liteNDN: QoS-Aware Packet Forwarding and Caching for Named Data Networks

Mohamed Abdelaal, Mustafa Karadeniz, Frank Dürr, Kurt Rothermel
Institute of Parallel and Distributed Systems, University of Stuttgart
Email: first.last@ipvs.uni-stuttgart.de

***Abstract*—Recently, named data networking (NDN) has been introduced to connect the world of computing devices via naming data instead of their containers. Through this strategic change, NDN brings several new features to network communication, including in-network caching, multipath forwarding, built-in multicast, and data security. Despite these unique features of NDN networking, there exist plenty of opportunities for continuing developments, especially with packet forwarding and caching. In this context, we introduce liteNDN, a novel forwarding and caching strategy for NDN networks. liteNDN comprises a cooperative forwarding strategy through which NDN routers share their knowledge, i.e. data names and interfaces, to optimize their packet forwarding decisions. Subsequently, liteNDN leverages that knowledge to estimate the probability of each downstream path to swiftly retrieve the requested data. Additionally, liteNDN exploits heuristics, such as routing costs and data significance, to make proper decisions about caching normal as well as segmented packets. The proposed approach has been extensively evaluated in terms of the data retrieval latency, network utilization, and the cache hit rate. The results showed that liteNDN, compared to conventional NDN forwarding and caching strategies, achieves much less latency while reducing the unnecessary traffic and caching activities.**

***Index Terms*—named data networking, forwarding strategy, caching policy, quality of service**

I. Introduction

Thanks to the current technological advancements in sensing and communication, more machines, controllers, consumer devices, and wearables are to be connected to the Internet. However, the current IP-based architecture has several limitations when dealing with large-scale data distribution, including high multicasting overhead especially for "resources-constrained" devices, high handshaking overhead, poor mobility support, limited routing scalability, and complex security models [1]. These limitations have confirmed the belief that the era of IoT and Industry 4.0 inevitably demands the design of a *second generation* of the Internet architecture. Hence, several novel architectures have recently been introduced to meet the accelerating expansion in Internet traffic.

In this context, *named data networking* is a data-centric Internet architecture solution. It has recently received much attention as an efficient alternative Internet architecture to the IP-based Internet architecture. Generally, NDN represents a clean-slate receiver-oriented design which utilizes the request/response model to retrieve data from possible source nodes [2]. NDN architecture is based on four main concepts, namely content forwarding, in-network caching, multicasting, and built-in security at the data level. Content forwarding leverages data names—that carry application semantics—to exchange two types of packets, i.e. Interest and Data. Such NDN packets carry only unique names to identify the requested data without embedding the source or destination addresses. Specifically, users specify what data they are interested in while NDN infrastructure takes care of searching and delivering the data back to the users.

In fact, we observed that the conventional NDN forwarding strategy, referred to as the best route strategy (BRS) [3], entirely overlooks the unique NDN primitives, such as in-network caching and multipath forwarding. Specifically, the BRS strategy estimates the best route for each name prefix towards its producer. Nevertheless, there may exist routers in the vicinity of data consumers which have cached versions of the requested data. Accordingly, the routes estimated by the BRS strategy can further be optimized to proactively forward the Interest packets to the closest data source. Aside from packet forwarding, the default caching strategy of NDN architecture relies on storing each received packet regardless of the network size and the routing costs. For medium- and large-sized networks, the NDN routers—in the vicinity of data producers—typically incur excessive overhead as a result of caching data requested by remote consumers. Accordingly, the cache memories can be exhausted in vain. Moreover, the current NDN architecture lacks an efficient mechanism for dealing with segmented data, where each segment requires an individual Interest packet, thus flooding the network with unnecessary traffic. Consequently, a challenge of improving the forwarding strategy together with increasing the caching efficiency emerges.

To tackle this challenge, we introduce liteNDN, a QoS-aware forwarding and caching strategy for NDN networks. liteNDN exploits the unique NDN primitives to broadly support the fast and reliable retrieval of data from cached versions at intermediate routers, thus reducing the packet delivery time. Moreover, liteNDN implement a decision making mechanism to proactively decide upon caching the received packets in light of the routing costs. Consequently, routers located close to a certain data producer can completely avoid caching Data packets from this producer. Additionally, liteNDN leverages the strategy of sending a portion of the segmented data without receiving corresponding Interest packets, before caching them close to the data consumers. Therefore, the time required to retrieve these segments can be drastically reduced together

This work is supported by the German Federal Ministry of Education and Research (BMBF) grant 01DH17059.

Published in the IEEE Consumer Communications and Networking Conference (CCNC), 10-13 January 2020 Las Vegas, USA

with reducing the network traffic. To the best of our knowledge, liteNDN is the first NDN forwarding strategy which harnesses the cached versions of the exchanged packets to optimize the forwarding decisions. Additionally, the proposed caching strategy represents the first work which simply exploits heuristics to efficiently deal with segmented data and to drastically reduce the caching activities.

In detail, liteNDN provides the following contributions: (1) We introduce a *cooperative* forwarding strategy which relies on sharing data names for identifying possible data sources close from consumers. In our proposed forwarding strategy, each router collects information about the names of data being exchanged by neighboring routers. Together with the routing costs, these shared names are locally exploited to estimate the probability of each path to swiftly retrieve the requested data (cf. Section III). (2) We devise a heuristic-based caching strategy which leverages the routing cost as a decision metric to sidestep the caching of data received from nearby producers. (3) We introduce an adaptive mechanism for dealing with segmented data. In this context, a weighted sum model (WSM) is used to determine the portion of segmented data to send back to a producer—according to the data type, size, and its significance—once the first segment has been requested. (4) We provide an extensive evaluation of liteNDN to assess its effectiveness compared to the default NDN strategies. The results show that using liteNDN, we obtain a significant improvement of the round trip time (at least by 43%) relative to the BRS strategy. Moreover, liteNDN requires much less time (at least by 91%) to retrieve segmented data compared to the baseline method.

II. System Overview

In this section, we provide an overview of named data networking, before introducing our proposed solution.

