Integrated Network Coding and Caching in Information-Centric Networks

Revisiting pervasive caching in the ICN framework

Abhiram Ravi · Parmesh Ramanathan · Krishna M. Sivalingam

Abstract Information-Centric Networks (ICNs) replace IP addresses with content names at the thin waist of the Internet hourglass, thereby enabling pervasive router-level caching at the network layer. In this paper, we revisit pervasive content caching and propose an algorithm for cache replacement at ICN routers by incorporating principles from network coding, a technique used to achieve maximum flow rates in multicast. By introducing a low computational cost in the system, Network Coded Caching better utilizes the available small storage space at the routers to cache more effectively in the network. Results of our experiments on the GENI (Global Enterprise for Network Innovations) testbed demonstrating the performance of our algorithm on a real network are included in the paper. We evaluate the algorithm in two different traffic scenarios (i) Video-on-Demand (VoD) (ii) Zipf-based Web traffic. Working with the Named Data Networking implementation of ICN, we also present the additional headers and logical components that are needed to enable Network Coded Caching. In a nutshell, we show that an integrated coding-and-caching strategy can provide significant gains in latency and content delivery rate for a small computational overhead.

Abhiram Ravi
Department of Computer Science and Engineering, Indian Institute of Technology Madras, Chennai, IN
India-UK Advanced Technology Center of Excellence in Next Generation Networks, Systems and Services (IU-ATC)
E-mail: abhiram@cse.iitm.ac.in

Parmesh Ramanathan
University of Wisconsin-Madison, Madison, USA
E-mail: parmesh@ece.wisc.edu

Krishna M. Sivalingam
Department of Computer Science and Engineering, Indian Institute of Technology Madras, Chennai, IN
India-UK Advanced Technology Center of Excellence in Next Generation Networks, Systems and Services (IU-ATC)
E-mail: skrishnam@cse.iitm.ac.in

1 Introduction

Recent interest in information-centric paradigms [1] of internet services has been motivated by an envisioning of a better internet where the focus of the consumer is to retrieve some desired *content* as opposed to connecting to an end host. These paradigms provide a mechanism for hosts to publish and users to request content at the network layer by replacing IP packets, which constitute the current narrow waist of the Internet hourglass, with content chunks. Furthermore, the notion of content as a *primitive* entity [2] implicitly enables several potential benefits that are crucial to the current Internet (e.g., caching, security, reliability, mobility).

In this paper, we are concerned with effectively utilizing the pervasive and typically small sized router-caches that the ICN design enables. Our work is heavily motivated by the massive growth of video content in today's internet traffic, with IP Video traffic being estimated to rise to 73% of the global IP traffic by 2017 [3]. To address the challenges in effective content delivery, we propose a caching strategy that incorporates principles from network coding, a technique that is used to achieve maximum-flow content delivery rates in multicast. The key idea is that the routers linearly combine content items when their content stores reach their capacity, and store these encoded data items, each of them consuming the same storage space as that of a non-encoded content item. Note that unlike multicast, where the objective is to maximize the rate at which content is delivered to the clients while ignoring the benefits of temporal locality in client requests, our objective here is to opportunistically cache content items

for future use i.e., on the lines of an Least-Recently-Used (LRU) like policy. Our goal is to show that network coded caching can increase the effective utilization of the total distributed storage space available on the network and thus improve latency measures observed at the clients. The clients reconstruct the requested content by solving a set of linear equations represented by the encoded data items received. Evaluation of our strategy is based on performance measures in two different traffic scenarios (i) Video-on-Demand (VoD), where users download and stream high-definition video on-demand, and (ii) Zipf-based Web Traffic, where users browse webpages that follow a Zipf popularity distribution; and has been substantiated by results obtained through implementation and analysis on the GENI testbed. We demonstrate that our caching strategy can show significant improvements in latency and content delivery rate when compared to an ordinary LRU-based caching scheme without network coded caching.

The rest of the paper is organized as follows: Section 2 provides a background in Information-Centric Networking (ICN) and discusses related work. Section 3 presents the concept of Network Coded Caching, the proposed approach to caching in ICNs. Section 4 discusses the algorithmic and architectural specifications of Network Coded Caching. Section 5 presents a brief discussion regarding the design of the proposed algorithms. Section 6 discusses in detail the implementation and the performance evaluation of the proposed approach. We conclude with Section 7 and discuss scope for future work.

