Computer Communications 36 (2013) 779–791

Contents lists available at SciVerse ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

A case for stateful forwarding plane

Cheng Yi [a], Alexander Afanasyev [b,*], Ilya Moiseenko [b], Lan Wang [c], Beichuan Zhang [a], Lixia Zhang [b]

[a] The University of Arizona, P.O. Box 210077, Tucson, AZ 85721-0077, USA
[b] University of California, Los Angeles, 4732 Boelter Hall, Los Angeles, CA 90095, USA
[c] The University of Memphis, Dunn Hall 209, Memphis, TN 38152-3240, USA

ARTICLE INFO

Article history:
Available online 29 January 2013

Keywords:
NDN
Forwarding plane
Adaptive forwarding

ABSTRACT

In Named Data Networking (NDN), packets carry data names instead of source and destination addresses. This paradigm shift leads to a new network forwarding plane: data consumers send *Interest* packets to request desired data, routers forward Interest packets and maintain the state of all pending Interests, which is then used to guide *Data* packets back to the consumers. Maintaining the pending Interest state, together with the two-way Interest and Data exchange, enables NDN routers' *forwarding* process to measure performance of different paths, quickly detect failures and retry alternative paths. In this paper we describe an initial design of NDN's forwarding plane and evaluate its data delivery performance under adverse conditions. Our results show that this stateful forwarding plane can successfully circumvent prefix hijackers, avoid failed links, and utilize multiple paths to mitigate congestion. We also compare NDN's performance with that of IP-based solutions to highlight the advantages of a stateful forwarding plane.

1. Introduction

A network's architecture design determines the shape and form of its forwarding mechanism. Today's IP Internet accomplishes packet delivery in two phases. At the *routing plane*, routers exchange routing updates and select the best routes to construct the forwarding table (FIB). At the *forwarding plane*, routers forward packets strictly following the FIB. Thus, IP routing is stateful and adaptive, while IP forwarding is stateless and has no adaptability of its own. This *smart routing, dumb forwarding* approach places the responsibility of robust data delivery solely on the routing system. Consequently IP's Routing plane is also referred to as the *control plane*, and its forwarding plane the *data plane*.

As a newly proposed Internet architecture, Named Data Networking (NDN) inherits the hourglass shape of the IP architecture, but replaces IP's host-to-host data delivery model at the hourglass thin waist by a data retrieval model [1,2]. NDN packets carry data names rather than source and destination addresses. Data consumers express Interests in the form of desired data names, without specifying where the data may be located. Routers satisfy the Interests by retrieving the data, which are bound to the names by cryptographic signatures, from router caches, intermediate data repositories, or original data producers. While routing in an NDN network serves the same purpose as in an IP network, i.e., computing routing tables to be used in forwarding NDN's Interest packets, the forwarding plane in an NDN network is split into a two-step process: consumers first send out *Interest* packets, then *Data* packets flow back along the same path in the reverse direction. Routers keep state of pending Interests to guide Data packets back to requesting consumers.

Obvious benefits of NDN's forwarding plane include built-in network caching and multicast data delivery. A less obvious but equally important benefit is its *adaptive forwarding* enabled by the state maintained at routers. By recording pending Interests and observing Data packets coming back, each NDN router can measure packet delivery performance (e.g., round-trip time and throughput), detect problems that lead to packet losses (e.g., link failures or congestion), and utilize multiple alternative paths to bypass problematic areas. With such an intelligent and adaptive forwarding plane, the routing plane in an NDN network only needs to disseminate long-term changes in topology and policy, without having to deal with short-term churns.

The seminal paper by Jacobson et al. [1] sketched out a blueprint of the overall NDN architecture, however the operations of its forwarding plane are not fully explained and the design specifics remain to be filled in. Our main goal in this paper is to explore the design space and identify critical research issues by sketching out an initial design of NDN's forwarding plane and evaluating its data delivery performance under adverse conditions.

The contributions of this paper are mainly twofold. First, we propose a concrete design of NDN's forwarding plane which includes specific mechanisms for routers to keep track of data delivery performance, control network load, and retry alternative paths.

* Corresponding author. Tel.: +1 3109097675.
E-mail addresses: yic@cs.arizona.edu (C. Yi), afanasev@cs.ucla.edu (A. Afanasyev), iliamo@cs.ucla.edu (I. Moiseenko), lanwang@memphis.edu (L. Wang), bzhang@cs.arizona.edu (B. Zhang), lixia@cs.ucla.edu (L. Zhang).

0140-3664/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.comcom.2013.01.005

We also introduce a new Interest NACK mechanism to enable NDN router to perform quick and informed recovery from network problems. Second, we use simulations to evaluate the performance of our design in terms of its resiliency against prefix hijacks, link failures, and network congestion. We compare the performance of our stateful forwarding plane with that of both IP and an IP-based multipath forwarding solution, Path Splicing [3], to identify the fundamental differences between stateless and stateful forwarding planes.

The rest of the paper is organized as follows. Section 2 presents an overview of NDN's forwarding plane. Section 3 describes our design of NDN's adaptive forwarding. Section 4 describes the simulation studies and analyze the results. In Section 5 we discuss the benefits and costs of NDN's stateful forwarding plane. We summarize related work in Section 6, and conclude the paper in Section 7.

2. Overview of NDN's Forwarding Plane

In this section we briefly introduce NDN with a focus on its stateful forwarding plane. NDN is a receiver-driven, data-centric communication protocol. All communication in NDN is performed using two distinct types of packets: *Interest* and *Data*. Both types of packets carry a *name*, which uniquely identifies a piece of data that can be carried in one Data packet. Data names in NDN are hierarchically structured, for example `/arizona.edu/cs/chengyi/papers/forwarding2012.pdf/seg1`. Network routing protocols will distribute name prefixes, such as `/arizona.edu/`, in a way similar to distributing IP prefixes in today's Internet.

To retrieve Data, a consumer puts the name of desired data into an Interest packet and sends it to the network. Routers use this name to forward the Interest towards the data producer, and the Data packet whose name matches the name in the Interest is returned to the consumer. All data packets carry a signature that binds the name to the data.

Similar to IP packet delivery, an NDN network performs best effort data retrieval. An Interest or Data packet can be lost, and it is the end consumer's responsibility to retransmit the Interest if it does not receive the desired Data after expected round trip time (RTT) and it still wants the Data.[1] However, unlike IP's location-centric approach to data delivery, NDN packets carry data names instead of addresses. This basic difference in design leads to two profound differences in data delivery operations. First, although the name in an Interest packet is used to guide its forwarding, in a way similar to how a destination address is used to guide the forwarding of an IP packet, the Interest may cross a copy of the requested Data at an intermediate router or data repository and bring the Data back, while an IP packet is always delivered to the destination (if not dropped along the way). Second, an Interest packet carries neither address nor name to identify the requesting consumer that can be used to return the requested Data packet. Instead NDN routers keep track of incoming interfaces for each forwarded Interest (a pending Interest) and use this information to bring matched Data packets back to consumers.

In addition to the data name, each Interest packet also carries a random nonce generated by the consumer. A router remembers both the name and nonce of each received Interest, hence it can tell whether a newly arrived Interest carrying the same name as a previously received Interest is from a different consumer, or a previously forwarded Interest looped back (in which case the Interest is dropped). Therefore Interest packets cannot loop. Because Data packets follow the reverse path of the corresponding Interest packets, they do not loop either. This enables routers to freely retry multiple alternative paths in Interest forwarding. Notice that retry should be limited in scope and duration because (1) routers are not ultimately responsible for getting the Data, and (2) if all routers along the path perform retry, it may potentially lead to Interest explosion and significant overhead.

[1] In this paper the term **retransmit** is used exclusively for end consumers re-expressing Interests; another term **retry** is used when intermediate routers explore alternative paths after network problems are detected.

2.1. Forwarding Process

Each NDN router maintains three major data structures: a *Content Store* for temporary caching of received Data packets, a *Pending Interest Table (PIT)*, and a *forwarding table (FIB)* (see Fig. 1). By its name, each PIT entry records an Interest packet that has been forwarded, waiting for the Data packet to return. The entry records the name, the incoming interface (s) of the Interest (s), and the outgoing interface (s) the Interest has been forwarded to. An NDN router's FIB is roughly similar to the FIB in an IP router except that it contains name prefixes instead of IP address prefixes, and it may show multiple interfaces for a given name prefix (see Section 3.3). In addition, each NDN router has a *strategy module* that makes forwarding decisions for each Interest packet (see Section 3.5).

When a router receives an Interest packet, it first checks whether there is a matching Data in its Content Store. If a match is found, the Data is sent back to the incoming interface of the Interest packet. If not, the Interest name is checked against the entries in the PIT. If the name exists in the PIT already, then it can be either a duplicate Interest (i.e., its nonce is remembered in PIT entry) that should be dropped, an Interest retransmitted by the consumer that may need to be forwarded using a different outgoing interface, or an Interest from another consumer asking for the same Data which requires the incoming interface of this Interest to be added to the existing PIT entry. If the name does not exist in the PIT, the Interest is added into the PIT and further forwarded to the interface chosen by the strategy module.

When a Data packet is received, its name is used to look up the PIT. If a matching PIT entry is found, the router sends the Data packet to the interface (s) from which the Interest was received, caches the data in the Content Store, and removes the PIT entry. Otherwise, the Data packet is unsolicited and discarded. Each Interest also has an associated lifetime; the PIT entry is removed when the lifetime expires.

