# Adaptive Video Streaming with Network Coding Enabled Named Data Networking

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

***Abstract*—The fast and huge increase of Internet traffic 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 multi-path 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, state-of-the-art works do not consider the multipath capabilities of NDN and the potential improvements that multipath communication brings, such as increased throughput and reliability, that are fundamental for video streaming systems. 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 our architecture leads to reduced data traffic load on the sources, increased cache-hit rate at the in-network caches and faster adaptation of the requested video quality by the clients. The performance gains are verified through simulations 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. Thus, the clients are able to decide the appropriate bitrate and resolution of the requested video, among the multiple options offered by the video sources. 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 rendered DASH an attractive option for video streaming in future Internet architectures and, in particular, for Information-Centric Networking (ICN) [3] architectures such as Content-Centric Networking (CCN) [4] and Named Data Networking (NDN) [5]. In NDN, 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. Hence, 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. This content retrieval mechanism of NDN resembles the video retrieval mechanism of DASH [6]. In particular, NDN and DASH are both client driven and, 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.

Motivated by the similarities of content retrieval mechanisms of NDN and DASH, in this work we propose an adaptive video streaming architecture over NDN that uses network coding [7] to enable optimal multipath video streaming. Video streaming applications require a moderately high and stable amount of bandwidth between the video sources and the clients, to avoid quality fluctuations and provide high Quality of Experience. Thus, using multiple paths to connect the clients to the sources increases the bandwidth seen by the clients, enabling the quality adaptation mechanisms of DASH to converge to better video qualities than with the use of a single communication path. NDN natively supports multipath communication without the need of any additional protocol. The clients can transmit Interests over all their network interfaces (*e.g.*, LTE and Wi-Fi) to retrieve the Data packets that compose the requested content. However, NDN does not provide any mechanisms for coordinating the forwarding strategies at the intermediate nodes. On one hand, without such coordination, Interest packets for the same content may be forwarded on different paths. This results in the delivery of multiple copies of the same content on different paths and, therefore, diminishes the gains of using multipath communication. On the other hand, coordination of the forwarding strategies would require the global knowledge of the network topology, would be computationally complex and would not scale in large and dynamic topologies.

In this paper, we propose an efficient way of using the multiple paths available for video content delivery by enabling network coding within the NDN architecture. Our scheme takes

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.

advantage of multipath communication by sending Interests for network coded Data packets over multiple network interfaces. Network coded Data packets then follow multiple paths (i.e., the reverse paths followed by the Interests) to reach the clients. Since network coded Data packets generated from the same set of original Data packets are equivalent in terms of information content, the clients and the intermediate nodes do not need to coordinate their forwarding decisions. Thus, our scheme does not require complex coordination of the forwarding strategies and exploits in a natural way the gains of multipath communication.

Our system is based on our previous work [8], where we have introduced network coding in CCN. This scheme showed significant gains compared to the original CCN in terms of both the achieved throughput and the experienced delay. These gains were achieved without requiring transmission of the Interests over optimal multicast trees [9], which are difficult to compute in large and dynamic networks. However, despite the throughput and delay gains in our previous work [8], our previous scheme cannot support optimally video streaming applications because it does not take into account the underlying video data structure. To address this issue, we made changes to the data structures of NDN. In particular, we have redesigned the Content Store (CS) and the Pending Interest Table (PIT) and have further modified the Interest and Data processing functions of the network coding enabled NDN so that DASH streaming and efficient CS and PIT handling are supported. Moreover, we propose a new model of a client and a source application that enables DASH communication over network coding enabled NDN. Finally, we have implemented the proposed video streaming architecture in the ndnSIM simulator [10]. We compare the video streaming performance of our architecture to the original 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 exploits more effectively the multipath communication and attains a higher cache hit rate in the router nodes. This translates into lower bandwidth consumption at the sources, 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 [8] with new features that enhance the support for the network coding functionalities;
- a new design of a client and a 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, DASH and network coding is presented in Section III. The motivation behind the use of network coding for NDN adaptive video streaming is provided in Section IV. We describe in detail the proposed architecture for 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 in the content retrieval mechanisms of DASH and NDN have attracted the attention of the research community [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 Wi-Fi. The results show that the users can drastically reduce the amount of data downloaded over the cellular network by exploiting users' cooperation, which may result in cost reductions. In a system integrating CCN and DASH [6], the *version* component of the CCN content naming is used to name the different representations of a DASH segment, and the *segment* component of the CCN content naming is used to divide segments into Data packets that fit into lower level Maximum Transmission Units (MTU). A client requests a segment by sending Interests over its network interfaces. These Interests are satisfied by a set of Data packets containing the requested DASH segment, following the same procedures as in the CCN architecture.

Most of the aforementioned DASH-based video streaming proposals for ICN consider the existence of a single path connecting the sources with the clients [6], [14], [15]. This is sub-optimal, as not all the paths connecting the client with the sources are exploited, hence, limiting the resources that the client can use to adapt the video streaming towards a high quality. 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 video data. It is worth noting that the Internet Research Task Force (IRTF) addresses the adaptation of current video streaming mechanisms to the ICN architecture [3]. It also defines some use cases for video streaming and their requirements, identifying the main issues associated with these streaming mechanisms in ICN.

Existing works for multipath communication in ICN [16], [17] 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 naïve 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 cache a copy of the same Data packet. Schneider *et al.* [17] presented a set of novel Interest forwarding strategies that improve end-user Quality of Experience (QoE) and reduce clients' access cost and power consumption. To accomplish this, the authors propose a new set of interface estimators that 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], *i.e.,* caching

the same content, by applying network coding [7] on the Data packets. In this case, 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, which contains the Data packet availability information of the client, similarly 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 provides 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 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 Coding-Cache, 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 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. 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 coded Data packets, and only request network coded Data packets when they detect Data packet losses. In this case, network coding is only exploited to deal with lost Data packets.

## III. Background

In this section, we introduce the main concepts that enable dynamic adaptive video streaming over named data networking with network coding. 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. Finally, we introduce network coding and describe briefly its operation within the NDN architecture.

