Adaptive Video Streaming with Network Coding Enabled Named Data Networking

Jonnahtan Saltarin, *Student Member, IEEE*, Eirina Bourtsoulatze, Nikolaos Thomos *Senior Member, IEEE,* and Torsten Braun *Senior Member, IEEE,*

***Abstract*—The volume of video data traffic on the Internet has increased drastically in the last years, and it is forecasted to increase further. This motivates the development of new communication methods that can deal with the growing volume of data traffic. To this aim, Named Data Networking (NDN) has been proposed as a future Internet architecture that enables ubiquitous in-network caching and naturally supports multipath data delivery. Particular attention has been given to using Dynamic Adaptive Streaming over HTTP (DASH) to enable video streaming in NDN as in both schemes data transmission is triggered and controlled by the clients. However, most of these works do not consider the multipath capabilities of NDN and the potential improvements that it can bring to the video streaming system. In this paper, we present a novel architecture for dynamic adaptive streaming over network coding enabled NDN. In comparison to previous works proposing dynamic adaptive streaming over NDN, our architecture exploits network coding to efficiently use the multiple paths connecting the clients to the sources. Moreover, our architecture enables efficient multi-source video streaming and improves resiliency to Data packet losses. The experimental evaluation shows that the proposed architecture leads to a reduced data traffic load on the sources, an increased cache hit rate at the in-network caches and a faster adaptation of the requested video quality by the clients. The performance gains are verified in a Netflix-like scenario.**

I. Introduction

As of 2015, video accounted for 70% of consumer Internet traffic, and it is expected to reach 82% by 2020 [1]. This increase in the volume of video traffic is fueled by the emergence of applications such as social video, virtual reality (VR), augmented reality (AR), *etc*., that have become popular and involve the delivery of large amounts of video data. To efficiently deliver the video, one of the most prominent approaches is Dynamic Adaptive Streaming over HTTP (DASH) [2]. The advantage of DASH compared to previous video streaming methods is that the clients are in control of the streaming logic. Meaning, they are able to decide the appropriate bitrate and resolution of the requested video, among the multiple options offered by the video sources (*e.g.*, content providers). This requires that the sources encode the video in different representations, *i.e.*, different bitrates and resolutions. Each representation is further divided into a series of video segments with a duration of a few seconds. This allows clients to adapt in real-time the demanded video resolution and bitrate in response to network changes.

The client driven video adaptation property has made of DASH an attractive method for video streaming. In particular for Information-Centric Networking (ICN) [3], one of the multiple proposals for future Internet architectures. In ICN, communication is not based on the addresses of clients and sources as in current Internet, but on the name of the content that is being requested. Two of the most popular ICN architectures are Content-Centric Networking (CCN) [4] and Named Data Networking (NDN) [5]. In these approaches, clients request content by sending an *Interest* that contains the name of the requested content. Any node that receives this Interest and stores a copy of the requested content can reply by sending a *Data* packet back to the client. If a node receives an Interest but does not store a copy of the requested content, it forwards the Interest to its neighboring nodes, using a name based routing scheme. This procedure continues until the Interest reaches a node that possesses the requested content. As the Data packet is travelling to the client, intermediate nodes can cache a copy of it, to reply to future Interests. This ubiquitous in-network caching provides better scalability by reducing the load on the sources. Clearly, the content retrieval mechanism of NDN resembles the video retrieval mechanism of DASH [6]. In particular, NDN and DASH are both client driven. Hence, clients request content by sending requests with the names assigned to each piece of content, i.e., a content object in NDN or a video segment in DASH. It is, therefore, natural to consider DASH for implementing adaptive streaming in the NDN architecture.

One of the appealing characteristics of NDN is that it enables multipath communication without the need of any additional protocol. Clients can transmit Interests over all their network interfaces (*e.g.*, LTE and WiFi) to retrieve the Data packets that compose a DASH segment. This leads to a better use of the network resources and reduces the time needed for the client to collect all the Data packets. However, when multiple clients are interested in the same Data packet, and/or when the Data packets that compose a DASH segment are distributed across multiple sources (*e.g.*, in a Content Delivery Network (CDN) with multiple video servers), the optimal content delivery rate is only attained if the Data packets are delivered over the optimal set of multicast trees [7]. This means that each node needs to know where to forward each Interest, such that the Interests and the Data packets follow the computed multicast trees, which does not scale in large and

J. Saltarin and T. Braun are with the University of Bern, Bern, Switzerland. (e-mail: {saltarin,braun}@inf.unibe.ch). E. Bourtsoulatze is with the Imperial College London, London, United Kingdom (e-mail: e.bourtsoulatze@imperial.ac.uk). N. Thomos is with the University of Essex, Colchester, United Kingdom (e-mail: nthomos@essex.ac.uk). This work was done while E. Bourtsoulatze was with the University of Bern. This work has been partially funded by the Swiss National Science Foundation under grant number 149225.

dynamic topologies. Moreover, the optimal set of multicast trees can only be computed by a central entity that is aware of the network topology, which is both computationally hard and difficult to implement in large dynamic networks. An alternative solution that avoids the computation of the optimal set of multicast trees is network coding [8]. When network coding is enabled, the nodes perform coding operations on the received Data packets (*i.e.*, combine the Data packets), instead of just forwarding them, as in traditional networks.

Unlike the NDN architecture, in which clients sequentially request the Data packets that compose a DASH segment, in a network coding enabled NDN architecture, the clients request a set of network coded Data packets. Meaning, Data packets that are generated by combining the Data packets that belong to the DASH segment. The clients can retrieve the original DASH segment as soon as they receive a decodable set of Data packets, *i.e.*, a set of Data packets that has as many linearly independent network coded Data packets as the number of Data packets belonging to the DASH segment. This removes the need to coordinate the forwarding of Interests over multicast trees, and leads to a more efficient use of the available network bandwidth. Moreover, this also simplifies the client's design, as they do not need to control the reception of each particular Data packet. They only need to keep track of the number of received network coded Data packets and to check whether a decodable set of Data packets is available.

In our previous work [9], we have explored the benefits of enabling network coding in CCN. This scheme proposed modified methods for Data and Interest processing so that the network node can process network coded Data packets along with traditional non-network coded Interests and Data packets. However, the presented protocol was not designed for video streaming. To address this issue, in this paper, we propose an adaptive video streaming architecture for network coding enabled NDN based on DASH. Our architecture aims at taking advantage of multi-path communication to improve the quality of the delivered video, reduce the resource utilization at the source and improve the resiliency to Data packet erasures. Extending our previous work [9], we have made further changes to the data structures of NDN, in order to provide better support for the network coding functionalities. In particular, we have redesigned the Content Store (CS) and the Pending Interest Table (PIT). The changes in the design of the CS and PIT have required modifications to the methods used for Interest and Data packet processing. Moreover, we propose a new model of client and source applications that enables DASH communication over network coding enabled NDN. Finally, we have implemented the proposed video streaming over network coding enabled NDN architecture in the ndnSIM simulator [10]. We compare the video streaming performance of our architecture to the unmodified NDN architecture in a Netflix-like scenario, designed with parameters available in the literature [11]–[13]. Our results demonstrate that by using network coding, our proposed video streaming architecture attains a higher cache hit rate in the router nodes. This translates into lower bandwidth consumption at the source, as well as higher bitrate seen at the clients. As a result, clients can reach their desired video quality faster.

In summary, the main contributions of this paper are the following:

- a new model of a network coding enabled NDN node, extending our previous work [9] with new features that enhance the support for the network coding functionalities;
- a new model of client and source applications that enable adaptive video streaming over network coding enabled NDN; and
- an evaluation of our proposed architecture in a Netflix-like scenario, using the ndnSIM simulator [10].

The rest of this paper is organized as follows. In Section II we discuss the related work. The background on NDN and DASH is presented in Section III. The motivation behind the use of network coding for NDN video streaming is provided in Section IV. We describe in detail the proposed adaptive video streaming over network coding enabled NDN in Section V. In Section VI we evaluate the performance of the proposed architecture. Finally, Section VII concludes our work.

II. Related Work

The similarities between NDN and DASH have been considered in previous works [3], [6], [14], [15]. Detti *et al.* [14] proposed a cooperative adaptive video streaming application for CCN. In this application, mobile users download video segments from the sources over the cellular network and also from other mobile users that are connected through WiFi. The results show that by exploiting users' cooperation, the users can drastically reduce the amount of data downloaded over the cellular network, which may result in cost reductions. An integration of CCN and DASH was presented by Lederer *et al.* [6]. This system makes use of the versioning and segmentation support that comes with CCN. The CCN versioning is used to name the different representations of a DASH segment, and segmentation is used to divide segments into Data packets that fit into lower level Maximum Transmission Units (MTU). A client requests the desired segment by sending Interests over its available interfaces. These Interests are satisfied by Data packets containing the requested DASH segment data, following the same procedures as in the CCN architecture. However, most of DASH-based video streaming proposals for ICN [6], [14], [15] consider the existence of a single path connecting the sources with the clients. This is sub-optimal as not all the paths connecting the client with the sources are exploited and, hence, network resources are wasted. Differently from previous works, in this paper we propose a novel DASH-based video streaming architecture for ICN that exploits multiple paths to receive the data. It is worth noting that the Internet Research Task Force (IRTF) is addressing the adaptation of current video streaming mechanisms to the ICN architecture in the RFC7933 [3]. It also defines some use cases for video streaming and their requirements, identifying the main issues associated with these streaming mechanisms in ICN.

The use of multipath communications in ICN has been studied by previous works [16], [17]. These works mainly focus on the design of Interest forwarding strategies that exploit all the available interfaces of the node to distribute the

Interests, without considering a specific application. Rossini *et al.* [16] investigated multipath Interest forwarding strategies, showing that the use of multiple paths simultaneously can increase resilience and reduce repository load. They also show that naive multipath Interest forwarding strategies (*e.g.*, round robin) could reduce the caching efficiency since when multiple clients decide to send the Interest for a Data packet over different paths, all the caches over the followed paths will cache a copy of the same Data packet. Schneider *et al.* [17] presented novel Interest forwarding strategies that improve end-user Quality of Experience (QoE) and reduce the clients' access cost and power consumption. To accomplish this, a new set of interface estimators are proposed, which enable fine-grained control and selection of interfaces in a multi-homed scenario. However, it is not clear how the QoE is affected when multiple multi-homed clients request the same content. In our work, we resolve the caching efficiency issue [16] by using network coding [8]. This is possible because the sources send different network coded Data packets over each path, increasing the diversity of Data packets that are cached in the network.