2.2. Datagram State

An NDN router maintains an entry in its PIT for *every* pending Interest packet, thus we say the router contains "datagram state." This state leads to a closed-loop, two-way symmetric packet flow:

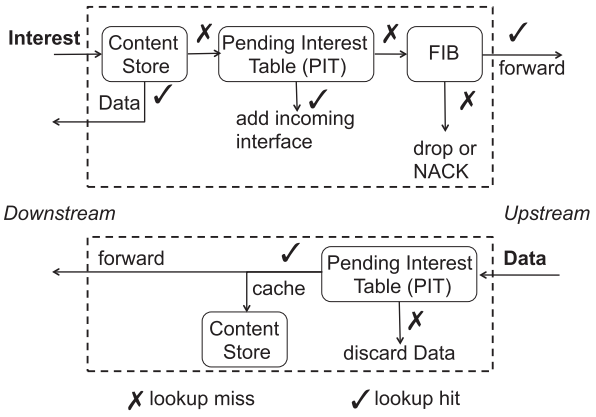


Fig. 1. Interest and Data processing in NDN.

over each link, every Interest packet pulls back exactly one Data packet, maintaining one-on-one flow balance, except in (rare) cases where packets get lost or matching data does not exist.

It is worth noting that NDN's datagram state differs in fundamental ways from the virtual circuit state for ATM or MPLS. First, a virtual circuit sets up a single path between an ingress-egress router pair; when it breaks, the state has to be recovered for the entire path. Second, a virtual circuit pins down the path to be used for packet forwarding; if any of the links along the path gets overloaded due to traffic dynamics, packets on the same virtual circuit cannot be diverted to adapt to the load changes. In contrast, NDN's datagram state is per-Interest, per-hop. At each hop, the router makes its own decision on where to forward an Interest. When a router crashes or a link fails, the failure only affects the Interests at that specific location; the previous hop routers can quickly detect the failure and get around the problematic area.

3. Adaptive forwarding

In this section we describe an initial design on how to utilize NDN routers' datagram state to build an intelligent and adaptive forwarding plane. The main goals are to retrieve data via the best performing path (s), and to quickly detect any packet delivery problem and recover from them.

3.1. Interest NACK

In the original sketch of NDN [1], routers discover failures by timeout only. More specifically, when a router *N* forwards an Interest, it starts a timer based on the estimated RTT to the data producer. If the corresponding Data packet comes back before the timer expires, the RTT is updated; otherwise *N* will try alternative path if one exists, or otherwise give up. However, this timer-based problem detection can be relatively slow. Furthermore, when the router *N* exhausts its options and gives up, the unsatisfied Interest (which we call the *dangling state*) is left on the PIT of those routers between the consumer and *N* that the Interest has traveled through, until the Interest's lifetime expires. Such dangling state can potentially block other consumers from getting the same data, since the intermediate routers believe that they have already forwarded the Interest and just wait for the Data to return.

We address these issues by introducing *Interest NACK*. When an NDN node N_u can neither satisfy nor further forward an Interest, it sends an Interest NACK back to the downstream node N_d. If N_d has exhausted all its own forwarding options, it will send a NACK further downstream. Note that Interest packets flow from downstream node to upstream node, Data packets flow from upstream node to downstream node, and Interest NACKs are always sent from upstream to downstream.

An Interest NACK carries the same name and nonce as the original Interest, plus a NACK code explaining why the Interest cannot be satisfied or forwarded so that proper actions can be taken accordingly. Below are the NACK codes in our current design; additional codes may be added as the need arises.

- *Duplicate*: A pending Interest with identical name and nonce has been received earlier by the upstream node. This occurs if the Interest is looped back to the upstream node, or if some node forwarded multiple copies of the same Interest that happen to meet at the upstream node.
- *Congestion*: The upstream node cannot forward the Interest further or return the Data back due to congestion. See Section 3.4 for details.
- *No Data*: The upstream node (which can be either a router or the producer) does not have the requested data and has no path to forward the Interest.

In the absence of packet losses, every pending Interest is consumed by either a returned Data packet or a NACK. Returning NACKs brings two benefits to the system: it cleans up the pending Interest state much faster than waiting for timeout, and it allows the downstream nodes to learn the specific cause of a NACK, so that they can take informed recovery actions. Note that an Interest NACK is different from an ICMP message; the former goes to the previous hop while the latter is sent to the source host, hence their effects are entirely different.

3.2. PIT

PIT maintains datagram forwarding state (Fig. 2). A PIT entry is created for each requested name. It contains a list of nonces that have been seen for that name, a list of incoming interfaces from which Interests for that name have been received, as well as a list of outgoing interfaces to which the Interest has been forwarded. In a PIT entry, each incoming interface records the longest Interest lifetime it has received; when the lifetime expires the incoming interface is removed from the PIT entry, and the entire PIT entry is removed when all its incoming interfaces have been removed. Each outgoing interface records the time when the Interest is forwarded via this interface, so that when Data packet returns RTT can be computed. The RTT measurement is then used to update the RTT estimate for the corresponding name prefix stored in the FIB (Section 3.3).

3.3. FIB

NDN FIB differs from IP FIB in two fundamental ways. First, an IP FIB entry usually contains a single best next-hop, while an NDN FIB entry contains a ranked list of *multiple* interfaces. Second, an IP FIB entry contains nothing but the next-hop information, while an NDN FIB entry records information from both routing and forwarding planes to support adaptive forwarding decisions (see Fig. 2).

3.3.1. Routing plane information

FIB entries are added for all name prefixes announced in routing. When a name prefix disappears from routing, it is not immediately removed from the FIB, but kept for a *stale time* period or longer, if Interests under that prefix continue to be satisfied. This helps reduce unreachability caused by routing convergence, when some reachable prefixes may undergo temporary withdrawals.

For each name prefix, its FIB entry lists all interfaces that are allowed by routing policy, together with their associated preferences. Routing preference reflects routing policy as well as path cost, typically calculated using static link metrics; it is one of the inputs that we use to rank the interfaces.

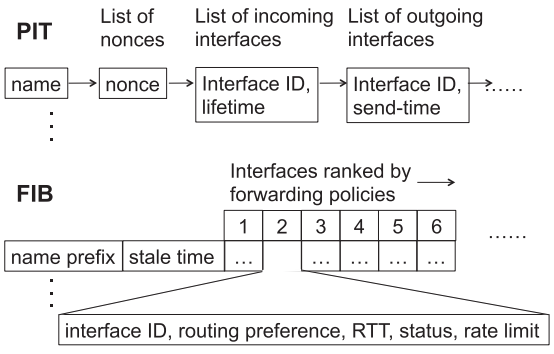


Fig. 2. Forwarding State in PIT and FIB.

3.3.2. Forwarding Performance Information

A FIB entry records the working status of each interface with regard to data retrieval. Interface status is per-name-prefix-per-interface. To search for the best way to represent this status, we start by experimenting with a simple coloring scheme:

- Green: the interface can bring data back.
- Yellow: it is unknown whether the interface may bring data back.
- Red: the interface cannot bring data back.

When a new FIB entry is created or a new interface is added to a FIB entry, the interface's initial status is Yellow. It turns Green when Data flows back from that interface. A Green interface turns Yellow when a pending Interest times out (i.e., no Data comes back within the expected time),[2] after Data ceases flowing for a certain amount of time, or upon the receipt of a "No Data" or "Duplicate" NACK. An interface is marked Red when it goes down. A "Congestion" NACK does not change the color of an interface but reduces the rate Interests can be sent through that interface (see Section 3.4). Green interfaces are always preferred over Yellow ones; Red interfaces are not used to forward Interests.

A FIB entry also maintains a per interface estimate of the RTT to retrieve data. It is a moving average of RTT samples taken every time a Data packet is received over the corresponding interface. This RTT estimate is used in setting up a *retry-timer*, which serves two purposes: (1) before the timer expires, subsequent Interests carrying a name that already exists in PIT will be suppressed because the router is waiting for Data to be retrieved by the previously forwarded Interest; (2) after the timer expires, the router will stop trying alternative interfaces upon receiving a NACK, to limit the overhead caused by local retry.[3]

3.3.3. Interface ranking

Interfaces in a FIB entry are ranked in order to help forwarding strategy choose the best interface (s) to use. When a router learns a new name prefix from routing, it ranks the interfaces for this prefix based on routing preference, since no forwarding performance has been observed yet. When information about forwarding performance becomes available, *forwarding policy* adjusts the interface ranking by taking into consideration both types of information.

A wide variety of forwarding policies can be supported in an NDN network. For example, if the policy is simply "follow routing", the interface ranking will be solely determined by routing preference; if the policy is "the sooner the better", an interface with smaller RTT will be ranked higher. Yet another example is to give higher preference to the current working path, which helps ensuring performance stability experienced by applications. Note that *routing policies* determine which routes to be made available to the forwarding plane. In the case of BGP, routing policies are reflected in routing announcements, which propagate from data producers to consumers. *Forwarding policies*, on the other hand, determine which routes actually get used and in which order. They are reflected in Interest forwarding, which go from the consumers towards producers.[4]

3.4. Rate limiting and congestion control

The one-to-one flow balance between Interest and Data packets gives NDN an effective way to prevent congestion inside networks. By pacing Interests sent to the upstream direction (towards producer) of a link, one can prevent congestion (caused by Data) on the downstream direction of the link.

