How to Establish Loop-Free Multipath Routes in Named Data Networking

Klaus Schneider, Beichuan Zhang
The University of Arizona
Email: {klaus, bzhang}@cs.arizona.edu

***Abstract*—Both IP networks and Named Data Networking (NDN) can be extended to support multipath forwarding. However, since the IP forwarding plane cannot detect loops, IP routing protocols are strictly required to produce loop-free paths. In contrast, NDN can choose between using a loop-free routing protocol and handling loops at the forwarding layer. In this paper, we explore the trade-offs that come with this choice.**

It is often useful to *split traffic* for one destination among multiple paths, for example, to balance the traffic load or to reduce network congestion. This traffic splitting requires the employed paths to be *loop-free*, lest they waste network resources, and the involved routers to have a high *path choice*, that is, a high number of potential nexthops to forward traffic to.

We show that we can achieve a higher loop-free path choice than state-of-the-art loop-free routing protocols by combining an *almost loop-free routing* protocol (ALR) with loop-removal at the forwarding layer. ALR's advantage comes from exploiting the ability of NDN's data plane to *always exclude the incoming interface* from forwarding, which broadens the routing task beyond the traditional goal of creating a Directed Acyclic Graph (DAG). Assuming this incoming interface-exclusion, ALR employs certain heuristics to minimize routing loops while, when possible, giving each router along a path at least two potential nexthops towards the destination. The quality of these nexthops is signaled to the forwarding layer, which then detects and permanently removes all remaining loops. Combined, these two mechanisms result in higher path choice and path quality than current alternatives, while being computationally efficient enough for practical implementation.

I. Introduction

IP networks follow the paradigm of *smart routing, dumb forwarding:* the routing protocol pre-establishes a single shortest path and the forwarding plane has no choice other than to use this path. This lack of choice prevents the IP forwarding plane from handling problems such as link failures or prefix-hijacking [32].

However, even if the IP architecture allowed multipath forwarding, it would still suffer from a second fundamental problem: its forwarding plane has no means to detect whether an incoming packet has *looped*, that is, whether it has been forwarded by the same router earlier. Since loops cannot be detected, they are very *expensive*; packets circle around until the TTL runs out, wasting network resources while doing so. Thus, IP networks, even when extended to allow multipath forwarding, require a strictly *loop-free* routing protocol.

For Named Data Networking (NDN), this requirement is loosened: loops can be detected via a *nonce* in the packet header and subsequently be handled by the forwarding layer. Thus, NDN routing does *not* have to be loop-free; indeed, the most common NDN routing protocols, NLSR [10] and Hyperbolic Routing [13], do produce paths that potentially result in loops.

These forwarding loops, although detectable, can still waste network resources, especially in the case of *traffic splitting*. NDN can split traffic to one destination at strategic points in the network in order to balance the load among multiple paths, gradually redirect traffic during congestion [5], [25], or exploit differences in costs and performance of paths (e.g. between WiFi and LTE [24]). When splitting traffic in this way, looping paths waste network resources and increase consumer delay, hence they need to be avoided at the forwarding layer. In other words, while NDN does not require loop-free routing, it still requires *loop-free forwarding*.

Thus, the research question becomes: *"How do we best establish loop-free paths at the NDN forwarding layer?"*

One option is to use a loop-free routing protocol, and it has been studied extensively in the IP literature (see Section II). NDN provides another option, one which might show to be superior: accepting some routing loops and handling them at the forwarding layer. Here we investigate this second option and split the research question into: *"1) Which non-loopfree routing protocol should we use? 2) How should the forwarding layer turn paths that contain a loop into loop-free paths? 3) And is the result better (in terms of path choice, path quality, and computational complexity) than using the best available loop-free routing protocol?"*

For the case of multipath traffic splitting (i.e. forwarding on more than a single nexthop), current work does not answer these questions. Instead, current forwarding strategies avoid the looping problem in one of three ways: 1) they restrict themselves to forwarding on a single best path (as done in the NCC, BestRouteStrategy, and AccessStrategy implemented in NFD, the Green/Yellow/Red-Strategy [32], and ASF [13]), 2) they split up traffic, but implicitly rely on loop-free routing [5], [30], [23], [33], [20], [34], [14], [25], 3) they completely ignore the problem and thus restrict their scalability (like the Multicast/Broadcast-Strategy in NFD).

In this work, we give a detailed answer to the first two questions, and answer the third one with a clear *Yes:* A combination of non-loopfree routing and loop-removal at the forwarding layer gives better performance than any reasonably complex loop-free routing protocol. Moreover, we show that the best trade-off between routing effort and forwarding effort is achieved by using an *Almost Loop-free Routing* (ALR) protocol (Section III) combined with intelligent loop removal at the

forwarding layer (Section IV).

ALR exploits the forwarding layer's ability to always exclude the incoming interface of a packet: when multiple nexthops are available, a router will never send the packet back to the previous hop. This *incoming interface exclusion* avoids one-hop loops, which broadens the routing problem from creating a Directed Acyclic Graph (DAG) to creating a directed graph without cycles longer than one hop. This broader routing problem then allows each node to have a higher nexthop choice, while still preventing loops. ALR's specific goal is to give each router at least two nexthops towards each destination, unless doing so would result in a loop. To achieve that, ALR determines "downhill nexthops" [8], nexthops that go closer to the destination and thus are guaranteed to avoid loops if all routers restrict their forwarding to downhill nexthops;[1] for non-downhill nexthops, ALR applies certain *heuristics* to estimate the likeliness of a loop, and thus to determine whether the nexthop should be used for forwarding. This information is passed to the forwarding layer by giving each FIB nexthop a type from the set { `Downward`, `Upward`, `Disabled` }.

At the forwarding layer, we exploit the knowledge of the nexthop type and, after detecting a loop, our scheme only disables upward nexthops; hence we call it *Uphill Nexthop Removal* (UNR). Our evaluation (Section V) shows that, given enough time, UNR results in a network that is completely and permanently loop-free, while retaining a higher path choice than state-of-the-art loop-free routing schemes.

The computational complexity of ALR's loop check heuristics is comparable or only slightly higher than ALR's most efficient competitors (see Section III-B). The complexity of the forwarding loop removal is also low: depending on the topology, only 0% to 1.1% of FIB entries need to be disabled (see Section V). Moreover, loop removal is seamless for endpoints: looped packets may see a higher latency, but they are never dropped.

II. Loop-Free Routing

Both IP and NDN employ *Destination-based Hop-by-Hop routing:* Each router independently decides how to forward a given packet based on its destination address or name (this differs from *source-based explicit routing* like MPLS [3], where the source host decides which path its packets will take and in-network routers blindly follow that decision). When extended for multipath forwarding [8], independent hop-by-hop routing allows in-network routers to solve certain problems like link failure and congestion on their own. Moreover, independent multipath routing moves the routing task from finding the k best paths to finding the *k best nexthops* at each node, resulting in a potentially much larger path choice.

However, when routers make independent forwarding choices based solely on the packet's destination, the forwarded packets might run into loops. To avoid these loops, routers must employ certain *heuristics* to decide which nexthops they will forward to. These heuristics depend on the *goal* of multipath forwarding and on how many routers along the path are using them. For example, handling a single link failure allows to use broader heuristics (like "Loop-Free Alternates" [2]), since only one router on the path (the one that is upstream of the failing link) applies the heuristic and all other routers are restricted to forward on the single shortest path. For traffic splitting, however, narrower heuristics are required, since all routers on the path are using them: all of them may send packets to nexthops other than the shortest path. Below, we restrict the discussion to designs that support this independent splitting of traffic. This restriction excludes all approaches based on Multi-Topology Routing, such as *Path Splicing* [18]. For a broader survey that includes these schemes, see [22].

Arguably the simplest loop-freeness heuristic is *Equal Cost Multi-Path* (ECMP) routing [1], in which a router uses the nexthop of the shortest path n_{sp}, plus any nexthop n_i with the exact same cost: $cost(n_i, dst) = cost(n_{sp}, dst)$. The shortest path and equal cost paths are always guaranteed to be loop-free. However, forwarding only on equal cost paths limits the achievable nexthop choice, since path costs need to match exactly; the more fine-grained the link cost metric, the less path choice ECMP can achieve (see Section V).

A higher path choice can be reached by non-equal cost multipath algorithms, the most prominent of which is the *Downward Path Criterion* [8]. Downward paths include the shortest path nexthop plus any nexthop n_i that is closer to the destination (has a lower cost) than the current node (x): $cost(n_i, dst) < cost(x, dst)$. Downward Nexthops are simple to compute, achieve a higher path choice than ECMP, and are still guaranteed to be loop-free. Thus, they are very commonly used in the literature, known under the names of Loop-Free Invariant (LFI) [28], Rule 1 (One Hop Down) Deflection Set [31], and Relaxed Best Path Criterion [27].

One extension of Downward Paths is to also consider nexthops with the same cost: $cost(n_i, dst) \leq cost(x, dst)$. However, to avoid the packet from directly looping back, one needs to add a *tiebreaker* which assures that traffic only crosses one direction of the equal-cost link. This tiebreaker could be based on the node id (shown below) or the node degree [11]:

$$cost(n_i, dst) < cost(x, dst) \vee$$
$$(cost(n_i, dst) = cost(x, dst) \wedge id(n_i) < id(x))$$

This heuristic is guaranteed to be loop-free and achieves at least as much nexthop choice as the Downward Criterion (and often more). It is quite common in the literature, and has been used in the context of ICN [9].

Yang et al. [31] provide two more heuristics, called "Rule 2 (Two Hops Down)" and "Rule 3 (Two Hops Forward)". However, these rules are more complex to compute and only prevent packets from revisiting the same link (link level loop), but not from revisiting the same node (node level loop). Since this approach wastes network resources, we require stricter guarantees for loop-freeness: in this work, we define "loop-free" to mean that loops are avoided at both link and node level.

Another approach for loop-free multipath routing is used by the three algorithms from the work "Maximum Alternative Routing Algorithm" (MARA) [21]. They are based on con-

[1]Throughout this paper, we use the terms "uphill" and "upward" ("downhill" and "downward") interchangeably to indicate whether, compared to the current node, a given neighbor is closer to the destination or further away. These terms should not be confused with the terms "upstream" and "downstream", which indicate the position of a router in the flow of a certain packet (packets flow from upstream routers to downstream routers).

Fig. 1. Abilene Topology [4]

structing an Directed Acyclic Graph (DAG) of the network and finding an optimal solution to the problems of 1) maximizing the minimum connectivity, 2) maximizing the minimum max-flow, and 3) maximizing the minimum max-flow as an extension of shortest path routing. These algorithms often result in a higher path choice than the Downward Criterion, but are also significantly more complex to compute (see Section V-A).

