not is indicated through a *discovery tag* field on every Interest. The consumer tags an Interest as "discovery" if it wishes to flood the Interest; otherwise, the Interest is tagged as "non-discovery". The inclusion of this tag allows the consumer to regulate the flooding frequency.

However, if multiple consumers concurrently request content under the same name prefix (but not necessarily the same data name), each joining consumer will initiate flooding of its first Interest, even if a network forwarder has already learned the path to the prefix. To avoid unnecessary flooding, when NFD receives a discovery Interest but already knows a working path, the Interest will be retagged as "non-discovery" and forwarded along the known path via unicast. This effectively "absorbs" Interest flooding from joining consumers. When a Data packet comes back to the same forwarder, the original prefix announcement that was used to create the FIB entry will be attached onto the Data packet, allowing new downstream nodes to learn the producer's prefix.

### C. Fast Reaction to Link Failures

Link failures are infrequent but inevitable in wired switched Ethernet environments. NFD constantly monitors the availability of learned paths, and informs the consumer when a link failure is detected so that it may initiate another flooding.

We use a variant of Bidirectional Forwarding Detection (BFD) [13] at the link layer to detect link failures between adjacent nodes. NDN-BFD treats any incoming NDN packet on a link as an indication that the link is up. If a node has not sent any NDN packets on a link during a specific period of time (e.g. 5ms), it transmits an "idle packet", which informs the neighbor that the link is still up, even though there is no current activity. If no NDN packets or idle packets arrive within a longer period of time (e.g. 50ms), NDN-BFD declares a link failure to the network layer.

Such link failures affect all pending Interests already forwarded via the failed link, as well as all future Interests that would have used the failed link as their nexthop. For already forwarded Interests, although NFD could attempt to retransmit them on an alternate path, it is computationally expensive to do so, because records about pending Interests are organized by name prefix and NFD cannot easily enumerate through them. Thus, as a trade-off between recovery speed and processing overhead, we rely on consumer retransmission (see Section IV-D) to retransmit the affected Interests.

NFD will react to link failure when it receives another Interest (which could be a new Interest or a retransmission) matching a FIB entry with the failed link as its nexthop. It does not forward the Interest to this nexthop. Instead, if the FIB entry has a known alternate path, NFD forwards the Interest on that path. If no alternative path exists and the incoming Interest is tagged as "discovery", it is flooded. Otherwise, since there is no alternative path and the Interest cannot be flooded, NFD informs the downstream about the link failure by returning a *Nack* against the non-discovery Interest. A Nack ("negative acknowledgment") is a packet returned in lieu of a Data when it is unable to be retrieved [14]. This packet contains the Interest name and indicates why the Data could not be retrieved (in this case, "link failure").

Upon receiving a Nack, a downstream forwarder will set a flag in the relevant FIB entry indicating that the current path to this prefix has a link failure. It will then run through the same procedure as the node adjacent to the failure (listed above). Eventually, if no functioning alternate path is found, the Nack will be returned, hop by hop, to the consumer. Then, the consumer can retransmit the Interest, tagged "discovery", to initiate a new flooding, which may find another path to the producer.

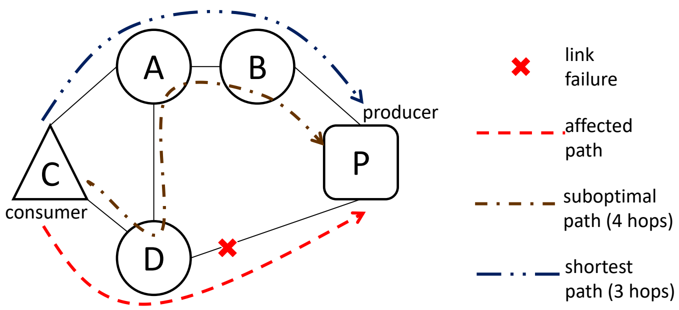

Fig. 2: Flooding from node D learns suboptimal path

We require flooding to be initiated from the consumer, instead of allowing NFD to flood a (non-discovery) Interest when there is a link failure, because flooding from an in-network node might cause the network to learn suboptimal paths. In Figure 2, consumer C initially retrieves Data from P via C-D-P path. When link D-P fails, if D floods the Interest, the path learned would be C-D-A-B-P, which is one hop longer than the shortest path C-A-B-P which could be learned if flooding is initiated from the consumer. This is also the motivation of introducing a “discovery tag” into the Interest packet to indicate whether an Interest is being flooded.

