# PopNetCod: A Popularity-based Caching Policy for Network Coding enabled Named Data Networking

Jonnahtan Saltarin*, Torsten Braun*, Eirina Bourtsoulatze† and Nikolaos Thomos‡
*University of Bern, Bern, Switzerland
†Imperial College London, London, United Kingdom
‡University of Essex, Colchester, United Kingdom
saltarinj@gmail.com, braun@inf.unibe.ch, e.bourtsoulatze@imperial.ac.uk, nthomos@essex.ac.uk

***Abstract*—In this paper, we propose PopNetCod, a popularity-based caching policy for network coding enabled Named Data Networking. PopNetCod is a distributed caching policy, in which each router measures the local popularity of the content objects by analyzing the requests that it receives. It then uses this information to decide which Data packets to cache or evict from its content store. Since network coding is used, partial caching of content objects is supported, which facilitates the management of the content store. The routers decide the Data packets that they cache or evict in an online manner when they receive requests for Data packets. This allows the most popular Data packets to be cached closer to the network edges. The evaluation of PopNetCod shows an improved cache-hit rate compared to the widely used Leave Copy Everywhere placement policy and the Least Recently Used eviction policy. The improved cache-hit rate helps the clients to achieve higher goodput, while it also reduces the load on the source servers.**

## I. Introduction

Data intensive applications, *e.g.*, video streaming, software updates, *etc.*, are the major sources of data traffic in the Internet, and their predominance is expected to further increase in the near future [1]. Moreover, nowadays Internet users are more concerned about *what* data they request, rather than *where* that data is located. To address the increased data traffic and the shift in interest from location to data, technologies like Content Delivery Networks (CDN) have been proposed. However, these solutions cannot fully exploit the network resources and deal effectively with the increasing amount of data traffic, since they work on top of the current Internet architecture, which is based on host-to-host communication. To address this issue, the Named Data Networking (NDN) architecture [2], [3] has been proposed, which replaces the addresses of the communicating hosts (*i.e.*, IP addresses) with the name of the data being communicated. In the NDN architecture, clients request data by sending an *Interest* that contains the name of the requested data. Any network node that receives the Interest and holds a copy of the requested data can satisfy it by sending a *Data packet* back to the client.

Two of the main advantages that the NDN architecture has over the traditional host-to-host architectures are: *(i)* the inherent use of in-network caching, and *(ii)* the built-in support for multipath communications. The pervasive in-network caching concept proposed by NDN reduces the number of hops that Interests and Data packets need to travel in the network. This reduces the delay perceived by the application retrieving the requested data. However, having caches in all the routers is not always necessary to yield the full benefits that caching brings to the data delivery process. Previous works [4]–[6] have shown that enabling caches only at the edge of the network may achieve performance improvements similar to those obtained when every router is equipped with a cache. Furthermore, NDN provides natural multipath support by allowing clients to distribute the Interests that they need to send to retrieve content objects over all their network interfaces (*e.g.*, LTE, Wi-Fi), which enables the applications to better use the clients' network resources. However, in the presence of multiple clients and/or multiple data sources, the optimal use of multiple paths requires the nodes to coordinate where they forward each Interest in order to reduce the number of Data packet transmissions and the network load.

To optimally exploit the benefits brought by in-network caching and multipath communication, previous works [7], [8] had proposed the use of network coding [9]. In a network coding enabled NDN architecture, the network routers code Data packets by combining the Data packets available at their caches prior to forwarding them. The use of network coding *(i)* increases Data packet diversity in the network, hence, the use of in-network caches is optimized, and *(ii)* in multi-client and multi-source scenarios it removes the need for coordinating the faces where the nodes forward each Interest, which enables efficient multipath communication. Although there are works that consider the use of network coding in NDN, they do not consider that caching capacity is limited [7], [8], [10], [11] or they assume that a centralized node coordinates the caching decisions [12], [13], which is unrealistic or difficult to deploy.

In this paper, our goal is to develop a distributed caching policy that preserves the benefits that network coding brings to NDN for the realistic case when the caches have limited capacity. We propose *PopNetCod*, a popularity-based caching policy for network coding enabled NDN architectures. PopNetCod is a caching policy in which routers distributedly estimate the popularity of the content objects based on the received Interest. Based on this information, each router decides which Data packets to insert or evict from its cache. The decision to cache a particular Data packet is taken before the Data packet arrives at the router, *i.e.*, while processing the corresponding Interest. Since the first routers to process Interests in their path to the source are the edge routers, this helps to cache the most popular Data packets closer to the network edges, which reduces the data delivery delay [4]–[6]. To avoid caching the same Data packet in multiple routers over the same path, routers communicate the Data packets that they decide to cache by setting a binary flag in the Interests to be forwarded upstream. This increases the Data packet diversity in the caches. When the cache of a router is full and a Data packet should be cached, the router decides which Data packet should be evicted from its cache based on the popularity information.

We implement the proposed caching policy on top of ndnSIM [14], based on the NetCodNDN codebase [8], [10]. We evaluate the performance of PopNetCod in a Netflix-like video streaming scenario, designed using parameters available in the literature [15]–[17]. In comparison with a caching policy that uses the NDN's default Leave Copy Everywhere (LCE) placement policy and the Least Recently Used (LRU) eviction policy, PopNetCod achieves a higher cache-hit rate, which translates into higher video quality at the clients and reduced load at the sources.

The remainder of this paper is organized as follows. Section II provides an overview of the related works. Section III describes the system architecture. Section IV introduces the problem of caching in network coding enabled NDN for data intensive applications. Then, Section V presents our caching policy, PopNetCod. A practical implementation of the PopNetCod caching policy is described in Section VI. Section VII presents the evaluation of the PopNetCod caching policy.