III. Almost Loop-free Routing

In this Section, we present a routing algorithm that produces a higher nexthop choice than loop-free routing, while keeping the chance of loops low. This higher path choice is possible because of one change in NDN's forwarding logic: forwarding decisions are based not only on the destination prefix, but also on the *incoming interface*; when considering its forwarding choices, a router can always exclude the interface on which the packet arrived.[2] This *incoming interface-exclusion* prevents one-hop loops at the forwarding layer, which changes the fundamental problem that a routing algorithm solves. All loop-free routing algorithms can be seen as solving a graph problem: turning a given undirected network graph into a Directed Acyclic Graph (DAG) [29]. With incoming interface-exclusion, the problem becomes wider: turning an undirected graph into a directed graph *without cycles longer than one hop*. This widening of the routing problem is the main cause for the higher path choice of our algorithm; it allows the use of many paths that would otherwise have looped.

Consider the example in the Abilene topology (Figure 1), when Washington (WA) wants to send packets to the destination Atlanta (ATL). Without excluding the incoming interface, Washington could never use New York (NY) as the nexthop, since New York may send a packet with destination Atlanta right back to Washington. With incoming interface-exclusion, however, Washington has at least two loop-free paths to Atlanta: WA→ATL and WA→NY→CH→IN→ATL.

Prior work has shown that raising the forwarding choice from one option to two is much more important than raising it from two to higher values, more succinctly called *the power of two choices* [17]. One way to give each router two nexthops is to simply choose the two with the lowest routing cost; but this naive solution leads to frequent loops and omits many nexthops that could be added without risking additional loops (see Section V). Thus, the specific goal of our Almost Loop-free Routing algorithm (ALR) is to give each router *at least two* possible nexthops for packet forwarding, unless adding the second nexthop would result in a loop. In addition, ALR provides the forwarding layer with a hint about which nexthops are most likely to result in a loop, hence which should be disabled after a loop has occurred.

Thus, the design of ALR is as follows: Each FIB nexthop is assigned a type { `Downward`, `Upward`, `Disabled` }, depending on whether it is closer to the destination than the current node (`Downward`) or further away (`Upward`). Certain upward nexthops are likely to cause a loop, and thus should be excluded from forwarding; they are assigned the type `Disabled`. While skipping them for traffic splitting, the forwarding layer may still use `disabled` nexthops for probing or as backup during failure of all other links.

Algorithm 1 Overview: Loop Check Heuristics

```
nexthops ← getDownwardNexthops();
disabledNexthops ← getUpwardNexthops();
if nexthops.size == 1 then
    forall n_i in disabledNexthops do
        if passesLoopcheckHeuristics(n_i) then
            Add n_i to nexthops;
            break;
        end
    end
end
```

A high-level overview of ALR is shown in Algorithm 1. For each destination prefix, ALR starts out by adding all Downward nexthops, together with equal-cost neighbors using the node degree as a tiebreaker (nodes with lower degree route towards ones with higher degree; for same-degree nodes the node id breaks the tie). If, after adding these downward nodes, the current node still only has one nexthop, ALR iterates through all disabled nexthops (in the order of routing cost) and applies a heuristic loop check to determine whether the nexthop should be added.

A. Heuristic Loop Checks

ALR uses certain heuristics to exclude nexthops that are guaranteed or very likely to result in a loop. These heuristics are best understood when focusing on routes towards a single destination prefix. In the next section we show how to implement them efficiently, using Dijkstra's algorithm, which takes one run to compute the distance to *all* destination prefixes.

In each round of Algorithm 1, we have a fixed current node x, a fixed destination d, and a changing candidate nexthop n_i. We use three heuristics to check whether adding n_i is likely to cause a loop or not; they are all based on a manipulation of the topology combined with an adjacency check or a shortest path calculation:

Heuristic 1: We remove node x from the topology. Now, if neighbor n_i doesn't have a path to destination d, we don't

[2]In contrast to other work on incoming-interface dependent forwarding [12], [19], [16], we do not require to use a different forwarding table for each incoming interface; it is sufficient to maintain *one* forwarding table and simply remove the incoming interface from the set of outgoing interfaces.

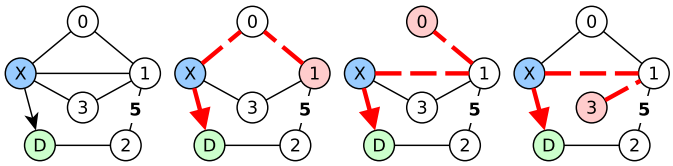


Fig. 2. Loop Check Heuristic 2

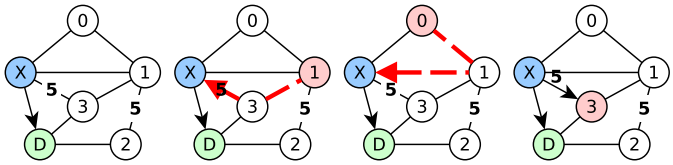


Fig. 3. Loop Check Heuristic 3

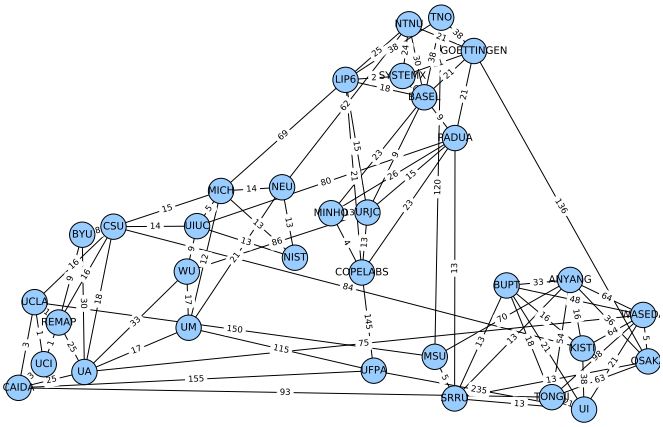


Fig. 4. NDN Testbed Topology

add n_i and move on to the next neighbor. If n_i can't reach the destination other than going back through x, it is clear that all packets sent from x to n_i would result in a loop. This heuristic is simple, straight-forward to implement, and also used by NLSR[3].

Heuristic 2: We remove link (x, n_i) from the topology and run Dijkstra's algorithm for n_i. If the shortest path of n_i to d still goes through x, we don't add n_i and move on to the next neighbor.

An example can be seen in Figure 2, where the current node X is show in blue, the destination D is shown in green, and the candidate nexthop (starting with node 1) is shown in red. All link weights are set to 1, except the link between node 1 and 2, which has the weight of 5. Heuristic 1 doesn't apply to the example, since after removing X, neighbor 1 still has a path (although more costly) to the destination. For Heuristic 2, we remove link (X, 1), and notice that the shortest path of 1 still goes through X, meaning that adding 1 would cause the loop $X \rightarrow 1 \rightarrow 0 \rightarrow X$. Thus, we don't add neighbor 1 to the FIB of X and move on to the next neighbor. For candidate nexthops 0 and 3 the same will happen: their shortest path would go through X and thus they are skipped. As a result, X will not use any nexthop to destination D other than the directly connected link $X \rightarrow D$.

Heuristic 3: For all Upward/Disabled neighbors $n_j \neq n_i$ of x: If n_i is adjacent to and upward of n_j, we don't add n_i and move on to the next neighbor. Our reasoning is that if n_i is upward of n_j, and n_j is upward of x, sending packets to n_i could result in the loop: $x \rightarrow n_i \rightarrow n_j \rightarrow x$.

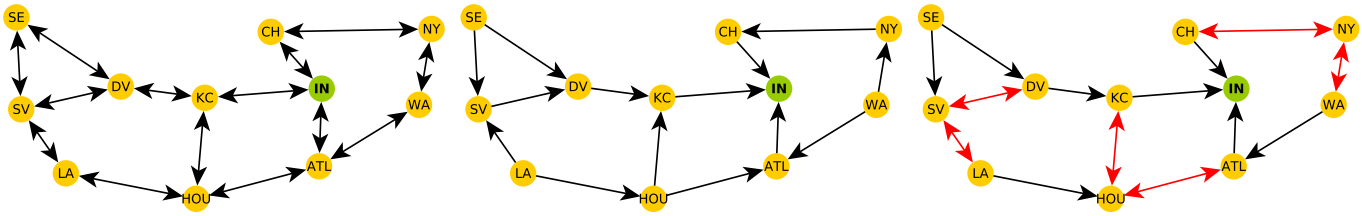
An example can be seen in Figure 3. For candidate nexthop 1, Heuristics 1 & 2 don't apply: after removing node X, Node 1 is not disconnected from D, and after removing link (X, 1), the shortest path from 1 to D ($1 \rightarrow 3 \rightarrow$ D) does not go through X. However, there is still the chance for a loop: Node 3 is uphill of X (equal cost to D, but 3 has a lower node degree), so 3 is allowed to send to X. Since Node 1 is uphill of 3, it will forward to 3, causing the loop $X \rightarrow 1 \rightarrow 3 \rightarrow X$. Thus, nexthop 1 needs to be excluded and Heuristic 3 does exactly that. For candidate nexthop 0 the same will happen: 0 is uphill of 1, which is uphill of X, so nexthop 0 is also skipped. Finally, for candidate nexthop 3 none of the heuristics apply; node 3 is added to the nexthop set. Thus, node X will have two nexthops to destination D: $X \rightarrow D$ and $X \rightarrow 3$.

Applying these heuristics avoids many potential loops, but can lead to a small number of *dead ends*, cases where a router receives a packet but its only forwarding option is to directly return the packet to the previous node (see Table IV in Section V-A). Fortunately, compared to loops, dead ends are less costly and easier to handle at the forwarding layer. The router detecting the dead end simply sends back a NACK and the router receiving the NACK disables this nexthop entry and forwards the Interest to a different nexthop.

To illustrate the effect of the three heuristics, we end this Section with two examples. First, we compare our heuristics (ALR) with NLSR and loop-free routing for the destination Indianapolis (IN) of the Abilene topology (Figure 5). Since NLSR only uses Heuristic 1, all nodes can choose any of their neighbors for packet forwarding, but this high path choice also leads to many potential forwarding loops (like $KC \rightarrow DV \rightarrow SV \rightarrow LA \rightarrow HOU \rightarrow KC$). Loop-free routing with the downward criterion avoids these loops, but also restricts the path choice: only 4 out of 10 routers have more than one nexthop to forward on. In contrast, ALR is still loopfree (when excluding the incoming interface), but gives routers a higher path choice: 10 out of 10 nodes have at least two nexthops towards the destination. Thus, in this example, ALR can provide every node with at least two nexthops; in some larger topologies, ALR needs to limit certain nodes to one nexthop in order to prevent loops (see Section V).