### D. Consumer Retransmission

When the consumer believes or is informed that an Interest or Data has been lost, but still wishes to retrieve the relevant Data, it should retransmit the Interest. The retransmitted Interest contains the same name, but has a new nonce, to avoid being detected as a duplicate. The consumer can either send a non-discovery Interest, allowing the network to explore known alternate paths, or send a discovery Interest, initiating a new round of flooding.

There are two situations when a consumer should retransmit an Interest:

*a) Nack:* As mentioned in the previous section, when the network returns a “link failure” Nack to the consumer, the consumer should retransmit a discovery Interest to re-trigger flooding. Retransmitting a non-discovery Interest would be useless in this situation, as the network has already exhausted all known alternate paths prior to returning the Nack to the consumer.

*b) Retransmission timer:* Once the consumer has learned the prefix granularity from the prefix announcement, it can measure the round trip time for data retrievals, and use it to calculate the retransmission timeout (RTO) using TCP’s algorithm [15]. After expressing an Interest, if no Data comes back within the RTO, it can retransmit the Interest. This is primarily used to recover Interests that were already forwarded to a link prior to its failure. [2]

### E. Handling Producer Mobility

Mobility is handled using the same procedures as link failure: a node that has moved is treated as a link failure between the previous hop and the moved node. When an Interest reaches the previous hop, a Nack will be returned so that the network can discover the new location of the Data. After finding a new path, the new nexthop is added to the FIB entry and will be used for subsequent Interests. The old path is retained in the FIB entry as an alternate. This is especially useful when a node is moving back and forth between several attachment points. In this case, the forwarder at the branching point could alternate between the paths without returning Nacks.

## V. Caching of Internet Contents

One of NDN’s benefits is in-network caching. Every node opportunistically caches Data packets passing through it, allowing them to satisfy future Interests requesting the same Data. Popular contents requested by multiple consumers can be satisfied from a nearby cache, resulting in bandwidth savings and latency reduction. However, for most local contents produced by a node within the LAN, these improvements are marginal, as internal links have abundant bandwidth and low latency. On the other hand, most LANs are connected to the Internet via Wide Area Network (WAN) connections which often have limited bandwidth and high latency. Maximizing the utilization of in-network caches can significantly improve the performance of Internet content retrieval.

### A. Caching Basics

Every forwarder is equipped with an opportunistic cache of Data packets, called the Content Store (CS). Data packets passing through a forwarder are added to its CS. When an Interest arrives, after confirming its nonce is not a duplicate, the forwarder queries the CS to look for a matching Data that can satisfy the Interest. If a match is found, it is returned to the downstream, and the Interest is not forwarded or flooded further. The CS is subject to a capacity limit set according to available memory on the node. When the CS is full, entries are evicted in either First-In-First-Out (FIFO) or Least-Recently-Used (LRU) order.

Both discovery and non-discovery Interests can be satisfied with cached Data. When answering a discovery Interest, NFD attaches the original prefix announcement onto the Data. Therefore, an Interest flooding may learn a path toward a cache, instead of a path toward the producer. If a subsequent Interest is forwarded to a cache but cannot be satisfied by a cached Data, the cache node should forward or flood (if the Interest is tagged “discovery”) the Interest as a forwarder normally would, or return a Nack if there is no available upstream. Returning a Nack would cause the network to find a path to another cache or producer.

### B. Internet Contents Retrieval

Internet contents are retrieved via a gateway router. When a consumer expresses an Interest requesting an Internet content, flooding is used to discover a path to the gateway, which then retrieves the content from the Internet. The gateway announces itself as the “producer” of the “/” prefix through

[2] The lack of a Data reply could also be caused by packet loss, but in NFD, packet losses are greatly reduced through link layer retransmissions [16].

a prefix announcement attached to the returning Data packet. This announcement causes network forwarders to insert a FIB default route toward the gateway, so that subsequent Interests for Internet contents are sent to the gateway without flooding.

One issue is that, a FIB entry at the "/" prefix will match not only Interests requesting Internet contents, but also Interests requesting local (internal) contents. To solve this problem, the gateway's announcement should carry a *local prefix list*, containing a list of administrator-defined prefixes internal to the LAN. If an Interest matches this root FIB entry, but falls under one of the local prefixes, it will not be forwarded toward the gateway, but instead flooded or Nacked as if there were no FIB match.