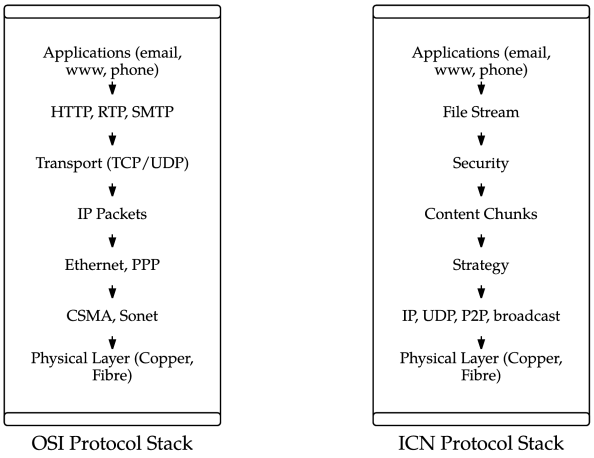

Fig. 1: **Information-Centric Networking**: The Protocol Stack

2 Background

We first provide a brief overview of ICNs and caching in ICNs along with a primer on network coding, the technique adapted in this paper.

2.1 ICNs and Caching in ICNs

In the contemporary design of the Internet, nodes on the network are labeled with IP addresses. Whenever a packet has to be sent to a particular node, the sender stamps the destination address onto the packet and the network figures out how to route the packet to its destination. Information-centric networks push more intelligence into the network and allow the user to specify *what* he wants, rather than *where* he wants it from. Instead of labeling nodes on the network, ICNs label content with *names*. The user presents the *name* of the content that he is interested in to the intelligent pool called the internet, and the internet figures out where the content is and brings it back to him.

ICN's protocol stack varies significantly from the OSI stack near the waist of the Internet hourglass. Content names replace IP addresses at the Network layer. The content names are hierarchically structured, similar to URLs in the current Internet (Eg., */youtube/favorite-artist/live-in-madison/720p*). Another major difference is that several of the transport layer functionalities are pushed below the network layer and into the *Strategy* layer. There is a security layer on top of the Network layer that adds security signatures to all data in order to make the content items self-certifying. Most of the power that ICN offers comes from functional capabilities that are enabled at the Network layer.

ICN communication is driven by consumers of data. At the network layer, there are two kinds of packets – *Interest* packets and *Data* packets. A consumer initiates a request for a desired content item by broadcasting an interest to all connected nodes or to a subset of them. Any node that *hears* the interest and has data corresponding to the interest can respond with a Data packet. Nodes have no identity in this architecture. Routing of content is completely *pull*-based i.e data is transmitted only in response to an interest and not otherwise. The senders are stateless. The interest leaves *breadcrumbs* on each node on its transmission path until it is served. The data packet traces the breadcrumbs back to the source(s) of the interest. Interest generators are unaware of whom they were served by and content servers are unaware of who they are serving. One of the ICN implementations that specifies the

protocols and data structures that can achieve the described functionality is Named Data Networking [4].

In this paper, we work in the context of the Named Data Networking (NDN) protocols for the ICN framework. Interest and data forwarding mechanisms are adapted from the NDN forwarding algorithm [5]. In the current NDN architecture, intelligence is embedded within the entire network and is not confined to the edge: each router maintains and works with three data structures: (i) Pending Interest Table (PIT) (ii) Forwarding Information Base (FIB), and (iii) Content store. The PIT is used to keep track of pending interests in the system, and the interfaces along which the received content must be disseminated. The FIB is used to identify the interfaces along which an incoming interest must be forwarded, forming the basis for the forwarding strategy at the router. In the case of flooding, incoming interests are forwarded on all outgoing interfaces. The Content Store is used to opportunistically cache retrieved content for use in the near future. Requests to the same content may arise close in time because of temporal locality of interests. Routers can directly serve requests to content that is cached in their respective content stores, and the requests need not travel all the way to the content source.

Ubiquitous content caching is one of the key benefits that ICN offers. In a broader sense, packet or object caching in the Internet is mainly motivated by the following two aspects.
Redundancy Elimination : In this context, the packets that are cached at the intermediate nodes are compared against the packets traveling in the network, and any redundancy, if detected, is removed [6], or duplicate packets are compressed in order to account for loss protection [7].
Content Service : In this context, content cached at the routers can be served for future requests to the same content, thereby reducing latency at the client.