Second, we show the result of running ALR in the NDN Testbed topology (Figure 4) for a sample of destination prefixes of the node CAIDA (Table I). We see that the ordering of nexthops by type (Downward > Upward > Disabled) can differ from the ordering by cost; some Disabled nexthops have a lower routing cost than other Upward or Downward nexthops. This discrepancy shows a trade-off between choosing shorter paths and choosing paths that are guaranteed to be loop-free; it also shows that the nexthop type is necessary: it can't be inferred by the routing cost.

[3]The NLSR paper [10] uses a different phrasing ("removing all immediately adjacent links except one"), but the results are identical.



(a) NLSR (b) Loop-free Routing (c) ALR

Fig. 5. Routing Entries in the Abilene Topology for Destination Indianapolis (IN)

TABLE I
EXAMPLE FIB ENTRIES OF NODE CAIDA

Destination Prefix	Nexthop	Cost	Type
/NEU	**UCLA**	**48**	**Downward**
	UCI	**49**	**Downward**
	UA	**63**	**Downward**
	TONGJI	236	Disabled
	UFPA	291	Disabled
/WASEDA	**UA**	**100**	**Downward**
	UCI	104	Disabled
	UCLA	104	Disabled
	TONGJI	**135**	**Downward**
	UFPA	354	Disabled
/TONGJI	**TONGJI**	**93**	**Downward**
	UCLA	137	Disabled
	UCI	138	Disabled
	UA	**142**	**Upward**
	UFPA	367	Disabled

B. Implementation & Computational Complexity

We implement the loop check heuristics in two steps: First, we perform multiple runs of Dijkstra's algorithm and store the result in hash tables, indexed by the destination prefix. Second, we iterate through each destination and perform the heuristics, benefiting from the efficient hash table lookup.

For the first step, we modify Dijkstra's algorithm to return not only the nexthop and shortest path cost, but also a hash set of the shortest path; the hash set allows us to check whether a node is part of the shortest path in an average time complexity of *O(1)*. We combine this information into a structure `FibNexthop = (Cost, NexthopId, hash_set<NodeId> path)`. We then create hash tables for the shortest path of the current node and for each of its neighbors, applying the following graph modifications (see Algorithm 2):

- `shortestPathMap`: The shortest path costs of the current node x.
- `nbSpSrcNodeRemoved`: The shortest path costs for all neighbors in a graph where node x has been removed. Without x, some of the neighbors may be disconnected from the destination, a result that is used in Heuristic 1.
- `nbSpSrcLinkRemoved`: The shortest path costs for all neighbors in a graph where the link between neighbor n_i and node x has been removed, as used in Heuristic 2.
- `nbSpMap`: The shortest path costs for all neighbors in an unmodified graph, used to compare the cost of the two neighbors in Heuristic 3.

Algorithm 2 Fill Hash Tables of Node x

```
map⟨DstId, FibNextHop⟩ shortestPathMap ← runDijkstra(x);
map⟨nId, map⟨DstId, FibNextHop⟩⟩ nbSpMap;
map⟨nId, map⟨DstId, FibNextHop⟩⟩ nbSpSrcNodeRemoved;
map⟨nId, map⟨DstId, FibNextHop⟩⟩ nbSpSrcLinkRemoved;
forall n_i in neighbors do
    nbSpMap[n_i] ← runDijkstra(g,n_i);
    Remove node x from graph g;
    nbSpSrcNodeRemoved[n_i] ← runDijkstra(g,n_i);
    Add node x back;
    Remove link (x,n_i) from graph g;
    nbSpSrcLinkRemoved[n_i] ← runDijkstra(g,n_i);
    Add link (x,n_i) back;
end
```

Dijkstra's algorithm can be implemented with a computational complexity of $O(m + n \log n)$, where **m** denotes the number of edges in the network, **n** the number of nodes in the network, and **k** the number of neighbors per node (used below). We need to run Dijkstra's algorithm once for the shortest path and three times for each neighbor. Thus, the total complexity for this step is (the O notation omits constant factors):

$$(1 + 3k) * O(m + n \log n) = O(km + kn \log n)$$

Next, we use these tables to fill the FIB, as described earlier and shown in Algorithm 3. The complexity of this algorithm is O(n) for one iteration over all destinations times O(k) for each iteration over all neighbors containing the heuristic checks. The heuristics perform the following operations:

- **H1–H3:** Looking up the cost or type of a neighbor: O(1)
- **H2:** Checking if the shortest path contains a certain neighbor: O(1) with our hash set data structure.
- **H3:** Checking if two neighbors are adjacent in the graph: O(1) for graphs implemented by an adjacency matrix.

Thus the complexity of each heuristic is:

- **H1,H2:** $O(n) * O(k) * O(1) = O(kn)$
- **H3:** $O(n) * O(k^2) * O(1) = O(k^2 n)$

And the total complexity of this step is:

$$2 * O(kn) + O(k^2 n) = O(k^2 n)$$

Combined with the step of filling the tables, the total complexity of our algorithm is:

$$O(km + kn \log n + k^2 n).$$

Skipping H3 reduces the total complexity to $O(km + kn \log n)$ and also removes the requirement to implement the graph as an adjacency matrix; but it will also increase the

number of loops that need to be handled by the forwarding layer. We explore this trade-off in Section V.

Even when keeping H3, ALR's complexity is the same or just slightly higher than the Downward Criterion or NLSR (both of them require $O(km+kn\log n)$), depending on whether k is smaller or larger than $\log n$. Fortunately, since the node degree follows a power-law distribution [7] the number k is small for most nodes; for example, even in the 315 node Sprint topology 68% of nodes have 5 or less links and 85% of nodes have 8 or less links. Moreover, after computing the shortest path, a router can already begin forwarding and finish the computation for the other paths as a lower-priority background task.

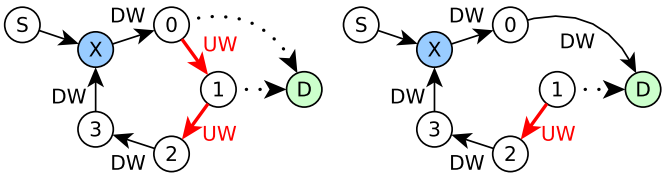
Algorithm 3 Apply Heuristics & Add Nexthops to FIB

```
forall singleDstEntry in shortestPathMap do
    dstId ← singleDstEntry.first;
    FibNextHop spNexthop ← singleDstEntry.second;
    Add spNexthop to FIB with type DOWNWARD;
    forall neighborEntry in nbSpSrcNodeRemoved do
        FibNextHop n_i ← neighborEntry.second[dstId];
        // nbSpSrcNodeRemoved includes Heuristic 1
        if n_i is downward of spNexthop then
            Add n_i to FIB with type DOWNWARD;
        else
            Add n_i to FIB with type DISABLED;
        end
    end
    numFibEntries ← count(FIB, type != DISABLED);
    if numFibEntries == 1 then
        bool mighLoop ← false;
        forall n_i in disabledFibEntries do
            // Perform Heuristic 2:
            neighborSp ← nbSpSrcLinkRemoved[n_i.id][dstId];
            mightLoop ← neighborSp.contains(x.id);
            forall n_j ≠ n_i in disabledFibEntries do
                // Perform Heuristic 3:
                if isConnected(n_i,n_j) && n_i.isUpwardOf(n_j) &&
                n_j.type == UPWARD then
                    mightLoop ← mightLoop || true;
                end
            end
            if mightLoop == false then
                Add n_i to FIB with type UPWARD;
                break;
            end
        end
    end
end
```

Note that the only change necessary to a link-state routing protocol (like NLSR) is the route calculation part. ALR neither requires further message exchange, nor any other modification to the link-state semantics; all required topology information is already signaled by the link state advertisements.

IV. Forwarding Loop Removal

Since ALR may produce some looping paths, we need a way to handle loops at the forwarding layer. After detecting a loop via the Interest nonce, a router can try different nexthops or backtrack to the previous router, so each Interest packet is guaranteed to eventually reach the destination. However, since this path exploration can be immensely costly (see Section V), we need a way to *permanently* remove loops at the forwarding layer; one of the routers involved in the loop needs to disable the nexthop it used to forward the packet on, not only for the current packet, but for all future packets (until the next routing change). The question arises: *which nexthop should be disabled?*



(a) Before (b) After

Fig. 6. Example of Forwarding Loop Removal

Current loop handling schemes either disable the nexthop at the router that detected the loop or at the router directly downstream to it (which is informed of the loop via a NACK) [32]. This mechanism mixes the two problems to be solved during a loop: 1) retrieving the Data for the current looped packet; 2) changing FIB entries to prevent loops of *future* packets. By disabling directly adjacent nexthops independent of their qualitative type (downward or upward), current schemes will often disable nexthops that go closer to the destination, leading to a higher number of removed nexthops and to longer remaining paths (see Section V).

We show that a better result can be achieved by splitting these two problems: For (1) we use the traditional method of trying different outgoing interfaces and sending back a NACK to the downstream once all outgoing faces are exhausted [32]. This recursive backtracking guarantees that every looped Interest packet will eventually retrieve the Data, making the process seamless for the data consumer.

For (2) we use our new mechanism of *Uphill Nexthop Removal* (UNR): we only remove FIB nexthops that lead further away from the destination; these nexthop entries may or may not be directly adjacent to the router detecting the loop. The looping path may contain multiple uphill nexthops, but to avoid future loops it is often sufficient to remove only one of them; thus the question becomes: *which uphill nexthop should be disabled?*

Through empirical tests, we found that we can preserve a higher path choice, while achieving complete loop-freeness, by removing the first uphill nexthop *on the path that the looped Interest took*; removing a random uphill nexthop or the last uphill nexthop on the Interest path (i.e. the first in the direction a NACK would take) leads to worse results. Thus, instead of signaling the Uphill Nexthop Removal via a NACK, we use a *loop signaling Interest*, a special Interest packet that is not expected to return any data, but follows the path of the original Interest to indicate that a loop has occurred. A router that receives the signaling Interest checks the type of the nexthop it used to forward the original Interest to. If the type is `Downward`, it forwards the signaling Interest to this nexthop. If the type is `Upward`, the router drops the signaling Interest and changes the type to `Disabled`. To prevent certain DoS attacks, signaling Interests should be signed by the sender and receiving routers should only consider signaling Interests from a trusted source.

An example of the UNR algorithm is shown in Figure 6.