The application of network coding in ICN has been explored by Montpetit *et al.* [18] who proposed an architecture called *NC3N*. In this approach, Interests have a new field that contains the Data packet availability information of the client that sends the Interest, similar to the approach proposed by Sundararajan *et al.* [19]. Nodes storing Data packets that match the name prefix of the received Interest, reply only if they can provide a network coded Data packet that conveys new information to the client. However, when there are multiple clients requesting the same content, *(i)* the aggregation of Interests is problematic, since Interests with the same name but coming from different clients contain different Data packets availability information; and *(ii)*, the pipelining of Interests, *i.e.*, sending multiple concurrent Interests for different Data packets, is also problematic, since all the pipelined Interests will have the same Data packets availability information. The latter is undesirable as a node that has a matching Data packet will reply to multiple Interests with the same Data packet. These Data packets will be considered as duplicates by the client as only one of these will carry novel information with respect to the Data packets that are already available at the client. Inspired by NC3N [18], Wu *et al.* [20] proposed CodingCache, where network coding is used to replace the Data packets in the cache of the network nodes. Due to the increased Data packet diversity in the network, the cache hit rate is improved. However, CodingCache suffers from the same drawbacks as in NC3N, namely, the Interest aggregation and Interest pipelining are problematic. In the work presented by Llorca *et al.* [21], multicast delivery in network coding enabled ICN is optimized by finding the evolution of the Data packets that are cached in the network. However, this approach needs a central entity that is aware of the network topology and the Interests, which does not scale well with the number of network nodes. The demonstrated benefits that network coding brings to ICN encouraged researchers to study applications that can take advantage of network coding enabled NDN. Matsuzono *et al.* [22] have proposed L4C2, a network coding enabled mechanism for low latency, low loss video streaming over CCN. In L4C2, the network nodes estimate the acceptable delay and Data packet loss rate in their uplinks, adjusting the requested video quality accordingly. The clients first request non-network coding Data packets, and only request network coded Data packets when they detect Data packet losses. In this case, the benefits of network coding are only exploited when Data packet losses occur.

TABLE I
NOTATION

$\mathcal{S}$	$\triangleq$	Set of source nodes
$\mathcal{R}$	$\triangleq$	Set of router nodes
$\mathcal{C}$	$\triangleq$	Set of client nodes
$\mathcal{Y}$	$\triangleq$	Set of DASH representations
$\mathcal{Z}^j$	$\triangleq$	Set of DASH segments that form representation j
$\mathcal{P}_n$	$\triangleq$	Set of Data packets that form a DASH segment with name n
$\mathcal{P}_n^\nu$	$\triangleq$	Subset of Data packets with name n, stored in node ν
$p_{n,j}$	$\triangleq$	The jth Data packet in the set $\mathcal{P}_n$
$\mathcal{I}_n$	$\triangleq$	Set of Interests that request the set of Data packets $\mathcal{P}_n$
$i_{n,j}$	$\triangleq$	The jth Interest in the set $\mathcal{I}_n$
$\hat{\mathcal{P}}_n$	$\triangleq$	Set of network coded Data packets belonging to the DASH segment with name n
$\hat{p}_{n,g}$	$\triangleq$	A network coded Data packet belonging to generation g in the set $\mathcal{P}_n$
$\hat{\mathcal{I}}_n$	$\triangleq$	Set of Interests that request the set of network coded Data packets $\hat{\mathcal{P}}_n$
$\hat{i}_{n,g}$	$\triangleq$	An Interest requesting a Data packet in the set $\mathcal{I}_n$
$\mathcal{F}_n^\nu$	$\triangleq$	Set of faces at node ν that are configured to forward Interests with name prefix n

III. Adaptive Video Streaming over Named Data Networking (NDN)

In this section, we introduce the main concepts that enable dynamic adaptive video streaming over named data networking. First, we describe the operation of Dynamic Adaptive Streaming over HTTP (DASH). Then, we describe the operation of Named Data Networking (NDN) and show the similarities between NDN and DASH. The main notation used throughout the paper is summarized in Table I.

A. Dynamic Adaptive Streaming over HTTP (DASH)

As we mentioned in the Introduction, one of the most prominent video streaming techniques used nowadays is adaptive video streaming, and, in particular, Dynamic Adaptive streaming over HTTP (DASH). One of the main characteristics of DASH is that the clients are in control of the streaming logic, deciding the bitrate and resolution of the streamed video. To enable the video quality adaptation by the clients, each video x is encoded with different parameters (e.g. bitrate, resolution, *etc.*), creating a set of *representations* $\mathcal{Y}^x$. The video data in each representation $y^x \in \mathcal{Y}^x$ is divided into a set of *segments* $\mathcal{Z}^{x,y}$. Every segment $z^{x,y} \in \mathcal{Z}^{x,y}$ has the same duration. This allows clients to request segments belonging to the representation that better adapts to their current network conditions, display capabilities, *etc*. Hence, the clients can switch to a different representation after receiving each segment if the network conditions change, for example. To inform the clients about the offered video qualities, a file called the Media Presentation Description (MPD) is associated with each video x. This file contains information about the available representations $\mathcal{Y}^x$ in which the video x has been encoded and the segments $\mathcal{Z}^{x,y}$ that compose each representation, among other parameters. A client that is interested in receiving the video x should first request the MPD file associated with this video. Then, after receiving and parsing the MPD file of the video x, the client knows what representations are available and what names it should use to request the video segments.

B. DASH over Named Data Networking (NDN)

We consider a network that is formed by *(i)* a set of source nodes $\mathcal{S}$ that generate and store DASH segments, *(ii)* a set of clients $\mathcal{C}$ that request DASH segments, and *(iii)* a set of router nodes $\mathcal{R}$ through which the DASH segments are requested and transmitted. Every node $\nu \in \mathcal{S} \cup \mathcal{C} \cup \mathcal{R}$ is connected with its neighboring nodes through a set of faces $\mathcal{F}^\nu$. We consider that the connection between nodes is wired, thus, no interference or overhearing is possible.

Each source $s \in \mathcal{S}$ stores DASH segments that can be requested by the clients. Since the size of DASH segments is usually larger than the Maximum Transmission Units (MTU), the DASH segments are divided into smaller Data packets that fit into an MTU. Therefore, we consider that a DASH segment $z^{x,y}$ is also a set of Data packets $\mathcal{P}_n = \{p_{n,1}, \dots, p_{n,|\mathcal{P}_n|}\}$, where n is a name based on x, y, and z. To retrieve a DASH segment composed by the set of Data packets $\mathcal{P}_n$, a client $c \in \mathcal{C}$ should send a set of Interests $\mathcal{I}_n = \{i_{n,1}, \dots, i_{n,|\mathcal{I}_n|}\}$, where $|\mathcal{I}_n|$ = $|\mathcal{P}_n|$, meaning that there is one Interest for each Data packet. Note that the actual number of Interests that the client c should send to retrieve $\mathcal{P}_n$ may be higher than $|\mathcal{P}_n|$, since in a lossy network some Interests and Data packets may be lost. Thus, some of the Interests in $\mathcal{I}_n$ will need to be sent more than once. The Interests are sent over a set of faces $\mathcal{F}_n^c$ that are configured to forward Interests with name prefix n. The information about the faces of a node that are configured to forward Interests for a specific name prefix is stored in the *Forwarding Information Base (FIB)* table. Other than the FIB, each NDN node has two additional tables: a *Content Store (CS)*, that is a cache where Data packets that pass through the node can be stored, and a *Pending Interest Table (PIT)*, that keeps track of the Interests that the node have forwarded and the faces over which those Interests have arrived.

In NDN, when a node receives an Interest, it either: *(i)* waits for the requested Data packet to arrive, if previously an Interest for the same Data packet has already been forwarded; *(ii)* replies to the Interest with a matching Data packet from its CS; or *(iii)* forwards the Interest to other nodes, in order to receive the requested Data packet. This is further explained in the following.

- *Waiting for a new Data packet* — Consider a node ν that receives an Interest $i_{n,j}$ over the face f. If the node ν finds in its PIT an entry that matches the name in the Interest, it means that it has already forwarded $i_{n,j}$ and hence the Data packet $p_{n,j}$ is expected. In this case, the node adds to the respective PIT entry the face f over which the Interest has arrived, and does not forward $i_{n,j}$ again.
- *Replying to an Interest* — When the PIT of the node ν does not have any entry that matches the Interest $i_{n,j}$, it looks for a Data packet with a matching name in its CS. If the CS stores a copy of the Data packet $p_{n,j}$, the node ν replies to the Interest $i_{n,j}$.
- *Forwarding an Interest* — If the CS of the node ν does not contain a Data packet matching the name of the Interest $i_{n,j}$, the node ν forwards the Interest to one or more of its neighboring nodes, according to its FIB. Moreover, the node ν also updates its PIT table to add the information of the forwarded Interests.

Once the requested Data packet $p_{n,j}$ is found in the CS of a router or in a source, it is sent to the client over the reverse path of that followed by the Interest. When a node receives a Data packet $p_{n,j}$ over a face f, it first looks up in its PIT for an entry that matches the name of the Data packet $p_{n,j}$. If no PIT entry matches the name n, j, the Data packet is considered unsolicited and it is discarded. If no PIT entry matches the name n, j, the Data packet is forwarded over all the faces specified in the corresponding PIT entry. Additionally, the Data packet $p_{n,j}$ may be added to the CS, according to the caching policy of the node. A more detailed description of the NDN operation is provided in the NFD Developer Guide [23].