One of the main disadvantages of current protocol stack is that the redundancy and request information is at the application layer, and cannot be easily detected at the network layer. This results in a high percentage of redundant packets traveling across the network. ICNs overcome this disadvantage by providing a framework to detect this redundancy at the network layer, thus enabling ubiquitous router-level caching. In addition to improving response time, this also reduces the load on the servers and increases the network's overall throughput. Our focus in this paper is on improving the *service* aspect of content caching.

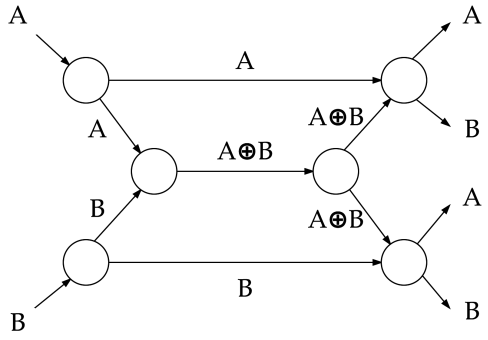

Fig. 2: Network Coding on the Butterfly Network. Multicast is achieved at the maximum flow rate to the servers at both the clients.

2.2 Network Coding

Network coding is a novel idea in which the nodes in the network do not merely forward incoming packets, but perform some operations on the packets ahead of forwarding them. The motive is to increase the overall throughput of the system. A common operation that is typically used is the *xor*, where a group of packets are *xor*-ed with weight coefficients chosen from some finite field F. The throughput increase due to network coding is evident in multicast in the famous butterfly network example (see Figure 2). The problem is to multicast both packets A and B to the two bottommost nodes shown in the figure. Without network coding, a rate of at most 1.5 packets per time unit can be achieved (see [8]) because of contention on the center link, whereas with network coding, a multicast rate of 2 packets per time unit can be achieved using the coding scheme shown in the figure. Random linear network coding [8] is typically implemented to achieve network coding at gigabit speeds.

Next, we briefly discuss and compare our proposed techniques to related work in the field.

2.3 Related Work

Existing literature on network coding focuses primarily on improving content delivery rates, but does little to exploit the benefits of temporal locality in content requests. For example, [9] and [10] have focused on improving multicast streaming by achieving maximum flow rates with network coding. On the other hand, [11] and [12] assume a stochastic model for client requests and apply network coding for long term content caching. The focus of these techniques is to optimally allocate caches or distribute content across *servers* for

the long-term (typically in Content Distribution Networks) to bring down the average latency of the system. These servers are small in number and are assumed to have large storage capacities. On the contrary, in this paper, we are concerned with applying network coding as a technique to improve short-term caching strategy and cache replacement decisions for a raft of small-sized pervasive router caches. The difference is that the potential performance gains that we are looking at arise opportunistically from the temporal locality of interests generated by clients in the system, and not from any optimization performed based on any stochastics induced in the system.

We exploit both ideas presented in this section to develop a ubiquitous network coded caching framework for ICNs.

3 Network Coded Caching Concept

In this section, we describe our proposed network coding integration for content caching in ICNs.

We work in the context of the Named Data Networking protocols for the Information-Centric Network architecture. Every router on the network is equipped with a content store to keep a local copy of content for serving future requests. Suppose the cache capacity of the each of the content routers is C packets. Once the cache reaches its capacity, typical caching algorithms use a contemporary cache replacement policy (like LRU). The standard proposal in the NDN system is to use an LRU based cache at each of the content stores. Instead, in our proposed Network Coded Cache, when the content store reaches its storage limit, incoming data packets are linearly coded with the existing content items in the store. The coefficients of the linear combination are chosen randomly from a finite field.

Consider an example where $C = 2$, as demonstrated in Figure 3. Let content items be represented by D_i. The packets arrive in the sequence D_1 to D_4. Snapshots of the cache over time are shown in the figure. When D_1 and D_2 arrive, there is enough space in the content store to accommodate them. When D_3 arrives, the content store has reached its capacity. But instead of replacing either D_1or D_2 with D_3, D_3 is linearly coded with either D_1 or D_2 (say D_1) and the encoded data item is stored as shown. Similarly, when D_4 arrives, it is linear combined with D_2 and the encoded data item is stored.