TABLE II
EVALUATION TOPOLOGIES

Name	Nodes	Links	Degree
Abilene	11	14	2.55
Geant	27	38	2.82
Testbed	33	87	5.27
Exodus	79	147	3.72
Ebone	87	161	3.70
Telstra	108	153	2.83
Abovenet	141	374	5.31
Tiscali	161	328	4.07
Sprint	315	972	6.17

The original Interest takes the path $S \to X \to 0 \to 1 \to 2 \to 3 \to X$, at which point X detects the loop by observing the duplicate nonce. A traditional loop-removal scheme would now disable either the nexthop $X \to 0$ or $3 \to X$, both of which are poor choices, as they lead closer to the destination. Instead, UNR sends the loop signaling Interest packet along the loop and router 0 will remove the uphill nexthop $0 \to 1$. Afterwards, this loop is avoided as node 0 will choose a different nexthop towards destination D. In Section V-B, we show that our scheme of removing only upward nexthops (UNR) converges to a loop-free network after disabling much fewer nexthops than schemes that simply disable nexthops adjacent to the router detecting the loop.

V. EVALUATION

In this section, we evaluate our proposed routing algorithm and forwarding scheme. To emulate NDN's forwarding behavior, we use a custom C++ simulator based on the LEMON graph library [6].

We compare 9 topologies (Table II) of different size, node degrees, and link metrics: 1) The real Abilene and GEANT topologies with randomly generated link weights in the range between 1 and 100, 2) the NDN Testbed (Figure 4) in its current size with the currently assigned link weights, 3) the six measured ISP topologies from the Rocketfuel [26] dataset together with their inferred link weights [15]: Exodus (AS3967), Ebone (AS1755), Telstra (AS1221), Abovenet (AS6461), Tiscali (AS3257), and Sprint (AS1239).

A. Almost Loop-free Routing

First, we compare Almost Loop-free Routing (ALR) to the loop-free routing algorithms discussed in Section II and to non-loopfree variants of NLSR, all of which differ in how they determine the set of nexthops at each router:

- **ECMP:** Equal Cost Multi-Path [1] uses the nexthop of the shortest path n_{sp}, plus any nexthop n_i with the same cost: $cost(n_i, dst) = cost(n_{sp}, dst)$.
- **DW:** Downward paths [8] include the shortest path nexthop plus any nexthop that is closer to the destination than the current node x: $cost(n_i, dst) < cost(x, dst)$.
- **DWE:** Downward Path + Equal Cost Nexthop. DWE includes all nexthops from DW and, in addition, all nexthops with an equal cost to the destination and a lower node id.
- **ALR/ALR-H2:** Our routing scheme, once using all three heuristics (ALR) and once using only the first two (ALR-H2).
- **MARA:** The algorithms from the paper "Maximum Alternative Routing Algorithm" [21]. We report the data from the original paper where it is available, namely for two of their algorithms (MARA-SPE and MARA-MC) and in four of the Rocketfuel topologies.[4]
- **NLSR:** NLSR [10] applies only the first of ALR's heuristics (see Section III); thus for node x, NLSR will include all possible nexthops except the ones that can reach the destination only by looping back through x.
- **NLSR2/NLSR3:** NLSR allows to specify the maximal number of nexthops per name-prefix that are included in the FIB. Thus, NLSR2 and NLSR3 restrict the FIB to the two, or respectively three, best (= lowest cost) nexthops.

1) Metrics: We evaluate the presented routing algorithms in terms of their *computational complexity*, resulting *path choice*, and *path quality.*

The *computational complexity* of a routing algorithm determines how scalable, and thus useful, it is for real-world deployment; a routing algorithm that needs exponential time is of little practical use, no matter how good the resulting paths. We quantify this scalability with the well-known O-notation.

For *path choice*, we measure the average number of nexthops in the FIB at each node, together with its standard deviation (given in parenthesis). We also measure the percentage of nodes that have two or more nexthops (NH>1) towards a given destination.

For *path quality*, we run an experiment between all source-destination pairs, that is, $N = n * (n-1)$ times, where *n* is the number of nodes in the network. At each hop, we randomly pick a nexthop from the available ones, until we either reach the destination, run into a loop (a node where the packet has previously been), or run into a dead end (a node with no forwarding options other than directly returning the packet to the previous node). Then we record the percentage of src-dst pairs that ran into a loop (L) or dead end (D). Whenever the packet runs into a loop, the node detecting the loop forwards it on a different interface; when all interfaces are exhausted or the packet runs into a dead end, it is backtracked to the previous node which then tries a different interface (similar Yi et al.'s NACK retry mechanism [32]). With this recursive backtracking, each packet will ultimately reach the destination. Once it did so, we record the *full path length*, including all hops of the backtracking steps. For example, a packet with destination D taking the path $A \to B \to A \to D$ will count as a path length of three.

2) Results: The computational complexity is given in Table III, where **k** denotes the number of neighbors per node, **m** the number of edges in the network, and **n** the number of nodes in the network. ECMP has the same complexity as running Dijkstra's algorithm, $O(m + n \log n)$, which is scalable and commonly used in IP link-state routing algorithms like OSPF or IS-IS. DW, DWE, NLSR, and ALR-H2 need to run Dijkstra's algorithm for each link of a router, leading to a complexity

[4]The authors measure the path length in number of involved nodes, rather than our measurement of the number of hops between those nodes. Thus we have to subtract 1 from their numbers (see Table VII).

TABLE III
ROUTING COMPLEXITY

Algorithm	Comp. Complexity
ECMP	$O(m + n \log n)$
DW, DWE	$O(km + kn \log n)$
NLSR (2/3)	$O(km + kn \log n)$
ALR-H2	$O(km + kn \log n)$
ALR	$O(km + kn \log n + k^2 n)$
MARA-MC	$O(mn + n^2)$
MARA-SPE	$O(mn + n^2 \log n)$

that is k times higher: $O(km + kn \log n)$. The two MARA algorithms perform more complex graph calculations, which need $O(mn)$ or even $O(n^2 \log n)$. ALR falls in the middle of these options: it is slightly more complex than DWE and NLSR, but significantly less complex than the two MARA algorithms.

The results on path choice and path quality are presented in Table IV. The first three routing algorithms (ECMP, DW, DWE) are guaranteed to be *loop-free*, but result in a limited choice of nexthops: the average number of nexthops and the percentage of nodes with more than one nexthop are both significantly lower than in the non loop-free algorithms. ECMP requires to match routing costs *exactly*, which in most cases restricts forwarding to only one nexthop; thus ECMP performs especially poor in topologies with fine-grained link costs like Abilene or GEANT. DW performs better than ECMP as it doesn't depend on exact cost matching. However, DWE is strictly superior to both schemes, as it has the same computational complexity and returns a path choice that is at least as high, and often higher. Thus, we consider DWE as reference point for loop-free routing algorithms.

The three NLSR variants provide a much higher nexthop choice than DWE. For example, NLSR always allows two or more nexthops, unless some routers have only one link, in which case none of the routing algorithms achieves a NH>1 of 100%. However, NLSR also results in frequent loops and a high total path length. With random forwarding, NLSR causes loops in over 90% of all src-dst pairs and the resulting average path length often surpasses the number of nodes in the network. NLSR3 and NLSR2 lead to slightly fewer loops, but their average path length is still many times higher than the optimum. As a result, all NLSR variants strictly require an intelligent forwarding strategy. In contrast, the loop-free routing schemes work quite well even with random forwarding or round robin.

ALR creates a higher path choice than the loop-free routing schemes, while leading to much fewer loops than NLSR (2/3). In most cases, ALR achieves its goal of providing at least 2 nexthops for each node: except in the Abilene topology, it is never more than 2.6% away from the maximal number of nodes with at least 2 nexthops (achieved by NLSR), and in the largest three topologies it is withing 0.2% of the maximum. Moreover, in all tested topologies, ALR's looping chance is at least an order of magnitude smaller than NLSR's and its average full path length is much closer to the one of DWE. In particular, ALR seems strictly superior (i.e. creating higher nexthop choice while leading to fewer loops) to all cases of NLSR2 and some cases of NLSR3 (Abovenet, Tiscali, Sprint).