IV. Motivation

A. Challenges of Multipath Streaming

Video streaming is a data intensive application that requires a moderately high and stable amount of bandwidth between the video sources and the clients. For this reason, utilizing multiple paths to connect the clients to the sources can improve the quality of the video received by clients. Nowadays, most client devices, *e.g.*, mobile phones, laptops, *etc.*, have two or more network interfaces that can connect them to the video sources. However, in the traditional host-centric networking, support for multipath is not extended. Recent efforts to support multipath communications on TCP (MP-TCP) [24] have been developed by the IETF. However, end-to-end connections need to be established for each interface, which makes difficult the use of in-network caching and dynamic selection of the sources. In comparison, NDN provides natural support of multipath content retrieval. This is achieved by allowing clients to distribute all the Interests needed to retrieve a DASH segment over all the available faces. However, multipath video content retrieval in NDN is well supported in the following scenarios:

- *Multi-source unicast* — Let us consider the case illustrated in Fig. 1a, where a client c_1 is interested in a DASH segment composed of the set Data packets $\mathcal{P}_n$. Let us also consider that the $|\mathcal{P}_n|$ Data packets that compose $\mathcal{P}_n$ are distributed across multiple sources $\mathcal{S}$, such that each source $s \in \mathcal{S}$ stores a subset of Data packets $\mathcal{P}_n^s \subset \mathcal{P}_n$. An example of this scenario is a Content Delivery Network (CDN) in which content is

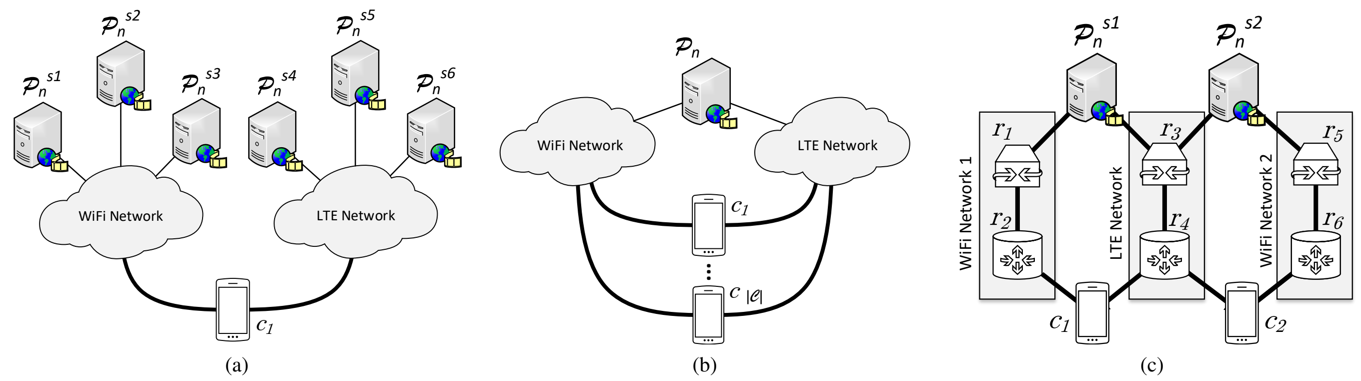

Fig. 1. Devices retrieving Data packets over LTE and WiFi: (a) multi-source unicast; (b) single-source multicast; (c) multi-source multicast (butterfly network).

distributed across multiple video servers. In this case, the client and the routers need to select properly the face that will forward each Interest, so that it reaches the source that stores the requested Data packet. This can be accomplished by carefully configuring the FIB table of all the nodes, such that each Interest reaches the right source. However, for contents comprised of a large number of Data packets, such as DASH segments, keeping the FIB tables of all the nodes updated for each Data packet does not scale well, and the FIB size could become very large. Moreover, in large networks and in the presence of unreliable sources that can become available or unavailable at any moment, keeping the FIB tables updated may require a lot of signaling messages, and thus lead to waste of resources.

- *Single-source multicast* — Let us now consider the case where a single source stores the complete set of Data packets $\mathcal{P}_n$ that compose a DASH segment, and that multiple clients $\mathcal{C}_n \subset \mathcal{C}$ are interested in $\mathcal{P}_n$, as illustrated in Fig. 1b. To minimize the time needed for each client to receive the complete set of Data packets $\mathcal{P}_n$, while also minimizing the number of duplicated Data packet transmissions in the network, the Data packets need to travel over an optimal set of multicast trees [7]. To accomplish this in NDN, each node should know where to forward each Interest $i_{n,j}$ such that all the Interests $i_{n,j}$ sent by different clients are aggregated at the optimal point in the network, minimizing the number of redundant transmissions of $p_{n,j}$. Moreover, computing these multicast trees has high computational complexity, is not reliable under network dynamics, and requires the knowledge of the network topology [7]. In the illustrative example shown in Fig. 1b, clients are connected to the source through both LTE and WiFi interfaces. If all the clients send the Interest $i_{n,j}$ over the LTE interface, the Data packet $p_{n,j}$ will only be sent through the LTE network. However, if a fraction of the clients decide to send the Interest $i_{n,j}$ over the WiFi interface, the Data packet $p_{n,j}$ will also be sent from the source to the WiFi network, wasting network resources.
- *Multi-source multicast* — Let us consider that multiple clients are interested in a DASH segment composed by a set of Data packets $\mathcal{P}_n$ that are distributed across multiple sources. This scenario presents the issues that appear in the multi-source unicast and the single-source multicast scenarios. To illustrate this, let us the network presented in Fig. 1c, where the two clients c_1 and c_2 need to coordinate where they need to send each Interest, such that the router r_4 is able to aggregate the Interests for the same Data packet. Moreover, when the each of the sources has a disjoint set of Data packets, *i.e.*, $\mathcal{P}_n^{s_1} \neq \mathcal{P}_n^{s_2}$, the clients also need to know which Data packets each source stores, to avoid sending Interests to the source that does not store a copy of the requested Data packet.

From the discussion above, it becomes obvious that there are open challenges related to the deployment of efficient adaptive video streaming systems in NDN. In the next section, we will explain how these challenges can be effectively addressed by introducing network coding in the NDN architecture.

B. Network Coding enabled NDN

The shortcomings of the NDN architecture discussed in the previous sub-section can be solved by enabling network coding [8], a technique in which the Data packets delivered to the clients are coded by means of combining the Data packets available at sources and routers prior to being forwarded. Hence, network coded Data packets contain information from all the Data packets that have been combined to generate them. The key idea behind introducing network coding in NDN is that clients no longer need to request specific Data packets, but rather network coded Data packets as they all have equivalent information. With network coding, the nodes do not need to coordinate the faces where they forward the Interests, which optimizes the network bandwidth utilization.

Differently from NDN, where an Interest $i_{n,j}$ requests a specific Data packet $p_{n,j}$, in a network coding enabled NDN, an Interest $\hat{i}_n$ requests a network coded Data packet $\hat{p}_n$, without specifying the particular Data packet ID j. In this case, the set of Interests needed to retrieve $\hat{\mathcal{P}}_n$ is $\hat{\mathcal{I}}_n = \{\hat{i}_n\}$, *i.e.*, the set contains a single Interest. To retrieve the demanded content, each client sends the Interest $\hat{i}_n$ at least $|\mathcal{P}_n|$ times. Note that more than $|\mathcal{P}_n|$ Data packets may be needed, as network coded Data packets are generated by randomly combining the Data packets with name prefix n. Hence, with some small probability when coding is done in a large finite field, these Data packets can be linearly dependent. Furthermore, due to losses, additional Interests may need to be sent to compensate for the lost Data packets. Any node ν can reply to these Interests with network coded Data packets. The network coded Data packets are generated by combining the set of Data packets $\mathcal{P}_n^\nu$ that are available in the CS or the repository of the node ν and that match the name prefix n.

The Data packet $\hat{p}_n$ can be considered as a vector $\hat{\mathbf{p}}_n$, where each element of the vector belongs to a finite field. Then, the set of network coded Data packets $\mathcal{P}_n$ can be expressed as a matrix $\hat{\mathbf{P}}_n$ where each row corresponds to a Data packet $\hat{\mathbf{p}}_n$. The operations performed by the node ν to generate a new network coded Data packet $\hat{\mathbf{p}}_n$ can be expressed as $\hat{\mathbf{p}}_n = \mathbf{A} \cdot \mathbf{P}_n^\nu$, where $\mathbf{A}$ is a matrix of coding coefficients drawn from a finite field, and $\mathbf{P}_n^\nu$ is the matrix formed with the set of Data packets $\mathcal{P}_n^\nu$. When the coding coefficients in $\mathbf{A}$ are randomly chosen from a large finite field, the generated Data packets have a high probability of being linearly independent with respect to the Data packets previously generated, and thus, innovative [25]. To decode the original Data packets that compose $\mathcal{P}_n$, a client should collect $|\mathcal{P}_n|$ innovative network coded Data packets $\hat{p}_n$.

To illustrate the benefits that network coding brings to video streaming over NDN, let us revisit the scenarios described in Section IV-A.

- *Multi-source unicast* — Differently to the original NDN, in a network coding enabled NDN the clients and the routers do not need to know which sources they can reach over each face, since they send Interests for network coded Data packets that are stored at any source rather than for specific Data packets that are stored at specific sources. This implies that the FIB tables can be smaller, since only one entry for the name prefix n, and not one for each Data packet name n, j, is needed. Each source then replies to these Interests with network coded Data packets $\hat{p}_n$, generated by combining the Data packets that match the name prefix n.
- *Single-source multicast* — When multiple clients send Interests for the same DASH segment in a network coding enabled NDN, they do not need to coordinate the Interests that they send over each face, since all of the Interests are for network coded Data packets and they can be aggregated at any node.
- *Multi-source multicast* — In this scenario, when network coding is allowed, no coordination is needed at the clients nor the routers, since all the Interests can be satisfied by any network coded Data packet.