## II. Related Work

Caching policies are needed to deal with caches that have limited capacity. Caching policies decide which Data packets are placed into the cache (*placement*), as well as which data packets are evicted from the cache when the cache is full and a new Data packet should be cached (*eviction*). There are placement algorithms that consider content popularity to decide which Data packets routers allow in their caches [18]–[21]. Specifically, *VIP* [18] is a framework for joint Interest forwarding and Data packet caching. This scheme uses a "virtual control plane" that operates on the Interest rate and a "real plane" which handles Interests and Data packets. It is shown that the design of joint algorithms for routing and caching is important for NDN. Thus, this scheme proposes distributed control algorithms that operate in the virtual control plane with the aim of increasing the number of Interests satisfied by in-network caches. *PopCaching* [19] is a popularity-based caching policy in which the popularity is computed online, without the need for a training phase. This makes *PopCaching* robust in dynamic popularity settings. However, *PopCaching* is designed for caching systems with a single cache in the path, while in this paper we are interested in networks of caches. *WAVE* [20] is a placement algorithm that determines the number of Data packets that should be cached for a given file with the help of an access counter. The number of Data packets to cache increases exponentially with the value of the access counter. The main idea of *WAVE*, which partially caches a content object according to the local popularity, is also adopted by the caching policy that we propose in this paper. However, *WAVE* does not facilitate edge caching, since the most popular data is cached closer to the source and slowly moves towards the edges as the number of requests increase. *Progressive* [21] is another partial caching algorithm, which exploits the content popularity to decide how many Data packets should be cached for each name prefix. The cache placement decision is taken when the Interests are received, which helps to cache the most popular content at the network edge. However, this approach lacks an eviction algorithm, and hence it cannot be deployed when the cache capacity is limited.

None of the approaches above consider the use of network coding [9], and all are evaluated in single-path scenarios. Given the benefits that network coding brings to multipath communications in NDN [7], [8], [10], [11], some approaches have been proposed to improve the benefits of caching in network coding enabled NDN architectures [12], [13], [22]. *NCCAM* [12] and *NCCM* [13] propose optimal solutions to the problem of efficiently caching in network coding enabled NDN. However, both approaches need a central entity that is aware of the network topology and the Interests, which does not scale well with the number of network nodes. *CodingCache* [22] is an eviction policy in which routers, before evicting a Data packet, apply network coding to the Data packet by means of combining it with other Data packets with the same name prefix that will remain in the cache. Due to the increased Data packet diversity in the network, the cache-hit rate is improved. However, in *CodingCache* Interest aggregation and Interest pipelining are problematic, limiting the benefits that network coding brings to the NDN architecture.

## III. Overview of Network Coding Enabled NDN

### A. Data Model

We consider a set of content objects $\mathcal{P}$ that is made available by a *content provider* to a set of *end users*. Each content object is uniquely identified by a name $n$. Clients use this name to request that particular content object. Each content object is divided into a set of Data packets $\mathcal{P}_n$, such that the size of each Data packet does not exceed the Maximum Transmission Unit (MTU) of the network. The set of Data packets $\mathcal{P}_n$ that compose a content object is divided into smaller sets of Data packets, which are known as *generations* [23]. The size of each generation $g$ is a design parameter chosen to enable network coding at scale. The set of Data packets that form the generation $g$ is denoted as $\hat{\mathcal{P}}_{n,g}$ and a network coded Data packet belonging to generation $g$ is represented by $\hat{p}_{n,g}$.

### B. Router Model

The routers have three main tables: a *Content Store (CS)*, where they cache Data packets to reply to future Interests, a *Pending Interest Table (PIT)*, where they keep track of the Interests that have been received and forwarded, to know where to send the Data packets backward to the clients, and a *Forwarding Information Base (FIB)*, which associates upstream faces with name prefixes, to route the Interests towards the sources. In order to enable the use of caching policies in the NetCodNDN architecture, we extend its design by adding a new module called Content Store Manager (CSM). The CSM manages the content store by enforcing a determined caching policy.

Whenever a router receives an Interest $\hat{i}_{n,g}$, it first verifies if it can reply to this Interest with the Data packets available in the CS. The router replies to the Interest if it is able to generate a network coded Data packet that has high probability of being innovative when forwarded on the path where the Interest arrived, *i.e.*, if the generated Data packet is linearly independent with respect to all the Data packets that have been sent over the face where the Interest arrived. In this case, the router generates a new Data packet by randomly combining the Data packets in its CS and then sends it downstream over the face where the Interest arrived. Otherwise, the router forwards the Interest to its upstream neighbors to receive a new Data packet that enables it to satisfy this Interest. However, if the router has already forwarded one or multiple Interests with the same name prefix $(n, g)$ and it expects to receive enough Data packets to reply to all the pending Interests stored in the PIT, the router simply *aggregates* this Interest in the PIT, and waits for enough innovative Data packets to arrive before replying to the Interest.

Whenever a router receives a Data packet $\hat{p}_{n,g}$, it first determines if the Data packet is innovative or not. A Data packet $\hat{p}_{n,g}$ is innovative if it is linearly independent with respect to all the Data packets in the CS of the router, *i.e.*, if it increases the rank of $\hat{\mathbf{P}}^r_{n,g}$. Non-innovative Data packets are discarded. If the Data packet $\hat{p}_{n,g}$ is innovative, the router sends the Data packet to the CSM, which decides to cache it or not according to the caching policy. Finally, the router generates a new network coded Data packet and sends it over every face that has a pending Interest to be satisfied.

### C. Content Store Model

The Content Store (CS) is a temporary storage space in which a router $r$ can cache Data packets that it has received and considers useful to reply to future Interests. The maximum number of Data packets that can be cached in the CS is given by $M$, while the set of Data packets that are cached in the CS is denoted as $\hat{\mathcal{P}}^r$. Thus, $|\hat{\mathcal{P}}^r| \leq M$.

Data packets in the CS are organized in CS entries. Each CS entry contains a set of network coded Data packets, $\hat{\mathcal{P}}^r_{n,g}$, that belong to the same generation $g$. Since the CS has a limited capacity of $M$ Data packets, then $\sum_{n,g} |\hat{\mathcal{P}}^r_{n,g}| \leq M$. The Data packets that compose a CS entry are stored in 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 $\hat{p}_{n,g}$.

Router $r$ generates a network coded Data packet $\hat{p}_{n,g}$ by randomly combining the Data packets $\hat{\mathbf{P}}^r_{n,g}$ 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 $\hat{\mathbf{p}}^{(j)}_{n,g}$ is the $j$th Data packet in $\hat{\mathbf{P}}^r_{n,g}$.

Additionally to the matrix $\hat{\mathbf{P}}^r_{n,g}$, each CS entry also stores a counter $\sigma^f_{n,g}$ for each face $f$ of router $r$. This counter measures the number of Data packets generated by applying network coding to the Data packets stored in matrix $\hat{\mathbf{P}}^r_{n,g}$ that have already been sent over face $f$, *i.e.*, it measures the amount of information from matrix $\hat{\mathbf{P}}^r_{n,g}$ that has been transmitted from router $r$ to its neighboring node connected over face $f$. The counter $\sigma^f_{n,g}$ is used to compute the number of network coded Data packets with name prefix $(n, g)$ that the router can generate with the Data packets cached in its CS and have high probability of being innovative to its neighboring node connected over face $f$. This number is denoted as $\xi^f_{n,g}$ and is computed as follows:

$$\xi^f_{n,g} = \mathtt{rank}(\hat{\mathbf{P}}^r_{n,g}) - \sigma^f_{n,g}. \tag{1}$$

When a Data packet with name prefix $(n, g)$ is evicted from the CS of router $r$, the amount of information in the matrix $\hat{\mathbf{P}}^r_{n,g}$ is reduced by 1. Correspondingly, the value of $\sigma^f_{n,g}$ is decreased by 1 for all faces.