Since this FIB entry will match every Interest requesting Internet contents, all Interests for Internet content will be forwarded as non-discovery, allowing only caches on the path between consumers and the gateway to be utilized. Although these on-path caches can satisfy some redundant Interests, requiring less content to be retrieved from the Internet, the caches in network switches and the gateway router have limited capacity and handle a high volume of traffic, and therefore can only offer limited assistance in this regard. However, there is abundant cache capacity on other, off-path nodes, especially on end hosts. Retrieved Data can be cached for longer on these nodes due to the less frequent replacement of old entries. Therefore, we extend the design in the next section to utilize off-path caches for Internet traffic by diverting Interests to them, in order to reduce WAN connection bandwidth usage.

### C. Diverting Interests to Off-Path Caches

To utilize off-path caches, all nodes remember where each Data packet they processed was forwarded to and, therefore, where they may be cached. After these Data packets have been evicted from the on-path cache, Interests requesting them can be forwarded to potential off-path caches.

Information about potential off-path caches is stored in the CS. When a Data is forwarded downstream, the downstream face is recorded as a potential off-path cache in the Data's CS entry. When a CS entry must be evicted, instead of deleting it altogether, it is converted to a "stub" entry, containing the Data name and the locations of potential off-path caches, but not the Data payload. Since a name is much smaller than a Data payload, a node can store many more stub entries than regular CS entries. These stub entries are subject to a separate capacity limit and can be evicted once that limit is exceeded.

When an incoming Interest matches a CS stub entry, the forwarder examines potential off-path cache locations, and decides whether to divert the Interest by predicting whether the Data is still cached on the off-path cache. We use a simple heuristic for this prediction: a Data is less likely to have been evicted if less Data packets have been sent to that downstream in the meantime. This heuristic is implemented in three steps: (1) NFD maintains an outgoing Data counter for each *face* (abstraction of "network interface"). This counter is incremented every time a Data packet is sent to the face. (2) When a downstream is recorded as a potential off-path cache, the current value of that face's counter is recorded in the CS entry. (3) To determine whether an Interest should be diverted to an off-path cache, NFD calculates the number of other Data packets sent to this cache after the requested Data. This is done by subtracting the value recorded in the CS entry from the current counter value. If the difference is less than or equal to a pre-determined *diversion threshold*, the Interest is diverted to the off-path cache, and not forwarded further toward the gateway. In case there are multiple eligible off-path caches, the Interest is diverted to the one with the least "other Data count", as it is most likely to still have the Data.

When an off-path node receives a diverted Interest, the forwarder on that node queries its CS to look for a match. If a regular CS entry is found, the Data packet is returned to the diverting node, which returns the Data to the downstream. If a stub CS entry is found, the Interest is passed to the downstream that is most likely to still have the Data, chosen with the same heuristic as above; however, it is not subject to the diversion threshold criteria [3]. If there is no match in the CS, most likely because the stub entry has been evicted, the off-path node returns a Nack to the diverting node, which then forwards the Interest toward the gateway. This Interest is also tagged "no diversion", disallowing other on-path nodes from diverting it again, in order to limit the extra latency introduced by Interest diversion.

Similar opportunistic off-path cache discovery mechanisms have been proposed for ISP networks. (1) In *S-BECONS* [17], an on-path node remembers which downstream has most recently retrieved a file, along with a timestamp of that retrieval. Whether an incoming query would be diverted or not depends upon whether the timestamp is recent enough. However, we argue the traffic volume sent to a downstream is a better predictor than when the downstream last retrieved the Data, hence our heuristic based upon "other Data" count. (2) In [18], when an incoming Interest matches the record of off-path caches, it is forwarded to a potential off-path cache; additionally, in most cases, the Interest is still forwarded toward the FIB nexthop, without waiting for an answer from the off-path cache. Although this design is effective in reducing latency when Data is found in an off-path cache, it would not save WAN connection bandwidth in our scenario. On the other hand, they limit diversion overhead by attaching some *quotas* on the Interest and letting every node decide whether to spend them. This is present in our design as the "no diversion" tag.

## VI. Evaluation

NDN self-learning was implemented in NFD, and evaluated in the Mininet network emulator [19], with both real world and synthetic topologies and traffic traces. Three additional forwarding schemes were implemented for comparison. (a) The "broadcast" scheme reuses NFD's multicast forwarding strategy, which floods Interests to every neighbor, with the exception of the downstream; however, Data only returns

[3] A node can tell that the Interest is already diverted because it is coming from an upstream in the direction of the gateway.