TABLE IV
ROUTING ALGORITHMS + RANDOM FORWARDING

Topo	Routing	NextHops (#)		NH>1	FullPathLen (Hops)		L(%)	D(%)
Abilene	ECMP	1.00	(±0.00)	0.0	2.80	(±1.47)	0.0	0.0
	DW	1.40	(±0.49)	40.0	2.85	(±1.52)	0.0	0.0
	DWE	1.40	(±0.49)	40.0	2.85	(±1.52)	0.0	0.0
	ALR	1.92	(±0.28)	91.8	3.67	(±1.90)	0.0	0.0
	ALR-H2	1.92	(±0.28)	91.8	3.66	(±1.89)	0.0	0.0
	NLSR2	2.00	(±0.00)	100.0	4.64	(±2.87)	15.8	0.0
	NLSR3	2.55	(±0.50)	100.0	8.32	(±7.20)	43.3	0.0
	NLSR	2.55	(±0.50)	100.0	8.29	(±7.21)	43.1	0.0
Geant	ECMP	1.00	(±0.00)	0.0	3.18	(±1.29)	0.0	0.0
	DW	1.46	(±0.71)	35.5	3.37	(±1.38)	0.0	0.0
	DWE	1.46	(±0.71)	35.9	3.38	(±1.38)	0.0	0.0
	ALR	1.75	(±0.66)	64.7	4.26	(±1.93)	0.8	0.0
	ALR-H2	1.75	(±0.66)	64.7	4.27	(±1.93)	0.8	0.0
	NLSR2	1.65	(±0.48)	65.4	7.45	(±6.19)	29.8	0.0
	NLSR3	2.05	(±0.86)	65.4	12.74	(±13.18)	50.0	0.0
	NLSR	2.46	(±1.67)	65.4	18.76	(±17.88)	63.0	0.0
Testbed	ECMP	1.06	(±0.24)	5.8	3.01	(±1.26)	0.0	0.0
	DW	2.60	(±1.30)	73.9	3.61	(±1.74)	0.0	0.0
	DWE	2.72	(±1.29)	77.7	3.77	(±1.84)	0.0	0.0
	ALR	2.91	(±1.06)	97.3	4.59	(±2.59)	1.6	0.5
	ALR-H2	2.92	(±1.04)	98.4	4.94	(±3.06)	7.0	0.5
	NLSR2	2.00	(±0.00)	100.0	9.43	(±9.31)	40.4	0.0
	NLSR3	3.00	(±0.00)	100.0	19.03	(±22.75)	60.6	0.0
	NLSR	5.27	(±1.31)	100.0	40.31	(±43.06)	78.5	0.0
Exodus	ECMP	1.20	(±0.47)	16.5	4.87	(±2.19)	0.0	0.0
	DW	1.80	(±1.05)	50.5	5.40	(±2.62)	0.0	0.0
	DWE	1.89	(±1.10)	54.2	5.57	(±2.72)	0.0	0.0
	ALR	2.21	(±0.86)	89.5	8.09	(±5.62)	6.8	0.9
	ALR-H2	2.21	(±0.86)	89.6	8.16	(±5.62)	8.8	0.9
	NLSR2	1.91	(±0.29)	91.0	13.50	(±10.80)	50.5	0.0
	NLSR3	2.58	(±0.65)	91.0	47.18	(±54.60)	76.1	0.0
	NLSR	3.63	(±2.06)	91.0	116.69	(±117.75)	88.9	0.0
Ebone	ECMP	1.20	(±0.45)	17.5	5.07	(±2.14)	0.0	0.0
	DW	1.75	(±1.06)	45.4	5.35	(±2.21)	0.0	0.0
	DWE	1.87	(±1.07)	53.2	5.45	(±2.27)	0.0	0.0
	ALR	2.17	(±0.91)	83.6	7.38	(±3.63)	2.9	1.1
	ALR-H2	2.18	(±0.90)	84.8	7.69	(±4.01)	6.9	1.0
	NLSR2	1.86	(±0.35)	86.0	13.59	(±10.21)	50.7	0.0
	NLSR3	2.50	(±0.73)	86.0	53.40	(±61.51)	76.7	0.0
	NLSR	3.54	(±1.98)	86.0	125.87	(±125.07)	89.1	0.0
Telstra	ECMP	1.08	(±0.28)	7.8	4.79	(±1.80)	0.0	0.0
	DW	1.32	(±0.75)	23.6	4.97	(±1.85)	0.0	0.0
	DWE	1.47	(±0.94)	29.1	5.43	(±2.17)	0.0	0.0
	ALR	1.62	(±0.89)	47.0	7.02	(±3.58)	3.1	0.0
	ALR-H2	1.62	(±0.89)	47.4	7.18	(±3.70)	7.7	0.0
	NLSR2	1.48	(±0.50)	47.6	12.36	(±9.65)	50.6	0.0
	NLSR3	1.72	(±0.83)	47.6	36.48	(±36.62)	70.9	0.0
	NLSR	2.37	(±2.17)	47.6	102.92	(±94.86)	85.1	0.0
Abovenet	ECMP	1.23	(±0.56)	17.8	4.64	(±2.09)	0.0	0.0
	DW	2.57	(±1.60)	69.6	5.21	(±2.43)	0.0	0.0
	DWE	2.72	(±1.68)	73.7	5.47	(±2.57)	0.0	0.0
	ALR	2.90	(±1.44)	93.4	6.90	(±3.74)	4.2	0.0
	ALR-H2	2.90	(±1.44)	93.4	7.08	(±3.98)	7.4	0.0
	NLSR2	1.93	(±0.25)	93.4	13.90	(±12.88)	47.4	0.0
	NLSR3	2.75	(±0.57)	93.4	64.48	(±88.84)	73.5	0.0
	NLSR	5.32	(±3.33)	93.4	318.18	(±329.86)	92.8	0.0
Tiscali	ECMP	1.17	(±0.47)	13.7	5.80	(±2.80)	0.0	0.0
	DW	2.00	(±1.81)	44.9	6.11	(±2.81)	0.0	0.0
	DWE	2.05	(±1.85)	46.8	6.26	(±2.90)	0.0	0.0
	ALR	2.29	(±1.73)	71.2	8.68	(±5.84)	7.6	0.0
	ALR-H2	2.29	(±1.73)	71.2	9.08	(±8.96)	8.9	0.0
	NLSR2	1.71	(±0.45)	71.3	18.77	(±19.61)	48.4	0.0
	NLSR3	2.21	(±0.86)	71.3	54.15	(±74.18)	69.0	0.0
	NLSR	3.77	(±3.61)	71.3	230.90	(±245.64)	90.3	0.0
Sprint	ECMP	1.39	(±0.86)	27.3	4.24	(±1.64)	0.0	0.0
	DW	2.57	(±2.41)	65.8	4.99	(±2.03)	0.0	0.0
	DWE	3.10	(±2.90)	75.3	5.91	(±2.47)	0.0	0.0
	ALR	3.19	(±2.37)	89.9	6.76	(±3.42)	1.7	0.0
	ALR-H2	3.19	(±2.37)	90.0	7.25	(±6.47)	6.8	0.0
	NLSR2	1.90	(±0.30)	90.1	16.03	(±18.01)	48.6	0.0
	NLSR3	2.61	(±0.66)	90.1	70.66	(±111.78)	74.4	0.0
	NLSR	6.07	(±6.76)	90.1	962.24	(±953.75)	96.9	0.0

ALR with Heuristic 3 disabled (ALR-H2) performs worse than the unmodified ALR (more loops and longer paths); the poorer performance comes as a trade-off against a lower computational complexity, one that matches NLSR. Still, compared to any of the NLSR variants, ALR-H2 performs

significantly better, causing fewer loops and creating shorter paths.

This evaluation was done assuming random nexthop selection at each node, and similar results can be achieved by using round-robin. We realize that this does not represent the behavior of real forwarding strategies, which most likely will use some sort of performance estimation of paths, so that the path length and the number of loops will be lower in practice. Nevertheless, this evaluation helps to quantify the *effort* the forwarding layer has to take up in order to avoid loops and find shorter paths, and the *maximal possible nexthop choice* it is given. As we have seen, the NLSR variants burden the forwarding layer with a much higher effort than their alternatives, and the loop-free routing schemes overly restrict the possible nexthop choice.

For all tested topologies, ALR(-H2) gives a good trade-off between increasing the nexthop choice, selecting shorter paths, and avoiding loops. In the next section, we look at the interaction between the presented routing algorithms and different forwarding strategies.

B. Forwarding Loop Removal

We now evaluate different loop removal strategies at the forwarding layer to show the benefit of knowing the nexthop type (upward/downward), and more specifically, of removing only upward nexthops (UNR). We compare UNR against the Green/Yellow/Red coloring scheme (COL) from [32], a strategy that is not aware of the nexthop type and disables nexthops at the node where a loop is detected or as close to that node as possible. Our earlier analysis (see Section IV) predicts that this type-agnostic loop handling disables the wrong nexthops, leading to less nexthop choice, longer remaining paths, and higher total forwarding effort.

1) Coloring Scheme: Yi et al's Green/Yellow/Red scheme [32] specifies the exact single outgoing face to use: choose the highest ranked Green face; if none is available, choose the highest ranked Yellow face. To allow for traffic splitting (using more than a single best face), we need to make two changes: 1) we initiate all interfaces to the state `Green` instead of `Yellow`; 2) we allow to use any available Green face, instead of just the highest ranked Green face. The detailed forwarding behavior is as follows:

1) Routers initiate all faces in their FIB to `Green`.
2) A Router will forward an Interest too a randomly chosen `Green` face, if one is available; if not, it will randomly choose a `Yellow` face.
3) If a router receives a looped Interest (one that arrived from a different face earlier), it forwards the packet on a yet unused face (following the Green/Yellow order).
4) If a router has exhausted all forwarding options (after a loop or in a dead end), it sends back a Negative Acknowledgement (NACK) to the previous router. The router receiving the NACK then switches the face to `Yellow`, tries a different one for forwarding, and if all faces are exhausted, returns a NACK itself.
5) If a router successfully receives a Data packet, it turns the corresponding face `Green`.

TABLE V
ACCURACY OF PATH CHOICE ESTIMATE

	Exodus		Ebone		Telstra		Tiscali	
	ECMP	DW	ECMP	DW	ECMP	DW	ECMP	DW
Exact[21]	1.89	75.98	2.07	16.59	1.37	3.37	2.14	607.56
Estimate	1.89	78.45	2.07	16.52	1.37	3.38	2.14	663.62
Err (%)	**0.00**	**3.25**	**0.00**	**-0.42**	**0.00**	**0.30**	**0.00**	**9.23**

With this recursive backtracking procedure, all packets will eventually reach the destination. After many runs, the remaining Green faces are very likely (above 99%) to be loop-free.

2) Scenario Description: In this experiment we run both schemes for all source-destination pairs until all loops in the network are removed. We then compare the results in terms of *path diversity*, *path quality*, and *forwarding complexity*.

We evaluate all routing algorithms listed in the last section. To allow NLSR to work with UNR, we extend its functionality to contain our qualitative nexthop type: the lowest cost nexthop (which is known to NLSR) is assigned the type `Downward`, all other nexthops get the type `Upward`. We label these extended NLSR schemes XNLSR, XNLSR2, and XNLSR3. The extension allows us to distinguish the benefit gained by UNR's type-aware nexthop removal from the benefit gained by ALR's loop-freeness heuristics.

3) Metrics: For *path choice*, we compare the number of remaining nexthops at each node together with their standard deviation and the percentage of nodes with at least 2 nexthops (NH>1). Moreover, we evaluate the number of possible paths between each source-destination pair (listed as the median, mean, and standard deviation). The exact number of paths between a src-dst pair can be determined by a depth-first search (DFS) with backtracking, which however has an exponential complexity, making it infeasible for the larger Rocketfuel topologies (compare [21]). Thus, we estimate the number of paths by the product of available nexthops at each node along a random path between each src-dst pair: $\#Paths \approx nh_0 * nh_1 * \cdots * nh_{d-1}$. When averaged over 100 runs, our estimate comes close to the exact numbers from Ohara et al. [21] (see Table V), at least in the smaller topologies where this comparison is feasible. Moreover, the conclusions drawn from the number of paths are also supported by the numbers of nexthops at each node, a much more stable measure.

For *path quality*, we consider the length of the remaining paths together with their standard deviation. Since all remaining paths are loop-free, this length does not include any backtracking steps like the path length in the last section. A higher average path length is acceptable as long as it is the result of higher path choice (if needed, the forwarding layer can still restrict traffic to shorter paths). However, when two algorithms provide similar path choice, a shorter path length is clearly more desirable.

For the *forwarding complexity*, we compare the number of FIB nexthop entries that need to be changed. For UNR, a change is always a permanent disabling of the nexthop entry; disabled nexthops will not be changed back unless there is a change in the routing topology (e.g. link removal or changed link cost). For COL, a change is when the FIB type switches

from Green to Yellow, or from Yellow to Green. These changes are reversible (a face turned Yellow might be turned Green again by returning Data); thus the percentage of changed FIB nexthops may exceed 100%.