TABLE I
Notation

| | | |
|---|---|---|
| $\mathcal{S}$, $\mathcal{R}$, $\mathcal{C}$ | $\triangleq$ | Set of source, router and client nodes, respectively |
| $\mathcal{Q}$ | $\triangleq$ | Set of DASH representations |
| $\mathcal{Z}^y$ | $\triangleq$ | Set of DASH segments that form representation $q$ |
| $\mathcal{P}_n$ | $\triangleq$ | Set of Data packets that form a DASH segment with name $n$ |
| $\mathcal{P}_n^\mu$ | $\triangleq$ | Subset of Data packets with name $n$, stored in node $\mu$ |
| $p_{n,j}$ | $\triangleq$ | The $j$th Data packet in the set $\mathcal{P}_n$ |
| $i_{n,j}$ | $\triangleq$ | An Interest requesting the data packet $p_{n,j}$ |
| $\hat{\mathcal{P}}_n$ | $\triangleq$ | Set of network coded Data packets generated from $\mathcal{P}_n$ |
| $\hat{p}_{n,g}$ | $\triangleq$ | A network coded Data packet belonging to generation $g$ in the set $\mathcal{P}_n$ |
| $\hat{i}_{n,g}$ | $\triangleq$ | An Interest requesting a network coded Data packet $\hat{p}_{n,g}$ |
| $\mathcal{F}_n^\mu$ | $\triangleq$ | Set of faces at node $\mu$ that are configured to forward Interests with name prefix $n$ |

### A. Dynamic Adaptive Streaming over HTTP

As we discussed in Section I, 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 $v$ is encoded with different parameters (*e.g.*, bitrate, resolution, *etc*.), creating a set of *representations* $\mathcal{Q}$. The video data in each representation $q \in \mathcal{Q}$ is divided into a set of *segments* $\mathcal{Z}$. Every segment $z \in \mathcal{Z}$ 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 representations, a file called the Media Presentation Description (MPD) is associated with each video $v$. This file contains information about the available representations $\mathcal{Q}$ in which the video $v$ has been encoded and the segments $\mathcal{Z}$ that compose each representation, among other parameters. A client that is interested in receiving the video $v$ should first request the MPD file associated with this video. Then, after receiving and parsing the MPD file of the video $v$, the client knows what representations are available and what names it should use to request the video segments. Each particular video segment is identified with a name $n \leftarrow \mathtt{v/q/z}$ that is composed of the ID $v$ of the video to which the segment belongs, the representation $q$ in which the segment has been encoded and the segment ID $z$.

### B. DASH over Named Data Networking

In order to describe the operation of DASH in NDN, let us 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 $\mu \in \mathcal{S} \cup \mathcal{C} \cup \mathcal{R}$ is connected with its neighboring nodes through a set of faces $\mathcal{F}^{\mu}$.

Each source $s \in \mathcal{S}$ stores a set of DASH segments that can be requested by the clients. Since the size of the DASH segments is usually larger than the Maximum Transmission Unit (MTU), the DASH segments are divided into smaller Data packets that fit into an MTU. Therefore, we consider that a DASH segment with name $n \leftarrow \mathtt{v/q/z}$ is composed of the set of Data packets $\mathcal{P}_n = \{p_{n,1}, \ldots, p_{n,|\mathcal{P}_n|}\}$. To retrieve a DASH segment composed of the set of Data packets $\mathcal{P}_n$, a client $c \in \mathcal{C}$ should send a set of Interests $\mathcal{I}_n = \{i_{n,1}, \ldots, i_{n,|\mathcal{I}_n|}\}$, 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 $|\mathcal{I}_n|$, 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 $\mathcal{F}_n^c$ of the client $c$ 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)* and a *Pending Interest Table (PIT)*. The CS caches Data packets that pass through the node. The PIT keeps track of the Interests that the node has forwarded and the faces over which these Interests have arrived.

In NDN, when a node receives an Interest, it either: *(i)* adds the information about the received Interest to the PIT and waits for the requested Data packet to arrive, if previously an Interest for the same Data packet had been forwarded; *(ii)* replies to the Interest with a matching Data packet from its CS; or *(iii)* forwards the Interest to its neighboring nodes. This is further explained in the following.

- *Waiting for a new Data packet* — Consider a router $r$ that receives an Interest $i_{n,j}$ over the face $f$. If the router $r$ finds in its PIT an entry that matches the name $(n, j)$, 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 router $r$ 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 router $r$ replies to the Interest $i_{n,j}$ with this Data packet.
- *Forwarding an Interest* — If the CS of the router $r$ does not contain a Data packet matching the name of the Interest $i_{n,j}$, the router $r$ forwards the Interest to one or more of its neighboring nodes, according to its FIB. Moreover, the router $r$ also updates its PIT table to add the information about 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 a PIT entry that matches the name $(n, j)$ is found, 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's Guide [23].

### C. Network coding in NDN

Network coding [7] is 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, when network coding is enabled in NDN, the network coded Data packets contain information from all the Data packets that have been combined to generate them. Differently from the original 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 $n$. Therefore, with some small probability when coding is done in a large finite field, the coded 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 can reply to these Interests with network coded Data packets. The network coded Data packets are generated by combining the set of Data packets that are available in the CS or the repository of the node 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 to generate a new network coded Data packet $\hat{\mathbf{p}}_n$ can be expressed as $\hat{\mathbf{p}}_n = \mathbf{A} \cdot \mathbf{P}_n^{\mu}$, where $\mathbf{A}$ is a matrix of coding coefficients drawn from a finite field, and $\mathbf{P}_n^{\mu}$ is the matrix formed with the set of Data packets $\mathcal{P}_n^{\mu}$. 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 [24]. To decode the original Data packets that compose $\mathcal{P}_n$, a client should collect $|\mathcal{P}_n|$ innovative network coded Data packets $\hat{p}_n$.

## IV. Motivation

Video streaming is a data intensive application, which requires a moderately high and stable amount of bandwidth between the video sources and the clients. Utilizing multiple paths to connect the clients to the sources permits to exploit network bandwidth resources, which remain underutilized with a single path streaming policy. Multipath streaming can, thus, significantly improve the quality of the video received by

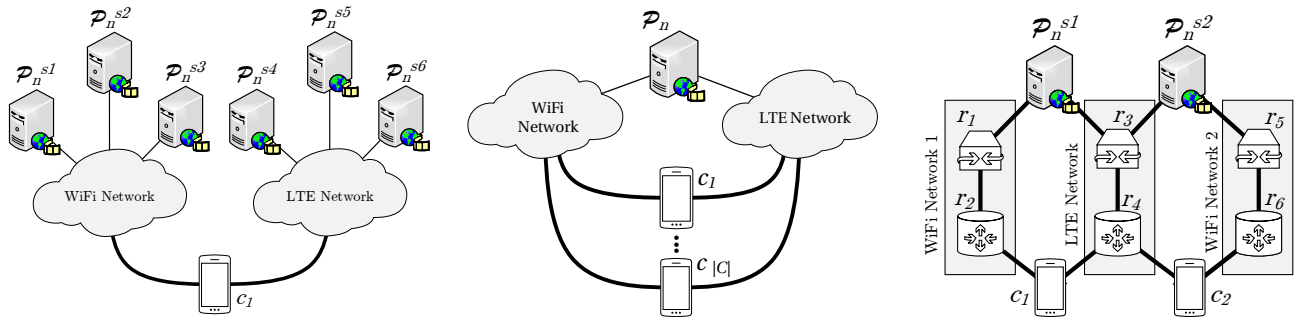



Fig. 1. Devices retrieving Data packets over LTE and Wi-Fi: (a) multi-source unicast; (b) single-source multicast; (c) multi-source multicast (butterfly network).

the clients compared to single path streaming. In addition, techniques that increase the network throughput, especially in the presence of network bottlenecks, can further enhance the clients' experience by enabling the delivery of better video representations that require higher throughput. Finally, resilience to packet losses is another important aspect of the video streaming systems. Packet losses require Interest retransmissions and result in network congestion and higher delays, which in turn have a direct negative impact on the clients' viewing experience. These three aspects of the NDN-based adaptive video streaming system, namely *(i)* multipath streaming, *(ii)* throughput and *(iii)* resilience to packet losses can be optimized by introducing network coding in the NDN architecture. In the following subsections, we provide the key motivating ideas behind the use of network coding in NDN for adaptive video streaming and illustrate those ideas through motivating examples.