The Router serves an interest for D from its cache with any data item that is a linear combination of D and possibly another D', where D' could be arbitrary. This introduces the need for a decoding algorithm at the client since it can now receive linear combinations of content items that it needs to resolve. In addition, this introduces the need for hardware that can enable network coding. Recent advances in NetFPGA hardware have shown that packets can be linearly combined and forwarded at gigabit speeds. We present a detailed description of this strategy in the next sections.

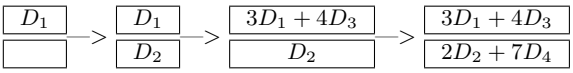

Fig. 3: **Content Store snapshots** - At an arbitrary router with a cache size of 2 content items. Content items are received in the sequence D_1 to D_4

4 Encoding and Decoding

Incorporation of network coded caching requires additional logic to be implemented at the clients and the routers, the details of which are presented in this section.

4.1 Encoding at the Router

In the traditional ICN framework, routers check their content store for direct matches with the content names represented by the received Interest packets. On the event of a match, the routers serve content items from their content stores without having to forward the requests any further. In the case of Network Coded Caching, routers not only modify their caching strategy, but also their lookup mechanism for serving Interests. Here we describe the encoding and request handling intelligence introduced at the routers.

4.1.1 Caching Strategy

When a data item D is received, if the data item is coded (i.e., already a linear combination of some content items) then it is inserted into the content store without any change. The LRU policy is adapted for cache replacement if the content store's capacity is exhausted. If the received data item is a non-coded content item, it is processed differently. If the content store's capacity is not yet completely utilized, the non-coded content item is stored without any modification. But if the content store's capacity is exhausted, the content store is searched for some *other* non-coded content item D' and D' is replaced with a random linear combination of D and D'. If no such D' exists (i.e., all data items in the content store are coded data items), then LRU

cache replacement is used to insert D as a non-coded content item into the cache.

4.1.2 Interest Handling

When an interest is received for content D, the router first checks if the interest can be locally served. The local servicing process for network coded caching comprises of a two-phase search. The router's content store is first searched for the non-coded content item D, and the item is served if present. If the search fails, a second search is initiated to find a coded data item in which D has a non-zero coefficient. If such a coded data item exists, then it is further verified that the nonce (see Section 4.3.2) associated with the coded data item is not among the *seen-nonces* list of the interest. This is done to ensure that the client doesn't receive multiple copies of the same linear combination. Once verified, the coded data item is served by the router. If the second search also fails, then the interest is forwarded based on the router's forwarding strategy.

4.2 Decoding at the Client

The client that requested content item D may receive a coded data item which contains a linear combination of D and another content item D'. In this case, the client can only decode D from the coded data item if it either has a copy of D', or if it can deduce a linear combination of D and D', from existing data items in its buffer, which is linearly independent of the coded data item received. If neither condition is satisfied, it must send out a request for D'. We term requests of this type *bouncing* requests. We note that the bouncing request for D' might be served with a data item that is a linear combination of D' and some D''. Similar to the previous situation, the client might have to send out another bouncing request, but now for D''. Note that the original intention is to decode D. We realize that this translates into a *dependency graph*, where each node represents a content item, and an (undirected) edge from node D to node D' indicates that the client contains a coded data item that is a linear combination of D and D'. The graph is *not* necessarily a simple graph, and dependencies on this graph are resolved *when cycles are formed.* This includes self cycles of length 1 formed by non coded content items. Cycles of size k immediately resolve k content items. The rest are resolved by propagation.

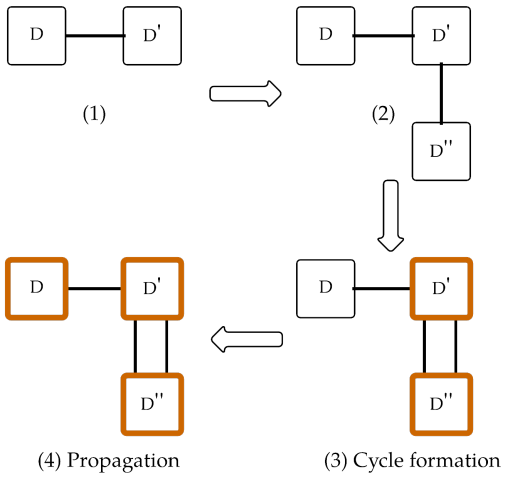

Fig. 4: **Visualization of the decoding process** at the client. Cycles resolve dependencies. In this case, two coded data items that are different linear combinations of D' and D'' are received, thus forming a cycle and resolving the dependency. D is also resolved via a propagation of resolution. Thick edged boxes indicate resolved (decoded) content items.

