NetCodCCN: a Network Coding approach for Content-Centric Networks

Jonnahtan Saltarin*, Eirina Bourtsoulatze*, Nikolaos Thomos† and Torsten Braun*
*University of Bern, Bern, Switzerland
†University of Essex, Colchester, United Kingdom
{saltarin,braun}@inf.unibe.ch, eirina.bourtsoulatze@gmail.com, nthomos@essex.ac.uk

***Abstract*—Content-Centric Networking (CCN) naturally supports multi-path communication, as it allows the simultaneous use of multiple interfaces (e.g. LTE and WiFi). When multiple sources and multiple clients are considered, the optimal set of distribution trees should be determined in order to optimally use all the available interfaces. This is not a trivial task, as it is a computationally intense procedure that should be done centrally. The need for central coordination can be removed by employing network coding, which also offers improved resiliency to errors and large throughput gains. In this paper, we propose NetCodCCN, a protocol for integrating network coding in CCN. In comparison to previous works proposing to enable network coding in CCN, NetCodCCN permits Interest aggregation and Interest pipelining, which reduce the data retrieval times. The experimental evaluation shows that the proposed protocol leads to significant improvements in terms of content retrieval delay compared to the original CCN. Our results demonstrate that the use of network coding adds robustness to losses and permits to exploit more efficiently the available network resources. The performance gains are verified for content retrieval in various network scenarios.**

I. Introduction

In the IP protocol, core of the current Internet architecture, each packet is routed based on the location of the host to which it is addressed. However, nowadays Internet users care more about the content they want to obtain rather than where it is stored. To address this mismatch, Jacobson *et al.* [1] proposed Content-Centric Networking (CCN), a new communication paradigm in which the importance is shifted from *where* the content is located, to *what* the content is. In the CCN model, the content is described by its *name* and the users demand content with the help of Interest messages that contain the name of the requested content. These Interests are transmitted over the network until they reach a node holding a copy of the content whose name matches that of the Interest message. This node creates a Data message that contains a copy of the requested content and sends it back to the requester. The Data message follows the reverse path of that followed by the Interest message. As the Data message is transmitted backwards to the requester, intermediate nodes can store copies of it, so they can reply to future Interests for the same content.

One of the advantages of CCN is that it allows clients to exploit multiple paths in a native way. Clients can simultaneously transmit Interests over all their network interfaces (*e.g.*, LTE and WiFi) to retrieve the content segments that comprise the requested content. This leads to a better use of the network resources and reduces the time needed to collect all the content segments. However, when multiple clients are interested in the same content (*e.g.*, a popular video stream), and/or when the content is distributed across multiple sources (*e.g.*, in a distributed storage system), the optimal content delivery rate is only attained if the segments are delivered over the optimal set of multicast trees [2]. This means that the Data and Interest messages should be transmitted over these multicast trees. The nodes need to know where they need to forward each Interest to follow these multicast trees, which does not scale for large and dynamic topologies. Furthermore, the computation of the optimal set of multicast trees needs a central entity that is aware of the network topology, which is hard to be done in dynamic networks. An alternative solution to the computation of the optimal multicast trees is to use network coding [3]. With network coding all the network nodes perform coding operations on the received packets instead of just replicating and forwarding them as in traditional networks. The receivers decode the information when they receive a decodable set of segments, *i.e.*, as many linearly independent coded segments as the number of source data segments.

The application of network coding in CCN has been explored in [4] where an architecture called *NC3N* is introduced. In this approach, Interests contain information about the content segments available at the client, based on the approach proposed in [5]. Nodes holding content segments that match the name prefix of the Interest reply only if they can provide a network coded content segment that is innovative to the client. However, in the presence of multiple clients, *(i)* the aggregation of Interests is problematic, since Interests for the same content segment from different clients contain different content availability information; and *(ii)*, when a client sends multiple Interests in parallel to receive different content segments, it includes the same information about the content that it already has. This is undesirable as a node that has a matching content segment will reply to these Interests with the same content segment, that will be duplicated for the client. Inspired by [4], CodingCache has been proposed in [6] which uses network coding to replace the content segments in the cache of the network nodes. Due to the increased content segment diversity in the network, the cache hit rate is improved. However, this approach suffers from the same drawbacks as the architecture in [4]. In [7], the multicast delivery of network coded content in Information-Centric Networks is optimized by finding the

978-1-4673-9953-1/16/$31.00 ©2016 IEEE

evolution of the content segments that are stored in the network. The drawback of this approach is that it does not scale well with the number of network nodes, because it needs a central entity that is aware of the network topology and the clients' requests.

In this paper we propose NetCodCCN, a novel protocol that enables network coding in CCN. Our proposed solution solves the shortcomings of the approaches presented in [4] and [6]. Specifically, *(i)* we eliminate the need to include in the Interests the information about the content available at the client, thus, simplifying Interest aggregation; *(ii)* we allow nodes to keep information about the content segments they have sent on each face, reducing the number of duplicate segments; and *(iii)* we allow clients to send multiple Interests in parallel, by modifying the way in which the nodes process the Interest messages.

We have implemented NetCodCCN by making the necessary changes to the CCNx [8] codebase, and performed experiments to compare it to unmodified CCNx. Our results demonstrate that NetCodCCN offers large gains in terms of the time needed to retrieve the original content object. Moreover, it adds robustness to losses and permits to exploit more efficiently the available network resources in multi-source multicast scenarios. To the best of our knowledge, this is the first practical implementation that enables network coding in CCNx.

II. Data retrieval in CCN

We focus on content communication over wired networks represented by directed acyclic graphs $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ and $\mathcal{E}$ denote the set of network nodes and the set of links connecting them, respectively. Each network consists of a set of source nodes $\mathcal{S}$ that generate and/or store content objects, a set of clients $\mathcal{U}$ that demand content objects and a set of intermediate nodes $\mathcal{R}$ through which the content objects are requested and transmitted. Hence, we have $\mathcal{V} = \mathcal{S} \cup \mathcal{U} \cup \mathcal{R}$, where every node $v \in \mathcal{V}$ is connected with its neighboring nodes through a set of faces $\mathcal{F}_v$.

In CCN, content objects are split into smaller segments that fit into Data messages. Each segment is uniquely identified by a name. We denote a content object as $C_p = \{c_{p,1}, \ldots, c_{p,N}\}$ where N is the number of segments in C_p and p is the name of the content object, which serves as a name prefix for the segments. The name of each segment $c_{p,n} \in C_p$ is generated by appending the segment id n to the content object's name p. For instance, the name of the segment $c_{p,1}$ is `/provider/videos/largevideo.h264/1`, where `/provider/videos/largevideo.h264` is the name prefix p and `1` is the segment id.