A. Preliminaries

In general, NDN networks rely on the request/response model to pull data, i.e. video, sensory data, etc. An Interest packet s_i typically comprises a hierarchical name of a desired piece of data, e.g. `ndn:/temp/01`, where NDN routers utilize such a name to forward the Interest packet towards all possible data sources (cf. Figure 1). In this context, *name-based* routing algorithms are de facto employed to distribute information about the contents' location based on the embedded names using the longest prefix matching. It is worth noting that hierarchical namespace is utilized for naming the exchanged data to achieve better routing scalability. Once an Interest is received by a node that possesses the required data, a Data packet d_i—comprising the data name, a cryptographically-signed version of the required data, and additional authentication and data-integrity information—is forwarded along the Interest's reverse route.

To exchange Interest and Data packets, NDN routers maintain three data structures, namely *content store* (CS), *pending interest table* (PIT), and *forwarding information base* (FIB) (cf. Figure 1). The former component is a temporary cache

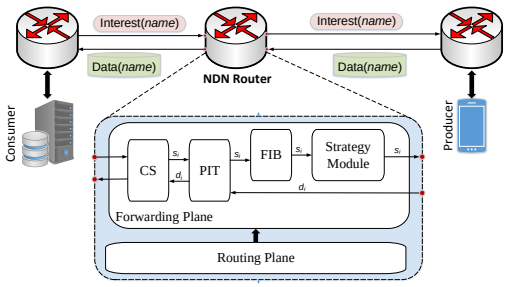


Fig. 1: Architecture of an NDN router

of the received Data packets. In fact, the in-network cashing mechanism enables the NDN routers from responding to similar data requests, thus reducing the network congestion together with improving the delivery speed. The PIT table stores all the incoming Interests that the NDN router has forwarded but not satisfied yet. If an incoming Interest s_i(`name`, `face ID`) has a matching entry in the PIT table, the NDN router records only the face ID without forwarding such an Interest. For the data requests, which do not have matching entries in the PIT table, the FIB component employs an adaptive forwarding strategy to forward them through multiple interfaces toward the data source(s). Such a per hop, per packet state enables routers to identify and discard looping packets, thus allowing them to freely use multiple paths toward the same data producer.

For the Data packets, they are typically not retrieved according to a routing table. Instead, they follow the reverse path toward consumers thanks to the interfaces recorded at the PIT table of each NDN router. In this context, the NDN routers perform component-wise longest prefix match of the *content name* from a Data packet against the FIB table. Once a Data packet is received by a router, the PIT table is initially checked for requests from consumers interested in this data packet. If there exist valid entries in the PIT table, the received data is stored in the CS cache, before forwarding the Data packet to the downstream neighbors, corresponding to all the matched PIT entries. When a Data packet is found to be unsolicited, the packet will be deliberately dropped as it may pose security risks to the forwarder. Nevertheless, there are cases in which unsolicited Data packets are cached if they arrive from a local face. Accordingly, the default caching policy stores all incoming data regardless of the network size.

B. liteNDN Architecture

Despite providing solutions to the predominant problems in IP-based networks, NDN networking is still in its infancy and there are plenty of opportunities for continuing research and developments. In this context, we introduce liteNDN, a novel forwarding and caching strategy for NDN networks which exploits the unique NDN features to broadly improve the relevant QoS metrics, i.e. data retrieval latency and throughput. As depicted in Figure 2, liteNDN comprises two main contributions, namely the cooperative forwarding strategy and the heuristic-based caching policy. The former contribution revolves around avoiding exceedingly long routes via deliberately retrieving

cached versions of the requested data from closer routers in lieu of obtaining them from the original producers. To this end, our proposed forwarding strategy, referred to as NDN-MPP, enables the various NDN routers to frequently share their knowledge, i.e. data names, to build a knowledge base of the data names exchanged among the neighboring routers.

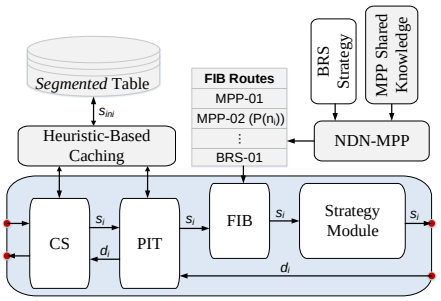


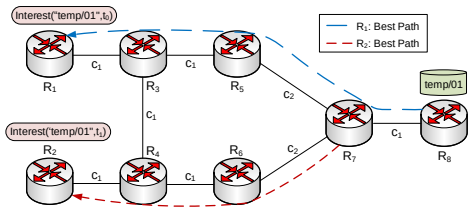
Fig. 2: liteNDN forwarding plane

In addition to the MPP shared knowledge, the NDN-MPP strategy takes as an input the ranked list of best paths estimated by the best route strategy, i.e. the default forwarding in current NDN architecture. In this manner, NDN-MPP generates, as an output, a new ranked list of paths where closer routers which *most probably* have the requested data are placed at the top of the list (cf. Figure 2). Aside from the NDN-MPP forwarding strategy, we also propose a heuristic-based caching policy which makes decisions of whether to store the received data. The core idea is to avoid exhausting the cache memories which mostly occurs due to storing Data packets from nearby data sources. To this end, we adopt the *routing distance*—defined in terms of the routing cost and the hop count between a certain router and the data producer—as a metric to properly make the caching decision. As a result, the received data by a router is cached only if the producer is exceedingly distant from that router. Moreover, we devise a mechanism for retrieving the segmented data without receiving corresponding Interests. Below, we explain our proposed methods in more detail, before experimentally assessing their impact in terms of the relevant QoS metrics.

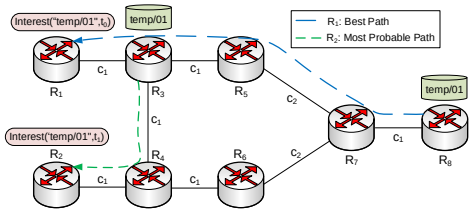
III. Cooperative Packet Forwarding

In this section, we introduce a novel forwarding strategy implemented as a component of the NDN Forwarding Daemon (NFD). Before delving into our proposed forwarding strategy, we first explain the conventional NDN forwarding strategy together with highlighting its drawbacks. In general, NDN employs the named-data link state routing (NLSR) protocol [4] as the default routing protocol. NLSR computes the best path—using the Dijkstra algorithm—for each data prefix to its destination. To implement multipath routing, NFD implements the *best route strategy* (BRS) which provides a ranked list of all possible paths towards each producer according to their routing cost, i.e. the *best path* is the one which has the lowest routing cost [3]. If the best path failed to retrieve the requested data, the BRS strategy selects the second best path to forward the Interest. This process continues till receiving a Data packet for such a request. In this manner, the BRS strategy tends to minimize the routing costs.