Apart from the gains in the above cases, network coding also improves *(i)* the throughput, in particular when bottlenecks are present in the network, as well as *(ii)* the resiliency to Data packet losses. To illustrate the throughput improvement, let us consider the network topology shown in Fig. 1c, which is also known as the butterfly network. We consider that the clients c_1 and c_2 are interested in a DASH segment composed by the set of Data packets $\mathcal{P}_n = \{p_{n,1}, p_{n,2}\}$ that is distributed across both sources, such that the source node s_1 stores a copy of the Data packet $p_{n,1}$ (*i.e.*, $\mathcal{P}_n^{s_1} = \{p_{n,1}\}$) and the source node s_2 stores a copy of the Data packet $p_{n,2}$ (*i.e.*, $\mathcal{P}_n^{s_2} = \{p_{n,2}\}$).

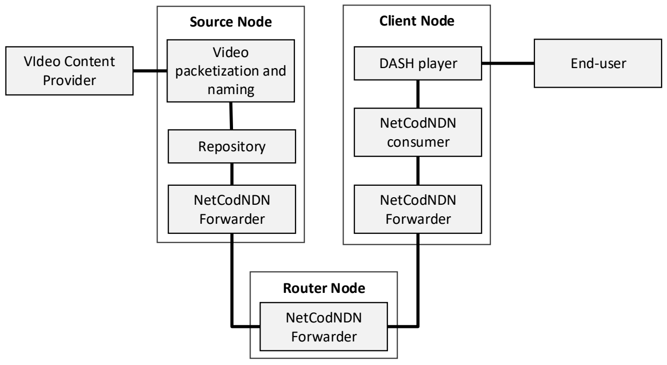

Fig. 2. Proposed architecture for adaptive video streaming over network coding enabled NDN.

In this case, if network coding is not enabled, the router r_4 cannot aggregate the Interests sent by the clients c_1 and c_2, since they are for different Data packets. This means that the link between the routers r_3 and r_4 becomes a bottleneck for the Data packets $p_{n,2}$ and $p_{n,1}$ traveling to the clients c_1 and c_2, respectively. Thus, one of the clients will see a higher delay in receiving the complete set of packets $\mathcal{P}_n$, which is critical for time-constrained applications such as video streaming. In contrast, in a network coding enabled NDN scenario, the router r_4 is able to aggregate the Interests sent by the clients c_1 and c_2, as these Interests are for network coded Data packets. If the router r_3 network encodes the Data packets received from the sources, the resulting network coded Data packet will be useful for both clients c_1 and c_2.

Furthermore, in order to illustrate how network coding improves the resiliency to Data packet losses, let us now consider a network in which Interests or Data packets losses can occur. Let us also consider that a client is interested in a set of Data packets $\mathcal{P}_n$, and that one of the Interests sent by the client or one of the Data packets sent to the client is lost in the network. Hence, one of the Data packets that the client is expecting will not arrive. If network coding is not enabled, the client should wait until an Interest expires before realizing which Data packet will not arrive. Then, the node should send the same Interest and wait again for the Data packet. In contrast, if network coding is enabled, a proactive node that knows the average Data packet loss rate can request $|\mathcal{P}_n| + \epsilon$ Data packets, where ϵ depends on the packet loss rate and the finite field where the coding operations are performed. This is possible because, in the event of an Interest or Data packet loss, any other network coded Data packet will be useful to the client, even if it is not an exact copy of the lost Data packet. This reduces the time that the client needs to retrieve the complete set of Data packets $\mathcal{P}_n$, since the clients do not need to wait until the lost Interest expires.

V. System Design and Implementation

In this section, we present our framework for dynamic adaptive streaming (DASH) over network coding enabled NDN. Our system is based on our previously proposed NetCodCCN architecture [9], which here is advanced to support DASH video streaming.

We consider a set of videos $\mathcal{X}$ that are made available by a *video content provider* to a set of *end-users*. The videos $\mathcal{X}$ are encoded into a set of different DASH representations $\mathcal{Y}^x$. Each representation $y^x \in \mathcal{Y}^x$ is divided into a set of segments $\mathcal{Z}^{x,y}$, as described in Section III-A. Each segment $z^{x,y} \in \mathcal{Z}^{x,y}$ has a name $n = x/y/z$ that identifies it uniquely. This name is composed of the video stream ID x, the representation ID y, and the segment ID z. The segments $\mathcal{Z}^{x,y}$ are stored at the sources. An end-user expresses its interest in watching a video $x \in \mathcal{X}$ to its client, which then requests the video x from the network. In Fig. 2, we show a simple network example composed of a source $s \in \mathcal{S}$, a router $r \in \mathcal{R}$, and a client $c \in \mathcal{C}$, and we illustrate the main components of these nodes. In the following sections, we discuss the source, router and client nodes implementation in more details.

A. Source Design

We consider that a source $s \in \mathcal{S}$ is a node that replies with the content in its *repository* to the Interests that it receives. Differently from a CS, a repository is designed to keep the Data packets for longer time periods, using persistent storage devices. Before the video content is requested by the clients, the sources are initialized as follows: *(i)* The source receives video content, in the form of DASH segments, from a video content provider. *(ii)* Each of the DASH segments is divided into video Data packets, and a name is associated with each Data packet. *(iii)* The video Data packets are loaded into the repository. Then, the repository replies with network coded versions of these Data packets whenever it receives an Interest that matches the name prefix. Next, we present further details about this process.

1) Video Data packetization and Naming: Each segment $z^{x,y} \in \mathcal{Z}^{x,y}$ is divided into a set of Data packets, $\mathcal{P}_n$, such that each Data packet $p_{n,j} \in \mathcal{P}_n$ fits into a Data packet. As presented in Section IV-B, the Data packet $p_{n,j}$ can be represented as a vector $\mathbf{p_{n,j}}$. To facilitate the deployment of network coding in practical settings [26], each Data packet $\mathbf{p_{n,j}}$ is prepended with an encoding vector $\mathbf{v_j}$ to inform the routers and clients about the coding operations that the network coded Data packet has been subjected to. At the sources, the initial value of this vector is set to be the jth unit vector, which has value 1 in the jth position and 0 otherwise.

Prepending encoding vectors to the Data packets introduces a communication overhead that consumes network resources, especially when the segment $z^{x,y}$ is divided into a large number of Data packets. To limit this overhead, we adopt the concept of *generations* [26], where the original set of Data packets that compose $\mathcal{P}_n$ are divided into smaller groups of Data packets, which are known as generations. The coding operations are restricted only between Data packets that belong to the same generation, in order to reduce the network coding overhead and make it appropriate for time-constrained applications. The set of Data packets that form the generation g is denoted as $\mathcal{P}_{n,g}$, where g is the generation id. Thus, $\mathcal{P}_n = \cup_{g=1}^{G} \mathcal{P}_{n,g}$, where G is the total number of generations. To avoid mixing Data packets from different generations, the generation id, g, is added to the name of the Data packet. The size of the generation controls the tradeoff between the overhead required to communicate the encoding vector and the Data packet diversity. Note that the size of the encoding vectors prepended to each Data packet is equal to the size $|\mathcal{P}_{n,g}|$ of the generation g, as the network coded Data packets may potentially carry information from all the Data packets in $\mathcal{P}_{n,g}$.

2) Repository: Each source s contains one or more *repositories* where the set of Data packets $\mathcal{P}^s_{n,g} \subset \mathcal{P}_{n,g}$ are stored. It is worth noting that a source s may not store the whole set of Data packets $\mathcal{P}_{n,g}$, but only a subset $\mathcal{P}^s_{n,g} \subset \mathcal{P}_{n,g}$. We should note that repositories and content stores have similar functionalities: *(i)* when they receive a Data packet $p_{n,g} \in \mathcal{P}_{n,g}$, they store it in their memory; and *(ii)* when they receive an Interest $\hat{i}_{n,g}$, they return a network coded Data packet $\hat{p}_{n,g}$ generated with the set of Data packets $\mathcal{P}^s_{n,g}$ that are available in their memory. Even though the design of repositories and content stores might differ, in this paper we consider that their functionality is the same, as will be further explained in Section V-C1.

B. Client Design

In our architecture, we consider that a client $c \in \mathcal{C}$ is a node that generates Interests for a video requested by an end-user. Our model of a client node consists of two main components: *(i)* a DASH player that decides the DASH representation that should be displayed and requests the appropriate segments, and *(ii)* a NetCodNDN consumer that receives requests for segments and generates Interests for network coded Data packets. Moreover, when the network coded Data packets arrive to the NetCodNDN consumer, it decodes them and reassembles the original segment before sending it to the DASH player. These components are further described in the following.

1) DASH player: The DASH player is the most direct interface between the end-user (*e.g.*, a person) and the video communication system. When a DASH player receives a request from an end-user to retrieve a video $x \in \mathcal{X}$, it first requests the MPD file. This file is typically of a small size and needs to be communicated only once. Thus, it is not network coded and is requested as traditional NDN content object. Given the MPD information, the available resources and the configuration parameters, the DASH player decides which representation $y \in \mathcal{Y}^x$ is the optimal every time that it has to request a new segment $z^{x,y} \in \mathcal{Z}^{x,y}$. The DASH player is agnostic to the protocol deployed in the network for requesting the segment, which can be either NetCodNDN, NDN or HTTP. It only takes into account the delay seen by the segment to be completely received.

2) NetCodNDN consumer: Whenever a NetCodNDN consumer receives requests for a DASH segment $z^{x,y} \in \mathcal{Z}^{x,y}$, it generates and forwards a set of Interests $\hat{\mathcal{I}}_{n,g}$, where n is the name prefix of the segment, and g is the generation id. The generations are requested sequentially, starting from $g = 1$ up to the last generation, G. The total number of generations G can be obtained as Data packet metadata, or simply by adding a flag to the last generation.