We set a limit on how fast Interest packets can be forwarded over an interface and experimented with a simple calculation of the Interest rate limit: $L_i = \alpha \times C_i/\overline{S}_i$, where L_i is the Interest rate limit of interface i, C_i is the upstream link capacity of i, $\overline{S}_i$ is an estimate of the size of the Data packets that have been received over i, and α is a configurable parameter. The ratio $C_i/\overline{S}_i$ is the maximum data rate that is allowed from upstream measured in packets per second (pps), which should be the same as the maximum Interest rate going upstream.[5] The coefficient α is used to compensate for errors in the calculations (e.g., imprecise data size estimate, link and network layer overheads).

Let us assume there two two neighbor nodes, N_u the upstream and N_d the downstream. N_d computes L_i and sends Interests to N_u no faster than L_i, which prevents the link between the two nodes from being congested. N_u will also respect a similar rate limit when it forwards Interests further upstream. If N_u's upstream link has less capacity than the link between N_d and N_u, it is possible that N_d sends more than N_u can forward. In this case, N_u will send Congestion NACKs back to N_d. Each node maintains a rate limit, $L_{i,n}$, for each outgoing interface i and each name prefix n, stored in the corresponding FIB entry (see Fig. 2). $L_{i,n}$ is reduced when a Congestion ACK is received, and increased when a Data is received. The specific adjustment algorithm is an area of our current research; one option is to use an Additive-Increase-Multiplicative-Decrease (AIMD) algorithm similar to TCP.

There are two other scenarios in which Congestion NACKs may be sent. Because L_i is estimated based on observed Data packet sizes, it may not accurately predict the actual link bandwidth that will be consumed by current and future Data packets. Since the upstream node knows what Data packets are waiting to be sent, upon receiving an Interest from downstream, it can preventatively return a Congestion NACK if it sees that the link $N_u \rightarrow N_d$ would be congested. Also, when N_u receives a Data packet from its upstream, but foresees imminent congestion if it sends these Data packets to N_d, it can send Congestion NACKs instead. As demonstrated in Early Random Detection (RED) [4], monitoring average queue length is an effective way to detect imminent congestion. In both case, the downstream node will reduce its $L_{i,n}$ so that Interests under prefix n will be sent to interface i at a lower rate.

In summary, we use per interface rate limit L_i to avoid congestion on a local outbound interface, and per prefix-interface rate limit $L_{i,n}$ to control congestion along a path (including a local interface) used by Interests under a particular name prefix. When neither L_i nor $L_{i,n}$ is reached, the interface i is said to be *available* for forwarding Interests under name prefix n, otherwise *unavailable*.

3.5. Forwarding strategy

Given the information stored in PIT and FIB, a router's strategy module determines when and which interface to use to forward an Interest, making forwarding decisions adaptive to network conditions. Our initial design includes the handling of new Interests, retransmitted Interests, Interest NACKs, and proactive

[2] When there is a timeout on the interface, it indicates something may be wrong with the current path. The interface is marked Yellow, even though the problem may or may not be due to the local interface. If any other Green interface exists, it will be used. If no Green interface exists, next incoming Interest will be still forwarded to the current interface if it is the best option.

[3] We are also investigating other solutions to limit the total resources spent on forwarding each Interest, one option is to add a hop count (TTL) field into each Interest packet and decrease it at every hop; if the Interest results in a NACK, the NACK will echo back the remaining TTL value which is then used in the retry.

[4] It is conceivable that forwarding policies could also decide whether the outgoing interfaces for a given name prefix should be, or should not be, limited to those learned from the routing protocols. An upstream router can easily reject an Interest it deems violating its policy by sending a NACK. We plan to look into this issue in our future study.

[5] A slightly more complicated formula can be obtained if we take the sizes of both Interest and Data packets into consideration.

probing of interfaces. The overall Interest processing mechanism is illustrated by Pseudocode 1.

New Interest: When a newly arrived Interest does not find a match in either Content Store or PIT, a new PIT entry will be created. The new Interest is then forwarded to the highest-ranked available Green interface if one exists, otherwise the highest-ranked available Yellow interface will be used (see Pseudocode 2). If there is no available interface, the forwarding strategy returns a NACK with the code "Congestion." When the router forwards the Interest, it starts a retry-timer, which is set to a small value at the time scale of RTT plus variance. If the router receives an Interest NACK before the retry-timer expires, it will try alternative interfaces to retrieve the Data.

Subsequent Interest: If an Interest matches an existing PIT entry, and its nonce does not exist in the nonce list, this Interest is considered a subsequent Interest. A subsequent Interest can be a retransmission from the same consumer, or originated from a different consumer requesting the same piece of Data. When a subsequent Interest is received before the retry-timer expires, it will not be forwarded because the router is still waiting for Data to be brought back by a previously forwarded Interest. Otherwise, this subsequent Interest will trigger the router to retry the Interest and start the retry-timer.

Interest NACK: When an Interest for data name N is returned in the form of a NACK, if the retry-timer is still running, a router will send an Interest with the same name and nonce to the next highest-ranked available interface (see Pseudocode 3). Ideally, we want routers to try a few alternatives but not for too long (the application may have moved on without the Data) nor consuming too much network resource. After the retry-timer expires, the PIT entry for N is kept until the Interest's life time expires, during this time retry can be triggered by a subsequent Interest of the same name.

In all the above situations, if a router needs to forward an Interest but it has no available Green or Yellow interface left that has not been tried, it will give up, delete the PIT entry and send a "Congestion" or "No Data" NACK back to the downstream router (s). Routers perform best effort to get around forwarding problems through local retries, however consumers are ultimately responsible for re-express the Interest if they still want the data.

Pseudocode 1 Interest processing

```
1: function Process (Interest)
2:   Name ← Interest.Name
3:   if Data ← ContentStore.Find (Name) then
4:     Return (Data)
5:   else if PitEntry ← PIT.Find (Name) then
6:     if Interest.Nonce ∈ PitEntry.NonceList then
7:       Return Interest NACK (Duplicate)
8:       Stop processing
9:     end if
10:    if PitEntry.RetryTimer is expired then
11:      Forward (Interest, PitEntry)
12:      Stop processing
13:    end if
14:    Add Interest.Interface to PitEntry.Incoming
15:  else
16:    PitEntry ← PIT.Create (Interest)
17:    PitEntry.Incoming ← Interest.Interface
18:    Forward (Interest, PitEntry)
19: end if
20: end function
```

Forwarding strategy

```
1: function Forward (Interest, PitEntry)
2:   if FibEntry ← FIB.Find (Interest.Name) then
3:     for each interface in FibEntry by rank do
4:       if interface ∉ PitEntry.Outgoing and
5:          interface ∉ PitEntry.Incoming then
6:         if interface.Avaialble then
7:           Set PitEntry.RetryTimer
8:           Transmit (interface, Interest)
9:           Add interface to PitEntry.Outgoing
10:          if ProbingDue (FibEntry) then
11:            Probe (Interest, PitEntry)
12:          end if
13:          Stop processing
14:        end if
15:      end if
16:    end for
17:    Return Interest NACK (Congestion)
18:    GiveUp (Interest)
19:  else
20:    Return Interest NACK (No Data)
21:    GiveUp (Interest)
22:  end if
23: end function
```

Interest NACK processing (retry)

```
1: function Process (NACK)
2:   PitEntry ← PIT.Find (NACK.Name)
3:   if PitEntry ≡ ∅ or
4:     PitEntry.RetryTimer expired or
5:     NACK.Nonce ∉ PitEntry.NonceList
6:   then
7:     Stop processing
8:   end if
9:   Forward (NACK.Interest, PitEntry)
10: end function
```

Interface probing: By default the forwarding strategy prefers currently working (Green) interfaces. It also performs periodic probing through Yellow faces in order to discover alternative available paths or paths with better performance that may appear after a link failure recovery, or if there is a cache closer than the producer.

A router checks whether probing is due every time when it successfully forwards an Interest in any of the three aforementioned situations. When probing is due, it will pick an available Yellow interface that has not been tried before, and forward a copy of the Interest. Note that the router will not do anything if there is no interface left for probing.

Interface probing helps discover availability and performance information for alternative paths, but also results in duplicate Data packet returns. One can control this overhead by limiting the probing frequency, sending a probing Interest after either a certain amount of time has passed or a certain number of packets have been forwarded. We are still working on the exact design and analysis of the probing frequency.

4. Simulation study

In this section we use simulations to evaluate how well NDN's adaptive forwarding plane works and whether it achieves robust

packet delivery under adverse conditions. We contrast with the behavior of IP to illustrate the difference between NDN's *stateful* forwarding plane and the traditional IP's *stateless* forwarding plane. We also include in the comparison with Path Splicing [3], an adaptive multipath enhancement to IP, to observe the differences in the performance between NDN and Path Splicing and to understand the underlying causes of these differences.

We implemented a basic NDN forwarding plane and forwarding scheme in the newly developed ndnSIM, a NS-3 based NDN simulator [5,6]. We also implemented Path Splicing in NS-3 according to [3]. In the rest of this section, we first present a brief description of Path Splicing, then three network fault scenarios used in the simulation, followed by our simulation results.

4.1. Path splicing

Path Splicing enhances enables source hosts to utilize multiple paths in IP packet delivery. In Path Splicing, each router maintains multiple routing tables, called *slices*. All the routers in a network are preconfigured with the same number of slices. A router computes its first routing table (the original slice) by using the standard routing protocol metrics, it then computes the rest of the slices by using the same topology and the same shortest-path algorithm but different sets of link weights, which are generated by randomly perturbing the original set of link weights learned from the routing protocol.