Each source $s \in \mathcal{S}$ stores content objects that can be requested by the clients. A client $u \in \mathcal{U}$ that is interested in a content object $C_p = \{c_{p,1}, \ldots, c_{p,N}\}$ should send a set of Interest messages $I_p = \{i_{p,1}, \ldots, i_{p,N}\}$, one for each segment. These interests are sent over a set of faces $\mathcal{F}_u^p$ that are configured to forward Interests for content with name prefix p. The information about which faces a node can use to send Interests for specific name prefixes is stored in the *Forwarding Information Base (FIB)* table.

In CCN, each node $v \in \mathcal{V}$ has a cache, or *Content Store (CS)* in CCN terminology, where segments that pass through the node can be stored. These segments can be used later to reply to Interests for segments with a matching name. Therefore, a node $v \in \mathcal{I} \cup \mathcal{S}$ holding a copy of the segment $c_{p,n}$ in its CS replies to any Interest $i_{p,n}$. If the CS of node v does not contain a segment matching the name of the Interest $i_{p,n}$, the node v first checks its *Pending Interest Table (PIT)*, that keeps track of the Interests forwarded by the node and all the faces over which those Interests have arrived. If the node v finds in its PIT an entry that matches the name in the Interest, it knows that it has already forwarded $i_{p,n}$ and hence the segment $c_{p,n}$ is expected. In this case, v does not forward $i_{p,n}$ again, but only adds the face f over which the Interest has arrived to the respective PIT entry. When the PIT does not have any entry that matches the Interest $i_{p,n}$, the node v forwards the Interest to its neighboring nodes over the set of faces $\mathcal{F}_u^p$ configured in its FIB and adds a corresponding entry to the PIT table.

Once the requested segment is found in the CS of an intermediate node or in a source node, it is transmitted to the client in a Data message over the reverse path of that followed by the Interest. When a node $v \in \mathcal{I} \cup \mathcal{U}$ receives a Data message with the segment $c_{p,n}$ over a face f, it first checks its CS. If a segment with the same name exists, the arrived segment $c_{p,n}$ is considered duplicated and it is not transmitted further. If there is no matching segment in the CS, the node checks its PIT for an entry that matches the name of the segment $c_{p,n}$. If there is no matching PIT entry, the segment $c_{p,n}$ is considered unsolicited and it is discarded. If a matching PIT entry exists, the segment is forwarded over all the faces specified in the corresponding PIT entry. Additionally, the segment $c_{p,n}$ may be added to the CS, according to the caching policy.

III. Towards Network coding enabled CCN

In this section we describe the benefits that network coding can bring to CCN. First, we motivate the use of network coding by presenting three scenarios in which CCN does not perform efficiently. Then, we show how network coding can alleviate the drawbacks of CCN in the mentioned scenarios, while also bringing additional benefits.

A. Motivation

Nowadays, communication devices usually come with multiple network interfaces that can be used to gather content, *e.g.*, smart-phones usually have WiFi and 3G/LTE interfaces. However, in the traditional host-centric networking, using multiple interfaces in parallel to retrieve content is a difficult task, as end-to-end connections need to be established for each interface. In CCN, multipath content retrieval is naturally supported, as the clients can distribute all the Interest messages needed to retrieve a content object over all available faces. However, there are some scenarios where CCN does not provide efficient support for multipath content retrieval:

- *Multi-source unicast:* Let us consider the case illustrated in Fig. 1a, where a client u_1 is interested in a content object

C_p. Let us also consider that the N segments that compose C_p are distributed across multiple sources $\mathcal{S}$, such that each source $s \in \mathcal{S}$ contains a subset $\Gamma_p^s \subset C_p$. In this case, the client and the intermediate nodes need to select properly the face over which they send the Interest for each segment, such that it reaches the right source. This is done using the information stored in the FIB table. However, keeping the FIB tables of all the nodes updated for each segment of C_p does not scale well, in particular in large networks and in the presence of unreliable sources that can become available/unavailable at any moment.

• *Single-source multicast:* Let us now consider the case where a single source stores the N segments that compose the content object C_p, but multiple clients are interested in C_p, as illustrated in Fig. 1b. In order to minimize the time needed for each client to receive the complete set of segments that compose C_p, while also minimizing the number of duplicated transmissions of the same segment in the network, the segments need to travel over cost-efficient multicast distribution trees [2]. Finding these trees in a distributed manner is a very complicated task [2], which necessitates the knowledge of the network topology. In CCN, this means that each node of the network should know where each Interest $i_{p,n}$ should be forwarded such that all the Interests for the segment $c_{p,n}$ from different clients are aggregated in the optimal point in the network and the number of duplicated transmissions of $c_{p,n}$ is minimized. In the simple example shown in Fig. 1b, if all the clients send the Interest $i_{p,n}$ over the LTE face, the segment $c_{p,n}$ will be transmitted from the source to the LTE network and then to the clients. However, if some of the clients decide to send the Interest $i_{p,n}$ over the WiFi face, the segment $c_{p,n}$ will also be transmitted from the source to the WiFi network, wasting resources that could have been used to transmit another segment.

• *Multi-source multicast:* Here we consider that multiple clients are interested in content that is distributed across multiple sources. In this case, content delivery through CCN suffers from problems that appear in both the multi-source unicast and the single-source multicast scenarios. This becomes obvious considering the case presented in Fig. 1c. In this case, both clients u_1 and u_2 need to coordinate where to send each Interest, so that the Interests for the same segment are aggregated in the node r_4. Moreover, when the sources have different sets of segments, *i.e.*, $\Gamma_p^{s1} \neq \Gamma_p^{s2} \neq C_p$, the clients also need to know what segments each source stores, in order to avoid sending Interests over the face connecting them directly to the source (*i.e.*, the WiFi Network) that does not hold a copy of the requested segment.

B. Enabling network coding in CCN

The shortcomings of the CCN architecture discussed in Section III-A can be dealt with by enabling network coding [3], a technique in which the segments delivered to the clients are coded at sources and intermediate nodes. The key idea behind introducing network coding in CCN is that clients no longer need to request specific segments, but rather encoded segments as they all have the same amount of information. This removes the need to coordinate the forwarding of Interests and leads to a more efficient use of the available network bandwidth.