## IV. Caching in Network Coding Enabled NDN

Whenever a router $r$ receives an Interest $\hat{i}_{n,g}$ over face $f$, it either *(i)* replies with a Data packet $\hat{p}_{n,g}$, if it can generate a network coded Data packet that has high probability of being innovative to its neighboring node connected over face $f$, *i.e.*, $\xi^f_{n,g} > 0$, or, otherwise, *(ii)* forwards the Interest $\hat{i}_{n,g}$ upstream.

If at time $t$ router $r$ receives the Interest $\hat{i}_{n,g}$, a cache-hit is defined as:

$$h^f_{n,g}(t) = \begin{cases} 1, & \text{if } \xi^f_{n,g} > 0 \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

Let us now assume that during a time period $[t, t+T]$ router $r$ receives a set of Interests $\mathcal{I}(t,T)$. The cache-hit rate during this time period is defined as follows:

$$H(t,T) = \frac{1}{T} \sum_{t'=t}^{t+T} h^f_{n,g}(t'). \tag{3}$$

The overall cache-hit rate seen by router $r$ at time $t$ can be computed as follows:

$$H(t) = \lim_{T\to\infty} H(t,T) = \lim_{T\to\infty} \frac{1}{T} \sum_{t'=t}^{t+T} h^f_{n,g}(t'). \tag{4}$$

To make optimal use of the limited CS capacity, the objective of each router is to maximize the number of Interests that it can satisfy with the Data packets available in its CS, *i.e.*, maximize its overall cache-hit rate. Achieving a high cache-hit rate at the routers is beneficial for both clients and sources. For the sources, an increased cache-hit rate reduces their processing load and bandwidth needs, since the number of Interests that they receive is reduced. For the clients, the delivery delay is reduced, since the Interests are satisfied with Data packets cached at routers closer to them.

It is clear from (2), (3), and (4) that in order to maximize the overall cache-hit rate, routers should maintain the value of $\xi^f_{n,g}$ high enough so that most of the Interests received can be satisfied with the Data packets in their CS. However, since in this paper we consider that the routers' CS have limited

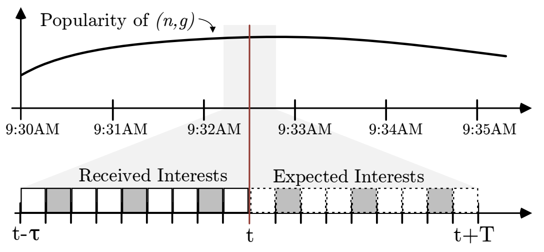

Fig. 1. Popularity prediction for the name prefix $(n, g)$.

capacity, it is unfeasible for a router to cache all the Data packets that it receives [7], [8], [10], [11]. Optimal solutions to this issue have been proposed in previous works [12], [13], which consider a central controller that knows the network topology and is aware of all the Interests received by the routers. However, these solutions do not scale well with the size of the network, since they require a high number of signaling messages and a powerful enough controller. Hence, in this work we consider that each router decides online and independently from other routers if a Data packet should be cached or not, and which Data packet should be evicted from the CS when it is full. This is achieved by using a distributed caching policy $\pi$ that maximizes the overall cache-hit rate $H(t)$ of each router,

$$\max_{\pi} H(t). \tag{5}$$

The optimal caching policy $\pi$ predicts which Interests will be received in the future, so that the router caches the Data packets that will be useful to satisfy those Interests.

## V. The PopNetCod Caching Policy

In this section, we present our popularity-based caching policy for network coding enabled NDN, called PopNetCod. To increase the overall cache-hit rate, the PopNetCod caching policy exploits real-time data popularity measurements to determine the number of Data packets that each router should cache for each name prefix. In order to determine which Data packets to cache in and/or evict from the CS, such that the overall cache-hit rate is maximized, PopNetCod performs the following steps. First, it measures the popularity of the different name prefixes contained in the Interests that pass through it. Then, it uses this popularity to predict the Interests that it will receive. Finally, it uses this prediction to determine in an online manner the Data packets that should be cached and the ones that should be evicted from the CS.

### *A. Popularity Prediction*

The popularity prediction in PopNetCod is based on the fact that the rate $\lambda^f_{n,g}(t)$ at which Interests for a particular content object arrive at a router $r$ over face $f$ at time $t$ tends to vary smoothly, as shown in Fig. 1. Thus, router $r$ can predict the rate of the Interests that it will receive in the near future by observing the Interests that it recently received. Let us denote $\mathcal{I}^f_{n,g}(\tau, t)$ as the set of Interests for the name prefix $(n, g)$ that router $r$ has received over face $f$ in the past period $[t-\tau, t]$, where $t$ is the current time and $\tau$ is the observation period. Let us also denote $\mathcal{I}^f(\tau, t)$ as the total set of Interests for all name prefixes received over face $f$ during the period $[t-\tau, t]$. Using the sets $\mathcal{I}^f_{n,g}(\tau, t)$ and $\mathcal{I}^f(\tau, t)$, router $r$ can compute the average Interest rate for the name prefix $(n, g)$ over face $f$ as follows:

$$\lambda^f_{n,g}(\tau, t) = \frac{|\mathcal{I}^f_{n,g}(\tau, t)|}{|\mathcal{I}^f(\tau, t)|}, \tag{6}$$

Note that since the average Interest rate does not vary abruptly, the average Interest rate $\lambda^f_{n,g}(\tau, t)$ of the recent period $[t-\tau, t]$ will be very close to that expected in the near future, *i.e.*, in the period $[t, t+T]$ where $T$ is the length of the prediction period. Thus, $\lambda^f_{n,g}(\tau, t) = \lambda^f_{n,g}(t, T)$, which hereafter we denote as $\lambda^f_{n,g}(t)$. The PopNetCod caching policy uses $\lambda^f_{n,g}(t)$ to predict the number of Interests with name prefix $(n, g)$ that will be received over face $f$ in the near future, and hence, to allocate more storage space in the CS to Data packets with higher cache-hit probability.