4) Results – Uphill Nexthop Removal vs. Coloring Scheme: The results are shown in Tables VI and VII. The loop-free routing algorithms (ECMP, DW, and DWE) leave no loops to be removed, thus there is no difference between UNR and COL. Similarly, ALR leaves very few loops to be removed, thus the difference between forwarding schemes is small. Nevertheless, for the small amount of remaining loops, UNR removes less FIB nexthops and retains more and shorter paths than COL.

For NLSR (2/3), we see the same results at a larger scale: UNR outperforms COL in all tested topologies and for all tested metrics.

We can see another result when considering the change from NLSR2 over NLSR3 to NLSR (increasing the initial path choice but also loop potential). Both forwarding schemes need to remove increasingly more nexthops to make the network loop-free, but the resulting path choice and quality are in stark contrast: UNR gets better, the higher the initial path choice; COL gets worse. UNR always retains more path choice when combined with NLSR than when combined with NLSR2/3; COL often retains less path choice with NLSR than with NLSR2/3. In addition, UNR keeps the paths similarly short in all NLSR schemes; COL's paths get increasingly longer.

How is that possible? The worsening results from NLSR2 to NLSR clearly show that COL removes the wrong nexthops: it often removes nexthops that go closer to the destination, leading to more nexthops needing to be removed and to longer remaining paths. For a type-agnostic strategy like COL, NLSR2 provides better guidance than the full NLSR: it pre-selects the two best nexthops, which are more likely to be loop-free already. However, a more intelligent loop removal scheme like UNR can take advantage of the full NLSR, retaining a higher path choice than NLSR2, combined with any forwarding scheme, could.

A final nail in the coffin of the Coloring scheme is that it doesn't always converge to a final state: sometimes nexthops indefinitely switch between the state `Green` and `Yellow` (indicated in the tables as ∞). This artifact comes from the fundamental problem that a packet loop depends on the whole path a given packet took and not only the current nexthop; turning nexthops to `Green` on received data can re-enable a nexthop that later will lead to a loop when used as part of a longer path. This loop will then turn the nexthop `Yellow` again, repeating this process indefinitely.

UNR does not suffer from this problem: since all of UNR's changes are permanent, a disabled nexthop will never be re-enabled by the forwarding layer, avoiding indefinite oscillations. Making changes permanent is only possible because UNR's changes consider the nexthop type, and thus are of much higher quality than COL's changes.

5) Further Results for Routing Algorithms: When using UNR together with non loop-free routing algorithms, we gain further results about their performance.

First, we compared UNR to the two MARA routing schemes [21], with somewhat ambiguous results. MARA-MC leads to higher path lengths and often doesn't include the shortest path, which increases the forwarding cost. In contrast, MARA-SPE always includes the shortest path and leads to overall shorter paths. Compared to ALR, MARA-SPE often produces a lower average path choice, but also a lower variance, which means that it could provide a higher number of paths for nodes that have few paths. However, both MARA algorithms show a significantly higher computational complexity than any other tested routing algorithm (see Table III), potentially hindering their deployment.

Second, we observe that the final loop-free path choice of ALR + UNR is always higher than the best loop-free routing algorithm (DWE). We conclude that the higher computational effort of almost loop-free routing pays off, at least in this regard.

Third, when compared with the nexthop type-extended XNLSR, ALR requires a much lower forwarding layer effort: depending on the topology, it only needs to remove 0% to 1.1% of all FIB entries (ALR-H2 removes 0% to 2.2%) rather than XNLSR's 22% to 42%. Moreover, even with this higher overhead, XNLSR's resulting path choice is *not* obviously better: while XNLSR + UNR achieves a higher number of average nexthops per node, ALR + UNR achieves a higher percentage of nodes with at least 2 nexthops! In addition, ALR + UNR often achieves a higher *median* number of paths at each node, implying that it achieves its goal of improving the path choice for the nodes that most lack it.

VI. Concluding Remarks

Our heuristics leave room for improvements that achieve better trade-offs between computational complexity, path choice, and path quality (see the differences between ALR and ALR-H2). Hence, rather than claiming that the two presented schemes are the best of all possible choices, we make the larger point that non loop-free routing combined with loop removal at the forwarding layer leads to better path choice than the best current loop-free routing schemes.

The fundamental reason for this performance improvement is that loop-free multipath routing is too complex to find an optimal solution. Instead, one has to use heuristics, the current best of which is the *downward criterion* (always sending packets closer to the destination). We have extended these heuristics by exploiting two functions of the NDN forwarding plane, *loop detection* and *incoming interface-exclusion*, allowing us to increase path choice at a reasonable complexity.

Acknowledgements

We want to thank Teng Liang and Eric Newberry for proof-reading, John DeHart for his help in creating the NDN Testbed topology map, and Lan Wang and Lixia Zhang for giving invaluable feedback on our design.

References

[1] Analysis of an Equal-Cost Multi-Path Algorithm. RFC 2992, Nov. 2000.
[2] A. K. Atlas and A. Zinin. Basic specification for ip fast-reroute: loop-free alternates. 2008.

TABLE VI
ROUTING ALGORITHMS + FORWARDING LOOP REMOVAL

Topo	Routing	FW	NextHops (#)		NH>1	# Paths (Median,Mean,STD)			PathLen (Hops)		ChangedFibs (#)		ChangedFibs (%)	
Abilene	ECMP		1.00	(±0.00)	0.0	1	1.00	(±0.00)	2.80	(±1.47)	0	(±0)	0.00	(±0.00)
	DW		1.40	(±0.49)	40.0	2	1.78	(±1.18)	2.85	(±1.51)	0	(±0)	0.00	(±0.00)
	DWE		1.40	(±0.49)	40.0	2	1.78	(±1.18)	2.84	(±1.52)	0	(±0)	0.00	(±0.00)
	ALR	COL	1.92	(±0.27)	91.8	4	4.06	(±2.93)	3.65	(±1.89)	0	(±0)	0.00	(±0.00)
	ALR	UNR	1.92	(±0.27)	91.8	4	4.05	(±2.91)	3.65	(±1.89)	0	(±0)	0.00	(±0.00)
	ALR-H2	COL	1.92	(±0.27)	91.8	4	4.07	(±2.95)	3.65	(±1.89)	0	(±0)	0.00	(±0.00)
	ALR-H2	UNR	1.92	(±0.27)	91.8	4	4.05	(±2.90)	3.64	(±1.89)	0	(±0)	0.00	(±0.00)
	NLSR2	COL	1.76	(±0.43)	76.2	2	2.52	(±1.52)	3.70	(±1.95)	26	(±2)	11.89	(±0.71)
	XNLSR2	UNR	1.86	(±0.35)	86.1	2	3.06	(±1.63)	3.72	(±1.92)	15	(±2)	6.93	(±1.11)
	NLSR3	COL	1.65	(±0.64)	55.5	2	3.21	(±1.88)	3.99	(±2.22)	173	(±22)	61.66	(±7.68)
	XNLSR3	UNR	1.91	(±0.66)	73.4	2	3.84	(±3.30)	3.70	(±2.02)	70	(±4)	25.02	(±1.40)
	NLSR	COL	1.64	(±0.64)	54.7	2	2.73	(±1.32)	3.77	(±1.94)	174	(±20)	62.27	(±7.11)
	XNLSR	UNR	1.98	(±0.63)	79.1	3	4.29	(±3.46)	3.81	(±2.09)	62	(±2)	22.25	(±0.80)
Geant	ECMP		1.00	(±0.00)	0.0	1	1.00	(±0.00)	3.18	(±1.29)	0	(±0)	0.00	(±0.00)
	DW		1.46	(±0.71)	35.5	2	2.82	(±3.09)	3.37	(±1.38)	0	(±0)	0.00	(±0.00)
	DWE		1.46	(±0.71)	35.9	2	2.85	(±3.10)	3.38	(±1.38)	0	(±0)	0.00	(±0.00)
	ALR	COL	1.75	(±0.66)	64.6	4	8.25	(±11.42)	4.24	(±1.86)	2	(±2)	0.16	(±0.19)
	ALR	UNR	1.75	(±0.66)	64.5	4	8.16	(±11.53)	4.21	(±1.85)	1	(±0)	0.09	(±0.03)
	ALR-H2	COL	1.75	(±0.66)	64.5	4	8.02	(±11.08)	4.22	(±1.85)	3	(±3)	0.25	(±0.21)
	ALR-H2	UNR	1.75	(±0.66)	64.5	4	8.06	(±11.31)	4.20	(±1.85)	1	(±0)	0.10	(±0.04)
	NLSR2	COL	1.50	(±0.50)	50.1	2	3.01	(±2.42)	4.20	(±2.03)	∞	(±∞)	∞	(±∞)
	XNLSR2	UNR	1.59	(±0.49)	59.3	2	3.48	(±2.72)	4.27	(±2.05)	42	(±3)	3.65	(±0.29)
	NLSR3	COL	1.51	(±0.69)	39.8	2	4.94	(±7.48)	4.66	(±2.56)	∞	(±∞)	∞	(±∞)
	XNLSR3	UNR	1.78	(±0.81)	53.7	3	6.02	(±11.40)	4.21	(±2.05)	194	(±8)	13.46	(±0.52)
	NLSR	COL	1.42	(±0.88)	29.0	2	4.71	(±9.56)	4.98	(±2.76)	1,439	(±73)	83.48	(±4.23)
	XNLSR	UNR	1.89	(±0.90)	59.9	6	11.42	(±23.62)	4.29	(±2.00)	399	(±6)	23.16	(±0.37)
Testbed	ECMP		1.06	(±0.24)	5.8	1	1.10	(±0.31)	3.01	(±1.26)	0	(±0)	0.00	(±0.00)
	DW		2.60	(±1.30)	73.9	5	42.53	(±484.05)	3.61	(±1.74)	0	(±0)	0.00	(±0.00)
	DWE		2.72	(±1.29)	77.7	6	139.00	(±5.63e3)	3.76	(±1.84)	0	(±0)	0.00	(±0.00)
	ALR	COL	2.89	(±1.07)	96.2	16	265.56	(±3.79e3)	4.41	(±2.22)	27	(±7)	0.87	(±0.23)
	ALR	UNR	2.90	(±1.07)	96.2	16	287.89	(±4.56e3)	4.40	(±2.21)	11	(±1)	0.37	(±0.04)
	ALR-H2	COL	2.83	(±1.09)	93.9	16	225.07	(±3.10e3)	4.47	(±2.29)	122	(±12)	3.97	(±0.39)
	ALR-H2	UNR	2.86	(±1.12)	92.0	12	245.69	(±5.76e3)	4.27	(±2.15)	67	(±2)	2.19	(±0.06)
	NLSR2	COL	1.68	(±0.47)	68.4	4	13.23	(±134.41)	4.41	(±2.60)	583	(±16)	27.60	(±0.78)
	XNLSR2	UNR	1.80	(±0.40)	79.6	4	10.16	(±94.95)	3.93	(±1.99)	216	(±7)	10.21	(±0.33)
	NLSR3	COL	1.86	(±0.81)	59.1	4	33.23	(±303.52)	5.24	(±3.14)	2,186	(±158)	69.02	(±5.00)
	XNLSR3	UNR	2.37	(±0.77)	82.1	6	45.99	(±635.94)	4.23	(±2.18)	669	(±11)	21.13	(±0.35)
	NLSR	COL	2.00	(±1.18)	60.1	4	23.46	(±703.68)	5.37	(±3.44)	6,098	(±234)	109.52	(±4.21)
	XNLSR	UNR	3.20	(±1.46)	83.7	12	655.54	(±2.59e4)	4.23	(±2.12)	2,193	(±10)	39.39	(±0.19)

[3] D. O. Awduche and J. Agogbua. Requirements for Traffic Engineering Over MPLS. RFC 2702, Sept. 1999.