### A. *Efficient multipath streaming*

Nowadays, most client devices, *e.g.*, mobile phones, laptops, *etc*., have two or more network interfaces *e.g.*, LTE, Wi-Fi, Bluetooth, *etc*., where they can receive the demanded video content from. However, in the traditional host-centric networking, support for multipath is not extended. Recent efforts to support multipath communications on TCP (MP-TCP) [25] have been developed by the IETF. The drawback of these proposals is that they require end-to-end connections to be established for each host, which makes the use of in-network caching and the dynamic selection of the sources difficult. In comparison, NDN provides natural support for multipath content retrieval, without requiring end-to-end connections. This is achieved by allowing clients to distribute all the Interests needed to retrieve a video segment over all the available interfaces, without knowing a priori which source or in-network cache will provide the content. However, despite having the necessary components for enabling multipath communication, the original NDN architecture still lacks appropriate mechanisms for the optimal use of the multiple paths available for video content retrieval. Optimizing multipath video content retrieval in the original NDN implies coordinating the forwarding of Interests, so that *(i)* the Interests for a specific Data packet are forwarded towards the source of this Data packet in multi-source systems, and *(ii)* the Interests for the same Data packet are forwarded on the same paths, such that Interest aggregation and Data packet caching is optimized. That would require devising explicit coordination mechanisms and complex forwarding strategies.

The need for sophisticated Interest forwarding coordination mechanisms can be resolved by enabling network coding in the NDN architecture. When network coding is enabled in NDN, the network coded Data packets contain information from all the Data packets that have been combined to generate them. Thus, all network coded Data packets with a specific name prefix are equivalent in terms of contained information. This reduces the granularity of the information source and subsequently of the data requests. As a result, clients do not need to request specific Data packets, but rather network coded Data packets. Therefore, the nodes do not need to coordinate the faces where they forward Interests, which enables efficient multipath communication without explicit coordination mechanisms and enhances the network bandwidth utilization.

**Motivating examples.** We illustrate the improvement in the use of multiple paths by introducing network coding to NDN for adaptive video streaming through three characteristic scenarios as depicted in Fig. 1.

- *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 of 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 distributed across multiple video servers. In this case, the client and the routers need to select properly the face over which they 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 wastes resources. 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 than those of original NDN. Specifically, only one entry for the name prefix $n$ is needed in network coding enabled NDN, while in original NDN a distinct entry is required for every Data packet with name $(n, j)$. 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* — Let us now examine 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 [9]. To accomplish this in original NDN, each node should know where to forward each Interest $i_{n,j}$ such that all the Interests $i_{n,j}$ that are 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 [9]. In the illustrative example shown in Fig. 1b, clients are connected to the source through both LTE and Wi-Fi 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 decides to send the Interest $i_{n,j}$ over the Wi-Fi interface, the Data packet $p_{n,j}$ will also be sent from the source to the Wi-Fi network, wasting valuable network resources. 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. This is due to the fact that the Interests are for network coded Data packets. Thus, they can be aggregated at any node, which leads to more efficient network utilization.
- *Multi-source multicast* — Another problematic scenario is when multiple clients are interested in a DASH segment composed of the set of Data packets $\mathcal{P}_n$ that are distributed across multiple sources. In this scenario, the multipath adaptive video streaming in original NDN suffers from the shortcomings of both the multi-source unicast and the single-source multicast scenarios that we previously discussed. To illustrate this, let us consider the network in Fig. 1c. In this network, two clients $c_1$ and $c_2$ need to coordinate where to send each Interest, such that the router $r_4$ is able to aggregate the Interests for the same Data packet. Moreover, when each of the sources has a disjoint set of Data packets, *i.e.*, $\mathcal{P}_n^{s_1} \cap \mathcal{P}_n^{s_2} = \emptyset$, 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. With network coding enabled, no coordination is needed at the clients nor at the routers. This is because all the Interests can be satisfied by any network coded Data packet.

### B. Throughput gains

Apart from the gains coming from the more optimal use of the multipath communication capability, enabling network coding in NDN can also improve the throughput, in particular when bottlenecks are present in the network. This property of network coding has been demonstrated in traditional host-centric streaming systems [26], [27]. Here, we illustrate the throughput improvements also in the context of the NDN architecture. Let us consider the widely known butterfly network topology shown in Fig. 1c. We consider that the clients $c_1$ and $c_2$ are interested in a DASH segment composed of the Data packets in $\mathcal{P}_n = \{p_{n,1}, p_{n,2}\}$, which are distributed across both sources. Without loss of generality, let us assume 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.*, $\mathcal{P}_n^{s_2} = \{p_{n,2}\}$). 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, when the Data packets are requested in a network coding enabled NDN architecture, 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 $c_2$.

### C. Resilience to packet losses

Finally, we illustrate how network coding improves the resiliency to Data/Interest packet losses. Again, this property has been widely studied in state-of-the-art host centric streaming scenarios [28]. Similarly to traditional streaming architectures, in adaptive video streaming over NDN, network coding can deal efficiently with packet losses eliminating the need for explicit packet retransmissions. To illustrate this, let us consider a NDN network with Interests or Data packets losses. Let us also consider that a client is interested in a DASH segment composed of the 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 re-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 $\epsilon$ depends on the packet loss rate

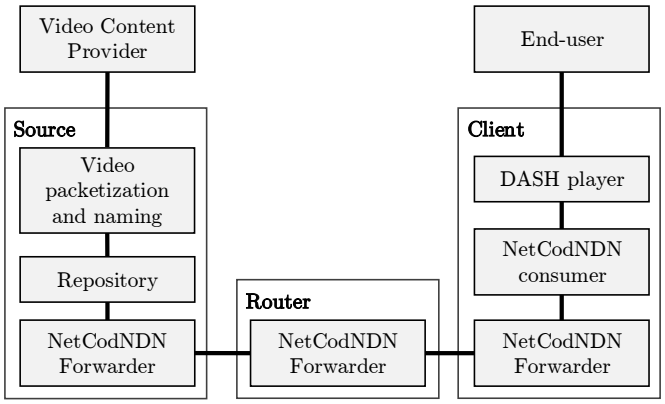



Fig. 2. Proposed architecture for adaptive video streaming over network coding enabled NDN.