Example

We illustrate the decoding process with an example, which is illustrated in Figure 4. Consider the following situation, in which the client initially requested for packet D:

- Client receives a linear combination of D and D'. Now the client sends out a bouncing request for D'.
- Client receives a linear combination of D' and D''. Now the client sends out a bouncing request for D''.
- Client receives a different linear combination of D' and D''. Now a cycle is formed. D' and D'' are resolved. D is resolved by propagation.

4.3 Architectural Requirements

Here we describe the additional information that needs to be carried in the Interest and Data packets in order to support network coded caching.

4.3.1 The Data packet

Every data packet must contain, in addition to the data item, the content names of the content items involved in the linear combination stored, and also their coding

coefficients. It must also contain a *nonce* that is a one-to-one function from the content items in the linear combination and the coding coefficients to a finite field.

4.3.2 The Interest packet

Every interest packet must contain, in addition to the name of the content being requested for, a list of *nonces* of content packets that it *does not* want to be served with. This is to ensure that no two data packets with the same content items *and* same coefficients are served when bouncing requests are made. This *seen-nonces* list ensures that the data packets received for a given interest are linearly independent.

5 Discussion

In this section, we address some of the challenges in the design of our strategy, and present an intuitive example to understand its benefits.

5.1 Dependency cycles

An immediate consequence of the client decoding scheme is that the dependency cycle sizes may grow indefinitely. If the cycles become extremely large, it could cause problems for small client buffers, and may considerably decrease performance. This is also the case for the interest packets, since the seen-nonces list may grow indefinitely. To avoid this, we prevent cycle sizes from growing beyond a certain threshold by dividing the entire space of content items into encode-able *groups* or *sets*. We enforce a restriction that only content items within the same *group* can be coded with each other. The size of each such group is k, thus restricting cycle sizes and sizes of the seen-nonces list to k. In the context of video data, an example would be to restrict content items to be coded with other content items that belong to the same movie. We work with encoding sets of equal size for the sake of evaluation, but in reality, these sets need not be of equal size. One possibility is to set an encoding restriction such that only content items that share certain namespace can be coded with each other. This relaxation will ensure that this process is scalable for a large number of content items.

Note that we have only considered linear combinations of *two* content items in the above discussion. Although it is possible to consider linear combinations of a larger number of content items, we do not address that in this work. Interpretations of dependency cycles in the cases of linear combinations of more than two content items will involve hyper-edges, and are a bit more involved. Our focus here is to show that network coded caching has the *potential* to improve performance measures at the client.

Also note that the dependency graph visualization of the received data items is a valid representation since it is assured that the client will eventually receive linearly independent data items for a given interest. This ensures that the cycles actually translate into a solvable set of linear equations.

5.2 Linear Independence

Since we deploy Random Linear Network Coding to implement Network Coded Caching, a client may receive two linearly dependent combinations of a pair of data items, although the event is of a low probability for a well chosen finite field F for network coding. In these cases, the client re-issues an interest for either one of the data items in the pair and includes the nonces of the linearly dependent combinations in the seen-nonces list of this interest.

5.3 Visualizing the Performance gain

Consider the network shown in Figure 5. (A) shows the system snapshot without network coded caching, and (B) shows the system snapshot with network coded caching. Node 1 is the request generator (client) and the other nodes are routers. The corresponding content stores are shown in rounded rectangles. The two snapshots shown differ only in the data items present in the content stores of the routers in the system. Suppose it is the client's intention to receive content item d. In the case of ordinary caching, d is four hops away from the client. In the case of network coded caching with the same cache size, we see that the d can be reconstructed by receiving the data items $2a + 3d$ and $3a + 4d$, which are just one hop away each. Although there is worst case latency of two hops (one for each data item), we see that both can be served in parallel, thus accounting for an effective latency of only one hop.