When a host sends a packet, it adds an ordered list of *tags* to the packet header. Each tag is an index to the slice to be used at each hop, and routers forward the packets according to the tags. A tag is removed from the list after it is used; when a packet's tag list becomes empty, routers will use the original slice to forward the packet. End-hosts can choose a *different* path for a packet by tagging it differently, although the hosts do not know the exact path the packet may take.

When a host detects a network fault, e.g. a packet loss, it retransmits the packet by using a different tag list. The recommended operation is for the end-host to examine the current list of tags and change each tag with a probability of 0.5. If a tag is to be changed, the new tag will be randomly chosen from the available slice numbers. For a client/server application to successfully retrieve a data packet, the tags on request and reply packets must both identify a working path. In simulating Path Splicing, unless otherwise specified, we use 10 slices by default and allow up to 20 retransmissions.

On the surface, Path Splicing seems similar to NDN forwarding in that, once hosts detect network delivery problems, they attempt to get around the problems by trying different paths. The additional state, i.e., multiple slices installed at each Path Splicing routers, enables the path adjustments by end hosts. However, there exist two fundamental differences between the two. First, the adaptability in Path Splicing can only be done by end-hosts, as opposed to by any nodes in an NDN network. Second, given the end hosts know neither the network topology nor the problem location, they adjust the path by random selections, as opposed to the informed decision by NDN nodes based on their observed performance and feedbacks. These functional differences lead to significant performance differences in the simulation results as we present below.

4.2. Simulation scenarios and setup

We examine a network's packet delivery performance under three fault scenarios: (1) *prefix hijack*, in which an attacker announces the victim's prefix and drops the traffic; (2) *link failure*, in which links randomly fail by certain probability; and (3) *congestion*, in which some links do not have enough bandwidth to carry the offered traffic. We evaluated the performance of NDN and IP under all the three scenarios, and evaluate Path Splicing in prefix hijack and link failure scenarios.[6] Since we focus our evaluation on the effectiveness of stageful adaptive forwarding, we disabled NDN's in-network caching in the simulation.

Except for a simple 6-node topology used in one congestion simulation, all other simulations are done using the Sprint PoP level topology [7],[7] the same topology used in [3] to show the improvement of Path Splicing over native IP. This topology has 52 nodes and 84 links. It should be noted that this topology contains 19 single-homed nodes. When their links fail, these single-homed nodes lose connectivity, making them vulnerable to failures as one can see from the simulation results.

In all simulations we assume that each IP (NDN) router announces one IP (name) prefix, and we precompute the best paths and install in each router's FIB. For IP, a single shortest path is installed for each IP prefix at a router. For Path Splicing, a number (default is 10) of slices are installed per IP prefix. For NDN, each router maintains a list of all outgoing interfaces ranked by the routing path length for each name prefix in the FIB. In prefix hijacking and congestion simulations, routing protocol do not react. In link failure simulations, we measure packet delivery performance before the routing protocol adapts to the failure, as is done in [3]. Our goal is to measure how well, or poorly, the forwarding plane can perform before routing converges.

In each simulation run, we create fault scenarios shortly after the simulation starts and observe the behavior of each scheme. Since NDN clients retrieve data by sending an Interest packet first, for fair comparison we use the same traffic pattern in IP and Path Splicing simulations, i.e., we run client/server applications where a client first sends a request to the server, which then sends a reply data packet back to the client. These request/reply packets have the same sizes as Interest/Data packets in NDN.

4.3. Prefix hijack

In prefix hijack simulations, an attacker announces the victim's prefix and silently drops all data traffic it receives, creating a "blackhole" of victim's traffic as what happened during the well publicized incident of YouTube's prefix hijack.[8]

End-hosts can detect the problem when they do not receive the content they are requesting. Traffic is said to be *affected* if it is forwarded towards the attacker, or unaffected if it is still forwarded to the true destination. If affected end-hosts have means to try other paths, they may find an alternative path to reach the true destination [8], in which case we say they have *recovered* from the hijack.

To simulate "blackhole" hijacks, in each simulation run we choose one node as the producer, one as the attacker, and the rest of the nodes as data consumers. The attacker announces to the routing system the IP prefix (for IP and Path Splicing) or the name prefix (for NDN) of the producer. We exhaust all combinations of (consumer, producer, attacker) tuples in the topology. We run NDN, IP, and Path Splicing to see whether the traffic will be affected and whether affected traffic can recover. The results are summarized in Fig. 3.

Let us consider the result for running IP simulation first as it is easier to understand. In an IP network, traffic will be affected if (1) the attacker is on the best path from the source to the destination,

[6] We skip the simulation results for Path Splicing under congestion because it only utilizes one path, thus the performance is no better than that of IP.

[7] We also evaluated NDN performance using much larger topologies and the results are similar to those reported in this paper.

[8] In an NDN network, a hijacker may return bogus Data instead of silently dropping Interest packets. How to handle bogus data attacks is an ongoing effort which is beyond the scope of this paper; we briefly discuss the solution space in Section 5.3.3.

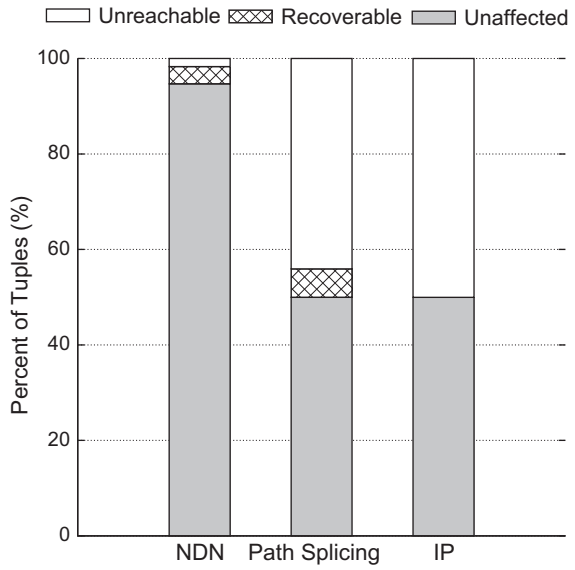


Fig. 3. Reachability during prefix hijack.

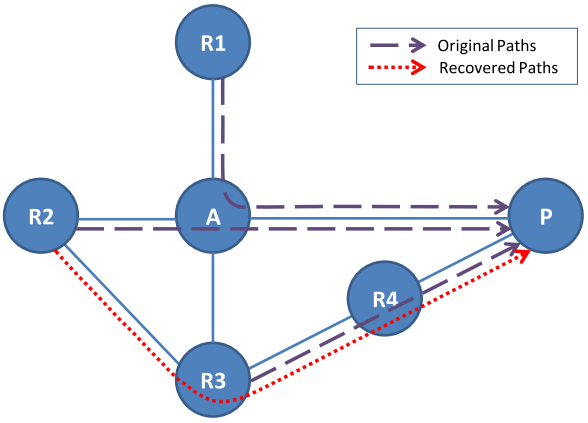


Fig. 4. A prefix hijack example.

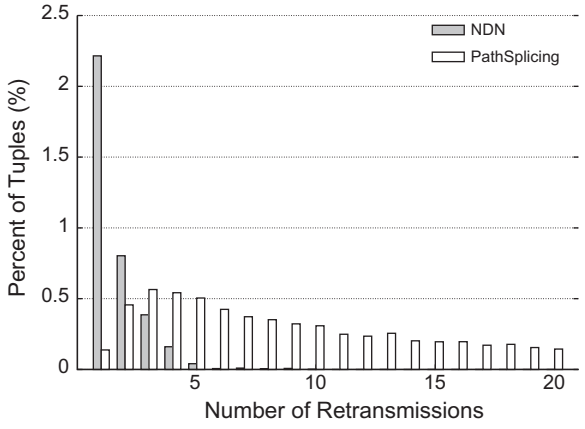


Fig. 5. Retransmissions needed during prefix hijack.

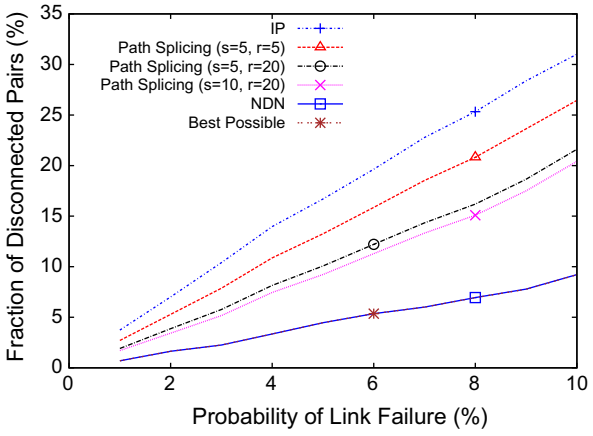


Fig. 6. Reachability after link failures.