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 architecture for dynamic adaptive streaming (DASH) over network coding enabled NDN. Our system is based on our previously proposed NetCodCCN architecture [8], which here is advanced to support DASH video streaming. The main notation used throughout the following section is summarized in Table I.

We consider a set of videos $\mathcal{V}$ that are made available by a *video content provider* to a set of *end-users*. The videos $\mathcal{V}$ are encoded into a set of different DASH representations $\mathcal{Q}$. Each representation $y \in \mathcal{Q}$ is divided into a set of segments $\mathcal{Z}$, as described in Section III-A. Each segment $z \in \mathcal{Z}$ has a name $n \leftarrow \mathtt{x/y/z}$ that identifies it uniquely. This name is composed of the video stream ID $v$, the representation ID $q$, and the segment ID $z$. Then, the segments are stored at the source's repositories. An end-user expresses its interest in watching a video $v \in \mathcal{V}$ to its client, which then requests the video $v$ from the network. In Fig. 2, we show a simple network example composed of a source, a router, and a client, 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. First, the source receives video content, in the form of DASH segments, from a video content provider. Then, each of the DASH segments is divided into Data packets, and a name is associated with each Data packet. Finally, the Data packets are loaded into the repository. The repository replies with network coded versions of these Data packets whenever it receives an Interest that matches the name prefix. In the following sections, we present further details about this process.

*1) Video data packetization and naming:* Each segment $z \in \mathcal{Z}$ 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 an MTU. As presented in Section III-C, 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 [29], each Data packet $\mathbf{p}_{n,j}$ is prepended with an encoding vector 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 the encoding vector associated with the Data packet $\mathbf{p}_{n,j}$ is set to be the $j$th unit vector, which has value 1 in the $j$th position and 0 otherwise.

Prepending encoding vectors to the Data packets introduces a communication overhead that consumes network resources, especially when the segment $z$ is divided into a large number of Data packets. To limit this overhead, we adopt the concept of *generations* [29], where the original set of Data packets that compose $\mathcal{P}_n$ is divided into smaller groups of Data packets, which are known as generations. The coding operations are restricted only among 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 appended to the name of the Data packet, such that $(n, g) \leftarrow \mathtt{v/q/z/g}$. 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 $\mathcal{P}^s_{n,g}$ 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 also 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; 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 storage. Even though the implementation of repositories and content stores might differ, in this paper we consider that their functionality is the same and 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, upon receiving a video request from an end-user, generates the Interests needed to collect its Data packets and assembles

the DASH segments that will be delivered to the end-user. Our model of a client 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 sections.

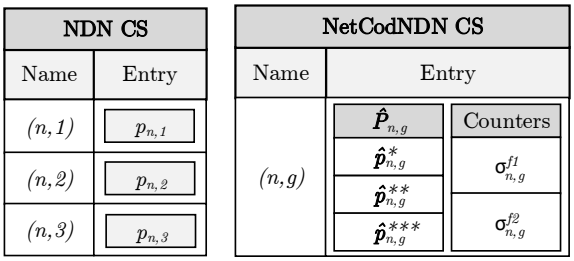
*1) DASH player:* The DASH player is the most direct interface between the end-user and the video communication system. When a DASH player receives a request from an end-user to retrieve a video $v$, it first requests the MPD file. This file is typically of a small size and needs to be communicated only 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 $q \in \mathcal{Q}$ is the optimal every time that it has to request a new segment $z \in \mathcal{Z}$. 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 bitrate measured in the reception of the video segments.

*2) NetCodNDN consumer:* Whenever a NetCodNDN consumer receives requests for a DASH segment, 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 Data packets of the last generation.

The NetCodNDN consumer keeps track of the received innovative Data packets $\hat{\mathbf{p}}_{n,g}$ in a matrix $\hat{\mathbf{P}}^c_{n,g}$, where $c$ stands for the consumer, so that the original set of Data packets can be retrieved by performing Gaussian elimination when the matrix $\hat{\mathbf{P}}^c_{n,g}$ 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, *(iii)* applying network coding operations on the Data packets before forwarding them, and *(iv)* keeping a cache with the received Data packets in order to reply to future Interests. 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 the routers when they receive Interests or Data packets. Note that in our previous work [8], we presented a network coding based content retrieval scheme that utilized the original CS and introduced a few modifications to the PIT, keeping the model of the proposed architecture 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. Moreover, other than the redesigned CS and PIT, the NetCodNDN forwarder still has the traditional CS and PIT to process non-network coded Interests and Data packets.



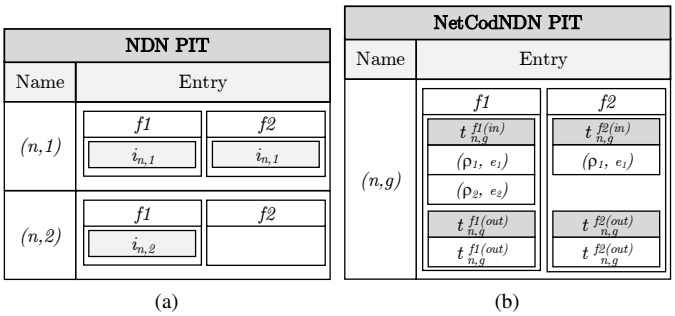

(a) (b)

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 $\sigma^f_{n,g}$.

It is worth noting that not every NetCodNDN router should keep a CS with the Data packets that it receives. As demonstrated by Fayazbakhsh *et al.* [30] and Sun *et al.* [31], the data delivery performance of an NDN network in which every router has a CS is not much better than that of an NDN network where only the edge routers have a CS, taking into consideration the computing power and storage capacity that a CS requires. It is also worth noting that not all the NetCodNDN routers need to apply network coding operations to the received Data packets. In fact, when a NetCodNDN router does not have a CS, it will not be able to apply network coding operations on the Data packets before forwarding them, since there will be no other Data packets cached in the router. If the routers that are able to apply network coding are chosen carefully, the benefits that network coding brings to the video delivery remain high, while the computing power and storage capacity of certain routers can be drastically reduced, as has been demonstrated by Cleju *et al.* [26].

*1) Content Store:* As presented in Section III-B, an NDN router $r$ maintains a cache with a set of Data packets $\mathcal{P}^r_n$ 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 $p_{n,j} \in \mathcal{P}^r_n$ is stored as an entry in the CS. When a router $r$ 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 can 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 router $r$ 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 router $r$ generates a network coded Data packet $\hat{p}_{n,g}$ by randomly combining the Data packets




(a) (b)

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.

$\hat{\mathbf{P}}^r_{n,g}$ in its CS. Thus, $\hat{p}_{n,g} = \sum_{j=1}^{|\hat{\mathbf{P}}^r_{n,g}|} a_j \cdot \hat{\mathbf{p}}^{(j)}_{n,g}$, where $a_j$ is a randomly selected coding coefficient and $\hat{\mathbf{p}}^{(j)}_{n,g}$ in the $j$th Data packet in $\hat{\mathbf{P}}^r_{n,g}$.