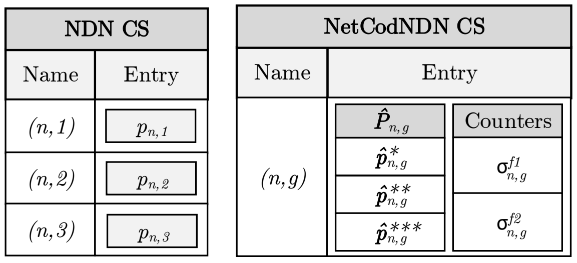

Fig. 3. Structure of a Content Store (CS) storing three Data packets with name prefix n: (a) In NDN, there are three CS entries, each one storing a single Data packet; (b) In NetCodNDN, there is a single entry that contains a matrix $\hat{\mathbf{P}}_{n,g}$ that stores the content of the three Data packets, and a set of counters $\sigma^{f}_{n,g}$.

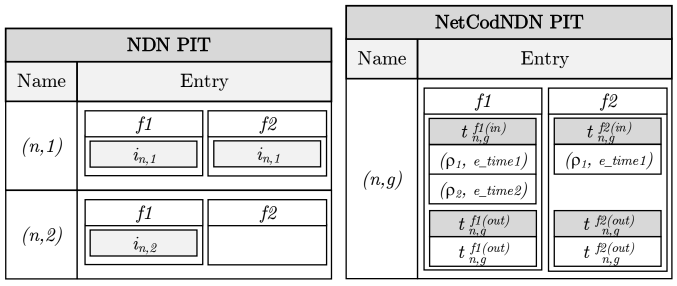

Fig. 4. Structure of a Pending Interest Table (PIT) storing two pending Interests with name prefix n: (a) In NDN, there are two PIT entries, each one storing a single Interest; (b) In NetCodNDN, there is a single entry that stores information about both Interests.

The NetCodNDN consumer keeps track of the received innovative Data packets $\hat{\mathbf{p}}_{n,g}$ in a matrix $\hat{\mathbf{p}}_{n,g}$, so that the original set of Data packets can be retrieved by performing Gaussian elimination when the matrix $\hat{\mathbf{p}}_{n,g}$ is full rank, *i.e.*, it contains $|\mathcal{P}_{n,g}|$ linearly independent Data packets.

C. The NetCodNDN forwarder

The NetCodNDN forwarder is in charge of *(i)* routing Interests towards the sources, *(ii)* forwarding Data packets back to the clients, and *(iii)* applying network coding operations on the Data packets before forwarding them. In the following we describe the architecture of the NetCodNDN forwarder, starting with the modified data structures: the new Content Store (CS) and the new Pending Interest Table (PIT). The Forwarding Information Base (FIB) of the NetCodNDN forwarder is the same as the FIB of the NDN forwarder [23], thus, we do not describe it here. Then, we present the algorithms followed by nodes when they receive Interests or Data packets. Note that in our previous work [9], we presented a network coding content retrieval scheme that utilized the original CS and introduced a few modifications to the PIT, keeping the model of the proposed NetCodCCN [9] as similar as possible to the model of CCN [4], at the expense of performance. The new CS and PIT models presented in this paper are redesigned to improve the performance of the network coding operations. Also note that in our architecture, other than the redesigned CS and PIT, the nodes still have the traditional CS and PIT to process non-network coding Interests and Data packets.

1) Content Store: As presented in section III-B, an NDN node ν can keep a cache with a set of Data packets $\mathcal{P}^{\nu}_{n}$ that it has received and considered useful to store, in order to reply to future Interests for the name prefix n. In traditional NDN, each Data packet $p_{n,j} \in \mathcal{P}^{\nu}_{n}$ is stored as an entry in the CS. When a node ν receives an Interest $i_{n,j}$, it looks into its CS to find all the entries that match the name prefix (n,j) of the Interest. Since the name prefix (n,j) refers to a specific Data packet, only one entry will match.

Differently from NDN, in a network coding enabled NDN the clients do not specify a precise name (n,j) for the Interests they send, but rather a name prefix (n,g) that refers to any network coded Data packet generated from the set $\hat{\mathcal{P}}_{n,g}$, where g is a generation id. When a node ν receives an Interest $\hat{i}_{n,g}$, it replies with a network coded Data packet $\hat{p}_{n,g}$ only if it considers that this Data packet has high probability of being innovative for the client. The node ν generates a network coded Data packet $\hat{p}_{n,g}$ by randomly combining the Data packets $\hat{\mathbf{P}}^{\nu}_{n,g}$ in its CS. Thus, $\hat{p}_{n,g} = \sum_{j=1}^{|\hat{\mathbf{P}}^{\nu}_{n,g}|} a_j \cdot \hat{\mathbf{p}}^{(j)}_{n,g}$, where a_j is a randomly selected coding coefficient and $\hat{\mathbf{p}}^{(j)}_{n,g}$ in the jth Data packet in $\hat{\mathbf{P}}^{\nu}_{n,g}$.

In our previous work [9], we used the CS model provided by CCN [4] for both, network coding and traditional Interests. In this case, generating a Data packet $\hat{p}_{n,g}$ requires up to $|\hat{\mathbf{P}}_{n,g}|$ lookups to the CS, which is inefficient. Differently, in this paper we propose a new design of the CS that facilitates the generation of network coding Data packets with respect to that of the NDN forwarder, as depicted in Fig. 3. In the NetCodNDN forwarder, each CS entry contains a set of network coded Data packets, $\hat{\mathcal{P}}^{\nu}_{n,g}$, where ν is the node ID and all the Data packets belong to the same generation g. This set of Data packets is stored as a matrix $\hat{\mathbf{P}}^{\nu}_{n,g}$, where each row is a vector $\hat{\mathbf{p}}_{n,g}$ that represents the network coded Data packet $\hat{p}_{n,g}$. This allows to reduce the number of lookups to the CS needed to generate a network coding Data packet to one, which is much more efficient than our previous work [9].

Moreover, each CS entry also stores a counter $\sigma^{f}_{n,g}$ for each face f of the node ν. Each counter $\sigma^{f}_{n,g}$ measures the number of Data packets generated with the content of the matrix $\mathbf{P}^{\nu}{}_{n,g}$ that have already been sent over the face f, *i.e.*, it measures the amount of information from the matrix $\hat{\mathbf{P}}_{n,g}$ that has been transferred from the node ν to the neighbor node connected over the face f. When a Data packet with name prefix (n,g) is removed from $\hat{\mathbf{P}}_{n,g}$ (*e.g.*, when the CS eviction policy decides that a Data packet with name prefix (n,g) needs to be removed from the CS), the amount of information in $\hat{\mathbf{P}}_{n,g}$ is reduced by 1, and the value of $\sigma^{f}_{n,g}$ needs to be decreased by 1 for all the faces.

2) Pending Interest Table: As described in section V-C1, node ν with a NetCodNDN forwarder receives an Interest $\hat{i}_{n,g}$, it uses the set of Data packets $\hat{\mathcal{P}}^{\nu}_{n,g}$ that are stored in its CS to generate a network coded Data packet and reply to the Interest. However, if the Data packets stored in the CS are not sufficient to generate an innovative Data packet, the node ν may need to wait until it receives new Data packets before replying to the Interest. In this case, the node forwards the Interest $\hat{i}_{n,g}$

to its neighbors and stores the face over which the Interest arrived and the face over which the Interest was forwarded in the Pending Interest Table (PIT).

To facilitate the new functionalities in the NetCodNDN forwarder, the design of the PIT has to be modified with respect to that of the NDN forwarder, as depicted in Fig. 4. The redesigned PIT is a collection of entries $\mathcal{T} = \{t_{n,g} \dots\}$. Each PIT entry $t_{n,g}$ keeps track of the received Interests with name prefix (n, g) that were forwarded and are pending, *i.e.*, have not been consumed by a Data packet. Each entry $t_{n,g}$ has two components for each face f, an *in-record* $t_{n,g}^{f(in)}$ that keeps track of the Interests that arrived over the face f and that have not been satisfied, and an *out-record* $t_{n,g}^{f(out)}$ that keeps track of the Interests that have been forwarded over the face f, and that are still pending.

The in-record $t_{n,g}^{f(in)}$ is a list that keeps track of the Interests $\hat{\imath}_{n,g}$ that arrived over the face f. Each element in this list is a tuple of the form $t_{n,g}^{f,\rho(in)} = (\rho, e\_time)$, where ρ is the rank that the matrix $\hat{\mathbf{P}}_{n,g}^{\nu}$ must have before replying to the Interest, and e_time is the expiration time of the Interest. The size of this list, denoted as $|t_{n,g}^{f(in)}|$, is the total number of Data packets with name prefix (n, g) that should be sent over face f.

The out-record $t_{n,g}^{f(out)}$ is a scalar that keeps track of the number of Interests with name prefix (n, g) that have been forwarded over the face f. The total number of Interests with name prefix (n, g) that have been forwarded by the node ν over all its faces is computed as $t_{n,g}^{(out)} = \sum_{f \in \mathcal{F}_{n,g}^{\nu}} t_{n,g}^{f(out)}$, where $\mathcal{F}_{n,g}^{\nu}$ is the set of faces of node ν that are configured to forward Interests with name prefix (n, g).

3) Interest Processing: In our architecture, video streaming is triggered by the clients that send Interests $\hat{\imath}_{n,g}$ for network coded Data packets, where n is the name of the DASH segment and g is the generation ID. The Interests $\hat{\imath}_{n,g}$ have a *NetworkCodingAllowed* field that signals the nodes to process the Interest following the NetCodNDN forwarder procedure, which is detailed below and summarized in Algorithm 1. If the *NetworkCodingAllowed* field is not present, the Interests are treated following the original NDN procedures [23].

When a node with a NetCodNDN forwarder receives an Interest for a network coded Data packet, it either *(i)* replies to the Interest with a network coded Data packet generated by combining the Data packets in its CS; *(ii)* forwards the Interest to its neighboring nodes, to receive an innovative network coded Data packet; or *(iii)* waits for a new network coded Data packet to arrive, if a previously received Interest with the same name prefix has already been forwarded. This is further explained in the below.