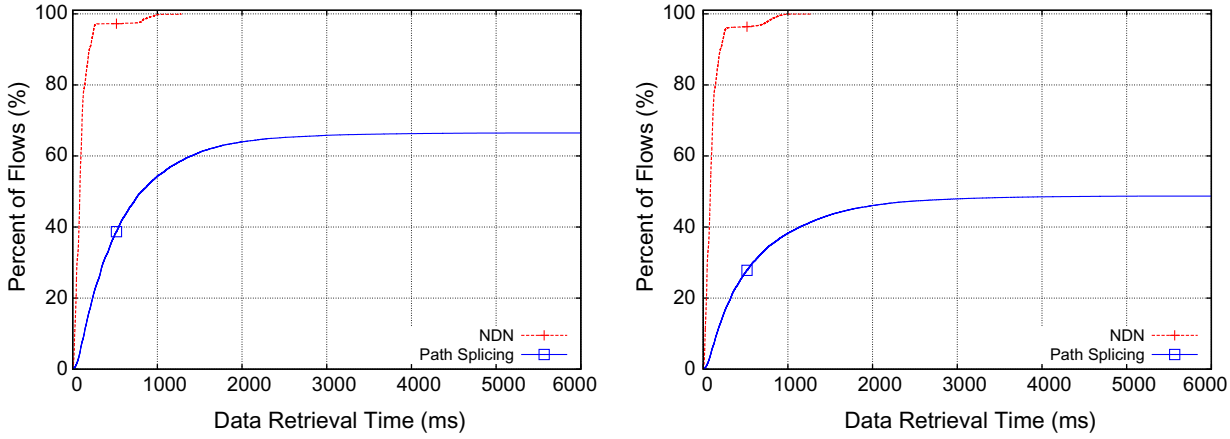
or (2) the attacker is closer to the source than the true destination. For the given topology, in less than 6% of the cases, the attacker is on the best path from the consumer to the producer and simply drops all requests, thus no consumer gets data back. In another 44% of the cases the attacker is closer to the consumer than the producer, thus traffic is affected as well. Note that in total 50% of all the cases traffic is affected and in the other 50% it is not. This is because the traffic that is affected under one producer/attacker pair will become unaffected when the producer and attacker switch roles. Since IP routers strictly follow the paths given by routing protocols, none of the affected traffic is recoverable. The end-hosts, although being able to detect the problem, cannot change the paths their packets take.

For NDN, in the less than 6% of cases where the attacker is on the original best path from the consumer to the producer, all traffic is recoverable except the cases where the consumer or producer is single-homed to the attacker. For the remaining over 94% of the cases, consumer-producer communications are not affected by the hijack. We use a simple topology shown in Fig. 4 to explain NDN's resilience to blackhole hijacks.

In this figure, A is the attacker, P is the producer, and other nodes are good routers. We first consider the routers who do not have the attacker on their shortest path to reach the producer, e.g., R3. When A announces P's name prefix, the routing system would rank the attacker path (R3-A) higher than the producer path (R3–R4-P). With NDN's intelligent forwarding plane, since the existing interface (R3–R4) to the producer has been bringing data back, it is colored Green. A's false routing announcement makes the interface to the attacker (R3-A) ranked higher by routing, but when R3 tries its out by sending an Interest to it now and then, it does not return a Data packet, thus it remains Yellow. Unlike IP routers, NDN routers do not direct traffic to a higher ranked path until it is observed to perform well.

Let us now consider the routers who have the attacker on their shortest path to the producer, e.g., R2. R2's Interest packets will be blackholed by the attacker, thus R2's retry-timer expires without getting data packets back, and interface (R2-A) will be marked Yellow; R2 does not automatically retry alternative interfaces because there is no Interest NACK or any feedback. End consumers will timeout and retransmit the Interests. Upon receiving a retransmitted Interest, if R2's retry-timer has expired, R2 will retry a different interface than the previously failed one. When the retransmitted Interest, following a different path, arrives at the producer and brings back the requested Data packet, R2 will mark the working interface (R2-R3) Green and keep using the working interface.

In simulating Path Splicing under the same hijack attacks, the results show the same 50% of affected traffic as in the IP case. However different from IP, when a client sends out a request packet and times out after RTT, it will retransmit the request with a different tag list, thus the packet will be routed along a different path. If the data packet comes back, the client knows the previous tags have



(a) 1% Link Failure Probability (b) 10% Link Failure Probability

Fig. 7. Data retrieval time under different link failure probability.

worked and will keep using the same tags. Thus, Path Splicing enables end-hosts to influence path selection in an attempt to escape routing hijack. This capability, however, is rather limited because the available options are bounded by the number of precomputed slices, and hosts perform *random* selections of tags because they have no knowledge about the network internals. Thus the recovered traffic is only 6% out of 50% of affected cases, and the random trials can take substantially long time. Fig. 5 shows the number of retransmissions needed to find a working path with each router keeping 10 slices. In most cases consumers in Path Splicing need many retransmissions, while NDN consumers only need a few.

NDN is both faster and more effective in finding working paths because every NDN router is able to make its own informed decisions on which alternative paths to try. These decisions are based on the observation from previously forwarded Interest packets, which is made available by maintaining per-datagram state at each router. In addition, the ability to retry alternative paths by every router whenever a problem is noticed speeds up recovery and does not require any precomputed tables.

4.4. Link failure

Robust packet delivery in the presence of link failures[9] is the classic measure used in Baran's early work to evaluate the resilience of packet switched networks [9]. We simulate link failures as follows. After the initial simulation warm-up, we associate each link with a uniform failure probability and fail links randomly according to this probability, producing one failure scenario. All packets sent over a failed link are dropped. If the original best path from A to B contains at least one failed link, the traffic flow between A and B is considered affected; if the network is able to switch packets from A to B to an alternative working path, we say this traffic flow can recover from the failure. We run NDN, IP, and Path Splicing respectively to see how many flows (i.e., consumer-producer pairs) can recover from given failures. We run each experiment 1000 times (i.e. running each of NDN, IP, and Path Splicing over 1000 randomly generated failure scenarios for each link failure probability) and our results are presented below.

Fig. 6 shows the percentage of host pairs that cannot recover, averaged over 1000 failure scenarios for each link failure probability. The "best possible" curve is the percentage of host pairs that are physically disconnected by the failed links, thus no solution can achieve disconnection ratio below this curve. The NDN curve overlaps with the best possible one, meaning that a consumer is able to retrieve data from a producer as long as *any* working path exists in between. Not only can NDN recover from link failures, it also finds alternative paths quickly. Fig. 7 shows the CDF of data retrieval time from 1000 failure scenarios,[10] which is from the first transmission of a request/Interest by the consumer to the arrival of the requested Data. It includes the time of possible retransmissions by the consumer in Path Splicing and NDN, and router retries in NDN. With 1% link failure probability, the median data retrieval time in NDN is 85 ms, and the 90th-percentile 198 ms; when the link failure probability is 10%, the median is not changed while the 90th-percentile increases slightly to 203 ms. The alternative paths that NDN finds are also of good quality. Fig. 8 shows the CDF of path stretch, which is the path length ratio of the selected path over the shortest path after failures. Under either 1% or 10% link failures, about 60% of paths in the NDN network have stretch of 1, which means that the adaptive forwarding plane found the shortest paths; the 90-percentile of path stretch increases marginally from 1.21 to 1.22 when failure probability increases from 1% to 10%.

NDN's resiliency to failures is due to its fast local recovery. When a link fails, the NDN router will mark this interface Red and try other interfaces following its forwarding strategy. If the router has tried and failed all possible interfaces, it returns a NACK to downstream node, which will then explore its own alternatives. When the Interest brings back Data via a working interface, this interface will be labelled Green and used to forward future Interests. Therefore, network retry starts from where the failure happens and pushes back towards the consumer until a working path is found, if one exists. Since a FIB entry's outgoing interfaces are ordered not only based on routing preference but also observed working status, a router tries most promising interfaces first, which leads to finding working paths sooner and finding good working paths.

In contrast, since IP's forwarding plane has no adaptability of its own, its percentage of disconnected pairs in Fig. 6 reflects the number of host pairs whose shortest paths contain at least one failed link. An IP network relies on routing protocols to handle link failures. Once a policy-compliant working path is found, routing will

[9] A node failure is equivalent to a simultaneous failure of multiple links. Thus, in this paper we do not treat node failures separately.

[10] In Fig. 7 and 8 we only consider packet exchanges that are affected by failures but recoverable (i.e., the failed link is on the original best path, but at least one alternative working path exists after the failure). These figures do not consider IP because although end hosts in an IP network may also retransmit, the packet will still be forwarded along the failed path, thus all affected pairs fail.

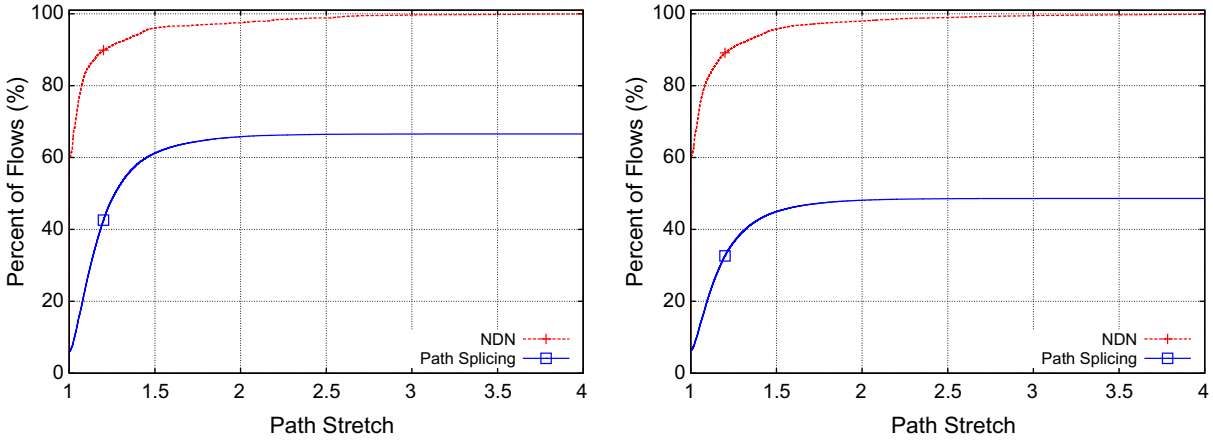


Fig. 8. Path stretch under different link failure probability.