Such a forwarding strategy completely ignores the in-network caching implemented in NDN routers. According to this policy, every NDN router—along the reverse path of a request packet—has to temporarily store the incoming data. Thus, the NDN routers employing the BRS forwarding strategy can readily overlook data sources in their vicinity which most probably have a copy of the requested data. As a result, the data retrieval latency can be broadly increased due to lacking flexibility while determining the most suitable path towards a data source. As an example, consider the routing scenario shown in Figure 3 to explain our observation. As depicted in Figure 3a, the network consists of eight NDN routers with two consumers connected to the routers R_1 and R_2 together with a data producer connected to the router R_8. The terms c_1 and c_2 denote the routing costs for each link where $c1 \ll c2$.



(a) Using NDN (BRS forwarding strategy)



(b) Using NDN-MPP

Fig. 3: An example of the proposed forwarding strategy

In this scenario, the NDN router R_1 sends an Interest at time t_0 to retrieve a data whose name is `temp/01`, i.e. the temperature value at room 01 of a certain building. Figure 3a demonstrates that the best path for the Interest triggered by R_1 goes through the routers R_3, R_5, R_7 and R_8 where the total cost in this path equals to $3c_1 + c_2$. Once the data is found in R_8, a Data packet is sent through the reverse path (cf. blue curve in Figure 3a) while caching the requested data in each router on that path. Subsequently, the router R_2 sends an Interest requesting the same data at time $t_1 = t_0 + \delta$, where $T_{rtt}^{R_1} < \delta < T_c^{R_1}$ given that $T_c^{R_1}$ is the caching time of data received by the router R_1 and $T_{rtt}^{R_1}$ is the round trip time of the Interest sent by R_1. For the Interest triggered by R_2, the BRS forwarding strategy chooses the lowest-cost route for this data. Accordingly, the Interest goes through routers R_4, R_6, and R_7 where the requested data is already cached in R_7. In this case, the total cost in this path equals to $2c_1 + c_2$, thus achieving a cost reduction of only one link.

To highly reduce the data retrieval latency and improve

the network throughput, we propose a *cooperative* forwarding strategy, referred to as NDN-MPP, which mainly relies on sharing knowledge among neighboring to properly exploit the NDN in-network caching, thus selecting shorter routes. As depicted in Figure 3b, the requested data already exists in R_3. If the request is forwarded to R_3, the routing cost of the Interest triggered by R_2 will be reduced to only $2c_1$. In NDN-MPP, the routers typically forward their requests to the *most probable path* (MPP), which *most probably* will retrieve the data from closer neighbors rather than employing the best path determined by the BRS strategy. Figure 4 depicts the NDN-MPP algorithm, where it comprises two main phases, namely *knowledge sharing* and *path estimation*.

In the knowledge-sharing phase, each NDN router shares with its direct neighbors information about the received data packets. Such Data packets $\mathcal{D}$ can either be (1) originated from data producers, e.g. R_8 in Figure 3b, or (2) they represent *virtual* Data packets generated by neighboring routers to share their MPP information, i.e. the name n_i, the face f_i, the routing cost c_i, and the hop count f_i from the data provider. For instance, when the router R_3, in Figure 3b, receives a Data packet from the router R_6. After parsing the packet, the data name together with the incoming face are stored in a special data structure, called the MPP table. Afterward, a virtual Data packet d_i is generated with a special fixed name, i.e. $n =$ "ndn:/mpp/Share", to share the MPP information with direct neighbors, i.e. R_1 and R_4. In this virtual packet, the routing cost and hop count are deliberately set to zero.

Once the virtual Data packet d_i is received by the router R_4, a new entry is created in the MPP table only if the name n_i does not already exist in the table (cf. line 4). If this condition is satisfied, the cost and hop count are updated by considering the link between R_3 and R_4 together with its cost c_{link} (cf. lines 5 and 6). It is worth mentioning that an NDN router may receive a certain data name several times from different neighbors, i.e. distinct faces. In this case, NDN-MPP utilizes the routing cost c_i and the hop count h_i to accurately select the optimal face which yields the lowest overhead. If the overhead $c_i \times h_i$ of the newly-arrived face f_i is greater than an existent one f_j in the MPP table, then the face f_i is ignored. Otherwise, the newly-arrived face f_i replaces the old face to further reduce the overhead.

The second phase of NDN-MPP is forwarding the received Interests based on the entries in the constructed MPP table. At the outset, the data name s_i of an incoming Interest is compared to the names $\mathcal{N}_{mpp}$ stored in the MPP table. If an exact match is found, the face f_j corresponding to that matched name is used to forward the Interest packet (cf. line 15). If the MPP table does not have enough knowledge to forward such an Interest packet, NDN-MPP estimates the probability P_{n_i} of each stored face f_j to be a probable path for satisfying this Interest. The intuition is to find a partial match between an incoming name prefix and the set of stored names $\mathcal{N}_{mpp}$. Equation 1 expresses the path probability P_{n_i} in terms of the word matching function $\phi(n_i, n_j)$ and the number of data names k_x for each neighbor $x \in X_{dir}$, where