In order to prepare the CS for the Interests that the router may receive, the PopNetCod caching policy maps the received Interest rate to the capacity of the CS, such that name prefixes with high rate are allocated more space in the CS. The number of network coded Data packets with name prefix $(n, g)$ that the router should cache in its CS at time $t$ to satisfy the Interests expected over face $f$ is denoted as $M^f_{n,g}(t)$ and computed as:

$$M^f_{n,g}(t) = \begin{cases} \lambda^f_{n,g}(t) \cdot M, & \text{if } \lambda^f_{n,g}(t) \cdot M < |\hat{\mathcal{P}}_{n,g}| \\ |\hat{\mathcal{P}}_{n,g}|, & \text{otherwise.} \end{cases} \tag{7}$$

### *B. PopNetCod Placement*

In the PopNetCod caching policy, the placement decision is taken following the reception of an Interest. Whenever a router decides to cache the Data packet that is expected as a reply to the received Interest, it sets a flag on the Interest signaling upstream routers about its decision. In the case of a set flag, the upstream nodes do not consider this Interest for caching. Since the edge routers (*i.e.*, the routers that are directly connected to the clients) are the first ones that have the possibility to decide whether they will cache a Data packet, the PopNetCod caching policy naturally enables edge caching. This is inline with recent works [4]–[6] arguing that most of the gains from caching in NDN networks come from edge caches, and thus, it is natural to cache the most popular content at edge routers.

Whenever a router receives an Interest $\hat{i}_{n,g}$ over face $f_t$ at time $t$, the PopNetCod caching policy follows the next steps to decide if the Data packet $\hat{p}_{n,g}$ should be cached. First, it uses popularity prediction to compute $M^f_{n,g}(t)$, *i.e.*, the total number of Data packets that it aims to cache for name prefix $(n, g)$, as defined in (7). Then, it computes the number of Data packets that it should cache in order to satisfy the expected Interests as:

$$\delta^f_{n,g}(t) = M^f_{n,g}(t) - \xi^f_{n,g}(t) \; \forall f \in \mathcal{F}. \tag{8}$$

Finally, the caching policy decides to cache the Data packet $\hat{p}_{n,g}$ that is expected as reply to the received Interest if the average number of Data packets needed by all the faces is greater than 0. However, it should be noted that the Data packet $\hat{p}_{n,g}$ will not be useful to the node connected over the downstream face $f_t$ over which the Interest arrived. This is because when the Data packet $\hat{p}_{n,g}$ arrives at the router, it is sent to face $f_t$ in order to satisfy the received Interest. Then, replying with the same Data packet to a subsequent Interest received over the same face $f_t$ does not add any innovative information, *i.e.*, the Data packet is considered as duplicated. Instead, the expected Data packet $\hat{p}_{n,g}$ is potentially useful for all the nodes connected over all the other downstream faces of the router. For this reason, the average number of Data packets needed is measured only over the downstream faces different to the one over which the Interest arrived. It is computed as:

$$\Delta^{+}_{n,g}(t) = \frac{1}{|\mathcal{F}^r| - 1} \sum_{\substack{f \in \mathcal{F} \\ f \neq f_t}} \delta^{f}_{n,g}(t) > 0, \tag{9}$$

where $\mathcal{F}^r$ denotes the downstream faces of router $r$.

### C. PopNetCod Eviction

The steps followed by the PopNetCod caching policy to decide how many Data packets with name prefix $(n, g)$ can be evicted from the router's CS are the following. Similarly to the placement case, first, the caching policy uses popularity prediction to compute $M^{f}_{n,g}(t)$, *i.e.*, the number of Data packets that it aims to cache for name prefix $(n, g)$. Then, it computes the number of Data packets that it can evict from its CS and still satisfy the expected Interests as:

$$\tilde{\delta}^{f}_{n,g}(t) = \mathtt{rank}(\hat{\mathbf{P}}^{r}_{n,g}) - M^{f}_{n,g}(t) \forall f \in \mathcal{F}. \tag{10}$$

Finally, the number of Data packets the router can evict from a particular name prefix $(n, g)$ is computed as the minimum number of Data packets that it can evict over all the faces:

$$\Delta^{-}_{n,g}(t) = \min_{f \in \mathcal{F}} \tilde{\delta}^{f}_{n,g}(t). \tag{11}$$

## VI. Practical Implementation of PopNetCod

In this section, we describe a practical implementation of the PopNetCod caching policy in the NetCodNDN architecture [10]. First, we describe the signaling between routers, which is used to prevent routers of the same path to cache duplicate Data packets. Next, we present the Interest processing algorithm, where placement decisions are made. Finally, we describe the Data packet processing algorithm for placement enforcement, eviction decision, and eviction enforcement.

### A. Signaling Between Routers

The PopNetCod caching policy is distributed and requires very limited signaling between routers. The only signaling that exists between routers to implement the PopNetCod caching policy is a binary flag added to the Interest and Data packets that is used to inform neighbor routers that an expected Data packet will be cached or that a received Data packet has been cached. Distributed caching policy decisions help to keep the complexity of the system low and to make our system scalable to a large number of routers.

Each Interest $\hat{i}_{n,g}$ carries a flag `CachingDown`, which is set to 1 by a router when it decides to cache the Data packet $\hat{p}_{n,g}$ that is expected to come as reply to the Interest. This flag informs upstream routers that another router downstream has already decided to cache the Data packet that is expected to come as reply to this Interest. The routers receiving an Interest with the `CachingDown` flag set to 1 do not consider to cache the Data packet that is expected to come as reply to this Interest, therefore reducing the number of duplicated Data packets in the path and the processing load in the nodes.

Since Interests for network coded data do not request particular Data packets, but rather any network coded Data packet with the requested name prefix, the routers need a way to know that a Data packet has been already cached by another router, so that they avoid caching duplicated Data packets. For this reason, each Data packet $\hat{p}_{n,g}$ has a flag `CachedUp`, which is set to 1 by a router when it caches this Data packet in its CS. This flag informs the downstream routers that another router has already cached this Data packet. A router receiving a network coded Data packet with the flag set to 1 does not consider it for caching. Instead, it waits for another Data packet with the same name prefix that has not been cached upstream. This ensures that a Data packet is cached by only one router on its way to the client.