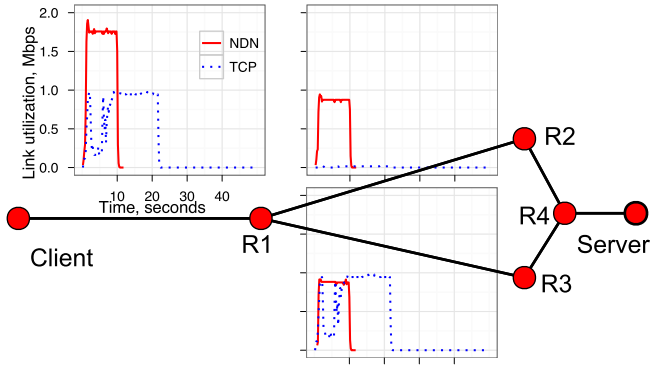


Fig. 9. Link utilization under congestion.

converge and packets will be delivered along the new path. However the convergence process, which includes failure detection, route propagation, route computation, and FIB update, can take time to complete. Measurements have shown that routing convergence may take tens of seconds to minutes, during which applications suffer from packet losses and long delays. Therefore, many efforts, including Path Splicing, have gone to improving packet delivery during the transient period after failures. The essence of these IP-based solutions is to find a loop-free path, either precomputed or computed on-the-fly, without relying on routing.

Path Splicing improves packet delivery under link failures over IP.[11] When links fail and packets are dropped, the consumer host will time out the request after RTT and retransmit using different tags. Fig. 7 and 8 show data retrieval time and path stretch of Path Splicing under different link failure probability. Among all the affected but recoverable host pairs, 66% and 49% of them succeed in data retrieval when the link failure probability is 1% and 10%, respectively.[12]

The performance of Path Splicing depends on the number of slices and the number of retransmissions allowed. The former represents the number of choices for alternative paths, the latter represents how many attempts are allowed to find a working path among the choices. We experimented with different settings to understand the impacts of these two tuning knobs. If the maximum number of retransmissions allowed is increased from 5 to 20 while the number of slices is kept at 5, the number of disconnected pairs of Path Splicing reduces significantly as shown in Fig. 6. But increasing the number of slices from 5 to 10 only makes a small improvement. This observation suggests that, for the specific topology used in this simulation, a small number of slices can provide adequate path diversity to get around the link failures, however end-hosts may do many random trials before they succeed. Figs. 7 & 8 show that for those flows which succeed in recovery by Path Splicing, they takes much longer time to retrieve data than NDN and in general the found paths are longer. This is because end-hosts randomly pick different tags, without knowing where the failures are to make an informed selection. Furthermore, such recovery attempts are initiated by end hosts after timeout, which necessarily takes much longer time compared to NDN routers performing local recovery.

4.5. Congestion

Today's Internet routing does not react to congestion due to concerns of routing oscillation and frequent routing updates. When a link is congested, the routing plane at each of the two incident routers either does not see the problem at all if routing protocols have their keep-alive messages pass through, or considers the link failed if enough keep-alive messages are lost. The responsibility of congestion control is solely on end-hosts, which run TCP to detect congestions and adjust sending rate reactively. In NDN, on the other hand, the forwarding state enables routers in the network to prevent, detect, and react to congestion by utilizing multiple paths when needed, resulting in effective and efficient congestion control.

Let us first use a simple 6-node topology to shed the light on the basic differences between NDN and TCP NewReno in their reactions to congestion (Fig. 9). The server and client each has a 10 Mbps link connecting to a router. Each router has buffer size of 20 packets and all the links between routers have 1 Mbps bandwidth. The lower path has an RTT of 130 ms, while the upper path's is 134 ms. Data packet size in both NDN and TCP is 1040 bytes, and both Interest size in NDN and TCP ACK size is 40 bytes. The client downloads content from the server and the figures show the link utilization achieved by NDN and TCP respectively. We can make

[11] The result in Fig. 6 appears to be worse than that in Fig. 6 of [3], because the result in [3] is for one-way traffic, and our result here is for two-way traffic, which requires working tags for paths in both directions. When we run simulations for one-way traffic only, the result is similar to that in [3].

[12] Fig. 6 shows that, with link failure probability of 10%, 31% of all host pairs are affected (i.e. the failure percentage of IP traffic), 9.2% are unrecoverable, hence 21.8% of host pairs are affected but recoverable. NDN can recover all the 21.8%, while Path Splicing can only recover 10.6% of them. Hence only about 49% of recoverable pairs succeed in Path Splicing as shown in Fig. 7b) and Fig. 8(b).

two observations from the results. First, while TCP/IP uses the shorter path only and saturates the bottleneck link, NDN is able to use both paths. In NDN, R1 first uses only the lower path because it is the most preferred, but when the rate-limit of the lower path is reached, R1 starts using the upper path too. Second, over each path, NDN is able to grab available bandwidth more quickly than TCP, which takes longer time to settle at a stable rate. Consequently TCP takes more than twice as long to download the same amount of data.

NDN has a number of means to prevent, control, and alleviate congestion. First, a downstream node N_d controls the rate of Interest forwarding based on its estimate of the bandwidth needed to carry the returning Data traffic. This prevents excessive Data from being pulled into the network, and is enabled by the symmetric two-way flow of Interest/Data packets. The estimate of needed bandwidth, though, can be off due to packet size variations, and the errors may lead N_d to send more Interests than it should. When that happens, the upstream node N_u can reject new Interests by sending a Congestion NACK downstream; or even when an excessive Data packet is retrieved but the downstream link cannot handle, N_u can simply cache the Data and send back an Interest NACK. In TCP/IP, on the other hand, because data is *pushed* from the sender to the receiver, when a data packet arrives at a link where it cannot be forwarded further, the router simply drops it, after the the packet has already consumed considerable bandwidth along the way from the sender to the congested link. While TCP congestion control also aims to achieve flow balance as an NDN network does, it sends data packets to probe the network's available bandwidth and takes much longer time to detect congestion (end-to-end vs hop-by-hop); meanwhile additional excessive packets may have been pumped into the network, which eventually get dropped.

Second, Interest NACKs allow NDN routers to adapt to congestion hop-by-hop. A Congestion NACK is generated if the Interest cannot be forwarded upstream due to congestion. The downstream node will try its other interfaces for this Interest. This hop-by-hop retry inside the network reacts much faster than the end-to-end solutions for stateless IP networks, leading to quick local workaround as we have seen in the case of link failure recovery. Upon receiving a Congestion NACK the router also adjusts its forwarding rate for the specific $[L_{i,n}]$ pair, where i is the interface and n is the name prefix. Further *excess* Interests will be diverted to other interfaces. When the network cannot satisfy the demand, Interest NACKs will eventually be pushed back to inform the consumer to adjust its Interest sending rate properly. This is in contrast to TCP, which can only guess whether congestion occurred in the network, and can only use AIMD window adjustment to tune towards the right sending rate.

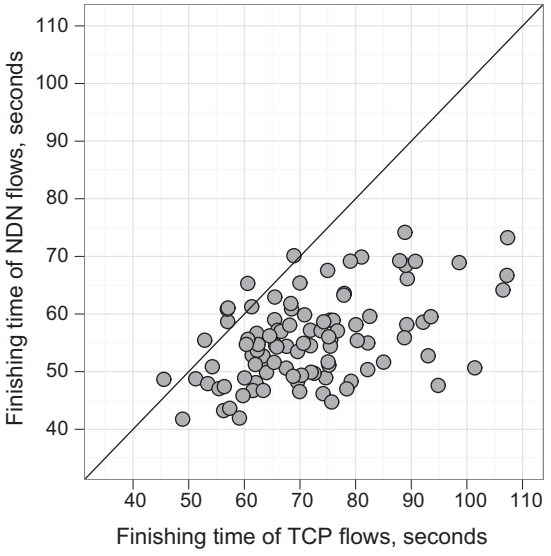


Fig. 10. Flow finish time under congestion.

Third, NDN can use multiple paths simultaneously to retrieve data whenever needed. As illustrated in the cases of hijack and link failure, NDN can find loop-free alternative paths quickly. When traffic is below the rate limit of a single upstream link, all will be forwarded along the best path. When traffic is over a single path's capacity, NDN can divert excess Interests to one or more alternative working paths. This capability of on-demand multipath forwarding enables efficient use of all available network resources.

Fourth, even though we did not simulate caching in this study, in a real NDN network, caching can further help speed up recovery from faults including congestion. When a Data packet arrives at a congested or failed link, it cannot be forwarded further but can be cached along the way. When downstream routers send another Interest, in response to either a NACK or end-host retransmission, via a different interface, this subsequent Interest will bring the requested data back as soon as it hits a cached copy of the data. With caching, recovery from packet losses can be much faster and more efficient in network resource usage than the end-to-end retransmission in IP-based solutions.