Require: path threshold α_p, link cost c_{link}, face f_{BRS}

1: **for all** Data packet $d_i \in \mathcal{D}$ **do** ▷ MPP Table construction
2: $n_i, f_i, c_i, h_i \leftarrow$ **ParsePacket**(d_i)
3: **for all** stored name $n_j \in \mathcal{N}_{mpp}$ **do**
4: **if** $n_i \neq n_j$ **or** $n_i ==$ "ndn:/mpp/Share" **then**
5: $c_i \leftarrow c_i + c_{link}$ ▷ Cost update
6: $h_i \leftarrow h_i + 1$ ▷ Hop count update
7: **Insert** d_i in the MPP table
8: **ForwardMPP**$(n =$ "ndn:/mpp/Share")
9: **else if** $c_i \times h_i \geq c_j \times h_j$ **then** ▷ Identical names
10: **Discard** the incoming face f_i
11: **else** **Replace** the face f_j with f_i
12: **for all** Data packet $s_i \in \int$ **do** ▷ Forwarding Interests
13: $n_i, f_i, c_i, h_i \leftarrow$ **ParsePacket**(s_i)
14: **for all** stored name $n_j \in \mathcal{N}_{mpp}$ **do**
15: **if** name $n_i == n_j$ **then**
16: **ForwardInterest**$(n = n_i, f = f_j)$
17: **else if** $P(n_j) \geq \alpha_p$ **then** ▷ Equation 1
18: **ForwardInterest**$(n = n_i, f = f_j)$
19: **else** **ForwardInterest**$(n = n_i, f = f_{BRS})$

Fig. 4: Cooperative forwarding algorithm for NDN networks

$|X_{dir}|$ denotes the number of direct neighbors. The function $\phi(n_i, n_j)$ estimates the level of similarity between two name prefixes n_i and n_j. For instance, assume an incoming Interest whose name $n_i =$ `ndn:/uni/temp/floor/06` is to be matched to a stored entry in the MPP table whose name is `ndn:/uni/lec/hum/03`. In this example, the output of the matching function $\phi(n_i, n_j)$ is equal to two since both prefixes share exactly two fields.

$$P(n_i) = \max_{x \in X_{dir}} \left(\frac{\sum_{j=1}^{k_x} \frac{\phi(n_i, n_j)}{k_x}}{\sum_{x=1}^{|X_{dir}|} \sum_{j=1}^{k_x} \frac{\phi(n_i, n_j)}{k_x}} \right) \quad (1)$$

In the MPP table, some data names $n_j, j = 1, \cdots, k_x$ are typically received from the same neighbor $x \in X_{dir}$. In this case, NDN-MPP computes the word matching function for each name, before averaging the results obtained for such a neighbor x, as expressed in Equation 1. After estimating the probability of each neighbor, we compare the maximum probability to a predefined threshold α_p (cf. line 17). If the probability P_{n_i} exceeds the threshold α_p, we utilize the face, corresponding to this neighbor, to forward the Interest packet. Otherwise, NDN-MPP selects the face f_{BRS}—determined by the BRS strategy—to forward the packet (cf. line 19). In case of receiving a negative acknowledgment (NACK) message for an Interest packet—sent over an MPP path, the NDN routers proactively switch back to the default best path strategy.

IV. Heuristic-based Caching

Before delving into our proposed caching policy, we first explain the default NDN caching strategy together with explaining its shortcomings. Indeed, the primarily reason for keeping a copy of the requested data in every node is to increase the chance of data hit in these nodes for incoming

similar Interests. Such a design takes into consideration an optimistic assumption that the stored data will be requested before being removed from the cache memory. However, caching the NDN packets in every router, regardless of the network size and the routing costs, certainly adds huge burden to the network resources. For intermediate and large networks, the NDN routers—located close from the data sources—will be extremely exhausted owing to the unnecessary caching activities, thus consuming the memory budget together with increasing the energy consumption. For instance,

In addition to the unnecessary caching activities, the request/response model of NDN networks may cause additional overhead when dealing with *segmented* data. In particular, the request/response model of NDN implies that each Data packet can be retrieved through sending an Interest packet. However, the size of the requested data, e.g. video and audio streams, is mostly larger than the size of a single packet [5]. In this case, the requested data has to be divided into a number of segments. Unfortunately, NDN lacks a suitable mechanism for dealing with the segmented data where each segment requires a single Interest packet to be retrieved. Accordingly, the bandwidth efficiency can be negatively harmed due to sending a myriad of Interest packets to request a sequence of segments constituting the required data. In addition to the bandwidth efficiency, the data retrieval latency is drastically increased owing to the time taken by each Interest to propagate throughout the network.

To overcome these challenges, we introduce our heuristic-based NDN caching policy (HBC) for the sake of improving the resources utilization together with reducing the delivery latency. Figure 5 depicts our proposed heuristic-based caching algorithm for making two primarily decisions: (1) whether to cache the received data in every router and (2) whether to send a series of segmented data once an Interest is received requesting the first segment. To properly make these decisions, it is necessary to embed *metadata* in the header field of each NDN packet about the downstream path and the significance of the requested data. In general, each NDN packet is encoded in a Type-Length-Value (TLV) format where a message body is mainly a collection of TLVs. Thanks to adopting the TLV format, NDN packets have the flexibility of adding new fields and types as the networking protocols evolve. Such flexibility enabled us from readily extending the header of the Data packets to keep track of the reverse path's cost, referred to as the travel cost c_{tvl}. Initially, the cost c_{tvl} is set to zero at the data source where its value is updated—at each router along the downstream path—through considering the cost of each link c_{link} (cf. line 3). After parsing a received Data packet, a router compares the updated travel cost c_{tvl} against a predefined caching threshold η_c. If the threshold is not exceeded, the router forwards the Data packet without caching the embedded content (cf. line 4). Otherwise, the default caching mechanism is adopted to store the content, before resetting the travel cost c_{tvl} to zero.

Regarding the segmented data, the first segment d_1—for which an initial Interest packet s_{ini} has already been received by the data source—is to be sent along the downstream

Require: caching threshold η_c, link cost c_{link}

1: **for all** Data packet $d_i \in \mathcal{D}$ **do**
2: $n_i, c_{tvl} \leftarrow$ **ParsePacket**(d_i)
3: $c_{tvl} \leftarrow c_{tvl} + c_{link}$ ▷ Cost update
4: **if** $c_{tvl} \geq \eta_c$ **then**
5: `cachingData` $\leftarrow$ `True` ▷ Caching the incoming Data d_i
6: $c_{tvl} \leftarrow 0$ ▷ Resetting the travel cost
7: **else** `cachingData` $\leftarrow$ `False`
8: **if isUnsolicited**(d_i) == `True` & **isSegmented**(d_i) == `True` **then**
9: **if** `seqNumber`(d_i) == 1 **then**
10: **Insert** s_{ini} in *Segmented* table
11: **else if** s_{ini}(name $= n_i$) $\in$ *Segmented* table **then**
12: **ForwardData** d_i to next router
13: **else** **Discard** the packet d_i

Fig. 5: Heuristic-based caching algorithm

path. In this context, we assume that consumers are mostly interested in retrieving *all* pieces of the segmented data $d_1, \cdots, d_m \in \mathcal{D}$. Based on this assumption, liteNDN implements a simple weighted sum model (WSM) [6] at the data source to determine whether to proactively send a portion of the remaining segments without receiving their corresponding Interest packets $s_2, \cdots, s_m$. To make such a decision, three distinct metrics are considered, including the data type, the data size, and the data significance (i.e. low-, medium-, or high-priority). If the decision is made to send the remaining segments $d_1, \cdots, d_m$, the data source forwards the segments along the downstream path till reaching the *closest* router to the data consumer.