- *Replying to an Interest* — The node ν replies to an Interest $\hat{\imath}_{n,g}$ when *(i)* it has collected $|\hat{\mathcal{P}}_{n,g}|$ innovative network coded Data packets, meaning that the generation g is decodable; or when *(ii)* a network coded Data packet generated by the node ν has high probability to be innovative for the neighbor node connected through the face f that sent the Interest. The number of network coded Data packets that can be generated by the node ν and that have a high probability to be innovative is given by $\xi_{n,g}^{f} = \texttt{rank}(\hat{\mathbf{P}}_{n,g}^{\nu}) - \sigma_{n,g}^{f}$. Recall that the parameter $\sigma_{n,g}^{f}$ denotes the number of network coded Data packets that have been sent over the face f. When $\xi_{n,g}^{f}$ is greater than 0, the node ν generates a new network coded Data packet $\hat{p}_{n,g}^{*} = \sum_{j=1}^{|\hat{\mathcal{P}}_{n,g}^{\nu}|} a_j \cdot \hat{\mathbf{p}}_{n,g}^{(j)}$ and sends it over the face f.
- *Forwarding an Interest* — If the number of network coded Data packets that can be generated by the node ν and that have a high probability to be innovative, $\xi_{n,g}^{f}$, is equal to 0, the node ν needs to receive a new innovative Data packet that increases the rank of $\hat{\mathbf{P}}_{n,g}^{\nu}$ before it is able to reply to the Interest $\hat{\imath}_{n,g}$. Before forwarding an Interest, the node ν checks its PIT. In order to support Interest pipelining, *i.e.*, sending multiple concurrent Interests for different Data packets of the same segment, the PIT lookup procedure of the NetCodNDN forwarder is different to that of the NDN forwarder. Specifically, if a matching PIT entry $t_{n,g}$ is found, the node ν adds a new tuple $(\rho + 1, \texttt{expire}(\hat{\imath}_{n,g}))$ to the in-record $t_{n,g}^{f(in)}$, where ρ is the highest rank on the in-record and $\texttt{expire}(\hat{\imath}_{n,g})$ is the expire time of the interest $\hat{\imath}_{n,g}$. The node ν forwards the Interest $\hat{\imath}_{n,g}$ if the number $\epsilon_{n,g}$ of innovative network coded Data packets with name prefix (n, g) that it is expecting to receive before the Interest $\hat{\imath}_{n,g}$ expires is not enough to satisfy all the pending Interests. To compute $\epsilon_{n,g}$, the node ν needs to take into consideration the Interest and Data packet loss rate and delays, among other variables. For the sake of simplicity, the NetCodNDN forwarder assumes that any forwarded Interest brings an innovative Data packet before its expiration. This assumption is similar to the one made by the original NDN forwarder, where received Interests are not further forwarded if a PIT entry matching the name of the Interest is found, since

Algorithm 1 Interest processing in the NetCodNDN forwarder

```
Require: $\hat{\imath}_{n,g}$, $f$, $\hat{\mathcal{P}}_{n,g}^{\nu}$
 1: if $\texttt{rank}(\hat{\mathbf{P}}_{n,g}^{\nu}) = |\hat{\mathcal{P}}_{n,g}|$ then {Generation is decodable}
 2:     $\xi_{n,g}^{f} = |\hat{\mathcal{P}}_{n,g}|$
 3: else
 4:     $\xi_{n,g}^{f} = \texttt{rank}(\hat{\mathbf{P}}_{n,g}^{\nu}) - \sigma_{n,g}^{f}$
 5: end if
 6: if $\xi_{n,g}^{f} > 0$ then
 7:     $\hat{\mathbf{p}}_{n,g} \leftarrow \sum_{j=1}^{|\hat{\mathbf{P}}_{n,g}^{\nu}|} a_j \cdot \hat{\mathbf{p}}_{n,g}^{(j)}$
 8:     Send Data packet $\hat{p}_{n,g}$ over face $f$
 9: else
10:     $\rho \leftarrow$ highest rank in $t_{n,g}^{f(in)}$
11:     $e\_time \leftarrow \texttt{expire}(\hat{\imath}_{n,g})$
12:     Insert $\{\rho + 1, e\_time\}$ into $t_{n,g}^{f(in)}$
13:     if $t_{n,g}^{(out)} \leq |t_{n,g}^{f(in)}|$ then
14:         $\texttt{send\_interest}(\hat{\imath}_{n,g})$
15:         $f' \leftarrow$ the face over which $\hat{\imath}_{n,g}$ was sent
16:         Update $t_{n,g}^{f'(out)}$
17:     end if
18: end if
```

Algorithm 2 Data packet processing in the NetCodNDN forwarder

```
Require: $\hat{p}_{n,g}$
1:  if $t_{n,g} = \varnothing$ then {Unsolicited}
2:      Discard $\hat{p}_{n,g}$
3:  else
4:      if $\texttt{rank}(\hat{\mathbf{P}}^{\nu}_{n,g} \cup \hat{p}_{n,g}) > \texttt{rank}(\hat{\mathbf{P}}^{\nu}_{n,g})$ then
5:          Insert $\hat{p}_{n,g}$ into $\hat{\mathbf{P}}^{\nu}_{n,g}$
6:          $\rho \leftarrow \texttt{rank}(\hat{\mathcal{P}}^{\nu}_{n,g})$
7:          for all $f \in \mathcal{F}^{\nu}$ do
8:              if $t^{f,\rho(in))}_{n,g}$ exists then
9:                  $\hat{p}^{*}_{n,g} = \sum_{j=1}^{|\hat{\mathcal{P}}^{\nu}_{n,g}|} a_j \cdot \hat{\mathbf{p}}^{(j)}_{n,g}$
10:                 Send the Data packet $\hat{p}^{*}_{n,g}$ over the face $f$
11:                 $\sigma^{f}_{n,g} \leftarrow \sigma^{f}_{n,g} + 1$
12:                 Remove $t^{f,\rho(in))}_{n,g}$
13:             end if
14:         end for
15:     else
16:         Discard $\hat{p}_{n,g}$
17:     end if
18: end if
```

the previously forwarded Interest is expected to bring the requested Data packet. This is because the NDN forwarder also considers that every forwarded Interest will bring the requested Data packet before its expiration. In this case, we can set $\epsilon_{n,g} = t^{(out)}_{n,g}$, where $t^{(out)}_{n,g}$ is the total number of Interests with name prefix (n, g) that have been forwarded by the node ν over all its faces. Thus, the node forwards the Interest $\hat{\imath}_{n,g}$ if $\epsilon_{n,g} \leq |t^{f(in)}_{n,g}|$, meaning that $t^{(out)}_{n,g} \leq |t^{f(in)}_{n,g}|$.

- *Waiting for a new network coded Data packet* — If $\epsilon_{n,g} > |t^{f(in)}_{n,g}|$, the node ν does not forwards the Interest $\hat{\imath}_{n,g}$ and keeps waiting, as it expects to receive enough network coded Data packets to satisfy all the pending Interests, including the received Interest.

4) Data Packet Processing: When a node ν receives a network coded Data packet $\hat{p}_{n,g}$ over the face f, it first determines whether this Data packet was expected or if it was unsolicited. The node ν accomplishes this by looking at its PIT. If no entry for the name prefix (n, g) exists in its PIT, the node considers the Data packet unsolicited and it is not further transmitted. Otherwise, if the Data packet was expected, the node ν determines if the Data packet $\hat{p}_{n,g}$ is innovative. The Data packet $\hat{p}_{n,g}$ is innovative for the node ν if it is linearly independent with respect to all the Data packets in the CS of the node ν, $\hat{\mathcal{P}}^{\nu}_{n,g}$, *i.e.*, if it increases the rank of $\hat{\mathbf{P}}^{\nu}_{n,g}$. If the Data packet is non-innovative, it is discarded by the node ν. If the Data packet $\hat{p}_{n,g}$ is innovative, the node ν inserts it into its CS. Then, the node ν generates a new network coded Data packet $\hat{p}^{*}_{n,g} = \sum_{j=1}^{|\hat{\mathcal{P}}^{\nu}_{n,g}|} a_j \cdot \hat{\mathbf{p}}^{(j)}_{n,g}$ and sends it over every face f for which $t^{f,\rho(in))}_{n,g}$ exists, *i.e.*, every face that has a pending Interest to be satisfied when the rank of $\hat{\mathcal{P}}^{\nu}_{n,g}$ is ρ. This procedure is outlined in Algorithm 2.

VI. Evaluation

In this section, we evaluate the performance of our proposed adaptive video delivery architecture based on the NetCodNDN forwarder (NetCodNDN-DASH) and compare it with an NDN variant without network coding capabilities (NDN-DASH). First, we describe the implementation of our architecture, the network topology used in the experiments, and the evaluation setup. Then, we evaluate the performance of our proposed video delivery architecture.

A. Implementation

We implemented our proposed adaptive video delivery architecture by extending both the named data layer to enable the NetCodNDN forwarder at every node, and the application layer to enable adaptive video streaming at the clients and the sources.

- *NetCodNDN forwarder* — The NetCodNDN forwarder is implemented by integrating the changes to the NDN architecture described in Section V-C into the NDN Forwarding Daemon (NFD) codebase [27]. We have modified two main modules of the NFD code to implement NetCodNDN. First, we have modified the *Tables* module, where two new tables were implemented: the modified CS (named in the code as Network Coding Matrix Table, or NCMT) and PIT (named in the code as Network Coding Forwarding Table, or NCFT), as described in Sections V-C1 and V-C2, respectively. In the modified Tables module, both the original and the modified versions of the CS and PIT coexist. The original version is used to process NDN Interests and Data packets, while the modified version is used in the processing of network coding enabled Interests and Data packets. We have also modified the *Forwarding* module, in order to add a new set of methods to process network coding enabled Interests and Data packets. The modified Forwarding module uses the original NDN procedures [23] to process traditional Interests and Data packets. Further, it uses the new NetCodNDN procedures, described in section V-C, to process the network coding enabled Interests and Data packets. We have used the Kodo C++ library [28] to enable network coding operations in the NetCodNDN forwarder.
- *Adaptive video applications* — To implement the sources and clients that enable adaptive video streaming with network coding, described in Sections V-A and V-B, respectively, we modified the Adaptive Multimedia Streaming with ndnSIM (AMuSt) codebase [29]. The AMuSt framework provides a set of applications for producing and consuming adaptive video, based on the DASH standard [2], but replacing HTTP with NDN. The DASH functionality is provided by the libdash library [30], an open-source library that provides an interface to the DASH standard. Currently, libdash is the official reference software of the DASH standard. We implemented a new set of applications in the AMuSt framework that use the Kodo C++ library [28] to enable network coding on both the sources and the clients.