In our previous work [8], we used the CS model provided by CCN [4] for both the traditional Interests and Interests for network coded data. 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 coded 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, $\hat{\mathcal{P}}^r_{n,g}$, where all the Data packets belong to the same generation $g$. This set of Data packets is stored as a matrix $\hat{\mathbf{P}}^r_{n,g}$, 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 coded Data packet to only one, which is much more efficient than our previous work [8].

Moreover, each CS entry also stores a counter $\sigma^f_{n,g}$ for each face $f$ of the router $r$. Each counter $\sigma^f_{n,g}$ measures the number of Data packets generated with the content of the matrix $\hat{\mathbf{P}}^{\mathbf{r}}_{n,g}$ 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 router $r$ 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. Therefore, the value of $\sigma^f_{n,g}$ is also decreased by 1 for all the faces, in order to reflect the current state of the CS in terms of the available information.

*2) Pending Interest Table:* As described in section V-C1, when a router $r$ with a NetCodNDN forwarder receives an Interest $\hat{i}_{n,g}$, it uses the set of Data packets $\hat{\mathcal{P}}^r_{n,g}$ 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 router $r$ may need to wait until it receives new Data packets before replying to the Interest. In this case, the router 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 PIT.

To facilitate the new functionalities of the NetCodNDN forwarder, the design of the PIT has to be modified with respect to that of the original 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* $t^{f(in)}_{n,g}$ that keeps track of the Interests that arrived over the face $f$ and that have not been satisfied, and an *out-record* $t^{f(out)}_{n,g}$ that keeps track of the Interests that have been forwarded over the face $f$, and that are still pending.

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

The out-record $t^{f(out)}_{n,g}$ 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 router $r$ over all its faces is computed as $t^{(out)}_{n,g} = \sum_{f \in \mathcal{F}^r_{n,g}} t^{f(out)}_{n,g}$, where $\mathcal{F}^r_{n,g}$ is the set of faces of router $r$ 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 $\hat{i}_{n,g}$ for network coded Data packets, where $n$ is the name of the DASH segment and $g$ is the generation ID. The Interests $\hat{i}_{n,g}$ have a `NetworkCodingAllowed` field which indicates whether the Interest is for a network coded Data packet or for a non-coded Data packet and, thus, determines the Interest processing procedures to be invoked by the NetCodNDN forwarder. If the `NetworkCodingAllowed` field is set to the value `1`, the NetCodNDN Interest processing procedures are invoked. If the `NetworkCodingAllowed` field is not present or set to the value `0`, the Interests are treated following the original NDN procedures [23].

When a router 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 procedure is further explained below and summarized in Algorithm 1,

- *Replying to an Interest* — The router $r$ replies to an Interest $\hat{i}_{n,g}$ 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 router $r$ has high probability to be innovative for the neighbor node connected through the face $f$ on which the Interest arrived. The number of network coded Data packets that can be generated by the router $r$ and that have a high probability to be innovative

**Algorithm 1** Interest processing in the NetCodNDN forwarder

**Require:** $\hat{i}_{n,g}$, $f$, $\hat{\mathcal{P}}^r_{n,g}$
**if** $\texttt{rank}(\hat{\mathbf{P}}^r_{n,g}) = |\hat{\mathcal{P}}_{n,g}|$ **then** {Generation is decodable}
  $\xi^f_{n,g} = |\hat{\mathcal{P}}_{n,g}|$
**else**
  $\xi^f_{n,g} = \texttt{rank}(\hat{\mathbf{P}}^r_{n,g}) - \sigma^f_{n,g}$
**end if**
**if** $\xi^f_{n,g} > 0$ **then**
  $\hat{\mathbf{p}}_{n,g} \leftarrow \sum_{j=1}^{|\hat{\mathbf{P}}^r_{n,g}|} a_j \cdot \hat{\mathbf{p}}^{(j)}_{n,g}$
  Send Data packet $\hat{p}_{n,g}$ over face $f$
**else**
  $\rho \leftarrow$ highest rank in $t^{f(in)}_{n,g}$
  $e \leftarrow \texttt{expire}(\hat{i}_{n,g})$
  Insert $\{\rho+1, e\}$ into $t^{f(in)}_{n,g}$
  **if** $t^{(out)}_{n,g} \leq |t^{f(in)}_{n,g}|$ **then**
    $\texttt{send\_interest}(\hat{i}_{n,g})$
    $f' \leftarrow$ the face over which $\hat{i}_{n,g}$ was sent
    Update $t^{f'(out)}_{n,g}$
  **end if**
**end if**

**Algorithm 2** Data packet processing in the NetCodNDN forwarder

**Require:** $\hat{p}_{n,g}$
**if** $t_{n,g} = \varnothing$ **then** {Unsolicited}
  Discard $\hat{p}_{n,g}$
**else**
  **if** $\texttt{rank}(\hat{\mathbf{P}}^r_{n,g} \cup \hat{p}_{n,g}) > \texttt{rank}(\hat{\mathbf{P}}^r_{n,g})$ **then**
    Insert $\hat{p}_{n,g}$ into $\hat{\mathbf{P}}^r_{n,g}$
    $\rho \leftarrow \texttt{rank}(\hat{\mathcal{P}}^r_{n,g})$
    **for all** $f \in \mathcal{F}^r$ **do**
      **if** $t^{f,\rho(in))}_{n,g}$ exists **then**
        $\hat{p}^*_{n,g} = \sum_{j=1}^{|\hat{\mathcal{P}}^r_{n,g}|} a_j \cdot \hat{\mathbf{p}}^{(j)}_{n,g}$
        Send the Data packet $\hat{p}^*_{n,g}$ over the face $f$
        $\sigma^f_{n,g} \leftarrow \sigma^f_{n,g} + 1$
        Remove $t^{f,\rho(in))}_{n,g}$
      **end if**
    **end for**
  **else**
    Discard $\hat{p}_{n,g}$
  **end if**
**end if**

is given by $\xi^f_{n,g} = \texttt{rank}(\hat{\mathbf{P}}^r_{n,g}) - \sigma^f_{n,g}$. Recall that the parameter $\sigma^f_{n,g}$ denotes the number of network coded Data packets that have been sent over the face $f$. When $\xi^f_{n,g}$ is greater than 0, the router $r$ generates a new network coded Data packet $\hat{p}^*_{n,g} = \sum_{j=1}^{|\hat{\mathcal{P}}^r_{n,g}|} a_j \cdot \hat{\mathbf{p}}^{(j)}_{n,g}$ and sends it over the face $f$.