We run a larger-scale simulation using the Sprint topology and generate a number of flows that lead to cross traffic at multiple locations in the network. All routers have 20 packets buffer each, and all links are assigned 1 Mbps bandwidth but different propagation delays according to the topology file. In each run, 20 client/server pairs are randomly selected and each client downloads the same amount of data from its server.[13] The clients start in a random order with 1 s apart. Packet size is the same as in the previous simulation. Fig. 10 shows the results from 100 runs, where each dot represents the finish time of the flow that finishes last. As the figure shows, NDN finishes sooner than TCP in all but 7 runs (including one run in which they finish almost the same time), demonstrating that NDN can utilize network resources more efficiently and handle congestion better.

We can explain the 6 cases where NDN took slightly longer time than TCP to finish as follows. In NDN, because all consumers try to retrieve data as fast as possible, and all routers explore multiple paths to satisfy consumers demand, consequently those pairs of nodes that have multiple parallel paths in between can capture more bandwidth and finish fast. However a number of flows in the simulation have only one single path between client C_i and server S_i, i.e. they must go through at least one specific link L_B to reach each other. If L_B is not shared with other traffic, C_i can finish data retrieval from S_i as soon as possible. But if L_B is shared by other traffic flows, which is more likely to be the case in NDN than in TCP/IP, C_i will take longer to finish.

The above observation suggests that multipath forwarding deployment should be accompanied by support for fair share of network resources. This fair share support can be added into the decision process when a node needs to return Congestion NACKs. The node has the discretion on which Interest to send a Congestion NACK back. Through the decision criteria one can achieve fair share goals, enforce bandwidth limit to downstream, maintain QoS targets, and even push back excessive Interests in the case of DDoS. Although the exact design and evaluation are still work-in-progress, we believe that the forwarding state in NDN routers make it easier and more effective to achieve these goals in an NDN network than in an IP network.

[13] We place clients/servers randomly and run the experiment multiple times in order to evaluate NDN and TCP in general situations.

In summary, our simulation results illustrate that maintaining datagram state at every router allows NDN to detect and react to packet delivery problems quickly and effectively. Since Data is expected to come back within an RTT period along the same path, forwarding plane problems can be detected at the time scale of an RTT. Interest NACKs allow a node to retry alternative paths quickly, enabling fast local recovery inside the network. The overall result is that NDN's forwarding resilience is significantly better than IP and Path Splicing across all fault scenarios we have examined.

5. Discussion

In this section, we first review and analyze the benefits brought by NDN's stateful forwarding plane and then identify a few key open issues for future research.

5.1. Benefits of Stateful Forwarding Plane in NDN

Datagram is the most basic unit in packet switched networks. Therefore, controlling traffic using per-datagram, per-hop state gives a network the most flexibility to support a wide variety of functions. While semantics of datagram state can be different (i.e., what information is remembered and how this information is used), it is the granularity of the state that allows (1) loop-free, multipath data retrieval, (2) native support of synchronous and asynchronous multicast (i.e., servicing requests from multiple consumers that are sent either at the same or different time), (3) efficient recovery from packet losses, (4) effective flow balancing (i.e., congestion avoidance), (5) robust recovery from network problems, such as link failures and hijacks, as illustrated in Section 4, and other important network functions.

The loop-free, multipath data retrieval in NDN contains two tasks that rely heavily on per-datagram state: loop-free forwarding of Interests and loop-free Data return. The first task is accomplished by remembering names and nonces for each forwarded Interest, thus looped (duplicated) Interests can be easily detected and discarded. It is this loop-free property that allows NDN routers to freely try multiple alternative paths in Interest forwarding. Delivery of a Data packet follows the per-Interest hop-by-hop state created at each router along the path. Therefore, if an Interest path is loop-free (and it is), then a Data path is by definition also loop-free.

Native support of synchronous and asynchronous multicast comes from two per-datagram pieces of NDN design: suppression of Interests for the same name and per-datagram in-network data storage (Content Store). If more than one consumer expresses Interests for the same Data around the same time, when these Interests' paths merge in the network, only the first Interest will be forwarded towards the producer; other Interests will only add additional incoming interfaces in the PIT entry, forming a multicast tree for Data to return. On the other hand, if the Data has been previously requested by someone else, it will be cached in a router's Content Store, from which any later request can retrieve Data without going all the way to the original producer.

The per-datagram in-network data storage also facilitates **efficient recovery from packet loss**. That is, if a Data packet is lost and the consumer retransmits the Interest, it will be satisfied when it hits the router on the producer side of the lossy link and bring the Data back, as opposed to pulling the Data from the original producer again like in the current Internet.

NDN's ability of **effective flow balancing and congestion control** comes from the combined effect of symmetric paths (of Interests and Data) and packet conservation: one Interest brings at most one Data back over each link. Therefore one can effectively regulate incoming Data rate by controlling outgoing Interest rate, which can be done on a per-interface basis, as well as per-prefix-per-interface basis. The simulation experiments reported in this paper used per-interface rate limiting, and we plan to explore the latter in our ongoing effort. The per-hop nature of datagram state, together with the feedback loop formed by Interests and Data, allows routers to divert traffic to alternative paths at any point in the network, without pre-coordination (e.g., flow set up).

Finally, the same datagram state that ensures path symmetry enables routers to measure forwarding plane performance (e.g., RTT) of each path. This information can then be used to retrieve data along best available paths, and to **robustly detect and recover from forwarding problems**, that may be caused by either physical failures or malicious attacks.

What we have explored so far is only a small subset of the benefits provided by NDN's datagram state. We believe that adaptive forwarding based on datagram state can be used to solve a number of other network control and security problems in the future, without requiring major changes to the basic mechanism.

5.2. State granularity and solution elasticity

Many efforts have been made over the years to add each of the above mentioned functions into IP networks, with each solution installing its own state into the network that cannot be used to solve other problems. For example, one may set up state of coarser granularity, e.g., per-connection state, for control purposes in IP or some other network architectures. However such state cannot be used to support IP multicast which requires per source-multicast group state.

When a coarser (than datagram) granularity of state is used for control purposes, it can be adequate to support a specific function, but is unlikely to be able to support other uses, simply because different control purposes require certain state information that is incompatible with the chosen granularity. For example, IP multicast requires state information associated with {host, multicast group} pair, which is incompatible with the state information needed by XCP [10] to control congestion. Similarly, the state information needed by XCP is different from that needed by PushBack [11] to mitigate DDoS attacks. Other piecemeal solutions include Pretty Good BGP [12] to mitigate route hijacks, Failure-Carrying Packets [13] to deliver packets under failures, etc. In contrast to NDN's per-hop datagram state that support all the above functions, each of these named solutions solves one problem by adding its own state or mechanisms tailored to the specific function; a network architecture with all of these functions and added state would become incoherent and cumbersome.

In choosing state granularity, there exist a tradeoff between the functionality to be supported versus the amount of resource it consumes. The downside of datagram state is the amount of state it incurs, which is perceived by many as infeasible based on today's technologies. For example, today's router hardware may not be able to hold a PIT or operate NDN at wire speed. Given where the technology is as of today, we admit that maintaining datagram state at routers represents formidable challenges. We consider these challenges as part of the research issues in realizing NDN, as we discuss next.

5.3. Research issues

In previous sections we have argued that adaptive forwarding with datagram state can achieve robust packet delivery as well as simplified routing. Below we briefly discuss a few important design choices and research challenges. Our short list is necessarily incomplete.

5.3.1. Forwarding state

NDN's datagram state brings with it a significant cost, both in router storage and in packet processing. More specifically, since an Interest stays in the PIT until the corresponding Data packet returns, the number of PIT entries (associated with each outgoing interface) is roughly on the order of $Bandwidth \times RTT/P$, where P is the average size of Data packets. For a path that can sustain 10 Gbps throughput, we need 100 K PIT entries assuming RTT = 100 ms and P = 1000 bytes. If a router has 10 such interfaces, then its PIT needs to hold 1 million entries. Although today's core routers can handle more than 1 M entries in IP routing tables, a PIT entry is larger in size than an IP routing entry due to variable-length names and other information, and PIT's memory cost is in addition to that of the NDN routing table. As routers get more interfaces and networks go faster over time, the PIT table will also grow proportionally. However PIT usage represents an even bigger challenge. IP routers perform lookups over FIB only, but NDN routers perform both read from (lookup), and write into, the PIT. For example, when a new Interest arrives, a PIT entry needs to be inserted; when a matching Data returns, a PIT entry needs to be removed. These are more expensive operations that require fast and scalable solutions. We need both novel data structures to store PIT efficiently and novel algorithms to operate on PIT at wire speed. There are already research efforts underway to tackle these issues (e.g., [14,15]).

5.3.2. Forwarding strategies

Forwarding strategy is a unique feature enabled by the NDN architecture. In this paper we have presented a simple strategy that works reasonably well in handling prefix hijack, link failure and congestion. However, it is clear that many possibilities exist in forwarding strategy designs to suite different network environment and problems. For example, different strategies may be used at core routers vs. edge routers. Exploring different design choices of forwarding strategy is one of our main goals in future work. Below we list three design questions we have encountered.