Differently from the original CCN where an Interest message $i_{p,n}$ requests a specific segment $c_{p,n}$, in a network coding enabled CCN variant an Interest message $\hat{i}_p$ requests a coded segment $\hat{c}_p$, without specifying a particular segment id. A source or intermediate node can reply to these Interests with network coded segments, generated by combining the segments from its CS that match the name prefix p. In matrix form, this can be expressed as $\hat{c}_p = \mathbf{A} \cdot \mathbf{CS_p}$, where $\mathbf{A}$ is a vector of coding coefficients drawn from a finite field, and $\mathbf{CS_p}$ is a vector of the segments from the node's CS that match the prefix p. When the coding coefficients in $\mathbf{A}$ are chosen uniformly at random from a large enough finite field, the generated segments have high probability of being linearly independent, and thus innovative. Whenever a client interested in a content object C_p collects N innovative coded segments $\hat{c}_p$, it can decode the original segments that compose C_p.

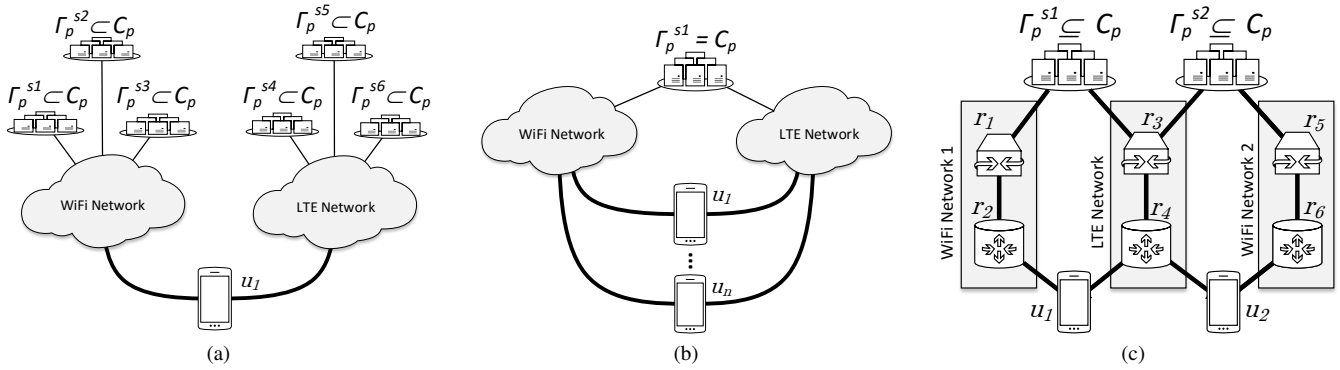
To illustrate the benefits that network coding brings to CCN, let us revisit the scenarios described in Section III-A.

• *Multi-source unicast:* Differently to the original CCN, when network coding is allowed, the client and the intermediate nodes do not need to know over which face they can reach a particular source, since they send Interests for coded segments rather than for specific segments. This implies that the FIB tables of the clients and intermediate nodes do not need an entry for each segment, but only a single entry for the name prefix is enough. Each source then replies to these Interests with coded segments $\hat{c}_p$, generated by combining the segments that match the name prefix p.

• *Single-source multicast:* When network coding is allowed, the clients do not need to coordinate what Interests they send over each face, since all of them are for coded segments. Thus, when all the clients send Interests $\hat{i}_p$ requesting a coded segment over a face (*e.g.*, LTE), they will be aggregated by the intermediate nodes, and only one Interest requesting a coded segment will reach the source.

• *Multi-source multicast:* In this scenario, when network coding is allowed, neither clients nor intermediate nodes need to coordinate the forwarding of the Interests, since they are for coded data and can be satisfied by any coded segment.

Network coding has also been shown to improve the throughput when bottlenecks are present in the network and the resiliency to packet erasures. In order to illustrate these benefits, let us consider the scenario in Fig. 1c, commonly known as the butterfly network. We consider that two clients are interested in a content object $C_p = \{c_{p,1}, c_{p,2}\}$ that is distributed across both sources, such that the source node s_1 holds a copy of $c_{p,1}$ (*i.e.*, $\Gamma_p^{s1} = \{c_{p,1}\}$) and the source node s_2 holds a copy of $c_{p,2}$ (*i.e.*, $\Gamma_p^{s2} = \{c_{p,2}\}$). In this case, if network coding is not enabled, the link between nodes r_3 and r_4 becomes a bottleneck, since the only way in which the clients u_1 and u_2 can get $c_{p,2}$ and $c_{p,1}$ respectively is through node r_4. Thus, the Interests sent by the clients u_1 and u_2 cannot be aggregated



(a) (b) (c)

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

in the node r_4, since they are for different segments, and one of the clients will receive the content with higher delay. In contrast, when network coding is enabled, the Interests sent by the clients u_1 and u_2 can be aggregated in the node r_4, as they are both for coded data. If the node r_3 applies network coding to the segments received from the sources, the resulting coded segment will be useful for both clients u_1 and u_2.

C. Challenges

As discussed in Section III-B, enabling network coding in CCN nodes brings benefits that can potentially improve the performance of content object retrieval under certain scenarios. However, some issues arise when the Interest messages do not specify the segment id.

One of the issues that arises is that any node that has a single coded segment $\hat{c}_p$ cached in its CS will reply with this segment to any Interest $\hat{i}_p$, as the name prefix in the Interest matches that of the cached segment $\hat{c}_p$. This is undesirable, since the intermediate nodes will always reply with the same cached segment $\hat{c}_p$, while clients need to receive N innovative coded segments in order to decode the original segments. Therefore, the intermediate nodes need a way to determine when they cannot provide a coded segment that is innovative to the client, and thus a new coded segment has to be retrieved. In [4] the authors propose to solve this problem by allowing the clients to include information about the coded segments they have collected so far. Intermediate nodes reply to an Interest only if they can provide innovative information. However, it is not clear how intermediate nodes can aggregate Interests with different information from the clients.

Another challenge that emerges when network coding is enabled in CCN is related to the *pipelining* procedure, *i.e.*, a client sending multiple concurrent Interests for different segments of the same content object. In the original CCN when a node receives an Interest that it cannot satisfy with content stored in its CS, the node checks its PIT. If the node finds an entry in the PIT indicating that an Interest for the same name has been received previously over the same face, it will consider this new Interest as a duplicate and do not forward it. Since Interests for different segments have different names, as the segment id is appended to the name prefix, pipelining is supported. However, when network coding is enabled in CCN, concurrent Interests for different coded segments of the same object have the same name. Therefore, pipelining is not supported as the Interests will be considered duplicated.