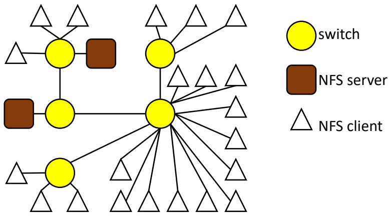

Fig. 3: Topology for NFS experiment

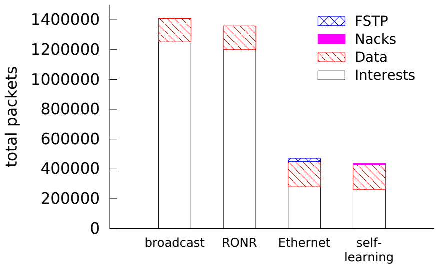

Fig. 4: NFS experiment, total packets

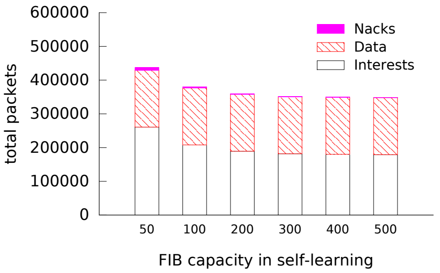

Fig. 5: Effect of FIB capacity in NFS experiment

via the shortest path. (b) RONR is implemented according to [6]. FIB granularity is derived from the Data name, removing the last component. The FIB expiration timer is set to 5 seconds. (c) "Ethernet-style self-learning" adapts Ethernet's self-learning algorithm to NDN. Interests and Data are restricted to the spanning tree, calculated by an FSTP implementation from the VDE switch emulator [20]. FIB granularity comes from prefix announcements, and FIB entries expire after 15 seconds.

### A. *Low Bandwidth Usage*

NDN self-learning can learn producers' accurate prefixes from prefix announcements. These announcements are used to build forwarding tables in the data plane with low overhead. We evaluated the performance of our design using a real-world university department topology and traffic trace. In Nov-Dec 2014, we captured Network File System (NFS) traffic with `nfsdump` [21], and implemented a pair of NDN programs to emulate the same functionality using NDN Interest-Data exchanges. The client program expresses Interests to read files and directories, as well as to execute commands. Interest names sent by the client start with the file name. [4] The network must learn which server serves the directory, as well as how to reach that server. In response, when NDN self-learning or Ethernet is in use, the server attaches a prefix announcement announcing the NFS mount point's prefix. To write a file, the client first sends a command to the server, and the server expresses Interests to retrieve the file from the client. These "reverse" Interests start with a name component that identifies the specific client, meaning that the network needs to learn a path to the client. Each Interest can be retransmitted up to 3 times upon a Nack or timeout. If no Data has been received after 3 retransmissions, the NFS operation that triggered the Interest is marked as having failed.

We took a subset of the traffic traces, and replayed them on the topology in Figure 3. This topology has 5 switches, 2 NFS servers, and 21 NFS clients. The experiment lasted 60 minutes, during which 36478 NFS operations were performed. We measured how many packet transmissions were needed to complete the NFS operations in each forwarding scheme, and plotted the results in Figure 4. The FIB capacity of NDN self-learning was set to 50 entries per node; RONR and Ethernet used a maximum of 510 and 37 FIB entries per node, respectively. The results demonstrate that NDN self-learning transmitted the least number of packets, while Ethernet-style self-learning, which also uses prefix announcements to determine FIB granularity, transmitted 7% more packets than self-learning, more than half of which can be attributed to the Fast Spanning Tree Protocol. RONR uses a 1-shorter prefix for its FIB entry granularity, which required one flooding per file for many NFS operations, as the Data name used ends with a segment number component (as per NDN naming conventions [22]). This pushed RONR's bandwidth usage close to that of broadcasting.

We studied the effect of FIB capacity on NDN self-learning, with results shown in Figure 5. It can be seen that NDN self-learning performs better with a higher FIB capacity limit, because every node is able to remember more forwarding paths for a longer period of time, resulting in less flooding. However, the improvement was marginal when the FIB capacity was greater than 300.