[4] O. Bonaventure. *Computer networking: Principles, protocols, and practice*. The Saylor Foundation, 2011.

[5] G. Carofiglio, M. Gallo, L. Muscariello, M. Papalini, and S. Wang. Optimal multipath congestion control and request forwarding in ICN. In *ICNP*, 2013.

[6] B. Dezső, A. Jüttner, and P. Kovács. Lemon–an open source c++ graph template library. *ENTCS*, 2011.

[7] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the internet topology. In *ACM SIGCOMM CCR*. ACM, 1999.

[8] J. He and J. Rexford. Toward internet-wide multipath routing. *IEEE Network*, 2008.

[9] E. Hemmati and J. Garcia-Luna-Aceves. A new approach to name-based link-state routing for ICN. In *ACM ICN 2015*.

[10] A. Hoque, S. O. Amin, A. Alyyan, B. Zhang, L. Zhang, and L. Wang. Nlsr: named-data link state routing protocol. In *ACM SIGCOMM ICN workshop*, 2013.

[11] A. Kvalbein, C. Dovrolis, and C. Muthu. Multipath load-adaptive routing: Putting the emphasis on robustness and simplicity. In *ICNP 2009*. IEEE.

[12] S. Lee, Y. Yu, S. Nelakuditi, Z.-L. Zhang, and C.-N. Chuah. Proactive vs reactive approaches to failure resilient routing. In *INFOCOM 2004*.

[13] V. Lehman, A. Gawande, B. Zhang, L. Zhang, R. Aldecoa, D. Krioukov, and L. Wang. An experimental investigation of hyperbolic routing with a smart forwarding plane in NDN. In *IWQoS 2016*. IEEE.

[14] C. Li, T. Huang, R. Xie, H. Zhang, J. Liu, and Y. Liu. A novel multi-path traffic control mechanism in named data networking. In *IEEE ICT*, 2015.

[15] R. Mahajan, N. Spring, D. Wetherall, and T. Anderson. Inferring link weights using end-to-end measurements. In *IMW*, 2002.

[16] P. Mérindol, J.-J. Pansiot, and S. Cateloin. Improving load balancing with multipath routing. In *ICCCN'08*. IEEE, 2008.

[17] M. Mitzenmacher. The power of two choices in randomized load balancing. *IEEE TPDS*, 2001.

[18] M. Motiwala, M. Elmore, N. Feamster, and S. Vempala. Path splicing. In *ACM SIGCOMM CCR*. ACM, 2008.

[19] S. Nelakuditi, S. Lee, Y. Yu, Z.-L. Zhang, and C.-N. Chuah. Fast local rerouting for handling transient link failures. *IEEE/ACM ToN*, 2007.

[20] D. Nguyen, M. Fukushima, K. Sugiyama, and A. Tagami. Efficient multipath forwarding and congestion control without route-labeling in CCN. In *IEEE ICCW*, 2015.

[21] Y. Ohara, S. Imahori, and R. Van Meter. Mara: Maximum alternative routing algorithm. In *INFOCOM 2009*. IEEE.

[22] J. Qadir, A. Ali, K.-L. A. Yau, A. Sathiaseelan, and J. Crowcroft. Exploiting the power of multiplicity: a holistic survey of network-layer multipath. *IEEE Communications Surveys & Tutorials*, 2015.

[23] H. Qian, R. Ravindran, G.-Q. Wang, and D. Medhi. Probability-based adaptive forwarding strategy in NDN. In *IM 2013 Symposium*. IEEE, 2013.

[24] K. Schneider and U. R. Krieger. Beyond network selection: Exploiting access network heterogeneity with NDN. In *ACM ICN*, 2015.

[25] K. Schneider, C. Yi, B. Zhang, and L. Zhang. A practical congestion control scheme for named data networking. In *ACM ICN 2016*.

[26] N. Spring, R. Mahajan, and D. Wetherall. Measuring isp topologies with rocketfuel. *ACM SIGCOMM CCR*, 2002.

[27] C. Villamizar. OSPF Optimized Multipath (OSPF-OMP). Internet-Draft draft-ietf-ospf-omp-02, IETF, Feb. 1999. Work in Progress.

[28] S. Vutukury and J. Garcia-Luna-Aceves. A simple approximation to minimum-delay routing. In *ACM SIGCOMM CCR*. ACM, 1999.

[29] S. Vutukury and J. J. Garcia-Luna-Aceves. Mdva: A distance-vector multipath routing protocol. In *INFOCOM 2001*. IEEE, 2001.

[30] Y. Wang, N. Rozhnova, A. Narayanan, D. Oran, and I. Rhee. An improved hop-by-hop interest shaper for congestion control in NDN. *ACM SIGCOMM CCR*, 2013.

[31] X. Yang and D. Wetherall. Source selectable path diversity via routing deflections. In *ACM SIGCOMM CCR*. ACM, 2006.

[32] C. Yi, A. Afanasyev, I. Moiseenko, L. Wang, B. Zhang, and L. Zhang. A case for stateful forwarding plane. *Elsevier*, 2013.

[33] F. Zhang, Y. Zhang, A. Reznik, H. Liu, C. Qian, and C. Xu. A transport protocol for content-centric networking with explicit congestion control. In *IEEE ICCCN*, 2014.

[34] J. Zhou, Q. Wu, Z. Li, M. A. Kaafar, and G. Xie. A proactive transport mechanism with explicit congestion notification for NDN. In *IEEE ICC*, 2015.

TABLE VII
ROUTING ALGORITHMS + FORWARDING LOOP REMOVAL (CONT.)