These challenges are addressed in our practical implementation of NetCodCCN, presented in section IV.

IV. The NetCodCCN Protocol

In this section we present NetCodCCN, a practical implementation of a network coding enabled content-centric networking protocol, based on the CCN architecture [1]. We start by defining the content segmentation and naming scheme in the proposed protocol. Then, we describe how Interests and Data messages are processed in NetCodCCN.

A. Content Segmentation

As in the CCN protocol, in NetCodCCN the content objects are split into smaller segments, $C_p = \{c_{p,1}, \ldots, c_{p,N}\}$, that fit into Data messages. Network coded segments $\hat{c}_{p,\mathbf{g}}$ are random linear combinations of original segments with name prefix p. Similarly to [9], $\mathbf{g}$ denotes the encoding vector associated with the network coded segment $\hat{c}_{p,\mathbf{g}}$. At the source nodes, network coded segments are generated by randomly combining the set of non-coded segments with name prefix p that are stored in their CS. Thus, $\hat{c}_{p,\mathbf{g}} = \mathbf{A} \cdot \mathbf{CS_p} = \sum_{l=1}^{L} a_l \cdot c_{p,l}$ where $\mathbf{A} = a_1, \ldots, a_L$ is a vector of coding coefficients randomly selected from a finite field, $\mathbf{CS_p} = \{c_{p,1}, \ldots, c_{p,L}\}$ is the vector of segments with name prefix p stored in the CS of the node, and $L = |\mathbf{CS_p}|$ is the size of the vector $\mathbf{CS_p}$, with $L \leq N$. At the intermediate nodes, network coded segments are generated by randomly combining the set of coded segments with name prefix p that are stored in their CS. Thus, $\hat{c}_{p,\mathbf{g}} = \sum_{l=1}^{L} a_l \cdot \hat{c}_{p,\mathbf{g_l}}$. The encoding vector $\mathbf{g}$ is generated as $\mathbf{g} = \sum_{l=1}^{L} a_l \cdot \mathbf{g_l}$, where $\mathbf{g_l}$ is the coding vector associated with the lth segment. For the special case when the lth segment is a non-coded segment, the vector $\mathbf{g_l}$ is a unit vector of size N that has value 1 in the lth position and 0 otherwise.

The clients and intermediate nodes keep track of the received innovative encoding vectors for prefix p in an encoding matrix $\mathbf{G_p} = [\mathbf{g_1}; \ldots; \mathbf{g_L}]$, with $L = |\mathbf{CS_p}|$. This allows the original set of segments to be retrieved at the clients by performing

Gaussian elimination when the matrix $\mathbf{G_p}$ is full rank. Since the matrix $\mathbf{G_p}$ only contains linearly independent encoding vectors, its rank can be computed as $Rank(\mathbf{G_p}) = L$. Thus, it is full rank when $L = N$.

The use of the encoding vectors introduces a communication overhead which depends on the number of content segments. To limit this overhead, we adopt the concept of *generations* [9], according to which the original set of segments that compose C_p is partitioned into smaller groups of segments, hereafter called generations, and the coding operations are restricted only between segments that belong to the same generation. For example, let us consider that C_p is partitioned into K generations of H_k segments each. Hence, the H_k segments of the kth generation are denoted as $C_{p,k} = \{c_{p,k,1}, ..., c_{p,k,H_k}\}$, where k is the *generation id*. The size of the generation, H_k, controls the tradeoff between the decoding delay, the segment diversity and the overhead required to communicate the encoding vector. Overall, the encoding vectors do not pose any limitations to our system as there are approaches to compress them efficiently [10], [11]. In order to avoid mixing segments from different generations, the segments are tagged with the generation id.

B. Content Naming

From the discussion above, it is obvious that the naming in NetCodCCN should have two additional components compared to the CCN protocol, namely the encoding vector $\mathbf{g}$ and the generation id k. For example, let us consider a content object C_p with name p = `/provider/videos/largevideo.h264`, that is partitioned into K generations of $H_k = 4$ segments each. Thus, in NetCodCCN the first segment of the kth generation, $c_{p,k,1}$, associated with the coding vector $[1, 0, 0, 0]$, is named $\{p, k, 1\}$ = `/provider/videos/largevideo.h264/k/1000`.

For the sake of clarity, and without loss of generality, hereafter we consider that the name prefix p in $\hat{c}_{p,\mathbf{g}}$, contains both the name prefix and the generation id. Note that, the proposed naming scheme is compatible with the original CCN and can support the delivery of non-coded segments.

C. Interest Message Processing

Similarly to CCN, in our protocol the data communication is triggered by the clients who send Interest messages $\hat{i}_p$ for data with name prefix p. In the proposed NetCodCCN protocol, the Interests have a *NetworkCodingAllowed* field that takes the value "1" when network coded segments are expected, otherwise, the field is not present or its value is set to "0". Nodes receiving an Interest with the *NetworkCodingAllowed* field activated process the Interest messages following the NetCodCCN procedure explained below and summarized in Algorithm 1. Otherwise, the Interests are treated following the original CCN procedures.

When a node $v \in \mathcal{V}$ receives an Interest $\hat{i}_p$ for a network coded segment over the face f, it either *(i)* replies to the Interest with a coded segment generated with the set of segments $\mathbf{CS_p}$; or *(ii)* forwards the Interest to other nodes in order to receive a new linearly independent segment. This is further explained in the following.

Algorithm 1 Interest Processing in NetCodCCN

Require: $\hat{i}_p, f$, $\mathbf{CS_p} \leftarrow$ segments that match p in the CS
1: **if** $Rank(\mathbf{G_p}) = N$ **then** {Generation is decodable}
2: $\quad \hat{c}_{p,\mathbf{g}} \leftarrow \sum_{l=1}^{L} a_l \cdot \hat{c}_{p,\mathbf{g}_l}$
3: $\quad$ Send segment $\hat{c}_{p,\mathbf{g}}$ over face f
4: **else**
5: $\quad \xi^{p,f} = Rank(\mathbf{G_p}) - \sigma_{sent}^{p,f}$
6: $\quad$ **if** $\xi^{p,f} > 0$ **then**
7: $\quad\quad \hat{c}_{p,\mathbf{g}} \leftarrow \sum_{l=1}^{L} a_l \cdot \hat{c}_{p,\mathbf{g}_l}$
8: $\quad\quad$ Send segment $\hat{c}_{p,\mathbf{g}}$ over face f
9: $\quad$ **else**
10: $\quad\quad$ InsertPIT (p, f)
11: $\quad\quad$ **if** $\sigma_{fwd}^{p} \leq \sigma_{pend}^{p,f}$ **then**
12: $\quad\quad\quad$ PropagateInterest($\hat{i}_p$)
13: $\quad\quad$ **end if**
14: $\quad$ **end if**
15: **end if**