### B. Status Information at Routers

Each router implementing the PopNetCod caching policy should store information that assists to identify the Data packets that should be cached or evicted. In particular, the router needs to keep the *Recently received Interests* information to compute the popularity prediction. Moreover, since the placement decision takes place when the Interest is received, the router needs to remember the *Names to be cached*, such that the selected Data packets are cached when they arrive. Finally, since the popularity information can vary over time, the routers should keep a list with the *Names to consider for eviction*, which is used when they decide about eviction. Below, we describe the data structures used to store this information.

• *Recently received Interests* — The router maintains a list $\mathbf{L}^f$ for each face $f$ of the router, where it stores the names of the Interests $\mathcal{I}^f(\tau, t)$ received over face $f$ during the period $[\tau, t]$. The parameter $\tau$ controls how much into the past is observed by the router to compute the popularity prediction. Together with the name prefix, each element in $\mathbf{L}^f$ also stores the time $t_i$ at which the Interest was received, such that it can be removed from $\mathbf{L}^f$ at time $t_i + \tau$.

• *Names to be cached* — The router maintains a table $\mathbf{A}$, where it stores the name prefixes (*i.e.*, the content object name appended with the generation ID) and the number of the Data packets that should be cached. When the router receives an Interest $\hat{i}_{n,g}$ and the PopNetCod caching policy decides that the network coded Data packet that is expected as reply should be cached, the router adds its name prefix $(n, g)$ to the list $\mathbf{A}$. Then, whenever a network coded Data packet arrives, the router looks for the name prefix of the Data packet in the list $\mathbf{A}$. If it finds a match, it caches the Data packet.

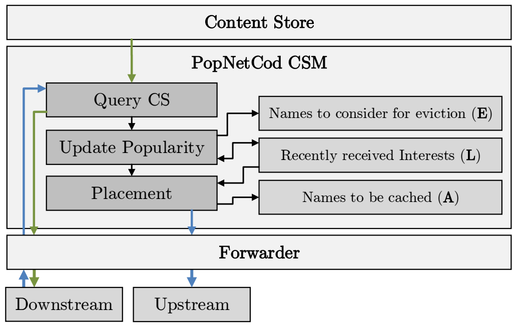

Fig. 2. Access to the CS and the Status Information during the Interest processing in a CSM configured with the PopNetCod caching policy.

**Algorithm 1** Interest processing at the CSM

```
Require: $\hat{i}_{n,g}$, $f$
 1: $t \leftarrow$ current time
 2: if Flag CachingDown in $\hat{i}_{n,g}$ is set to 1 then
 3:     if $\xi^f_{n,g} > 0$ then        ($\hat{i}_{n,g}$ can be satisfied from the CS)
 4:         Generate a Data packet $\hat{p}_{n,g}$ from the CS
 5:         Return $\hat{p}_{n,g}$
 6:     else
 7:         Return $\hat{i}_{n,g}$
 8:     end if
 9: else
10:     Add $(n, g)$ to $\mathbf{L}^f$
11:     if $\xi^f_{n,g} > 0$ then        ($\hat{i}_{n,g}$ can be satisfied from the CS)
12:         Generate a Data packet $\hat{p}_{n,g}$ from the CS
13:         Return $\hat{p}_{n,g}$
14:     else if $\hat{i}_{n,g}$ will be aggregated by the PIT then
15:         Return $\hat{i}_{n,g}$
16:     else
17:         Update $\mathbf{L}$.                        (Algorithm 2)
18:         if $\Delta^+_{n,g}(t) > 0$ then        ($\hat{p}_{n,g}$ should be cached)
19:             Insert $(n, g)$ into $\mathbf{A}$
20:             Set the flag CachingDown of $\hat{i}_{n,g}$ to 1
21:             Return $\hat{i}_{n,g}$
22:         else
23:             Return $\hat{i}_{n,g}$
24:         end if
25:     end if
26: end if
```

• *Names to consider for eviction* — The router also maintains a queue $\mathbf{E}$, where it stores the name prefixes of the CS entries that can be considered for Data packet eviction. When a name prefix $(n, g)$ is removed from the list $\mathbf{L}^f$, the popularity of this name prefix decreases, *i.e.*, it is a good candidate to consider for eviction. Thus, each time a name prefix is removed from $\mathbf{L}^f$, it is added to $\mathbf{E}$.

## C. Interest Processing

As depicted in Fig. 2, when a CSM configured with the PopNetCod caching policy receives an Interest $\hat{i}_{n,g}$ from downstream, it *(i)* determines if the Interest can be replied from the CS. Then, if the CSM could not reply to the Interest with

**Algorithm 2** Update $\mathbf{L}$

```
1: for all $f \in \mathcal{F}^r$ do
2:     for all expired entries $(n_l, g_l)$ in $\mathbf{L}^f$ do
3:         Remove $(n_l, g_l)$ from $\mathbf{L}^f$
4:         Add $(n_l, g_l)$ to $\mathbf{E}$
5:     end for
6: end for
```

the content of its CS, it *(ii)* updates the popularity information, and, *(iii)* determines if the Data packet that is expected as reply to this Interest should be cached. The CSM should provide the NetCodNDN forwarder with either a Data packet that should be sent as reply to the Interest, or an Interest that should be forwarded upstream. Below we describe the details of this procedure, which is summarized in Algorithm 1.

After receiving an Interest $\hat{i}_{n,g}$, the CSM first checks the flag `CachingDown` to see if any previous node downstream in the path has decided to cache the Data packet that is expected as reply to this Interest (lines 2 to 8). If the flag `CachingDown` is set to 1, then the CSM only checks its CS to determine if the Interest can be satisfied from the CS. If this is possible, *i.e.*, if $\xi^f_{n,g}$ is greater than 0, it generates a network coded Data packet from the CS and provides it to the NetCodNDN forwarder, which sends it over face $f$. If the Interest can not be satisfied from the CS, the CSM provides the same Interest to the NetCodNDN forwarder, which forwards it upstream.

If the flag `CachingDown` is set to 0, the CSM first inserts name $(n, g)$ of the Interest into the list $\mathbf{L}^f$ (line 10). Then, the CSM checks if it can satisfy the Interest with the content of the CS (lines 11 to 13). If this is possible, *i.e.*, if $\xi^f_{n,g}$ is greater than 0, it generates a network coded Data packet from the CS and provides it to the NetCodNDN forwarder which sends it over face $f$. Otherwise, the node needs to forward the Interest to its neighbor nodes. If the router does not send the Interest upstream, but aggregates it in the PIT with a previously received Interest, the CSM does not need to do anything else and provides the Interest to the NetCodNDN forwarder, which aggregates it (line 15). If the Interest will not be aggregated, then the CSM determines if it will cache the Data packet with name prefix $(n, g)$ that is expected as reply to this Interest, by computing $\Delta^-_{n,g}(t)$ using Eq. (9).