After sending the first segmented Data packet d_1 to the consumer node, the NFD module at each router automatically removes the corresponding entry from its PIT table. Accordingly, the reverse path going back to the data consumer(s) is completely lost. To overcome this problem, liteNDN implements in the NFD module of each NDN router a PIT-like data structure, referred to as the *Segmented* table (cf. Listing 1). Such a table is utilized to temporarily cache the initial Interests—which request segmented data—and their interfaces, thus guiding the rest of the segmented data $d_2, \cdots, d_m$ while traveling along the downstream path. Specifically, the table records the name prefix of the first requested segment d_1 and the list of interfaces from which the corresponding Interest(s) s_{ini} has been received.

Listing 1: Segmented PIT-like table

```
/*Structure of the segmented Data packets*/
struct Segmented_Table{
Name;                  //Name of the inital Interest
FaceId;                //Face identification
Original_Segment_No;   //Requested segment number
Nonce;                 //Nonce of the inital Interest
SegmentList;           //Segment list for downstream path
Entry_Creation_Time;   //Used for deleting the Timeout};
```

Once the rest of the segmented data $d_2, \cdots, d_m$ are received by an intermediate router, the packets are checked for being

unsolicited messages through searching for corresponding Interests in the PIT table. If no Interests exist, liteNDN checks if the received packets are pieces of a data block. According to the NDN naming convention, each segment is identified by a unique sequence number appended to the data name, e.g. `ndn:/uni/stgt/video/sequence_number` [7]. Thus, the word matching function $\phi(n_i, n_j)$ can be utilized to implement the boolean function *isSegmented*(d_i) which determines whether the packets $d_2, \cdots, d_m$ are pieces of a data block. If these two conditions, i.e. *isSegmented()* and *isUnsolicited()*, are satisfied, liteNDN search the *Segmented* table for a corresponding initial Interest s_{ini} (cf. line 8). If that Interest is found, liteNDN utilizes the interface of such an Interest to forward the packet $d_2, \cdots, d_m$ to the next router (cf. line 11). This forwarding operation continues till reaching the closest router to the data consumer. In this case, the packets are cached waiting for requests from the data consumer. In this manner, such a router can swiftly respond to future Interests requesting these segmented packets.

V. Performance Evaluation

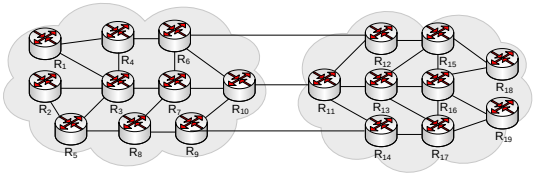
In this section, we assess the effectiveness of liteNDN in improving the performance of NDN networks while delivering Interest and Data packets. We first describe the setup of our evaluations. Subsequently, we discuss the obtained results.

A. Experimental Setup

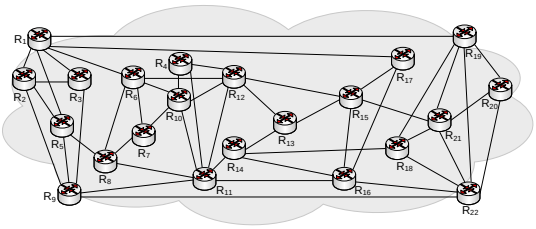
To examine the performance of liteNDN, we utilized a Mininet-based NDN emulator, referred to as the Mini-NDN network emulator [8]. The links between various NDN routers are presented as virtual Ethernet pairs. The size of each Interest packet, Data packet, and data object is set to 56 KB, 388 KB, and 380 KB, respectively. To run our experiments, we utilized a machine equipped with Core i5 CPU of 2.50 GHz and 6 GB of DRAM. Beside the Mini-NDN emulator, we employed several NDN helper tools, such as `ndn-traffic-generator`, `ndndump`, `ndnping`, `ndnputchunks`, and `ndncat- chunks`, to generate the network traffic and to examine the system performance. To test our proposed approach in different environments, we utilized two network topologies in each experiment. Figure 6a shows the first topology which consists of 19 NDN routers divided between two clusters. Such a topology serves as a realistic implementation of the NDN architecture as local area networks. Through such a topology, we can differentiate between local and global data names, thus overcoming the scalability problem of NDN naming schemes [9]. Figure 6b demonstrates the second topology which represents the current NDN testbed consisting of 22 NDN routers distributed over three continents [10]. In both topologies, the link capacity in is set to 2.37 Gbits/sec.

B. NDN-MPP Forwarding

In this section, we evaluate the performance of our proposed MPP-based forwarding strategy in terms of the average round trip time (RTT) for Interest/Data exchange, utilization of the routers and the network links, and the overall bandwidth usage.