### B. *Fast Link Failure Recovery*

Each forwarding scheme was evaluated for their reaction to link failure on the topology in Figure 6. The topology contains 12 NDN switches and 12 end hosts. Node P is a producer, serving content under `/P`. Node C is the consumer executing `ndnping` that continually sends Interests for `/P/ping/<sequence-number>` at a 1-second interval. The consumer application will retransmit an Interest at most twice upon receiving a Nack, but will not retransmit after a timeout.

The experiment lasted 60 seconds. 30 seconds into the experiment, link 1 experienced a failure. Since link 1 sits on the shortest path between C and P, it is expected that traffic will be affected, and therefore the forwarding scheme should find an alternate path.

We ran this experiment for the broadcast, Ethernet, RONR, and NDN self-learning schemes. Since the location of the root bridge in Ethernet affects which links are on the spanning tree, we ran Ethernet scenarios with R as the root bridge, in addition to scenarios with S as the root bridge. We also ensured that link 1 was on the initial spanning tree.

[4] File names in the trace were anonymized as SHA-1 digests to respect privacy, but the name hierarchy was preserved.

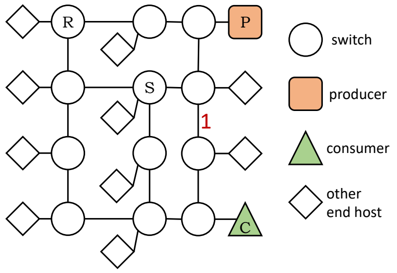

Fig. 6: Topology for link failure experiment

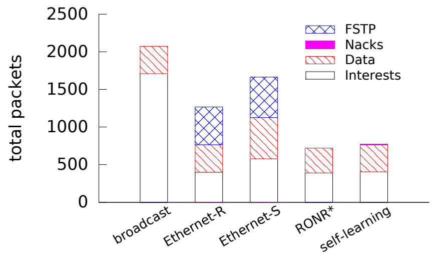

Fig. 7: Link failure experiment, total packets * RONR has 8.3% end-to-end loss

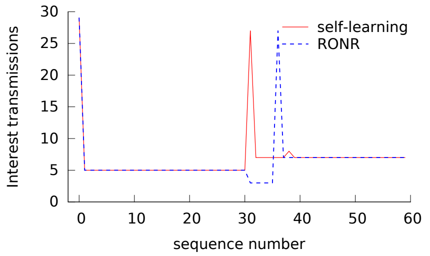

Fig. 8: Link failure experiment, Interest transmissions

Emulation revealed that, although every forwarding scheme can eventually detect a link failure and switch to an alternate path, the bandwidth usage and recovery speeds varied greatly. Figure 7 shows the total number of packet transmissions across the network by each scheme. While RONR demonstrated slightly lower bandwidth usage compared to NDN self-learning, it had a 8.3% end-to-end packet loss. The three other tested schemes completed the communication in its entirety, without any end-to-end packet loss. Among these, NDN self-learning transmitted less packets than broadcast and Ethernet.

We also compared the number of Interest transmissions in RONR and NDN self-learning for each Interest (shown in Figure 8). Both schemes flooded `/P/ping/0` and learned the shortest path. Link 1 failed after `/P/ping/30` was retrieved. The sharp increase in Interest transmissions indicates that NDN self-learning detected the link failure immediately and responded by promptly flooding `/P/ping/31`. On the other hand, RONR was stuck on the failed path for 5 seconds until the FIB entry expired, causing packet loss during these 5 seconds, and then discovered a new path by flooding `/P/ping/36`.

### C. *Off-path Cache Utilization*

Our Interest diversion design was also evaluated for its effectiveness in utilizing off-path caches for Internet content retrieval. This experiment used the tree topology shown in Figure 9. In this experiment, each end host ran a consumer program that retrieved 5000 Data packets from the Internet over a period of 1000 seconds, with a 200ms interval between Interests. Interest names followed a Zipf distribution with the parameters $N = 200000$ and $s = 0.955$ [23]. The end hosts also retrieved Data from each other at regular intervals, such that local traffic accounted for either 20%, 50%, or 80% of the total traffic volume of the network, depending upon the scenario. The Content Store (CS) capacity on each node was set to 2000 Data packets and 4000 stub entries, both using the FIFO replacement policy.

The main benefit of diverting Interests to off-path caches is to decrease the amount of traffic on the WAN connection, which is often the source of much latency when accessing Internet content. Figure 10 shows the number of packets traversing the WAN connection when running self-learning without diversion (“nodivert”) and with diversion, under three different traffic ratios. We can see that the Interest diversion mechanism can effectively reduce utilization of the WAN connection by up to 13%. The total amount of bandwidth saved by this mechanism is affected by what percentage of the LAN’s traffic volume consists of Internet content. It was more effective when Internet traffic made up a greater share of the LAN’s total traffic, as demonstrated in the lower curves.