Topo	Routing	FW	NextHops (#)		NH>1	# Paths (Median,Mean,STD)		PathLen (Hops)		ChangedFibs (#)		ChangedFibs (%)	
Exodus	ECMP		1.20	(±0.47)	16.5	1	1.89	(±1.94)	4.87	(±2.19)	0	(±0)	0.00	(±0.00)
	DW		1.80	(±1.05)	50.5	6	78.45	(±901.79)	5.40	(±2.63)	0	(±0)	0.00	(±0.00)
	DWE		1.89	(±1.10)	54.2	6	152.17	(±2.23e3)	5.57	(±2.72)	0	(±0)	0.00	(±0.00)
	MARA	SPE		(±)			32.52	(±76.92)	6.33	(±2.84)	0	(±0)	0.00	(±0.00)
	MARA	MC		(±)			42.08	(±66.94)	7.62	(±3.31)	0	(±0)	0.00	(±0.00)
	ALR	COL	2.16	(±0.84)	86.6	32	2,439.05	(±5.75e4)	7.21	(±3.70)	∞	(±∞)	∞	(±∞)
	ALR	UNR	2.19	(±0.88)	87.4	32	2,451.54	(±5.69e4)	7.11	(±3.63)	133	(±4)	0.98	(±0.03)
	ALR-H2	COL	2.16	(±0.85)	86.0	32	2,747.49	(±6.00e4)	7.15	(±3.67)	∞	(±∞)	∞	(±∞)
	ALR-H2	UNR	2.18	(±0.88)	86.7	32	2,337.49	(±4.81e4)	7.06	(±3.63)	179	(±3)	1.32	(±0.03)
	NLSR2	COL	1.70	(±0.46)	69.8	4	10.59	(±27.85)	6.62	(±3.68)	∞	(±∞)	∞	(±∞)
	XNLSR2	UNR	1.75	(±0.43)	75.2	4	9.66	(±31.87)	6.37	(±3.24)	974	(±13)	8.27	(±0.11)
	NLSR3	COL	1.68	(±0.74)	50.8	4	29.79	(±418.30)	8.27	(±4.74)	∞	(±∞)	∞	(±∞)
	XNLSR3	UNR	2.16	(±0.76)	77.8	8	74.66	(±1.28e3)	6.93	(±3.66)	2,572	(±24)	16.18	(±0.15)
	NLSR	COL	1.34	(±0.81)	22.6	4	52.16	(±2.69e3)	10.34	(±6.69)	30,567	(±617)	136.72	(±2.76)
	XNLSR	UNR	2.41	(±1.20)	79.0	24	4,475.50	(±3.42e5)	6.81	(±3.52)	7,519	(±34)	33.63	(±0.15)
Ebone	ECMP		1.20	(±0.45)	17.5	2	2.07	(±2.04)	5.07	(±2.14)	0	(±0)	0.00	(±0.00)
	DW		1.75	(±1.06)	45.4	4	16.52	(±110.54)	5.35	(±2.21)	0	(±0)	0.00	(±0.00)
	DWE		1.87	(±1.07)	53.2	4	23.17	(±207.62)	5.44	(±2.27)	0	(±0)	0.00	(±0.00)
	MARA	SPE		(±)			48.04	(±111.18)	6.56	(±2.85)	0	(±0)	0.00	(±0.00)
	MARA	MC		(±)			872.53	(±2.34e3)	10.71	(±4.53)	0	(±0)	0.00	(±0.00)
	ALR	COL	2.14	(±0.92)	81.5	24	422.62	(±9.03e3)	7.14	(±3.24)	282	(±18)	1.73	(±0.11)
	ALR	UNR	2.15	(±0.93)	81.2	24	384.06	(±9.56e3)	7.00	(±3.13)	178	(±3)	1.10	(±0.02)
	ALR-H2	COL	2.12	(±0.90)	81.2	24	386.46	(±7.32e3)	7.20	(±3.28)	665	(±41)	4.07	(±0.25)
	ALR-H2	UNR	2.14	(±0.93)	80.7	24	317.87	(±7.63e3)	6.96	(±3.12)	300	(±5)	1.83	(±0.03)
	NLSR2	COL	1.60	(±0.49)	60.3	4	18.02	(±93.78)	6.77	(±3.40)	∞	(±∞)	∞	(±∞)
	XNLSR2	UNR	1.70	(±0.46)	70.0	4	16.73	(±96.90)	6.51	(±3.14)	1,203	(±19)	8.64	(±0.13)
	NLSR3	COL	1.54	(±0.66)	44.7	4	154.04	(±9.25e3)	9.40	(±5.72)	17,516	(±793)	93.52	(±4.23)
	XNLSR3	UNR	2.06	(±0.77)	73.2	8	84.58	(±3.41e3)	6.93	(±3.45)	3,295	(±33)	17.59	(±0.18)
	NLSR	COL	1.32	(±0.78)	22.1	4	36.64	(±3.55e3)	10.90	(±6.89)	38,063	(±1,731)	143.88	(±6.54)
	XNLSR	UNR	2.33	(±1.21)	72.8	18	1,096.18	(±7.96e4)	6.89	(±3.26)	9,011	(±19)	34.07	(±0.07)
Telstra	ECMP		1.08	(±0.28)	7.8	1	1.37	(±0.66)	4.79	(±1.80)	0	(±0)	0.00	(±0.00)
	DW		1.32	(±0.75)	23.6	2	3.38	(±7.25)	4.97	(±1.85)	0	(±0)	0.00	(±0.00)
	DWE		1.47	(±0.94)	29.1	4	18.36	(±88.84)	5.43	(±2.16)	0	(±0)	0.00	(±0.00)
	MARA	SPE		(±)			16.18	(±22.63)	5.96	(±2.37)	0	(±0)	0.00	(±0.00)
	MARA	MC		(±)			26.27	(±31.47)	7.24	(±2.99)	0	(±0)	0.00	(±0.00)
	ALR	COL	1.59	(±0.85)	45.5	16	64.99	(±299.27)	6.80	(±3.03)	275	(±41)	1.59	(±0.24)
	ALR	UNR	1.61	(±0.89)	46.2	16	64.26	(±282.67)	6.76	(±3.00)	86	(±2)	0.50	(±0.01)
	ALR-H2	COL	1.58	(±0.85)	45.0	16	59.60	(±273.66)	6.73	(±2.97)	432	(±39)	2.49	(±0.23)
	ALR-H2	UNR	1.60	(±0.89)	45.9	16	58.34	(±251.74)	6.69	(±2.96)	159	(±4)	0.92	(±0.02)
	NLSR2	COL	1.36	(±0.48)	35.7	4	5.38	(±7.03)	6.19	(±2.87)	1,802	(±103)	11.40	(±0.65)
	XNLSR2	UNR	1.42	(±0.49)	42.0	4	5.94	(±8.90)	6.20	(±2.87)	598	(±11)	3.78	(±0.07)
	NLSR3	COL	1.28	(±0.53)	24.3	4	18.05	(±102.87)	7.10	(±3.58)	7,457	(±125)	40.60	(±0.68)
	XNLSR3	UNR	1.53	(±0.69)	41.2	4	15.80	(±82.16)	6.28	(±2.88)	1,997	(±17)	10.87	(±0.09)
	NLSR	COL	1.14	(±0.65)	7.9	2	16.86	(±378.02)	8.16	(±4.43)	24,537	(±786)	96.50	(±3.09)
	XNLSR	UNR	1.71	(±1.18)	41.2	8	82.93	(±1.94e3)	6.31	(±2.77)	7,152	(±36)	28.13	(±0.14)
Tiscali	ECMP		1.17	(±0.47)	13.7	1	2.14	(±2.30)	5.80	(±2.80)	0	(±0)	0.00	(±0.00)
	DW		2.00	(±1.81)	44.9	12	663.62	(±5.13e4)	6.11	(±2.81)	0	(±0)	0.00	(±0.00)
	DWE		2.05	(±1.85)	46.8	16	1,153.51	(±4.39e4)	6.26	(±2.90)	0	(±0)	0.00	(±0.00)
	MARA	SPE		(±)			3,742.34	(±1.46e4)	9.36	(±4.26)	0	(±0)	0.00	(±0.00)
	MARA	MC		(±)			1.02e4	(±3.71e4)	11.40	(±4.54)	0	(±0)	0.00	(±0.00)
	ALR	COL	2.26	(±1.72)	70.0	112	6.22e4	(±7.95e6)	7.75	(±3.75)	974	(±66)	1.66	(±0.11)
	ALR	UNR	2.27	(±1.74)	69.2	104	5.55e4	(±7.91e6)	7.65	(±3.73)	509	(±8)	0.86	(±0.01)
	ALR-H2	COL	2.26	(±1.72)	69.8	108	8.18e4	(±4.02e7)	7.75	(±3.75)	1,173	(±78)	1.99	(±0.13)
	ALR-H2	UNR	2.26	(±1.74)	68.9	96	3.72e4	(±4.15e6)	7.61	(±3.69)	585	(±6)	0.99	(±0.01)
	NLSR2	COL	1.59	(±0.50)	58.7	8	26.68	(±155.73)	7.38	(±3.93)	∞	(±∞)	∞	(±∞)
	XNLSR2	UNR	1.63	(±0.48)	62.5	4	20.12	(±115.76)	6.78	(±3.36)	2,243	(±26)	5.08	(±0.06)
	NLSR3	COL	1.62	(±0.73)	46.7	8	338.30	(±1.21e5)	8.48	(±4.87)	33,093	(±1,296)	58.19	(±2.28)
	XNLSR3	UNR	1.98	(±0.83)	64.9	12	273.29	(±1.73e4)	6.92	(±3.39)	5,821	(±45)	10.24	(±0.08)
	NLSR	COL	1.33	(±1.30)	17.1	4	167.63	(±1.73e4)	12.20	(±8.23)	1.3e5	(±1,464)	135.77	(±1.51)
	XNLSR	UNR	2.45	(±2.00)	63.3	52	1.40e5	(±5.62e7)	7.08	(±3.35)	33,802	(±60)	34.85	(±0.06)
Abovenet	ECMP		1.23	(±0.56)	17.8	1	2.53	(±4.91)	4.64	(±2.09)	0	(±0)	0.00	(±0.00)
	DW		2.57	(±1.60)	69.6	16	1,908.94	(±1.34e5)	5.21	(±2.43)	0	(±0)	0.00	(±0.00)
	DWE		2.72	(±1.68)	73.7	30	9,360.51	(±1.18e6)	5.47	(±2.57)	0	(±0)	0.00	(±0.00)
	ALR	COL	2.87	(±1.45)	92.0	108	6.77e4	(±7.39e6)	6.51	(±3.02)	458	(±16)	0.84	(±0.03)
	ALR	UNR	2.88	(±1.46)	91.6	108	8.12e4	(±3.73e7)	6.41	(±2.94)	328	(±7)	0.60	(±0.01)
	ALR-H2	COL	2.87	(±1.44)	91.9	120	1.79e5	(±1.21e8)	6.54	(±3.05)	646	(±16)	1.18	(±0.03)
	ALR-H2	UNR	2.88	(±1.47)	91.2	96	4.90e4	(±4.57e6)	6.32	(±2.91)	413	(±4)	0.75	(±0.01)
	NLSR2	COL	1.81	(±0.39)	81.1	8	28.97	(±144.83)	6.76	(±3.57)	∞	(±∞)	∞	(±∞)
	XNLSR2	UNR	1.86	(±0.34)	86.2	8	21.89	(±90.23)	6.23	(±3.11)	1,359	(±13)	3.72	(±0.03)
	NLSR3	COL	1.81	(±0.83)	54.2	8	644.21	(±6.10e4)	9.00	(±5.37)	36,420	(±781)	70.17	(±1.50)
	XNLSR3	UNR	2.41	(±0.74)	85.0	12	180.13	(±1.15e4)	6.37	(±3.12)	6,364	(±26)	12.26	(±0.05)
	NLSR	COL	1.36	(±1.17)	18.2	6	566.37	(±1.80e5)	12.88	(±8.91)	1.5e5	(±1,397)	151.47	(±1.39)
	XNLSR	UNR	3.18	(±1.76)	85.0	72	4.39e5	(±1.61e8)	6.20	(±2.94)	40,474	(±56)	40.23	(±0.06)
Sprint	ECMP		1.39	(±0.86)	27.3	1	2.10	(±2.88)	4.24	(±1.64)	0	(±0)	0.00	(±0.00)
	DW		2.57	(±2.41)	65.8	12	823.94	(±1.67e5)	4.99	(±2.03)	0	(±0)	0.00	(±0.00)
	DWE		3.10	(±2.90)	75.3	48	3.24e6	(±5.12e9)	5.91	(±2.47)	0	(±0)	0.00	(±0.00)
	ALR	COL	3.19	(±2.37)	89.8	128	6.42e5	(±8.02e8)	6.59	(±2.88)	323	(±28)	0.10	(±0.01)
	ALR	UNR	3.19	(±2.37)	89.8	128	3.38e5	(±2.39e8)	6.54	(±2.85)	132	(±3)	0.04	(±0.00)
	ALR-H2	COL	3.18	(±2.37)	89.5	120	7.01e5	(±1.21e9)	6.56	(±2.90)	1,447	(±51)	0.46	(±0.02)
	ALR-H2	UNR	3.18	(±2.37)	89.3	96	6.65e5	(±7.77e8)	6.37	(±2.82)	687	(±2)	0.22	(±0.00)
	NLSR2	COL	1.81	(±0.39)	81.3	8	96.53	(±5.43e3)	6.89	(±3.66)	∞	(±∞)	∞	(±∞)
	XNLSR2	UNR	1.85	(±0.35)	85.4	8	63.19	(±6.11e3)	6.15	(±3.04)	4,706	(±42)	2.50	(±0.02)
	NLSR3	COL	2.06	(±0.84)	67.4	18	3.08e4	(±1.46e7)	9.03	(±5.31)	1.0e5	(±2,153)	39.92	(±0.83)
	XNLSR3	UNR	2.41	(±0.74)	84.6	12	451.99	(±1.03e5)	6.14	(±2.96)	19,663	(±85)	7.62	(±0.03)
	NLSR	COL	1.32	(±1.87)	10.7	3	1.27e4	(±2.03e7)	18.72	(±14.32)	∞	(±∞)	∞	(±∞)
	XNLSR	UNR	3.51	(±2.70)	86.1	96	1.30e7	(±1.18e10)	6.49	(±2.91)	2.6e5	(±636)	42.65	(±0.11)