(a) Toplogy 1: NDN custom toplogoy



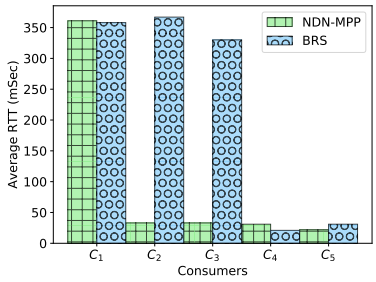
(b) Topology 2: NDN testbed topology

Fig. 6: Topologies adopted in the evaluations

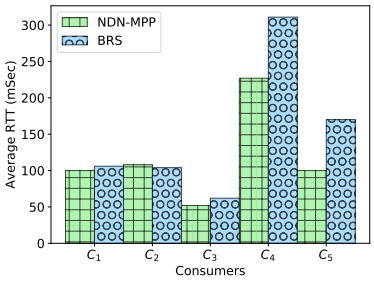
Based on our experiments, we found that the value of $\alpha_p = 0.6$ is convenient for making proper forwarding decisions (cf. Figure 4). In this set of experiments, five randomly-selected consumers successively send Interest packets to retrieve the same data from its provider. In this scenario, we compare the performance of NDN-MPP against the BRS forwarding strategy. Figure 7a shows the average RTT time of the five requests in the first topology. Once the consumer C_1 retrieved its requested data, the various routers—along the downstream path—share the data name to update their MPP tables. As a result, NDN-MPP requires for the consumers C_2 and C_3 much less RTT time (91% for C_2 and 90% for C_3) than the BRS strategy. Thanks to these exchanged packets, many nodes in the network have already cached the requested data. Therefore, both NDN-MPP and BRS approximately require for the consumers C_4 and C_5 circa the same RTT time.

In the second topology, similar results were obtained (cf. Figure 7b). After populating the MPP tables, we found that NDN-MPP requires 27% and 41% less RTT time than the BRS strategy for the consumers C_4 and C_5, respectively. Figure 7c compares the RTT distributions of NDN-MPP and the BRS strategy obtained through repeating such a scenario five times in both topologies. As the figure shows, NDN-MPP requires much less RTT time than the baseline method (on average less RTT time by 43%). Clearly, the RTT distribution of BRS has high variability, depending on the distance between consumers and producers. In case of NDN-MPP, the distribution is highly tight (the RTT density is broadly concentrated below 150 ms) thanks to sharing the routing knowledge among the various routers in the network. It is worth mentioning that NDN-MPP behaves exactly similar to the BRS strategy in two cases: (1) if the MPP Paths failed to retrieve the requested data or (2) if the MPP table is not yet populated. Therefore, the RTT distribution of NDN-MPP extends up to 358 ms. However, the density of such extreme values of RTT in case of NDN-MPP is highly constrained, as it can be seen in Figure 7c.

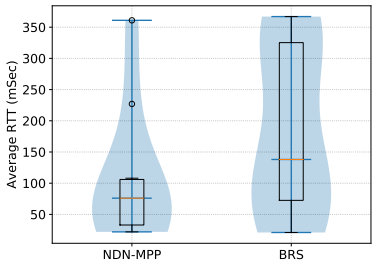
Figure 7d demonstrates the utilization of the various links and routers in the first topology. In this context, the link utiliza-



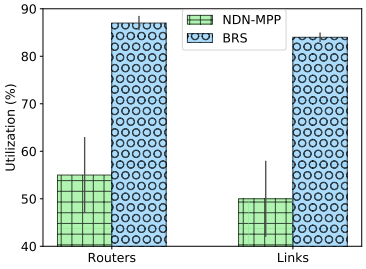
(a) RTT in topology 1



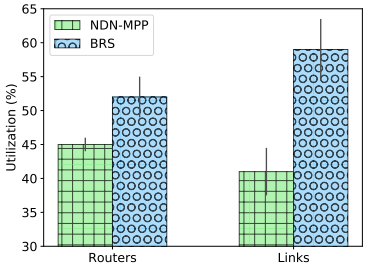
(b) RTT in topology 2



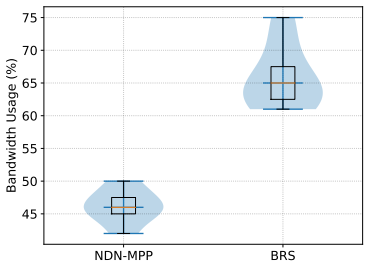
(c) Latency distribution in both topologies



(d) Utilization in topology 1



(e) Utilization in topology 2



(f) Network utilization in both topologies

Fig. 7: Performance of the proposed cooperative forwarding strategy

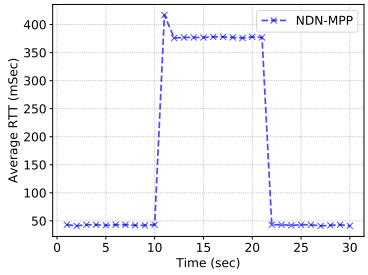


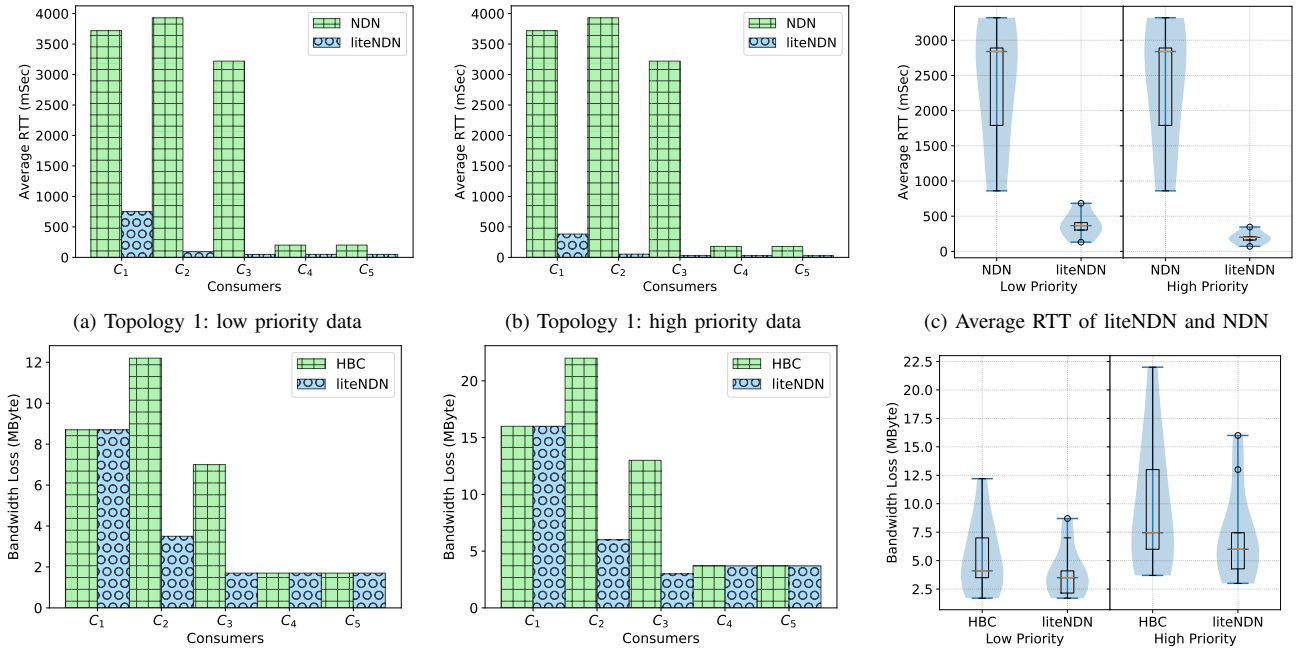
Fig. 8: Behavior of NDN-MPP in case of nodes failure