The diversion threshold parameter (X-axis), used by the forwarder to determine whether a potential off-path cache may still have the Data (mechanism described in Section V-C), has a significant impact on the effectiveness of the Interest diversion mechanism. When the diversion threshold was set to a small value (e.g. 250), no diversions occurred in our experiments, so that there was no change in performance. Larger diversion thresholds tended to achieve greater benefit, as long as they were set below the CS capacity ($\leq$ 2000). Setting the threshold to be greater than the CS capacity (e.g. 3000) caused a slight reversal in the trend of decreased WAN connection utilization, but still allowed for better performance than self-learning without diversion.

The processing overhead of diverted Interests is shown in Figure 11. In this plot, “hits” indicate how many diverted Interests were satisfied by an off-path cache, while “misses” indicate how many diverted Interests missed an off-path cache and were Nacked. A “hit” is desirable because, while it incurs a small processing overhead within the LAN, the significantly larger delay of sending an Interest onto the Internet is avoided; a “miss”, on the other hand, incurs internal processing overhead and adds a small, but undesirable, delay before Data is retrieved from the Internet. We can see that, in our experiments, there were a negligible number of cache misses in scenarios with smaller diversion thresholds ($<$ 1500). A significant number of cache misses begin to appear when the diversion threshold approaches 2000 because, when the diversion threshold equals the CS capacity, it is more likely that some Internet content has been evicted from the Content Store by a local content. For example, if A1 fetches a local Data from A2 and causes an Internet Data to be evicted from A’s CS, R will not know of this eviction and may still divert an Interest toward A, causing a cache miss. When the diversion threshold is set to 3000, a value greater than the CS capacity, there is a significant increase in cache misses but

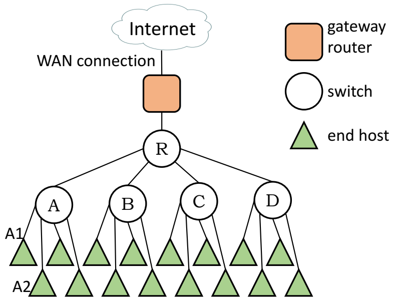

Fig. 9: Topology for the Internet content retrieval experiment

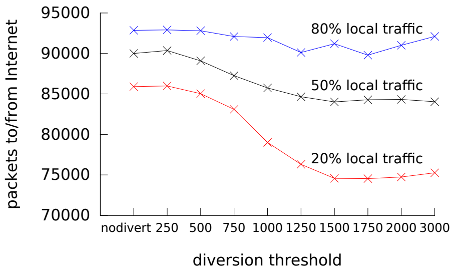

Fig. 10: WAN connection utilization

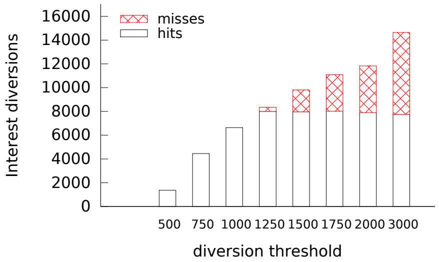

Fig. 11: Interest diversion hits and misses, 50% local traffic

only a marginal difference in off-path cache hits. Therefore, taking both WAN connection utilization and internal processing overhead into account, for best performance, the diversion threshold should be set to a value around 75% of the CS capacity.

## VII. Conclusion

This paper examines how broadcast-based self-learning can be applied to Named Data Networking. We discussed two previously overlooked issues, namely, the FIB granularity problem and the trust model for prefix announcements. We also proposed a specific NDN self-learning design for local-area switched Ethernet, which can build forwarding tables in the data plane with low overhead, recover quickly from link failures, and efficiently utilize off-path caches for Internet contents.

Emulation showed that NDN self-learning consumes 68% less bandwidth than RONR with file access traffic due to the more accurate prefixes learned from prefix announcements. We also demonstrated that, due to the use of Nacks, NDN self-learning can recover from link failues without waiting for FIB entry expiration or incurring the overhead of FSTP convergence. Our Interest diversion design was shown to reduce bandwidth usage on the WAN connection by up to 13% (when 80% of LAN traffic was composed of Internet content retrieval).