Finally, we have installed the implementations of the NetCodNDN forwarder and the adaptive video applications into ndnSIM [10] nodes, which is an NDN simulator based on the NS-3 network simulator [31]. This simulator is used to generate the network nodes, *i.e.*, sources, routers and clients, and connect them with point-to-point links.

B. Network Topology

We evaluate our proposed video delivery architecture in a layered topology, as presented in Fig. 5. The design of this topology is inspired by Netflix's OpenConnect Content Delivery Network (CDN) [12], and it is composed of four layers. The first layer of our topology contains a source that is able to provide all the video segments that the clients might request. The source can be considered as a server that stores all the video segments or as a connection to a CDN that is able to provide the video segments from any of its servers. The second layer of our topology contains a set of routers that represent Internet exchange points (IXP). In our evaluation, we consider 10 IXP nodes, each one connected directly to the source. The third layer of the topology is another set of routers that represent Internet service provider (ISP) nodes. Each ISP node is connected to two IXP nodes. Moreover, the ISP nodes are clustered into 16 groups that represent the European countries served by Netflix [12]. Netflix deploys video delivery servers in certain IXPs and ISPs. These servers store the complete or a fraction of the Netflix video catalog. In our topology, the IXP and ISP nodes have content stores that can cache all the incoming Data packets. When cache space is limited, our architecture needs a cache replacement strategy to decide the cached content. This is, however, out of the scope of this paper which aims to study the behavior of the proposed adaptive video streaming protocol. The study of such strategies is one of our future research investigations. Each country (*i.e.*, cluster) has between 4 and 10 ISP nodes. Finally, the fourth layer of the topology consists of a set of clients, each of them is connected to two ISP nodes that belong to the same country as the client. Each country has between 10 and 20 clients. We choose to connect each client with two ISP nodes to evaluate multi-path video delivery, considering that nowadays most end-user devices come with multiple interfaces, *e.g.* LTE and WiFi. The bandwidth of the links connecting the sources with the IXP nodes, as well as the IXP nodes with the ISP nodes, is set to 50Mbps. The bandwidth of the links connecting the clients to the ISP nodes is based on the one reported on the Netflix ISP Speed Index [13]. In detail, the bandwidth of each link is randomly selected from a normal distribution, with mean similar to the average ISP speed reported on the Netflix ISP Speed Index, and standard deviation 1.5. It is worth noting that the standard deviation reported on the Netflix ISP Speed Index tends to be much lower than 1.5, but we choose this value in order to allow more client diversity. The distribution of the total bandwidth of the clients used in the simulations, considering the two interfaces on the clients, is shown in Fig. 6.

C. Evaluation Setup

We consider that the end-users are interested in a video x that is available in three different representations, $\mathcal{Y}^x = \{480p,\ 720p,\ 1080p\}$ with bitrates $\{1750kbps,\ 3000kbps,\ 5800kbps\}$, respectively, that are similar to the ones that Netflix has been using in the past [11]. Each representation is divided into a set of 30 segments, each of the segments having a duration of 2 seconds. These segments are further divided into generations and Data packets, as presented in Section V-A1. When network coding is enabled, the operations are performed in a finite field of size 2^8. To evaluate our architecture, we consider a scenario where all the clients send Interests during the same interval of time, with starting time variations that are smaller than 100ms. This allows us to demonstrate the Interest aggregation capabilities of the NetCodNDN forwarder.

D. Evaluation Results

We start by evaluating the cache hit rate at the nodes of the ISP and IXP layers. In Fig. 7, we can see that at the ISP layer, the cache hit rate is constantly higher for NetCodNDN-DASH. Between segments 20 and 30, the cache hit rate increases from around 35%, for NDN-DASH to around 58% for NetCodNDN-DASH, *i.e.*, more than 20% higher. The reason for the lower cache hit rate of the traditional NDN architecture is that, since the clients are distributing the set of Interests that request a particular Data packet over both of their faces, they need to coordinate the face over which each Interest is sent, so that they are aggregated at a router closer to the clients. However, due to the high granularity of the content (each Data packet is unique and can satisfy only the Interest with the specific matching name), such coordination is not possible for each Data packet, as it requires centralized control and does not scale with the size of the network and the length of the video sequence. On the contrary, the need for coordination is eliminated with network coding, since clients do not send Interests for a particular Data packet, but for any network coded Data packet. Thus, an Interest can be satisfied with any innovative Data packet available in the CS of a node. These results verify our initial motivation for using network coding to enable a more efficient Interest Data packet aggregation and improve the use of the available network resources.

At the IXP layer, both the traditional and the network coded architectures show the same cache hit rate, which is around 85%, as illustrated in Fig. 8. The cache hit rate is high at the IXP layer because the 10 nodes that belong to this layer are receiving Interests from more than 200 clients, meaning that the probability of aggregating Interests at this layer is high.

This increased cache hit rate of the network coding architecture has two major performance consequences: *(i)* the number of Interests that reach the source is reduced and, *(ii)* the data bitrate seen by the client increases, since the Data packets are found closer to them.

First, let us investigate the impact of using network coding on the load of the source. In Fig. 9, we can see that there is a decrease in the total number of bytes that are provided

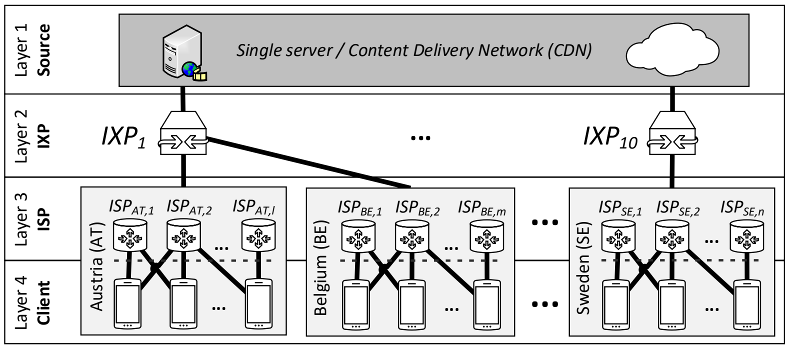

Fig. 5. Layered topology used in our evaluation. The first layer is the source, that can be a single server or a connection to a CDN, for example. The second and third layers are routers that represent IXP and ISP nodes, respectively, and that connect the sources with the clients. Finally, the fourth layer contains the clients that request the DASH segments.

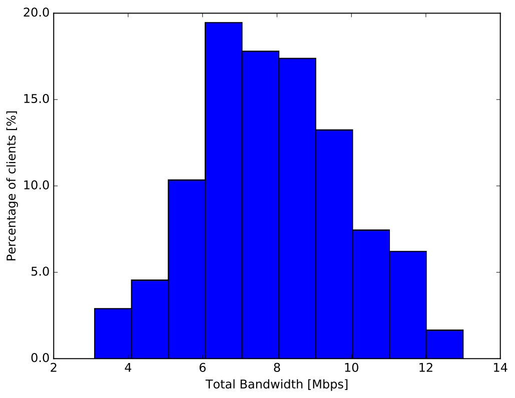

Fig. 6. Bandwidth distribution on the clients.

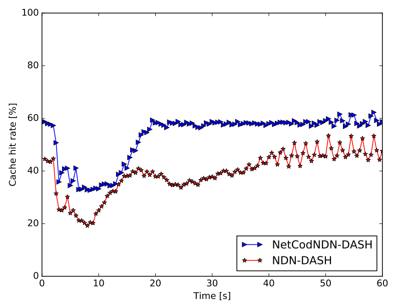

Fig. 7. Cache hit rate at the ISP layer.

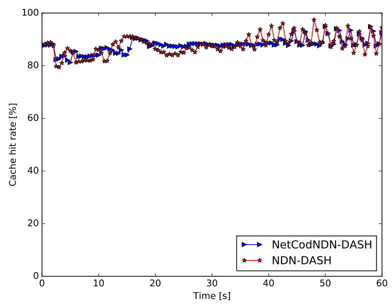

Fig. 8. Cache hit rate at the IXP layer.

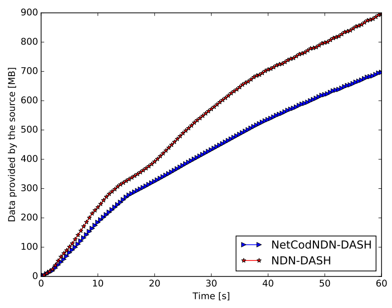

Fig. 9. Total volume of data delivered by the source.

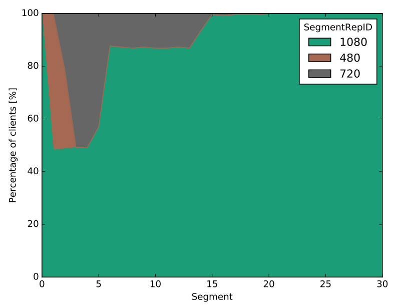

Fig. 10. Quality requested by the NetCodNDN-DASH clients.

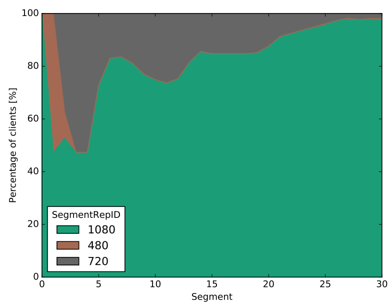

Fig. 11. Quality requested by the NDN-DASH clients.