tion is defined as the percentage of a network's bandwidth that is currently being consumed by the network traffic. Likewise, the router utilization is the fraction of NDN packets handled by that router relative to the network traffic. As expected, NDN-MPP achieves on average 37% and 40% less routers and links utilization than the baseline method. For the second topology, NDN-MPP again reduces the routers and links utilization by 13% and 30%, respectively (cf. Figure 7e). Figure 7f compares the performance of NDN-MPP and the baseline method in terms of the overall network bandwidth usage, i.e. the fraction of traffic on the network relative to the peak amount that the network can support. It is worth noting that high bandwidth usage may drastically increase the data delivery latency due to networks congestions. As depicted in Figure 7f, NDN-MPP achieves a significant reduction in the bandwidth utilization, on average by 30%, compared to the baseline method.

All these advantages of adopting NDN-MPP come at the expense of sharing routing information among the various routers to populate the MPP table. Such an overhead mainly depends on the network size and the size of the shared name prefixes. Nevertheless, we found that the overhead of adopting NDN-MPP does not exceed 20 KByte in both topologies. As Figure 7f demonstrates, NDN-MPP still outperforms the BRS strategy even with considering the overhead of sharing the name prefixes and other metadata. Finally, Figure 8 demonstrates the behavior of our proposed forwarding strategy in the presence of network failures or when receiving NACK or timeout messages. To examine such a behavior, we carried out the following scenario: The router R_1—in the first topology—requests multiple distinct data from the router $R18$, before R_2 requests the same data. According to NDN-MPP, R_2 can be served from R_4. During this experiment, we deliberately deactivated the router R_4 and measured the RTT time. As it can be seen in Figure 8, the RTT time—in case of NDN-MPP—is highly increased starting from time $t = 10$ sec from 50 ms to 417 ms when the router R_4 has been deactivated. This behavior occurs due to switching from NDN-MPP to the BRS strategy to serve these data requests. After reactivating the router R_4, the RTT time is reduced again to below 50 ms.

C. HBC Caching

In this section, we examine the performance of our proposed caching strategy with/without the NDN-MPP forwarding strategy. In these experiments, we adopted two scenarios in both topologies, including (1) the retrieval of *low-priority* data and (2) the retrieval of *high-priority* data. In the first scenario, five consumers successively send requests to retrieve ten identical equally-sized data segments whose priority is set to `low`. Such a priority level is utilized by the WSM model to decide upon the transmission of segmented data without receiving their corresponding Interests. Since the data priority is low, the WSM model decides to send only 50% of the segmented



(a) Topology 1: low priority data (b) Topology 1: high priority data (c) Average RTT of liteNDN and NDN

(d) Topology 1: low priority data (e) Topology 1: high priority data (f) Bandwidth loss of NDN-DAC and liteNDN

Fig. 9: Comparing the proposed liteNDN system to the current implementation of NDN

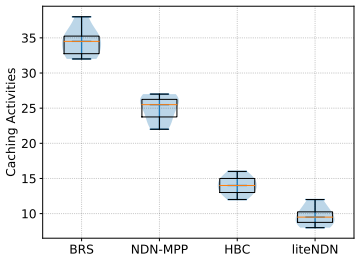


Fig. 10: Impact of liteNDN on the caching activities

data. Figure 9a depicts the average RTT time measured for retrieving the ten segments. Thanks to caching 50% of the remaining segments close to the data consumer, liteNDN requires much less time, on average by 91%, for retrieving the segments relative to the conventional NDN caching and forwarding strategies. Such a result highly depends on the number of segments within a data block, i.e. the higher the number of segments, the lower latency savings are obtained in the first scenario.

Figure 9b depicts the average RTT time in case of retrieving high-priority segmented data. In this case, the WSM model decides to send all the remaining segments. Consequently, liteNDN reduces the RTT time by circa 95% relative to the conventional NDN caching and forwarding strategies. Figure 9c summarizes the comparison between liteNDN and the default NDN strategies for dealing with low- and high-priority segmented data. Clearly, the distribution of NDN has a high variability, depending on the distance between the consumers and the data sources. Conversely, liteNDN exhibit a tight distribution thanks to the prior retrieval of segments. Additionally, the figure shows that high-priority data is served relatively faster, on average by 48%, than the low-priority data.

As aforementioned, our strategy for dealing with segmented data is mainly based on the assumption that consumers are interested in obtaining all the segmented packets of a data block whenever they request the first segment. Despite being a valid assumption, it is worth investigating the overhead incurred by our caching strategy if this assumption has been violated. In the next experiment, we answer the question of what is the penalty—in terms of the bandwidth usage—for violating this assumption. Again, five consumers successively send Interests to retrieve the first segment of a data block consisting of ten segments. However, they do not send further Interests for the remaining segments. In this case, the retrieved data is treated as unsolicited packets causing unnecessary bandwidth usage. Figure 9d compares the performance of liteNDN with and without the cooperative forwarding strategy while retrieving low-priority data. Obviously, integrating the cooperative forwarding strategy and the HBC caching policy reduces the bandwidth losses, on average, by 44%. For the high-priority data, such an integration achieves a comparable result where it reduces the losses, on average, by 45% (cf. Figure 9e).