In the future, we plan to extend the NDN self-learning design to incorporate congestion control, and apply the design to wireless networks and data centers.

## VIII. Acknowledgement

We would like to thank the anonymous reviewers for their comments. This material is based upon work supported by the National Science Foundation under Grant No. 1345142, 1513505, and 1629009. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

## References

[1] C. Perkins, E. Belding-Royer, and S. Das, "Ad hoc On-Demand Distance Vector (AODV) Routing," RFC 3561 (Experimental).

[2] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, kc claffy, P. Crowley, C. Papadopoulos, L. Wang, and B. Zhang, "Named Data Networking," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, Jul. 2014.

[3] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking Named Content," in *CoNEXT*, 2009.

[4] M. Meisel, V. Pappas, and L. Zhang, "Listen First, Broadcast Later: Topology-Agnostic Forwarding under High Dynamics," in *ITA 2010*.

[5] C. Partridge, R. Walsh, M. Gillen, G. Lauer, J. Lowry, W. T. Strayer, D. Kong, D. Levin, J. Loyall, and M. Paulitsch, "A Secure Content Network in Space," ser. CHANTS '12, 2012.

[6] 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 *ICN'2014*.

[7] A. Afanasyev *et al.*, "NFD Developer's Guide," NDN, Technical Report NDN-0021, Revision 7, 2016.

[8] P. Gusev, Z. Wang, J. Burke, L. Zhang, T. Yoneda, R. Ohnishi, and E. Muramoto, "Real-Time Streaming Data Delivery over Named Data Networking," *IEICE TRANSACTIONS on Communications*, 2016.

[9] G. Grassi, D. Pesavento, G. Pau, L. Zhang, and S. Fdida, "Navigo: Interest forwarding by geolocations in vehicular named data networking," in *WoWMoM 2015*.

[10] V. Lehman, A. K. M. M. Hoque, Y. Yu, L. Wang, B. Zhang, and L. Zhang, "A Secure Link State Routing Protocol for NDN," NDN, Tech. Rep. NDN-0037, 2016.

[11] Y. Yu, A. Afanasyev, D. Clark, k. claffy, V. Jacobson, and L. Zhang, "Schematizing trust in named data networking," in *ACM-ICN*, 2015.

[12] S. DiBenedetto and C. Papadopoulos, "Mitigating poisoned content with forwarding strategy," in *INFOCOM 2016*.

[13] D. Katz and D. Ward, "Bidirectional Forwarding Detection (BFD)," RFC 5880 (Proposed Standard).

[14] C. Yi, A. Afanasyev, I. Moiseenko, L. Wang, B. Zhang, and L. Zhang, "A Case for Stateful Forwarding Plane," *Comput. Commun.*, vol. 36, no. 7, Apr. 2013.

[15] V. Paxson, M. Allman, J. Chu, and M. Sargent, "Computing TCP's Retransmission Timer," RFC 6298 (Proposed Standard).

[16] S. Vusirikala, S. Mastorakis, A. Afanasyev, and L. Zhang, "Hop-by-hop best effort link layer reliability in Named Data Networking," NDN, Tech. Rep. NDN-0041, 2016.

[17] E. J. Rosensweig and J. Kurose, "Breadcrumbs: Efficient, best-effort content location in cache networks," in *IEEE INFOCOM 2009*.

[18] O. Ascigil, V. Sourlas, I. Psaras, and G. Pavlou, "Opportunistic off-path content discovery in information-centric networks," in *LANMAN 2016*.

[19] B. Lantz, B. Heller, and N. McKeown, "A Network in a Laptop: Rapid Prototyping for Software-defined Networks," in *HotNets-IX*.

[20] M. Goldweber and R. Davoli, "VDE: An Emulation Environment for Supporting Computer Networking Courses," *SIGCSE Bull.*, vol. 40, no. 3, Jun. 2008.

[21] D. Ellard and M. Seltzer, "New NFS Tracing Tools and Techniques for System Analysis," in *LISA 2003*.

[22] Y. Yu, A. Afanasyev, Z. Zhu, and L. Zhang, "NDN Technical Memo: Naming Conventions," NDN, Tech. Rep. NDN-0023, 2014.

[23] Y. Liu, F. Li, L. Guo, B. Shen, S. Chen, and Y. Lan, "Measurement and analysis of an internet streaming service to mobile devices," *IEEE Transactions on Parallel and Distributed Systems*, 2013.