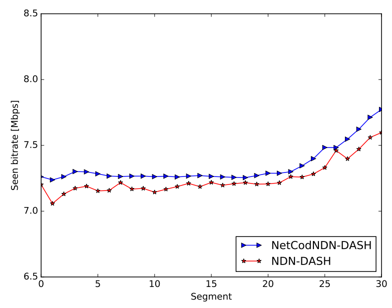

Fig. 12. Bitrate seen by the clients.

by the source, from 895MB for NDN-DASH to 696MB for NetCodNDN-DASH. This means that the use of network coding decreases the load on the source by 22%.

We now examine the benefits that network coding brings to the clients in terms of perceived quality. The percentage of clients that decide to request each DASH representation for each segment can be seen in Figs. 10 and 11, for NetCodNDN-DASH and NDN-DASH, respectively. Even if most of the clients are able to request the highest quality available ($1080p$), we can see that when NetCodNDN-DASH is used, the clients reach this representation faster than when NDN-DASH is used. The reason for this is that since more Data packets are served from closer caches, the bitrate seen by the clients is higher with NetCodNDN-DASH, as can be seen in Fig. 12. This increases the speed at which the DASH adaptation algorithm in the clients upgrade the DASH representation that is requested, meaning that the clients are able to receive the best quality available earlier.

It is worth mentioning that the NetCodNDN-DASH clients enjoy an increased bitrate compared to NDN DASH clients and hence improved quality despite the fact that the implementation of our NetCodNDN client introduces a small delay between the requests of two consecutive generations. Specifically, the client needs to wait until a generation is decoded before requesting the next one, meaning that no Data packets are flowing to the client in the time interval between the arrivals of the last Data packet of generation g and the first Data packet of generation $g+1$. NDN-DASH does not suffer from this problem as it does not consider generations. The bitrate achieved by our architecture can be further improved by a more advanced client implementation, that would allow the clients to start requesting the next generation before decoding the current one. We have successfully deployed such approaches in [32], [33] for video streaming in host-centric networks deploying generation-based network coding and will investigate them in the NetCodNDN client implementation in our future work.

VII. Conclusions

In this paper, we have presented an adaptive video streaming architecture based on DASH for network coding enabled NDN, based on DASH. Our architecture takes advantage of multi-path communication and uses network coding to eliminate the need for coordination between the nodes. This improves the quality of the delivered video, reduces the resource utilization at the sources and improves the resiliency to Data packet erasures. We implemented our architecture by modifying the original NDN codebase, to enable network coding operations at the sources, routers, and clients. We evaluated our architecture using a network topology similar to the one used by video content providers. We have observed large performance gains in terms of the load on the source, as well as an increased cache hit rate. We can conclude that our approach brings significant gains to video content providers, by reducing the traffic load on the servers and improving the use of network resources. This will, in turn, have an impact on the expenses for the video content providers, leading to reduced cost for the video content consumers. Moreover, network coding also improves the speed at which the clients obtain the desired DASH representation. Our future research includes the investigation of optimal caching algorithms for video streaming in network coding enabled NDN.

References

[1] "Cisco Visual Networking Index," Cisco Systems Inc., White Paper, June 2016.

[2] ISO/IEC JTC 1/SC 29, "ISO/IEC 23009-1:2014: Information technology – Dynamic adaptive streaming over HTTP (DASH) – Part 1: Media presentation description and segment formats," May 2014.

[3] C. Westphal *et al.*, "Adaptive Video Streaming over Information-Centric Networking (ICN)," Internet Requests for Comments, RFC Editor, RFC 7933, Aug. 2016. [Online]. Available: http://www.rfc-editor.org/rfc/rfc7933.txt

[4] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking Named Content," in *Proc. of ACM CoNEXT'09*, Rome, Italy, Dec. 2009.

[5] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, k. claffy, P. Crowley, C. Papadopoulos, L. Wang, and B. Zhang, "Named Data Networking," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 66–73, Jul. 2014.

[6] S. Lederer, C. Mueller, B. Rainer, C. Timmerer, and H. Hellwagner, "An experimental analysis of Dynamic Adaptive Streaming over HTTP in Content Centric Networks," in *Proc. of IEEE ICME'13*, San Jose, USA, Jul. 2013.

[7] Y. Wu, P. Chou, and K. Jain, "A Comparison of Network Coding and Tree Packing," in *Proc. of IEEE ISIT'04*, Chicago, USA, Jun. 2004, pp. 143–.

[8] R. Ahlswede, N. Cai, S.-Y. Li, and R. Yeung, "Network Information Flow," *IEEE Trans. Information Theory*, vol. 46, no. 4, pp. 1204–1216, Jul. 2000.

[9] J. Saltarin, E. Bourtsoulatze, N. Thomos, and T. Braun, "NetCodCCN: A network coding approach for content-centric networks," in *Proc. of IEEE INFOCOM'16*, San Francisco, USA, Apr. 2016.

[10] S. Mastorakis, A. Afanasyev, I. Moiseenko, and L. Zhang, "ndnSIM 2: An updated NDN simulator for NS-3," NDN, Technical Report NDN-0028, Revision 2, November 2016.

[11] A. Aaron, Z. Li, M. Manohara, J. D. Cock, and D. Ronca, "The Netflix Tech Blog: Per-Title Encode Optimization," http://techblog.netflix.com/2015/12/per-title-encode-optimization.html, December 2015.

[12] T. Böttger, F. Cuadrado, G. Tyson, I. Castro, and S. Uhlig, "Open Connect Everywhere: A Glimpse at the Internet Ecosystem through the Lens of the Netflix CDN," *CoRR*, vol. abs/1606.05519, 2016. [Online]. Available: http://arxiv.org/abs/1606.05519

[13] Netflix Inc., "The Netflix ISP Speed Index," https://ispspeedindex.netflix.com/, December 2016.

[14] A. Detti, M. Pomposini, N. Blefari-Melazzi, S. Salsano, and A. Bragagnini, "Offloading cellular networks with Information-Centric Networking: The case of video streaming," in *Proc. of IEEE WoWMoM'12*, Macao, China, Jun. 2012.

[15] L. Wang, I. Moiseenko, and D. Wang, "When video streaming meets named data networking: A case study," in *Proc. of IEEE HPCC/SmartCity/DSS'16*, Sydney, Australia, Dec. 2016.

[16] G. Rossini and D. Rossi, "Evaluating CCN multi-path interest forwarding strategies," *Computer Communications*, vol. 36, no. 7, pp. 771 – 778, 2013.

[17] K. M. Schneider and U. R. Krieger, "Beyond network selection: Exploiting access network heterogeneity with named data networking," in *Proc. of ACM ICN'15*, San Francisco, USA, Sept. 2015, pp. 137–146.

[18] M.-J. Montpetit, C. Westphal, and D. Trossen, "Network Coding Meets Information-Centric Networking: An Architectural Case for Information Dispersion Through Native Network Coding," in *Proc. of the 1st ACM NoM Workshop*, Hilton Head, USA, Jun. 2012.

[19] J. Sundararajan, D. Shah, M. Medard, S. Jakubczak, M. Mitzenmacher, and J. Barros, "Network coding meets tcp: Theory and implementation," *Proc. of the IEEE*, vol. 99, Mar. 2011.

[20] Q. Wu, Z. Li, and G. Xie, "CodingCache: Multipath-Aware CCN Cache with Network Coding," in *Proc. of the 3rd ACM ICN Workshop*, Hong Kong, China, Aug. 2013.

[21] J. Llorca, A. Tulino, K. Guan, and D. Kilper, "Network-Coded Caching-Aided Multicast for Efficient Content Delivery," in *Proc. of IEEE ICC'13*, Budapest, Hungary, Jun. 2013.

[22] K. Matsuzono, H. Asaeda, and T. Turletti, "Low Latency Low Loss Streaming using In-Network Coding and Caching," in *Proc. of IEEE INFOCOM'17*, Atlanta, United States, May 2017.

[23] A. Afanasyev *et al.*, "NDN, Technical Report NDN-0021.Revision 7," https://named-data.net/publications/TECHREPORTs/ndn-0021-7-nfd-developer-guide/, October 2016.

[24] A. Ford *et al.*, "TCP Extensions for Multipath Operation with Multiple Addresses," Internet Requests for Comments, RFC Editor, RFC 6824, Jan. 2013. [Online]. Available: http://www.rfc-editor.org/rfc/rfc6824.txt

[25] T. Ho, M. Medard, J. Shi, M. Effros, and D. R. Karger, "On Randomized Network Coding," in *Proc. of 41st Annual Allerton Conference on Communication, Control, and Computing*, 2003.

[26] P. Chou and Y. Wu, "Network Coding for the Internet and Wireless Networks," *IEEE Signal Processing Magazine*, vol. 24, no. 5, pp. 77–85, Sept. 2007.

[27] Named Data Networking (NDN) Project, "Named Data Networking Forwarding Daemon," https://github.com/named-data/NFD.

[28] M. Pedersen, J. Heide, and F. H. P. Fitzek, "Kodo: An Open and Research Oriented Network Coding Library," in *Proc. of IFIP NETWORKING Workshops'13*, New York, USA, May 2011.

[29] C. Kreuzberger, D. Posch, and H. Hellwagner, "AMuSt Framework - Adaptive Multimedia Streaming Simulation Framework for ns-3 and ndnSIM," https://github.com/ChristianKreuzberger/amust-simulator, 2016.

[30] C. Mueller, S. Lederer, J. Poecher, and C. Timmerer, "Demo paper: Libdash - An open source software library for the MPEG-DASH standard," in *Proc. of IEEE ICMEW'13*, San Jose, USA, Jul. 2013.

[31] "The network simulator - ns3," http://www.nsnam.org/.

[32] E. Bourtsoulatze, N. Thomos, and P. Frossard, "Distributed Rate Allocation in Inter-Session Network Coding," *IEEE Transactions on Multimedia*, vol. 16, no. 6, pp. 1752–1765, Oct. 2014.

[33] N. Thomos, E. Kurdoglu, P. Frossard, and M. van der Schaar, "Adaptive Prioritized Random Linear Coding and Scheduling for Layered Data Delivery From Multiple Servers," *IEEE Transactions on Multimedia*, vol. 17, no. 6, pp. 893–906, Jun. 2015.