How to discover working paths quickly and efficiently? There is a spectrum of choices between trying a single interface at a time (our current approach) and flooding an Interest to all interfaces at the same time, with the tradeoff between the overhead and delay to retrieve data. Flooding explores all possible paths at once, but it incurs most overhead in terms of redundant packets and network state. The number of interfaces to try at one time can be adjusted depending on the situation. For example, the first transmission of an Interest is forwarded to only a single interface, but if Data does not return in time and the consumer retransmits the Interest, this copy can be forwarded to multiple other interfaces in order to discover a working path quickly. The tradeoff between quickly retrieving data and minimizing overhead depends on the problem context. For example, flooding may be acceptable in home networks but not suitable in the core Internet.

How to utilize multiple paths? Our current design uses a single best path towards each name prefix as long as it is able to meet the consumer demands. Only after a problem occurs to that path, such as congestion, will a router divert excess traffic to other paths. In other words multiple paths are used on demand. A different approach is to proactively split traffic along multiple paths. This way, a router continuously receives feedback on forwarding plane performance from multiple paths, and a failure may only affect a smaller portion of the traffic. There can also be a hybrid approach which uses a couple of paths proactively but use more when existing paths encounter problems. Further research is needed to investigate how to best utilize NDN's multipath capability.

How to probe unused interfaces? In our current design, routers periodically send Interests to previously failed or unused paths to measure their performance. The questions are when to probe and which interfaces to probe. A simple design is trigger probing every N seconds or every M packets. The exact numbers of N and M, however, depend on the effectiveness of probing and the overhead it incurs. As to which interface to probe, one way is to probe all unused interfaces with equal probability, another is to probe higher-ranked interfaces with a higher probability since they might lead to better paths.

5.3.3. Bogus data packets

Malicious attackers in an NDN network may launch an attack by injecting bogus Data packets. The bogus Data packets can be either unrequested, or injected as responses to valid Interests in a hijack attack. In the former case, since those packets are not requested by anyone, they have no corresponding PIT entries at routers and will be discarded. Therefore NDN is able to dispel the brute force DoS attacks that happens everyday in today's IP networks. In the latter case, there exists a wide spectrum of possible solutions, ranging from router signature verification to end consumer feedback.

Since every Data packet carries a signature, one easily detects bogus data packets through signature verification. However, verifying the signature for every Data packet can be too expensive to perform by routers. Instead routers may perform signature checking over sample packets, which could be done by an auxiliary processor. We are also looking into hardware-assisted signature verification at routers. A more promising direction is to utilize end consumer feedback. Once end consumers identify bogus Data through signature verification, they can notify first hop routers to take proper actions. The routers may start sampling signature verifications as well as trying alternative paths to fetch data. Interested readers please refer to [16] for a more comprehensive discussion.

6. Related work

Today's IP architecture takes the "smart routing, dummy forwarding" approach. Due to its stateless nature, the forwarding plane strictly follows the routing state. Recent research efforts have recognized that introducing adaptability to forwarding plane is a promising approach. Wendlandt et al. [8] and Caesar et al. [17] argue that networks should provide end-hosts with multiple path choices, and end-hosts should be responsible for choosing different paths based on their observed forwarding plane performance. Works such as Pathlet routing [18], routing deflections [19] and Path Splicing [3] are specific designs along this direction. The basic difference among them is in the specifics of how alternative paths are obtained.

Multipath TCP [20] utilizes multiple interfaces/IP addresses of multihomed hosts to set up multiple sub-TCP connections between the two ends. Assumed that each source/destination IP address pair represents a different physical path, one can split traffic over the multiple end-to-end paths. TeXCP [21] adds adaptability to edge routers to handle network congestion. In TeXCP, multiple MPLS tunnels are precomputed between each pair of ingress-egress routers, and ingress routers split traffic over multiple tunnels based on congestion feedback from internal routers. All these efforts introduce some adaptability to the forwarding plane, but the adaptability is limited to end-hosts or edge routers only. In an NDN network, all hosts and routers are able to select alternative paths.

Although NDN is considered as one of the Information-Centric Networking (ICN) architectures, NDN's forwarding plane design is drastically different from that of other ICN architecture proposals. For example, PURSUIT [22] is a publish/subscribe architecture. PURSUIT employes source routing for packet forwarding. Its

forwarding path is encoded in a bloom filter carried in the packet header; routers forward packets strictly according to the path information carried in the bloom filter [23]. In case of failures, PURSUIT relies on preset backup paths to get packets delivered, therefore the network's adaptability is limited. Since each packet carries its own path in a PURSUIT network, PURSUIT routers do not maintain packet state as NDN routers do. At the same time, individual PURSUIT routers also cannot measure the forwarding plane performance nor collect feedback to adapt to failures as NDN routers do.

7. Conclusion

NDN's communication model of retrieving data by names leads to a forwarding plane design that keeps datagram state at every router. Because datagram is the basic unit in packet switched networks, this per-hop datagram state provides the flexibility to solve a host of existing problems that have resisted effective solutions up to now. In this paper we described a specific design on how to utilize this datagram state to provide high performance and resilience in an NDN network. We also quantitatively evaluate the data delivery performance of NDN under adverse conditions. Our results show that NDN's adaptive forwarding mechanism can provide excellent performance in handling blackhole hijacks, link failures and network congestion.

At the same time, we are fully aware that installing datagram forwarding state at routers brings largely open issues in terms of both technical feasibility and economical viability. The history of IP development shows, however, when a new architecture solution provides significant functional advantages as well as new application opportunities, even though its overhead may seem higher and its initial implementation offers inferior performance compared to the highly engineered implementation of the incumbent architecture, research and technology advancements would eventually catch up to close the gap and even go further. Thus we remain modestly optimistic about the future of NDN and its stateful forwarding plane. This paper serves as our invitation to the research community to further examine this new direction for building resilient networks and tackle the open research issues.

References

[1] V. Jacobson, D.K. Smetters, J.D. Thornton, M.F. Plass, N.H. Briggs, R.L. Braynard, Networking named content, in: Proceedings of ACM CoNEXT, 2009.
[2] L. Zhang et al., Named data networking (NDN) project, Tech. Rep. NDN-0001, NDN Project (October 2010).
[3] M. Motiwala, M. Elmore, N. Feamster, S. Vempala, Path splicing, in: Proceedings of ACM SIGCOMM, 2008.
[4] S. Floyd, V. Jacobson, Random early detection gateways for congestion avoidance, IEEE ACM Trans. Networking 1 (4) (1993) 397–413.
[5] NS-3 Network Simulator, <http://www.nsnam.org/>
[6] A. Afanasyev, I. Moiseenko, L. Zhang, ndnSIM: NDN simulator for NS-3, Tech. Rep. NDN-0005, NDN Project (July 2012).
[7] N. Spring, R. Mahajan, D. Wetherall, T. Anderson, Measuring ISP topologies with Rocketfuel, IEEE/ACM Transactions on Networking 12 (1) (2004) 2–16.
[8] D. Wendlandt, I. Avramopoulos, D.G. Andersen, J. Rexford, Don't secure routing protocols, secure data delivery, in: Proceedings of ACM HotNets, Workshop, 2006.
[9] P. Baran, On distributed communications networks, IEEE Trans. Commun. Syst. 12 (1) (1964) 1–9.
[10] D. Katabi, M. Handley, C. Rohrs, Congestion control for high bandwidth-delay product networks, in: Proc. of SIGCOMM, 2002.
[11] J. Ioannidis, S.M. Bellovin, Router-based defense against DDoS attacks, in: Proc. of NDSS, Symposium, 2002.
[12] J. Karlin, S. Forrest, J. Rexford, Pretty good BGP: improving BGP by cautiously adopting routes, in: Proceedings of IEEE ICNP, 2006.
[13] K. Lakshminarayanan, M. Caesar, M. Rangan, T. Anderson, S. Shenker, I. Stoica, Achieving convergence-free routing using failure-carrying packets, in: Proceedings of ACM SIGCOMM, 2007.
[14] Y. Wang, K. He, H. Dai, W. Meng, J. Jiang, B. Liu, Y. Chen, Scalable name lookup in NDN using effective name component encoding, in: Proceedings of IEEE ICDCS, 2012.
[15] H. Yuan, T. Song, P. Crowley, Scalable NDN forwarding: Concepts, issues, and principles, in: Proc. of IEEE ICCCN, 2012.
[16] P. Gasti, G. Tsudik, E. Uzun, L. Zhang, DoS & DDoS in named-data networking (August 2012). arXiv:1208.0952.
[17] M. Caesar, M. Casado, T. Koponen, J. Rexford, S. Shenker, Dynamic route recomputation considered harmful, Comput. Commun. Rev. (CCR) 40 (2) (2010) 66–71.
[18] P.B. Godfrey, I. Ganichev, S. Shenker, I. Stoica, Pathlet routing, in: Proceedings of ACM SIGCOMM, 2009.
[19] X. Yang, D. Wetherall, Source selectable path diversity via routing deflections, in: Proc. of ACM SIGCOMM, 2006.
[20] D. Wischik, C. Raiciu, A. Greenhalgh, M. Handley, Design, implementation and evaluation of congestion control for multipath tcp, in: Proc. of Usenix NSDI, 2010.
[21] S. Kandula, D. Katabi, B. Davie, A. Charny, Walking the tightrope: responsive yet stable traffic engineering, in: Proceedings of ACM SIGCOMM, 2005.
[22] PURSUIT Publish-Subscribe Internet Technology, <http://www.fp7-pursuit.eu/PursuitWeb/>
[23] P. Jokela, A. Zahemszky, C. Esteve Rothenberg, S. Arianfar, P. Nikander, Lipsin: line speed publish/subscribe inter-networking, in: Proceedings of ACM SIGCOMM, 2009.