- *Forwarding an Interest* — If the number of network coded Data packets that can be generated by the router $r$ and that have a high probability to be innovative, $\xi^f_{n,g}$, is equal to 0, the router $r$ needs to receive an innovative Data packet that increases the rank of $\hat{\mathbf{P}}^r_{n,g}$ before it is able to reply to the Interest $\hat{i}_{n,g}$. Prior to forwarding an Interest, the router $r$ 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 from that of the NDN forwarder. Specifically, if a matching PIT entry $t_{n,g}$ is found, the router $r$ adds a new tuple $(\rho+1, \texttt{expire}(\hat{i}_{n,g}))$ to the in-record $t^{f(in)}_{n,g}$, where $\rho$ is the highest rank on the in-record and $\texttt{expire}(\hat{i}_{n,g})$ is the expire time of the Interest $\hat{i}_{n,g}$. The router $r$ forwards the Interest $\hat{i}_{n,g}$ if the number of innovative network coded Data packets with name prefix $(n, g)$ that it is expecting to receive before the Interest $\hat{i}_{n,g}$ expires is not enough to satisfy all the pending Interests. To compute the number of expected innovative Data packets, the router 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 aligned with 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 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, the number of expected innovative Data packets with name prefix $(n, g)$ is equal to the total number of Interests with the same name prefix that have been forwarded, denoted as $t^{(out)}_{n,g}$. Thus, the router forwards the Interest $\hat{i}_{n,g}$ if $t^{(out)}_{n,g} \leq |t^{f(in)}_{n,g}|$.
- *Waiting for a new network coded Data packet* — If $t^{(out)}_{n,g} > |t^{f(in)}_{n,g}|$, the router $r$ does not forward the Interest $\hat{i}_{n,g}$ and waits for a new network coded Data packet to arrive, 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 router $r$ 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 router $r$ accomplishes this by looking at its PIT. If no entry for the name prefix $(n, g)$ exists in its PIT, the router considers the Data packet unsolicited and it is not further transmitted. Otherwise, if the Data packet was expected, the router $r$ determines if the Data packet $\hat{p}_{n,g}$ is innovative. The Data packet $\hat{p}_{n,g}$ is innovative for the router $r$ if it is linearly independent with respect to all the Data packets in the CS of the router $r$, $\hat{\mathcal{P}}^r_{n,g}$, *i.e.*, if it increases the rank of $\hat{\mathbf{P}}^r_{n,g}$. If the Data packet is non-innovative, it is discarded by the router $r$. If the Data packet $\hat{p}_{n,g}$ is innovative, the router $r$ inserts it into its CS. Then, the router $r$ generates a new network coded Data packet $\hat{p}^*_{n,g} = \sum_{j=1}^{|\hat{\mathcal{P}}^r_{n,g}|} a_j \cdot \hat{\mathbf{p}}^{(j)}_{n,g}$ and sends it over every face that has a pending Interest to be satisfied when the rank of $\hat{\mathcal{P}}^r_{n,g}$ is $\rho$. This procedure is outlined in Algorithm 2.

## VI. Evaluation

In this section, we evaluate the performance of our proposed adaptive video streaming 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 show the performance evaluation results of our proposed adaptive video streaming architecture.

### A. Implementation

We implemented our proposed adaptive video streaming 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 [32]. 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 and PIT, 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 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 Interests and Data packets. We have used the Kodo C++ library [33] 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, as described in Sections V-A and V-B, respectively, we modified the Adaptive Multimedia Streaming with ndnSIM (AMuSt) codebase [34]. 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 [35], 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 [33] to enable network coding at 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. ndnSIM is an NDN simulator based on the NS-3 network simulator [36]. This simulator is used to generate the network nodes, *i.e.*, sources, routers and clients, and connect them with point-to-point links.

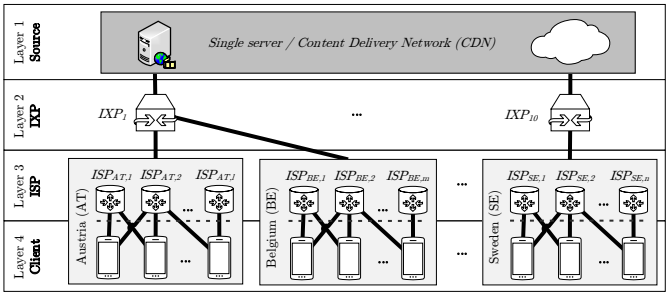



Fig. 5. Layered topology used in our evaluation. The first layer is the source, while the second and third layers represent IXP and ISP routers, respectively, and connect the source with the clients, that form the fourth layer.

### B. Network Topology

We evaluate our proposed adaptive video streaming architecture in a layered topology, presented in Fig. 5. The design of this topology is inspired by Netflix's OpenConnect Content Delivery Network [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 Content Delivery Network (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 routers, each one connected directly to the source. The third layer of the topology is another set of routers that represent Internet service provider (ISP) routers. Each ISP router is connected to two or three IXP routers. Moreover, the ISP routers 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 routers 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 routers. Finally, the fourth layer of the topology consists of a set of clients, each of them is connected to two ISP routers 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 routers to evaluate multi-path adaptive video streaming, considering that nowadays most end-user devices come with multiple interfaces, *e.g.*, LTE and Wi-Fi. The bandwidth of the links connecting the clients to the ISP routers is based on the values reported in the Netflix ISP Speed Index [13]. In detail, the bandwidth of each link is randomly selected from a normal distribution, with mean close to the average ISP speed reported

in the Netflix ISP Speed Index, and standard deviation 1.5. It is worth noting that the standard deviation reported in 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. In average, the used topologies have more than 110 ISP routers and more than 230 clients, additionally to the source and the 10 IXP routers.

### C. Evaluation Setup

We consider that the end-users are interested in a video $v$ that is available in three different representations, $\mathcal{Q} = \{480p,\ 720p,\ 1080p\}$ with bitrates $\{1750kbps,\ 3000kbps,\ 5800kbps\}$, respectively, that are a subset of the ones used by Netflix in the past [11]. Each representation is divided into a set of 50 segments, each of 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 coding operations are performed in a finite field of size $2^8$.

To select the representation that better adapts to their condition, the clients use an adaptation logic that considers the throughput measured by the client and the number of DASH segments that are buffered. The adaptation logic used in our evaluation is based on the one used by the DASH reference client, *dash.js*. As it has been demonstrated in the literature [37], the simple design of the *dash.js* adaptation logic performs better than other more sophisticated adaptation logics.

To demonstrate the Interest aggregation capabilities of the NetCodNDN forwarder, we consider a scenario where all the clients start requesting the video segments within the first 100ms of the simulation, and then carry on until they receive all the video segments. It is worth noting that since each client has different access bandwidth, and its adaptation logic works independently from other clients, the requested representation and segment IDs may vary across the clients. This means that at the same moment, different clients may be requesting different segments and representations, and thus some clients may finish retrieving all the segments earlier than others.