Replying to an Interest: The node v replies to an Interest $\hat{i}_p$ when *(i)* it has collected enough network coded segments to decode the original set of segments C_p; or when *(ii)* a segment generated by the node v has high probability to be innovative for the node connected through the face f from where the Interest arrived. The number of coded segments $\xi^{p,f}$ that can be generated by the node v and have high probability to be innovate for the node connected through the face f, is given by $\xi^{p,f} = rank(\mathbf{G_p}) - \sigma_{sent}^{p,f}$. The parameter $\sigma_{sent}^{p,f}$ denotes the number of coded segment that have been previously sent over the face f. When $\xi^{p,f}$ is greater than 0, the node v generates a new coded segment and sends it over the face f.

Forwarding an Interest: The node v forwards an Interest $\hat{i}_p$ to its neighbors when $\xi^{p,f}$ is equal to 0, since it needs a new segment that increases the rank of $\mathbf{G_p}$ before it can reply to the Interest $\hat{i}_p$. As in CCN, prior to forwarding an Interest the node v checks its PIT. However, in order to support pipelining, in NetCodCCN the PIT verification procedure is modified. Specifically, if the node finds a matching PIT entry indicating that an Interest for the same name prefix p has been previously received over the same face f, the Interest $\hat{i}_p$ is not considered duplicated, but it is treated as a request for an additional network coded segment from the same face. This means that the face f can appear multiple times in the PIT entry for the name prefix p. To decide whether the Interest $\hat{i}_p$ should be forwarded, the node v computes the number ν_p of innovative coded segments matching the name prefix p that it expects to receive before the Interest $\hat{i}_p$ expires. If $\nu_p > \sigma_{pend}^{p,f}$, where $\sigma_{pend}^{p,f}$ is the number of Interests received over the face f that are pending for a reply, the node v does not forward the Interest $\hat{i}_p$, as it expects to receive enough coded segments to satisfy all the pending Interests, including the received Interest $\hat{i}_p$. Otherwise, if $\nu_p \leq \sigma_{pend}^{p,f}$, the node v forwards the Interest.

To compute the expected value of ν_p, the node v needs

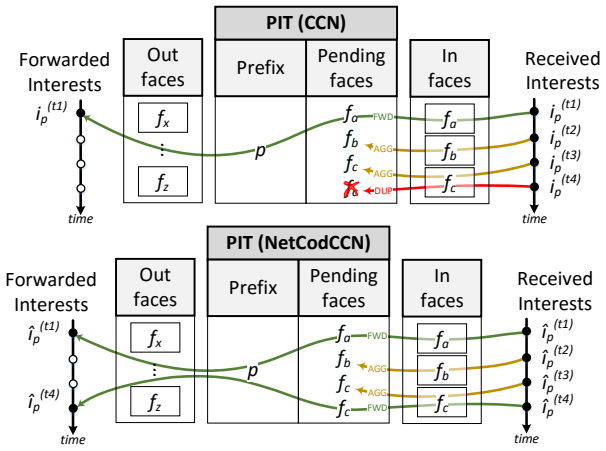


Fig. 2. Comparison of the PIT in CCN and in NetCodCCN.

a probabilistic model that takes into consideration the loss rate of the system and the delays that segments may suffer, among other variables. For the sake of simplicity, we make the assumption that nodes follow a simple model in which any forwarded Interest brings an innovative segment before its expiration. In this case, $\nu_p = \sigma^p_{fwd}$, where σ^p_{fwd} is the total number of Interests that have been forwarded by the node v for the name prefix p. Thus, the node v forwards the Interest if $\sigma^p_{fwd} \leq \sigma^{p,f}_{pend}$. This assumption is close to the model followed by CCN nodes, where Interests are not forwarded if they match a PIT entry, since the previously sent Interest is expected to bring the requested segment. This is because the CCN nodes also consider that any forwarded Interest will bring the requested segment before its expiration.

To further illustrate the difference in the processing of the Interests between CCN and NetCodCCN, let us consider the example presented in Fig. 2. In both CCN and NetCodCCN, the first three Interests are processed in a similar way: the Interest $i_p^{(t1)}$ is forwarded and the Interests $i_p^{(t2)}$ and $i_p^{(t3)}$ are aggregated. However, the Interest $i_p^{(t4)}$ is processed differently in both schemes: in CCN, it is considered duplicated since an Interest for the same name has been received over the face f_c previously; while in NetCodCCN, the Interest is forwarded since $\sigma^p_{fwd} = \sigma^{p,f_c}_{pend} = 1$.

When a segment with name prefix p is removed from the CS of node v (*e.g.*, when the CS eviction policy decides that segments of name prefix p needs to be removed from the cache), the corresponding vector should also be removed from $\mathbf{G_p}$, and $\sigma^{p,f}_{sent}$ needs to be decreased by 1 for all the faces.

D. Data Message Processing

The use of network coding in CCN also imposes modifications on the Data message forwarding procedure. Specifically, when a node $v \in \mathcal{V}$ receives a coded segment $\hat{c}_{p,\mathbf{g}}$ over the face f, it should determine whether the segment is innovative or not. The segment $\hat{c}_{p,\mathbf{g}}$ is innovative for the node v if the encoding vector $\mathbf{g}$ is linearly independent of the encoding vectors in $\mathbf{G_p}$, *i.e.*, if it increases the rank of $\mathbf{G_p}$. A non-innovative segment is considered as a duplicated segment and therefore it is discarded by the node v. When the segment $\hat{c}_{p,\mathbf{g}}$ is innovative, the node v inserts it into its CS, and updates the encoding matrix $\mathbf{G_p}$ that should now contain the received encoding vector $\mathbf{g}$. Then, the node v checks its PIT. If the node v finds a matching PIT entry, meaning that an Interest with name prefix p is pending, it generates a network coded segment $\hat{c}_{p,\mathbf{g}'} = \sum_{l=1}^{L} a_l \cdot \hat{c}_{p,\mathbf{g_l}}$ and sends it once over each of the faces specified in the PIT entry. It is important to note that since the face f may appear multiple times in the PIT entry for the name prefix p, as a consequence of allowing pipelining, a coded segment sent over face f consumes only one of the appearances of f in the corresponding PIT entry. If no matching PIT entry for the name prefix p is found, the received coded segment is considered unsolicited and it is not further transmitted. However, it can be kept in the CS, according to the CS insertion policy, as it can be useful to serve future Interests. This procedure is outlined in Algorithm 2.