In order to obtain an accurate value of $\Delta^-_{n,g}(t)$, the CSM first updates the popularity information, removing all the expired elements from $\mathbf{L}^f$ and adding their name prefix to the list $\mathbf{E}$ of name prefixes to be considered for eviction (line 17). This procedure is summarized in Algorithm 2. Then, the CSM computes the value of $\Delta^+_{n,g}(t)$. If $\Delta^+_{n,g}(t) > 0$, it means that the Data packet should be cached. In this case, the CSM inserts name prefix $(n, g)$ into the list $\mathbf{A}$, sets the flag `CachingDown` on the Interest $\hat{i}_{n,g}$ to 1 and, finally, provides the modified Interest to the NetCodNDN forwarder, which forwards it upstream (lines 18 to 21). If $\Delta^+_{n,g}(t) \leq 0$, then the CSM provides the same Interest to the NetCodNDN forwarder, which forwards it upstream (line 23).

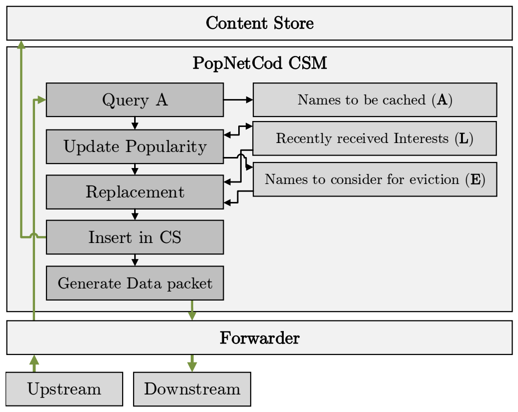

Fig. 3. Access to the CS and the Status Information during the Data packet processing in a CSM configured with the PopNetCod caching policy.

**Algorithm 3** Data packet processing at the CSM

```
Require: $\hat{p}_{n,g}$
1: if Flag CachingUp in $\hat{p}_{n,g}$ is set to 1 then
2:     Return $\hat{p}_{n,g}$
3: else if $(n,g) \notin \mathbf{A}$ then
4:     Return $\hat{p}_{n,g}$
5: else
6:     Update $\mathbf{A}$
7:     if $|\mathcal{P}^r| == M$ then                    (The CS is full)
8:         Update $\mathbf{L}$                            (Algorithm 2)
9:         while $|\mathcal{P}^r| == M$ do
10:            Select an element $(n_e, g_e)$ from $\mathbf{E}$
11:            if $\Delta^-_{n_e,g_e}(t) > 0$ then
12:                Evict $\Delta^-_{n_e,g_e}(t)$ Data packets with name prefix
                   $(n_e, g_e)$ from the CS
13:            end if
14:        end while
15:    end if
16:    Insert $\hat{p}_{n,g}$ into the CS
17:    Generate a Data packet $\hat{p}^*_{n,g}$ from the CS
18:    Set the flag CachingDown of $\hat{p}^*_{n,g}$ to 1
19:    Return $\hat{p}^*_{n,g}$
20: end if
```

### D. Data Packet Processing

As depicted in Fig. 3, when a CSM configured with the PopNetCod caching policy receives a network coded Data packet $\hat{p}_{n,g}$ from upstream, it *(i)* determines if the Data packet should be cached in the CS, by consulting $\mathbf{A}$. If the Data packet should be cached, the CSM ensures that there is enough free space in the CS, *(ii)* updating the popularity information and *(iii)* executing the cache replacement procedure if needed. Finally, the CSM *(iv)* inserts the received Data packet into the CS, and *(v)* generates a new network coded Data packet that should be forwarded downstream. This procedure is detailed below and summarized in Algorithm 3.

After receiving a Data packet $\hat{p}_{n,g}$, the CSM first checks the flag `CachedUp` to determine if any router upstream has already cached this Data packet. If the flag `CachedUp` has been set to 1, then, the CSM understands that another router upstream has already cached this Data packet. In this case, the CSM returns the Data packet to the NetCodNDN forwarder, which replies to any matching pending Interest (line 1).

When the flag `CachedUp` is set to 0, then the CSM first verifies if any entry in $\mathbf{A}$ matches name prefix $(n, g)$. If there is no matching entry, the CSM returns the Data packet to the NetCodNDN forwarder (line 3). If there is a match, the Data packet should be cached, and $\mathbf{A}$ is updated by increasing the counter of the matching entry by one (line 6). However, if the CS is full, the CSM first needs to release some space in the CS (lines 7 to 15). To evict Data packets, the CSM goes through the list $\mathbf{E}$, each time selecting a name prefix $(n_e, g_e)$ and computing the number of Data packets that can be evicted for the name prefix using Eq. (11). If this number is greater than 0, then the CSM evicts the corresponding number of Data packets from the CS and interrupts the scan of the list. Note that, since the cached Data packets are network coded, the CSM does not need to decide which particular Data packets from the CS entry $\hat{\mathcal{P}}_{n,g}$ it should evict from the CS, but it can select randomly network coded Data packets from the CS entry and evict them. After evicting at least one Data packet, the CSM caches the received Data packet $\hat{p}_{n,g}$. Then, the router generates a new Data packet $\hat{p}^*_{n,g}$ by applying network coding to the cached Data packets with name prefix $(n, g)$. Since the new Data packet $\hat{p}^*_{n,g}$ contains the cached Data packet $\hat{p}_{n,g}$, the router sets the flag `CachedUp` of $\hat{p}^*_{n,g}$ to 1. Finally, the CSM provides Data packet $\hat{p}^*_{n,g}$ to the NetCodNDN forwarder, which uses it to reply to pending Interests with name prefix $(n, g)$.

## VII. Evaluation

In this section, we evaluate the performance of the PopNetCod caching policy in an adaptive video streaming architecture based on NetCodNDN [10]. First, we describe the evaluation setup. Then, we present the caching policies with which we compare the PopNetCod caching policy. Finally, we show the performance evaluation results.