### D. Evaluation Results

We start by evaluating how the bandwidth of the links in the core network, *i.e.*, the links connecting the sources with the IXP routers, as well as the IXP routers with the ISP routers, affects the video quality received by the clients. Figs. 7 and 8 show the percentage of segments corresponding to each of the available video representations delivered for different values of the core links bandwidth, with NetCodNDN-DASH and NDN-DASH, respectively. As can be seen in Fig. 7, the percentage of segments delivered at the highest representation available (*i.e.*, 1080p) whith NetCodNDN-DASH, stabilizes at around 70% of the total number of delivered segments for core links bandwidth higher than 7.5Mbps. However, as can be seen in Fig. 8, with NDN-DASH the percentage of segments delivered at 1080p reaches the same stability only for core links bandwidth higher than 17.5Mbps. This behavior is observed as the use of network coding alleviates the competition between the clients for the core network resources. For the remainder of the evaluation, we consider that core links have a bandwidth of 10Mbps.

We now evaluate the cache-hit rate at the ISP and IXP layers. In Fig. 9, we can see that at the ISP layer, the cache-hit rate is constantly higher for NetCodNDN-DASH, as compared to NDN-DASH. After 60 seconds of streaming, the cache-hit rate for NDN-DASH is around 30%, while for NetCodNDN-DASH it is around 50%, *i.e.*, more than 20% higher. The reason for the lower cache-hit rate of NDN-DASH is that, since the clients are distributing the set of Interests that request a particular video segment 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. Further, it does not scale with the size of the network and the length of the video. On the contrary, the need for coordination is eliminated with NetCodNDN-DASH, 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 aggregation and improve the use of the available network resources through efficient multipath communication.

At the IXP layer, both the traditional and the network coded architectures show a higher cache-hit rate than the one achieved at the ISP layer. After 60 seconds of streaming, the cache-hit rate for NDN-DASH is around 75%, while for NetCodNDN-DASH it is around 95%, as illustrated in Fig. 10. The cache-hit rate is high at the IXP layer because the 10 routers 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. The 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. 11, we can see that there is a decrease in the total number of bytes that are provided by the source, from 1200MB for NDN-DASH to 450MB for NetCodNDN-DASH. This means that the use of network coding decreases the load on the source by more than 60%. This source load reduction translates into lower costs for the video content provider, that can reduce the number of servers at its data centers or reduce the amount of bandwidth needed by a CDN provider.

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 a given video representation is shown in Figs. 12 and 13, for NetCodNDN-DASH and NDN-DASH, respectively. With NetCodNDN-DASH more than 70% of the clients are able to receive the highest quality available ($1080p$) after a short adaptation period of around 8 segments. In contrast, with NDN-DASH less than 20% of the clients are able

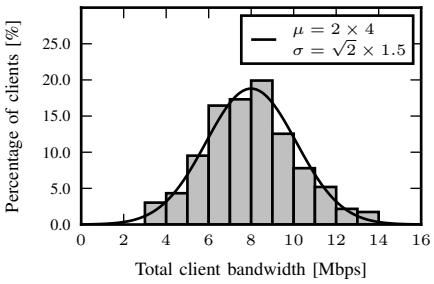



Fig. 6. Clients' bandwidth distribution. Each client has two faces, the average bandwidth is $\mu = 2 \times 4$Mbps and the standard deviation is $\sigma = \sqrt{2} \times 1.5$.

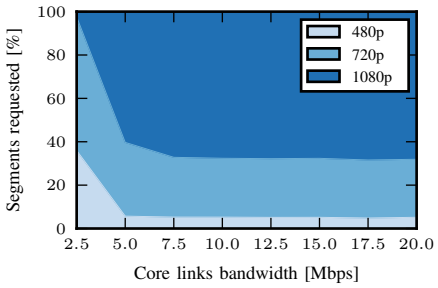



Fig. 7. Representation of the segments received by the clients with respect to different core links bandwidth, for NetCodNDN-DASH.

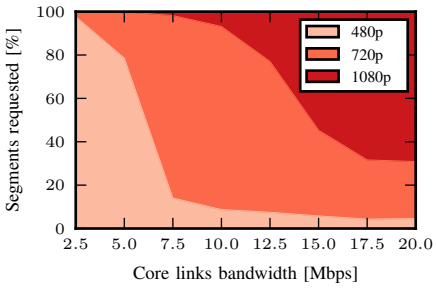



Fig. 8. Representation of the segments received by the clients with respect to different core links bandwidth, for NDN-DASH.

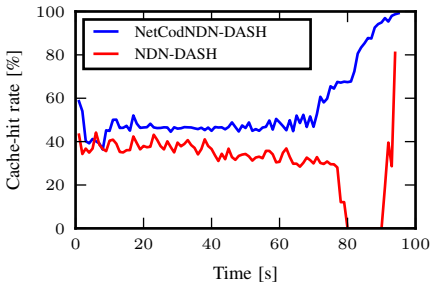



Fig. 9. Cache-hit rate at the ISP layer.

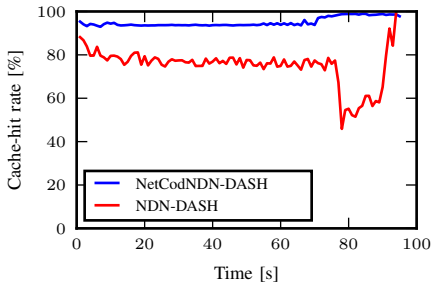



Fig. 10. Cache-hit rate at the IXP layer.

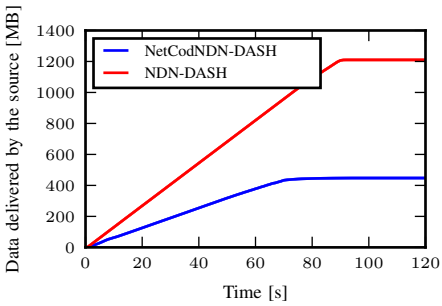



Fig. 11. Total data delivered by the source.

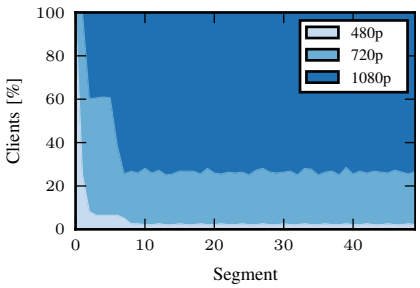



Fig. 12. Representations requested by the clients with NetCodNDN-DASH.

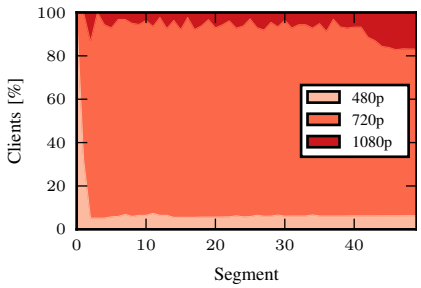



Fig. 13. Representations requested by the clients with NDN-DASH.