Algorithm 2 Content Processing in NetCodCCN

Require: $\hat{c}_{p,\mathbf{g}}$
if $\mathbf{g}$ increases the rank of $\mathbf{G_p}$ **then**
  Insert $\hat{c}_{p,\mathbf{g}}$ in the CS
  $PIT_p \leftarrow$ PIT entry for prefix p
  if $PIT_p \neq \varnothing$ **then**
    for all f in PIT_p **do**
      if f has not been served **then**
        $\hat{c}_{p,\mathbf{g}'} = \sum_{l=1}^{L} a_l \cdot \hat{c}_{p,\mathbf{g_l}}$
        Send segment $\hat{c}_{p,\mathbf{g}'}$ over face f
        $\sigma^{p,f}_{sent} \leftarrow \sigma^{p,f}_{sent} + 1$
        Remove one appareace of f from PIT_p
      end if
    end for
  end if
else
  Discard $\hat{c}_{p,\mathbf{g}}$
end if

It is important to note that network coding adds complexity to both the Interest and Data message processing in NetCodCCN. Performing algebraic operations on the segments before forwarding them adds, effectively, some complexity to the CCN node. In particular, as we have seen in Section IV-A, a node generates a new coded segment as $\hat{c}_{p,\mathbf{g}} = \sum_{l=1}^{L} a_l \cdot \hat{c}_{p,\mathbf{g_l}}$. If we consider that the operations are performed in a finite field of size 2^8 and that segments are of size X symbols, each time a node needs to generate a new coded segment, it performs $X \cdot L$ multiplications and $X \cdot (L-1)$ additions. This complexity does not pose limitations to our scheme as there are efficient implementations of network coding [12]. Further, as it has been shown in [13], a network coding coder and decoder can operate at wire-speed with rates of up to 1000Mbps.

V. NetCodCCN Evaluation

In this section, we evaluate the performance of NetCodCCN in various scenarios, and compare the results to the performance

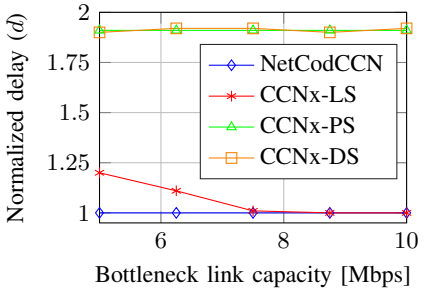


Fig. 3. Normalized delivery delay vs. the capacity of the bottleneck link in the butterfly network.

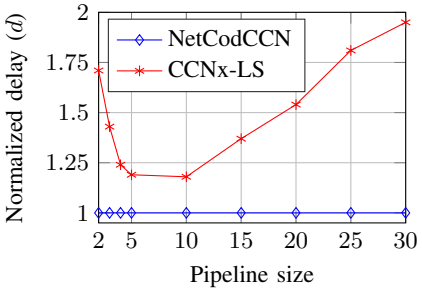


Fig. 4. Normalized delivery delay vs. the pipeline size in the butterfly network.

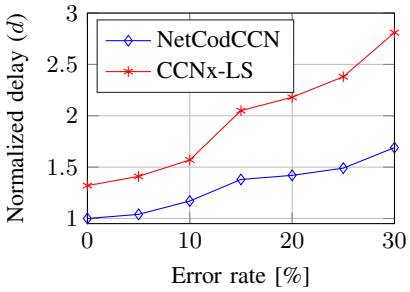


Fig. 5. Normalized delivery delay vs. the error rate in the butterfly network.

of the standard CCN. First, we describe the simulation setup. Then, we evaluate the performance of NetCodCCN in the butterfly network. This toy network provides a controllable environment which permits to verify the expected behavior of NetCodCCN, and facilitates the illustration of its benefits. Finally, we present the simulation results in a more realistic network topology, which is generated based on real network measurements taken from the Planetlab project [14].

A. Simulation Setup

We implemented NetCodCCN by integrating the changes to the CCN architecture described in Section IV into the CCNx 0.8.2 [8] code, and we compare its performance to that of the unmodified CCNx. The network topology is simulated using the NS-3 network simulator [15]. The software forwarders/routers for CCNx and NetCodCCN are installed on NS-3 nodes using the Direct Code Execution framework (DCE) [16].

We consider that the clients are interested in a content object composed of $N = 100$ segments. The size of each segment is 5KB. The segments are stored in a set of sources that are connected to the clients through a network of intermediate nodes. We consider that the intermediate nodes have sufficient CS space to store all the incoming segments. We assume that the $N = 100$ source segments comprise a single generation, *i.e.*, $H = N$ and $K = 1$. The finite field in which the network coding operations are performed is of size 2^8. In order to evaluate our protocol in a challenging scenario, we consider that all the clients send Interests during the same interval of time. In this way, we demonstrate that by using our protocol, the nodes are able to aggregate Interests adequately.

For the evaluation of CCNx, we consider the three main Interest forwarding strategies implemented in CCNx 0.8.2, and described in [17]:

- The *default* (DS) strategy selects the fastest responding face based on the face statistics.
- The *loadsharing* (LS) strategy distributes the Interest forwarding load over all the available faces, sending each Interest over the face with the smallest pending Interest queue.
- The *parallel* (PS) strategy sends the Interests in parallel over all the faces indicated in the FIB.

For the evaluation of NetCodCCN, we always consider the *parallel* strategy, since by sending a single Interest over all its faces the client can receive multiple useful (*i.e.*, linearly independent) segments.

To evaluate the performance of NetCodCCN, we measure the time $\Delta t_{measured}$ that a client needs in order to get the N segments available at the sources. In the standard CCN, $\Delta t_{measured}$ is defined as the elapsed time between the transmission of the first Interest and the reception of the Nth missing segment. In NetCodCCN, $\Delta t_{measured}$ is defined as the elapsed time between the transmission of the first Interest and the reception of the Nth linearly independent network coded segment, which permits to decode the whole generation of segments. We consider that clients can have heterogeneous network resources, thus, in order to make a fair comparison of the delivery delay, we define the *normalized delivery delay* as $d = \Delta t_{measured}/\Delta t_{min}$, where Δt_{min} is the theoretical lower bound on the time that a client would need in order to receive all the segments if it was alone in the network and was able to receive at max-flow rate. Thus, a normalized delivery delay equal to 1 means that the client was able to receive the complete set of segments at the maximum rate. Note that $\Delta t_{measured} \geq \Delta t_{min}$, or equivalently, $d \geq 1$ always holds.