Figure 9f demonstrates the advantage of employing the WSM model for deciding upon the portion of segmented to be proactively sent according to the data type, size, and significance. The figure demonstrates that the WSM model reduces the bandwidth losses, on average, by 45% through sending only a fraction of the segmented data in case of dealing with low-priority data. Finally, Figure 10 evaluates the performance of liteNDN in terms of the caching activities.

According to our HBC caching policy, routers in the vicinity of data producers can skip caching the exchanged data. In this experiment, the scenario revolves around sending three different Interest packets from three consumers requesting data from the same producer. During our experiment, we set the value of the caching threshold η_c to 100 ms. As it can be seen in the Figure, our caching strategy, compared to the default NDN strategies, highly reduces the overall caching activities (at least by 59%). To examine the integration of NDN-MPP and HBC strategies, the scenario has been repeated with requesting the same data from a certain producer. In this case, the integration of our proposed caching and forwarding strategies further reduces the caching activities (by at least 72%) relative to the baseline method.

VI. Related Work

In this section, we discuss related work in the realm of packet forwarding and caching in NDN networks. In fact, the problem of improving the performance of the NDN forwarding and caching strategies has been tackled in several research works [11]–[15]. For instance, MUCA [11] introduces an NDN forwarding strategy which forwards similar Interests—sent by different consumers—over the same route to use the cache resources more efficiently. To this end, MUCA relies on sharing statistical information to determine the most probable path. As analogous to MUCA, Chen et al. [12] propose a caching and forwarding strategy based on rendezvous points. Through adding a detoured routing capability to NDN, all requests for similar data are forwarded to the same router. Accordingly, each NDN router serves only a subset of the whole name space. Alternatively, liteNDN employ the principle of the most probable path to serve similar as well as different Interests through finding cached versions of the requested data in the vicinity of the corresponding consumers.

Dai et al. [13] propose BFAST, an index data structure for the name-based FIB tables. BFAST employs a counting bloom filter to balance the load among hash table slots. As a result, the searching time in each slot can be reduced, thus improving the performance in lookup, insertion and deletion. Similarly, Ghasemi et al. [14] propose a fast and memory-efficient data structure, referred to as NameTrie, to efficiently index forwarding table entries, thus accelerating the name lookups. The core idea behind NameTrie is to employ Patricia tries to compactly storing data names through removing the redundant information among name prefixes. We believe that NameTrie or BFAST can be integrated with liteNDN to further improve the performance of our cooperative forwarding strategy. Regarding the in-network caching, Sourlas et al. [15] introduce a distributed cache management architecture to reduce the content access latency and to control the network congestion. To this end, a set of distributed cache managers are responsible for dynamically reassigning information items to caches based on the observed item request patterns. Additionally, the authors develop a cooperative algorithm to coordinate the replacement process. However, such an approach suffers from being relatively complex due to requiring additional hardware components. Moreover, it may reduce the bandwidth utilization as a result of exchanging many control messages among the distributed cache managers.

VII. Conclusion & Future Work

In this paper, we presented liteNDN, a QoS-aware forwarding and caching strategy for NDN networks. liteNDN comprises two main components, including cooperative forwarding and heuristic-based caching. The former component leverages shared data names among routers to estimate the most probable paths towards cached versions of the requested data. Moreover, liteNDN avoids caching data at routers located close from the data producers. Besides, liteNDN implements a WSM model and a new *Segmented* PIT-like table for guiding the segmented packets sent without corresponding Interests. The evaluation results showed that liteNDN highly reduces the RTT time and bandwidth usage compared to the conventional NDN forwarding and caching strategies. In the future, we plan to further improve the RTT time of NDN packets through integrating our cooperative forwarding strategy with a fast data structure for the FIB tables, such as BFAST or NAmeTrie.

References

[1] W. Shang and et al., "Challenges in iot networking via tcp/ip architecture," *Technical Report NDN-0038. NDN Project*, 2016.

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

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

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

[5] I. Moiseenko, "Fetching content in named data networking with embedded manifests," *NDN, Tech. Rep. NDN-0025*, 2014.

[6] A. Kolios, V. Mytilinou, and E. Lozano-Minguez, "A comparative study of multiple-criteria decision-making methods under stochastic inputs," *Energies*, vol. 9, no. 7, 2016.

[7] Y. Yu, A. Afanasyev, Z. Zhu, and L. Zhang, "Ndn technical memo: Naming conventions," *NDN, NDN Memo, Technical Report*, 2014.

[8] M.-N. Contributors, "Mini-ndn github source code," https://github.com/named-data/mini-ndn, 2018.

[9] H. Yuan and P. Crowley, "Reliably scalable name prefix lookup," in *Proceedings of the Eleventh ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, 2015, pp. 111–121.

[10] Z. Lailari, H. B. Abraham, B. Aronberg, J. Hudepohl, H. Yuan, J. DeHart, J. Parwatikar, and P. Crowley, "Experiments with the emulated ndn testbed in onl," in *Proceedings of the 2nd ACM Conference on Information-Centric Networking*. ACM, 2015, pp. 219–220.

[11] C. Ghasemi, H. Yousefi, K. G. Shin, and B. Zhang, "Muca: New routing for named data networking," in *2018 IFIP Networking Conference (IFIP Networking) and Workshops*. IEEE, 2018.

[12] X. Chen, G. Zhang, Q. Gao, and H. Cui, "Improving ndn forwarding engine performance by rendezvous-based caching and forwarding," *Computer Networks*, vol. 145, pp. 232–242, 2018.

[13] H. Dai, J. Lu, Y. Wang, T. Pan, and B. Liu, "Bfast: High-speed and memory-efficient approach for ndn forwarding engine," *IEEE/ACM Transactions on Networking*, vol. 25, no. 2, pp. 1235–1248, 2016.

[14] C. Ghasemi, H. Yousefi, K. G. Shin, and B. Zhang, "A fast and memory-efficient trie structure for name-based packet forwarding," in *2018 IEEE 26th International Conference on Network Protocols (ICNP)*, 2018.

[15] V. Sourlas, L. Gkatzikis, P. Flegkas, and L. Tassiulas, "Distributed cache management in information-centric networks," *IEEE Transactions on Network and Service Management*, vol. 10, no. 3, pp. 286–299, 2013.