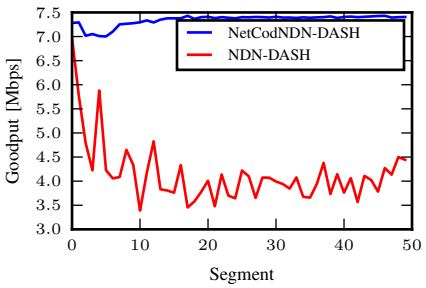



Fig. 14. Average goodput measured by the clients.

to receive the same quality, and only after a long adaptation period of around 45 segments. The reason for this is that since more Data packets are served from closer caches, the goodput measured by the clients is higher with NetCodNDN-DASH, as illustrated in Fig. 14. Specifically, the goodput measured by the NetCodNDN-DASH clients is around 7.5Mbps, while it is approximately 4Mbps for NDN-DASH clients. This is inline with the average client bandwidth of 8Mbps, which after excluding the protocol overhead data, leaves a goodput of around 7.5Mbps. The increased goodput accelerates the DASH adaptation towards the highest representation, meaning that the clients are able to receive the best possible quality earlier.

It is worth mentioning that the NetCodNDN-DASH clients enjoy an increased goodput compared to NDN-DASH clients and hence improved quality, despite the fact that the implementation of our NetCodNDN-DASH client introduces a small delay between the requests of two consecutive generations. This delay is due to the fact that the NetCodNDN-DASH 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 $g+1$. The NDN-DASH client does not suffer from this problem, as it does not consider generations. The goodput 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 in [38], [39] such approaches for video streaming in host-centric networks deploying generation-based network coding and will investigate them in the NetCodNDN-DASH 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. Our architecture takes advantage of multi-path communication and uses network coding to eliminate the need for coordination

between the network 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 cost 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: Forecast and Methodology, 2016-2021," White Paper, Cisco Systems Inc., Jun. 2016.

[2] ISO/IEC JTC 1/SC 29, *Information technology – Dynamic adaptive streaming over HTTP (DASH) – Part 1: Media presentation description and segment formats*, ISO/IEC 23 009-1:2014, May 2014.

[3] C. Westphal *et al.*, "Adaptive video streaming over information-centric networking (ICN)," RFC 7933, IRTF, 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, pp. 1–12.

[5] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, K. Claffy, P. Crowley, C. Papadopoulos, L. Wang, and B. Zhang, "Named data networking," *SIGCOMM Computer Communication Review*, 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 Int. Conf. Multimedia and Expo (ICME'13)*, San Jose, CA, USA, Jul. 2013, pp. 1–6.

[7] 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.

[8] 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, CA, USA, Apr. 2016.

[9] Y. Wu, P. Chou, and K. Jain, "A comparison of network coding and tree packing," in *Proc. of IEEE Int. Symp. Information Theory (ISIT'04)*, Chicago, IL, USA, Jun. 2004, p. 145.

[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, Nov. 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, Dec. 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," *arXiv preprint arXiv:1606.05519*, Jun. 2016.

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

[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*, San Francisco, CA, USA, 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, pp. 166–173.

[16] G. Rossini and D. Rossi, "Evaluating CCN multi-path interest forwarding strategies," *Computer Communications*, vol. 36, no. 7, pp. 771 – 778, Apr. 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, Sep. 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 1st ACM NoM Workshop*, Hilton Head, South Carolina, USA, Jun. 2012, pp. 31–36.

[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, no. 3, pp. 490–512, Mar. 2011.

[20] Q. Wu, Z. Li, and G. Xie, "CodingCache: multipath-aware CCN cache with network coding," in *Proc. of 3rd ACM SIGCOMM Workshop on Information-Centric Networks (ICN'2013)*, Hong Kong, China, Aug. 2013, pp. 41–42.

[21] J. Llorca, A. Tulino, K. Guan, and D. Kilper, "Network-coded caching-aided multicast for efficient content delivery," in *Proc. of IEEE Int. Conf. Communications (ICC'13)*, Budapest, Hungary, Jun. 2013, pp. 3557–3562.

[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, USA, May 2017.

[23] A. Afanasyev *et al.*, "NFD developer's guide," NDN, Technical Report NDN-0021 Revision 7, Oct. 2016.

[24] 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*, Oct. 2003.

[25] A. Ford *et al.*, "TCP extensions for multipath operation with multiple addresses," RFC 6824, IETF, Jan. 2013. [Online]. Available: http://www.rfc-editor.org/rfc/rfc6824.txt

[26] N. Cleju, N. Thomos, and P. Frossard, "Selection of network coding nodes for minimal playback delay in streaming overlays," *IEEE Transactions on Multimedia*, vol. 13, no. 5, pp. 1103–1115, Oct. 2011.

[27] C. Wu and B. Li, "rStream: Resilient and optimal peer-to-peer streaming with rateless codes," *IEEE Trans. Parallel and Distributed Systems*, vol. 19, no. 1, pp. 77–92, Jan. 2008.

[28] A. M. Sheikh, A. Fiandrotti, and E. Magli, "Distributed scheduling for low-delay and loss-resilient media streaming with network coding," *IEEE Trans. Multimedia*, vol. 16, no. 8, pp. 2294–2306, Dec. 2014.

[29] P. Chou and Y. Wu, "Network coding for the Internet and wireless networks," *IEEE Signal Processing Magazine*, vol. 24, no. 5, pp. 77–85, Sep. 2007.

[30] S. K. Fayazbakhsh, Y. Lin, A. Tootoonchian, A. Ghodsi, T. Koponen, B. Maggs, K. Ng, V. Sekar, and S. Shenker, "Less pain, most of the gain: incrementally deployable ICN," in *Proceedings of the ACM SIGCOMM 2013 Conference(SIGCOMM '13)*, 2013, pp. 147–158.

[31] Y. Sun, S. K. Fayaz, Y. Guo, V. Sekar, Y. Jin, M. A. Kaafar, and S. Uhlig, "Trace-driven analysis of ICN caching algorithms on video-on-demand workloads," in *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies (CoNEXT '14)*, 2014, pp. 363–376.

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

[33] M. Pedersen, J. Heide, and F. H. P. Fitzek, "Kodo: an open and research oriented network coding library," in *Proc. of IFIP 10th Int. Conf. Networking (NETWORKING'11)*, Valencia, Spain, May 2011, pp. 145–152.

[34] 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.

[35] 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 Int. Conf. Multimedia and Expo Workshops (ICMEW'13)*, San Jose, CA, USA, Jul. 2013.

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

[37] C. Timmerer, M. Maiero, and B. Rainer, "Which adaptation logic? An objective and subjective performance evaluation of http-based adaptive media streaming systems," *arXiv preprint arXiv:1606.00341*, Jun. 2016.

[38] E. Bourtsoulatze, N. Thomos, and P. Frossard, "Distributed rate allocation in inter-session network coding," *IEEE Trans. Multimedia*, vol. 16, no. 6, pp. 1752–1765, Oct. 2014.

[39] 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 Trans. Multimedia*, vol. 17, no. 6, pp. 893–906, Jun. 2015.