B. Butterfly Topology

We begin by evaluating NetCodCCN in the butterfly topology presented in Fig. 1c. We consider that every segment is stored randomly in at least one of the two sources, and a copy of the same segment is also placed in the CS of the other source with a duplication probability $\phi \in [0, 1]$. We set the capacity of every link in the network to $5Mbps$.

In the first set of experiments, we consider that $\phi = 1$, *i.e.*, both sources hold a copy of each segment in their CS. This corresponds to the single source multicast case presented in Section III-A. In this case, clients u_1 and u_2 can reach a copy of any segment over any of their faces. However, as explained in Section III-B, with the original CCN protocol the maximum performance can be achieved only if both clients coordinate and send Interest messages for the same segments over the faces that connect them to the node r_4. In contrast, when network coding is employed, the need for coordination is eliminated, since clients do not send Interests for a specific segment but rather for any network coded segment.

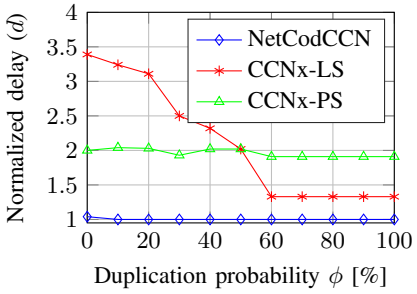


Fig. 6. Normalized delivery delay vs. source content duplication probability in the butterfly network.

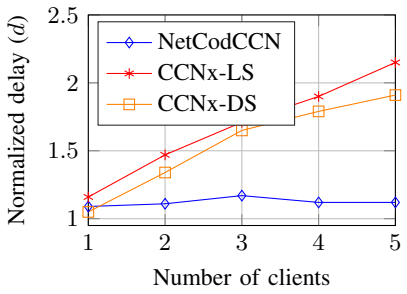


Fig. 7. Normalized delivery delay vs. the number of clients in the network in the PlanetLab topology.

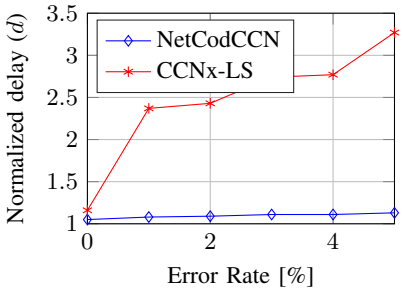


Fig. 8. Normalized delivery delay vs. the segment transmission error rate in the PlanetLab topology.

Fig. 3 depicts the normalized delivery delay as a function of the capacity of the bottleneck link between the nodes r_3 and r_4. We can see that NetCodCCN achieves the optimal performance in the whole range of link capacity values. This is due to the fact that network coding removes the need for coordinating the forwarding of Interest messages. In contrast, the CCN forwarding strategies perform poorly and only the LS strategy can achieve the performance of NetCodCCN but it requires significantly higher link capacity. When the bottleneck link has the same capacity as all the other links, the average delivery time d of CCNx-LS is approximately 1.2 times the minimum delivery delay, Δt_{min}. This is caused by the randomness introduced by the LS strategy when choosing the faces over which Interests are transmitted when all the faces have the same load. This creates two extreme cases. In one case, all the Interests sent by both clients to node r_4 are the same, thus d tends to one. In the other case, all the Interests are different, thus d tends to 1.33. This happens because each node receives 2/3 of the segments through the link connecting them to the sources, and 1/3 over the face connecting them to the node r_4, which means that 2/3 of the total segments travel on the bottleneck link. With the DS and the PS strategies, the average delivery time is close to 2, as expected. With the DS strategy, the face connecting the clients to the sources is chosen as the best, and thus most of the segments are received over that face. With the PS strategy, each client forwards every Interest over both faces, thus bringing one copy of each segment over each face.

We now investigate how the number of concurrent Interests that a client can send, also known as the pipeline size, affects the performance in terms of the average normalized delivery delay. As shown in Fig. 4, the performance of CCN is optimized for a pipeline size value between 5 and 10, where the normalized delivery delay seen by the clients is 1.2. This is due to the fact that clients need to send at least 4 Interest messages over the faces connecting them to the node r_4 in order to create a continuous flow of segments. Since the LS strategy distributes the Interests over all available faces, a client has to send 4 Interests over each face while it also has sent 3 or 4 Interests over the other face, which amounts to 7 or 8 Interests in total. For smaller pipeline sizes, the continuous flow is not set, while for larger pipeline sizes the number of Interests sent over the bottleneck link increases, thus worsening the client coordination problem. In contrast, the performance of NetCodCCN is not affected by the pipeline size, as can be verified in Fig. 4. This can be explained by the fact that NetCodCCN eliminates the necessity that the clients request the same segments over the bottleneck link. For the rest of the experiments, without loss of generality, we choose a pipeline size of 10.

In Fig. 5, we depict the influence of the segment loss rate on the performance of NetCodCCN and of the original CCN. We consider losses that are caused both by the transmission losses and the errors during the processing of the segments. We can see that the performance of CCN with the LS strategy degrades faster than the performance of NetCodCCN as the segment error rate increases. This is caused by the fact that in CCN, the client will be able to react to a segment loss only when the corresponding Interest expires, since any earlier re-transmission of an Interest with the same prefix will be prevented by the PIT. Instead, with NetCodCCN, the clients can send Interests for new coded segments until they have a sufficient number of coded segments in order to recover the original ones. It is important to note that the maximum amount of concurrent Interests that a client can send is controlled by the pipeline size.