### A. Evaluation Setup

We consider a layered topology consisting of 1 source, 123 clients, and 45 routers connecting the clients and the sources. The routers are arranged in a two-tier topology, with 10 routers directly connected to the source and 35 edge routers directly connected to the clients. The links connecting the routers between them and the links connecting the routers to the source have a bandwidth of $20Mbps$. The bandwidth of the links connecting the clients to the routers follow a normal distribution, with mean $4Mbps$ and standard deviation 1.5. These values are chosen based on the Netflix ISP Speed Index [17]. Each client is connected with two routers, considering that nowadays most end-user devices have multiple interfaces, *e.g.*, LTE, Wi-Fi.

For the evaluation, we consider that the source offers 5 videos for streaming, each one composed of 50 video segments with a duration of 2 seconds each, *i.e.*, in total, each video has a duration of 100 seconds. The video segments are available in three different representations, $\mathcal{Q} = \{480p,\ 720p,\ 1080p\}$ with bitrates $\{1750kbps,\ 3000kbps,\ 5800kbps\}$, respectively. These values for the representations and bitrates are according to the values that had been used by Netflix [15]. As presented in Section III-A, the content objects (*i.e.*, the video segments in our evaluation scenario) are divided into Data packets and generations, in order to implement network coding. In particular, for the representations $\mathcal{Q} = \{480p,\ 720p,\ 1080p\}$, each video segment is divided into $\{359, 615, 1188\}$ Data packets of 1250 bytes each, and $\{4, 7, 12\}$ generations, respectively. Thus, in total, the source stores $540,500$ Data packets. All the routers are equipped with content stores able to cache between $0.9\%$ and $2.3\%$ of the total Data packets available at the source.

The clients randomly choose a video to request and start the adaptive video retrieval process at a random time during the first 5 seconds of the simulation. The network coding operations are performed in a finite field of size $2^8$. The clients use the *dash.js* adaptation logic [24] to choose the representation that better adapts to the current conditions, *i.e.*, the measured goodput and the number of buffered video segments.

## B. Benchmarks

We compare the performance of our caching algorithm with the following benchmarks:

- *LCE-NoLimit* — The placement policy is Leave Copy Everywhere (LCE). We assume that the CSs of the routers have enough space to store all the videos.
- *LCE+LRU* — The placement policy is LCE, while the eviction policy is Least Recently Used (LRU), which evicts Data packets with the least recently requested name.
- *NoCache* — In this setting, the routers do not have a CS, *i.e.*, all the Data packets should be retrieved from the source.

## C. Evaluation Results

We first evaluate the average cache-hit rate at the routers. In Fig. 4, we can see that by using the PopNetCod caching policy, the routers achieve a higher cache-hit rate than with LCE-LRU. This is because with PopNetCod the number of Data packets cached for a certain name prefix increases smoothly, according to the popularity. In comparison, with LCE+LRU all Data packets received by the router are cached, and the least recently used are evicted from the CS when the capacity is exceeded. Thus, if a router receives Data packets that are requested by a single client, the router still caches them, wasting storage capacity that could be used to cache more popular Data packets that are requested by multiple clients. We can also see that the LCE+NoLimit caching policy defines an upper bound to the cache-hit rate at the routers, since caching all the Data packets with unlimited CS capacity represents the best caching scenario. On the contrary, the NoCache case, where the routers do not have CS capacity, defines a lower bound to the cache-hit rate. Note that in our evaluation the NoCache policy has a non-zero cache-hit rate because our measurement of cache-hit rate also includes Interest aggregations, which is what is being measured in this case.

The increased cache-hit rate that the PopNetCod caching policy brings to the routers has two major consequences: *(i)* the goodput at the clients increases, which enables the adaptation logic to choose higher quality representations when bandwidth is sufficient, and *(ii)* the source receives less Interests, meaning that its processing and network load is reduced.

Let us first evaluate the impact that the increased cache-hit rate at the routers has for the clients. In Fig. 5, it is shown that by using PopNetCod, the clients benefit from an increased goodput, compared to the LCE+LRU policy. This is a consequence not only of the increased cache-hit rate in the network, but also because PopNetCod caches the most popular content in the network edge, which reduces the content retrieval delay. The percentage of video segments delivered to the clients for each of the available representations (*i.e.*, $480p$, $720p$, and $1080p$) with the PopNetCod and LCE+LRU caching policies is shown in Figs. 6 and 7, respectively. We can see that, compared to the LCE+LRU policy, with the PopNetCod caching policy a higher percentage of video segments are delivered in the highest representation available, *i.e.*, $1080p$. This happens because the Data packet retrieval delay is reduced, since more Interests are being satisfied from the routers' content stores, which increases the goodput measured by the clients. The percentage of video segments delivered to the clients in each of the available representations with the upper bound LCE+NoLimit caching policy can be seen in Fig. 8.

Finally, we analyze the impact that the increased cache-hit rate in the routers has for the sources by measuring the load reduction at the source. This metric measures the percentage of Data packets received at the clients that have not been directly provided by the source. It is computed as $1 - N_S^{sent}/N_C^{rcvd}$, where $N_S^{sent}$ denotes the total number of Data packets sent by the source, and $N_C^{rcvd}$ denotes the total number of Data packets received by all the clients. In Fig. 9, we can see that by using the PopNetCod caching policy, the source load is reduced by up to 10% more than by using LCE+LRU, when the CS size is 12.5K Data packets. Note that the load reduction on the source in the NoCache scenario is larger than 0, even if no Data packet is being served from the CSs. This is because the Interest aggregation at the routers makes it possible to serve multiple Interests with the same Data packet, reducing the number of Data packets delivered by the source.

## VIII. Conclusions

In this paper, we have presented PopNetCod, a popularity-based caching policy for data intensive applications communicating over network coding enabled NDN. PopNetCod is a distributed caching policy, where each router aims at increasing its local cache-hit rate, by measuring the popularity of each content object and using it to determine the number of Data packets for each content object that it caches in its content store. PopNetCod takes cache placement decisions when Interests arrive at the routers, which naturally enables edge caching. The evaluation of the PopNetCod caching policy is performed in a Netflix-like video streaming scenario. The results show that, in comparison with a caching policy that uses the LCE placement

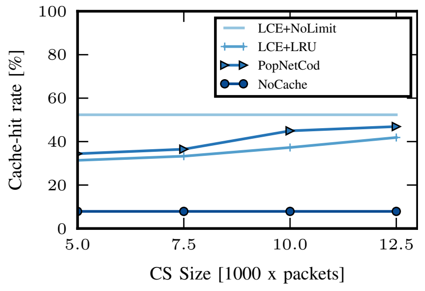

Fig. 4. Average cache-hit rate in the routers.