Finally, we evaluate the performance of NetCodCCN for different values of the duplication probability ϕ. This corresponds to the multi-source multicast case presented in Section III-A. In CCNx, when $\phi < 1$, the clients should not only coordinate the requests sent over the bottleneck link as in the previous scenario, but they also should have the knowledge of the segments that each source stores, in order to avoid sending Interests over the face connecting them directly to the source that does not hold a copy of the requested segment. In Fig. 6, we can see that CCN with the LS strategy takes 3.4 times longer to deliver all the segments to the clients, when each segment is stored only in one of the sources. When the PS strategy is employed, the clients do not need to know how the content is distributed since each Interest message is sent over both faces. However, since a copy of every segment cross the bottleneck link, the traffic over the bottleneck link is doubled compared to the network coding case. When the probability that the segments are stored in both sources increases, the performance of CCNx with the LS strategy improves, but eventually saturates at 1.2

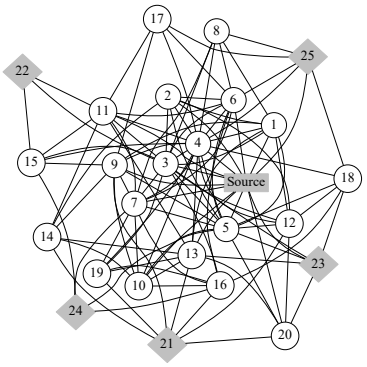


Fig. 9. Planetlab topology used.

times the minimum delay, which is consistent with the results depicted in Fig. 4.

C. Planetlab Topologies

We now evaluate our protocol in more realistic network topologies captured by the PlanetLab project [14]. We use the network topology shown in Fig. 9 that consists of one source node, 5 client nodes and 20 intermediate nodes. The links connecting the nodes have a capacity of 12Mbps. The topology was generated using the procedure described in [18]. We measure the normalized delivery delay d for each client and then compare its average.

First, we investigate how the performance is affected by the number of clients in the network. In Fig. 7, we can see that with a single client (in this case node 24), NetCodCCN and CCNx perform similarly. In this case, network coding does not introduce any gains since there is only one client in the network and no losses are considered. However, the performance of CCNx starts to degrade with the introduction of more clients, as they start to compete for the network resources. In contrast, we can see that the performance of NetCodCCN does not deteriorate with the addition of new clients to the network topology. These results show that the NetCodCCN protocol uses more efficiently the available network resources.

We also evaluate how the error in segment transmission affects the performance of the NetCodCCN for larger topologies. For this evaluation, we choose to keep only one client, in order to compare the results with the performance of the CCN. As with the butterfly topology, we consider losses that are caused both by transmission losses and errors introduced during the processing of the segments. In Fig. 8, we can see that NetCodCCN maintains the delivery delay close to the expected one, while the performance of CCN degrades very fast with the introduction of errors. As in the butterfly topology, this fast degradation is due to the fact that when a segment is lost, the client needs to wait until the corresponding Interest expires before it can re-send a new one.

VI. Conclusions

In this paper, we have presented NetCodCCN, a protocol that integrates network coding in CCN. In NetCodCCN, the clients express Interest messages for coded segments of a given prefix instead of asking a specific segment as in CCN. The network nodes combine the Data messages by means of RLNC before forwarding them in order to take advantage of the network diversity. Our protocol is able to *(i)* simplify the aggregation of Interests for coded content; *(ii)* reduce the number of duplicate segments; and *(iii)* allow clients to send multiple Interests for the same content in parallel. The overall system has been tested in networks with multiple clients and sources, where we have observed large performance gains in terms of the time needed to retrieve the demanded content.

Our future research includes the investigation of optimal Interest forwarding strategies that enable flow control in the case of multiple different content objects. We will also consider the transmission of video content characterized by strict delivery deadlines. Furthermore, we will focus on enabling content security in NetCodCCN.

Acknowledgment

This work has been partially funded by the Swiss National Science Foundation under grant number 149225.

References

[1] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking Named Content," in *ACM CoNEXT'09*, Dec. 2009.
[2] Y. Wu, P. Chou, and K. Jain, "A Comparison of Network Coding and Tree Packing," in *IEEE ISIT'04*, Jun. 2004.
[3] R. Ahlswede, N. Cai, S.-Y. Li, and R. Yeung, "Network Information Flow," *IEEE Trans. Inf. Theory*, vol. 46, no. 4, pp. 1204–1216, Jul. 2000.
[4] 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 *1st ACM NoM Workshop*, Jun. 2012.
[5] J. Sundararajan, D. Shah, M. Medard, S. Jakubczak, M. Mitzenmacher, and J. Barros, "Network Coding Meets TCP: Theory and Implementation," *Proc. IEEE*, vol. 99, no. 3, pp. 490–512, Mar. 2011.
[6] Q. Wu, Z. Li, and G. Xie, "CodingCache: Multipath-Aware CCN Cache with Network Coding," in *3rd ACM ICN Workshop*, Aug. 2013.
[7] J. Llorca, A. Tulino, K. Guan, and D. Kilper, "Network-Coded Caching-Aided Multicast for Efficient Content Delivery," in *IEEE ICC'13*, Jun. 2013.
[8] "CCNx ®, version 0.8.2," http://www.ccnx.org/releases/ccnx-0.8.2/doc/.
[9] P. Chou and Y. Wu, "Network Coding for the Internet and Wireless Networks," *IEEE Signal Process. Mag.*, vol. 24, no. 5, pp. 77–85, Sep. 2007.
[10] N. Thomos and P. Frossard, "Toward one Symbol Network Coding Vectors," *IEEE Commun. Lett*, vol. 16, no. 11, pp. 1860–1863, Nov. 2012.
[11] D. E. Lucani, M. V. Pedersen, J. Heide, and F. H. P. Fitzek, "Fulcrum Network Codes: A Code for Fluid Allocation of Complexity," *available at http://arxiv.org/abs/1404.6620*, 2014.
[12] M. Pedersen, J. Heide, P. Vingelmann, and F. Fitzek, "Network coding over the $2^{32} - 5$ prime field," in *IEEE ICC'13*, Jun. 2013.
[13] M. Zhang, H. Li, F. Chen, H. Hou, H. An, W. Wang, and J. Huang, "A general co/decoder of network coding in hdl," in *2011 Intl. Symp. on Network Coding*, Jul. 2011.
[14] "PlanetLab," https://www.planet-lab.org/.
[15] "The network simulator - ns3," http://www.nsnam.org/.
[16] "Direct Code Execution (DCE)," https://www.nsnam.org/overview/projects/direct-code-execution/.
[17] "CCNDC(1) Manual Page, Project CCNx ®, version 0.8.2," https://www.ccnx.org/releases/ccnx-0.8.2/doc/manpages/ccndc.1.html.
[18] N. Cleju, N. Thomos, and P. Frossard, "Selection of Network Coding Nodes for Minimal Playback Delay in Streaming Overlays," *IEEE Trans. Multimedia*, vol. 13, no. 5, pp. 1103–1115, Oct 2011.