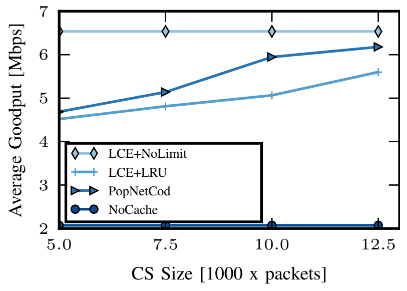

Fig. 5. Average goodput perceived by the clients.

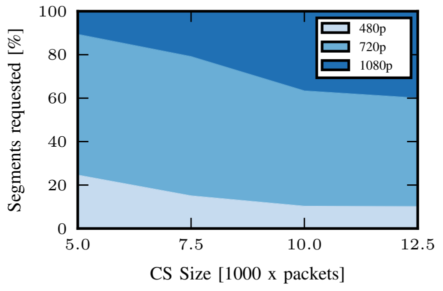

Fig. 6. Percentage of video segments delivered in each of the representations, with PopNetCod.

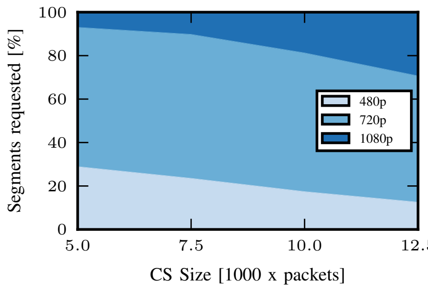

Fig. 7. Percentage of video segments delivered in each of the representations, with LCE+LRU.

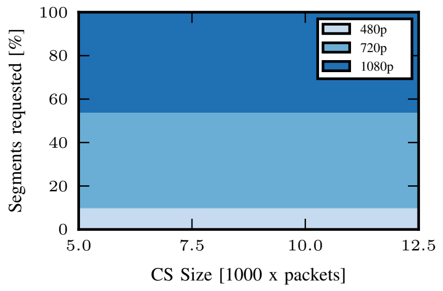

Fig. 8. Percentage of video segments delivered in each of the representations, with LCE+NoLimit.

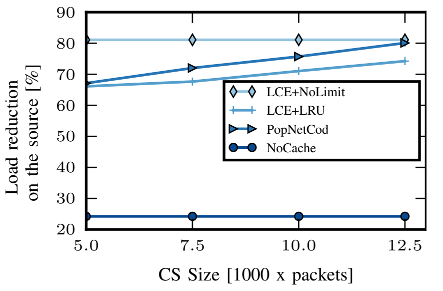

Fig. 9. Load reduction in the source, measured as the percentage of Data packets provided by caches.

policy and the LRU eviction policy, PopNetCod achieves a higher cache-hit rate. The increased cache-hit rate reduces the number of Interests that the source should satisfy, and also increases the goodput seen by the clients. Thus, our caching policy presents benefits for the content providers, by reducing the load of its servers and hence its operative costs, and for the end-users, who are able to watch higher quality videos.

## References

[1] "Cisco Visual Networking Index: Forecast and Methodology, 2016-2021," White Paper, Cisco Systems Inc., Jun. 2016.

[2] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, K. Claffy, P. Crowley, C. Papadopoulos, L. Wang, and B. Zhang, "Named data networking," *SIGCOMM Comp. Comm. Review*, vol. 44, no. 3, pp. 66–73, Jul. 2014.

[3] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," in *Proc. ACM CoNEXT'09*, Dec. 2009.

[4] A. Dabirmoghaddam, M. Mirzazad-Barijough, and J. J. Garcia-Luna-Aceves, "Understanding Optimal Caching and Opportunistic Caching at the Edge of Information-Centric Networks," in *Proc. ACM ICN'14*, 2014.

[5] 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 *Proc. ACM SIGCOMM'13*, 2013.

[6] 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 *Proc. ACM CoNEXT'14*, 2014.

[7] 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. ACM NoM Workshop*, Jun. 2012.

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

[9] 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.

[10] J. Saltarin, E. Bourtsoulatze, N. Thomos, and T. Braun, "Adaptive video streaming with network coding enabled named data networking," *IEEE Trans. on Multimedia*, vol. 19, no. 10, Oct. 2017.

[11] A. Ramakrishnan, C. Westphal, and J. Saltarin, "Adaptive video streaming over ccn with network coding for seamless mobility," in *Proc. IEEE ISM'16*, Dec. 2016.

[12] J. Llorca, A. Tulino, K. Guan, and D. Kilper, "Network-coded caching-aided multicast for efficient content delivery," in *Proc. ICC'13*, 2013.

[13] J. Wang, J. Ren, K. Lu, J. Wang, S. Liu, and C. Westphal, "An optimal cache management framework for information-centric networks with network coding," in *Proc. IFIP Networking'14*, Jun. 2014, pp. 1–9.

[14] S. Mastorakis, A. Afanasyev, I. Moiseenko, and L. Zhang, "ndnSIM 2: an updated NDN simulator for NS-3," NDN, Tech. Rep. 28, Nov. 2016.

[15] A. Aaron, Z. Li, M. Manohara, J. D. Cock, and D. Ronca, "The Netflix tech blog: Per-title encode optimization," https://medium.com/netflix-techblog/per-title-encode-optimization-7e99442b62a2, Dec. 2015.

[16] 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.

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

[18] E. Yeh, T. Ho, Y. Cui, M. Burd, R. Liu, and D. Leong, "VIP: A Framework for Joint Dynamic Forwarding and Caching in Named Data Networks," in *Proc. ACM ICN'14*, 2014.

[19] S. Li, J. Xu, M. van der Schaar, and W. Li, "Popularity-driven content caching," in *Proc. IEEE INFOCOM'16*, Apr. 2016.

[20] K. Cho, M. Lee, K. Park, T. T. Kwon, Y. Choi, and S. Pack, "WAVE: Popularity-based and collaborative in-network caching for content-oriented networks," in *Proc. IEEE INFOCOM'13 Workshops*, Mar. 2012.

[21] N. Abani, G. Farhadi, A. Ito, and M. Gerla, "Popularity-based partial caching for information centric networks," in *Proc. MedHocNet'16*, 2016.

[22] Q. Wu, Z. Li, and G. Xie, "CodingCache: multipath-aware CCN cache with network coding," in *Proc. ACM ICN'13 Workshop*, Aug. 2013.

[23] P. Chou and Y. Wu, "Network coding for the Internet and wireless networks," *IEEE Sig. Proc. Mag.*, vol. 24, no. 5, pp. 77–85, Sep. 2007.

[24] 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.