Note that the other content items in the content store snapshots are exactly the same for both the cases. We have demonstrated that with the same storage space, we are also able to serve d quickly, *without* affecting the latencies incurred on requests to a, b, and c by the same client. Overall, (A) can serve content items a, b and c in one hop each, but requires 3 hops to serve content d. (B) is capable of serving a, b, c and d, each with an effective latency of one hop. In fact, a request to both a and d can be simultaneously served with a latency of one hop, since both the coded packets can be served in

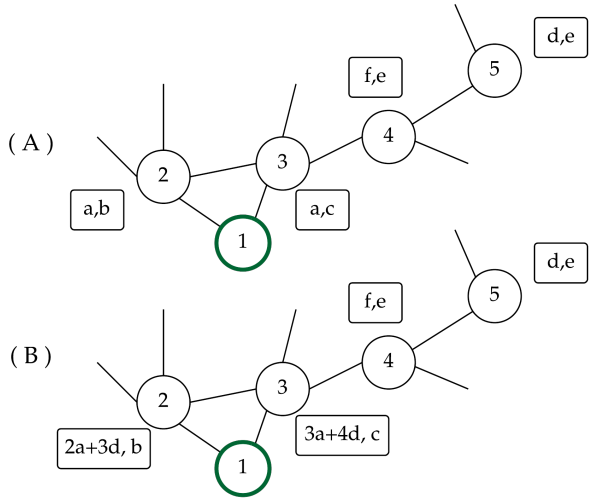

Fig. 5: **Visualizing the potential gains** of network coded caching. Content store snapshots for the corresponding nodes are shown in rounded rectangles.

parallel. This demonstrates how network coded caching can take advantage of multi-path routing to improve throughput.

6 Performance Evaluation

In this section, we evaluate the performance of network coded caching on a real network through experiments on the GENI network testbed. By implementing the Named Data Networking protocols on GENI, and our algorithms on top of it, we carry out a performance evaluation comparing *latency* and *rate* measures observed at the client with and without network coded caching enabled.

First, we performed preliminary simulations on a remote computer to evaluate the performance of network coded caching. The simulations demonstrated significant potential for improvement in latency measures. Assured by these findings, we proceeded to the GENI testbed to evaluate our algorithms on a real computer network.

6.1 Information-Centric Networking on GENI

The GENI (Global Enterprise for Network Innovations) testbed is an ultra-fast network of computers across the United States. By leveraging the concept of Network virtualization, it serves as an experimental testbed for Networking researchers to deploy and test their new algorithms, architectures, protocols etc., on real networks at a large scale.

We implement ICN (i.e the NDN framework, caching and forwarding algorithms) as an application level overlay over TCP/IP sockets. We spawn desired topologies on the testbed through the GENI portal [13]. We then assign each node the role of either client, router or server. Forwarding mechanisms and algorithms for each of the roles are implemented in accordance with the NDN forwarding algorithm [5]. We use flooding as the forwarding *strategy* at the routers, instead of the red, green, yellow port adaptive forwarding discussed in [5]. We make this simplification because our aim is to present the potential performance improvements that network coded caching can provide, over that of ordinary caching, irrespective of the forwarding mechanism used.

We use GENI to set up a topology of 20 nodes, with 2 servers, 4 clients, and 14 routers, a virtual instantiation of the real network topology at the University of Wisconsin-Madison [14]. At each server, we store real files for a subset of the content space, such that all servers together span the universe of content items.

6.2 Performance in Video-on-Demand (VoD)

We evaluate our algorithm in a scenario where users are streaming videos on demand. The content space is a set of movies, and each movie is associated with a sequence of content items (frames). We assume a *Zipf* distribution (with a suitable exponent) over the movies in the content space. As a request pattern, each client first chooses a movie from the Zipf distribution over movies, and then requests the frames for the chosen movie in sequence. We assume that the client is playing the requested movie at a constant frame rate, which corresponds to a constant content consumption rate. The client also has a *playback buffer*, which represents the maximum amount of *un-played* content that the client can store for future play. In a parallel view, this sets a limit on the number of requests that can be pending at the client at a given time, since in the best case, any further content served in addition to those requested has to be dropped , owing to a lack of buffer space. Table 1 lists all the parameters chosen for the emulation. For the encoding restriction, we divide each movie into disjoint groups of 10 frames each (consecutive frames) and enforce the restriction that frames can be linearly combined only with other frames of the same encoding group. The setup essentially emulates a Ultra-HD Video On-Demand scenario implemented over an information-centric network.

Parameter	Value
Topology	Wisconsin sub-network
Number of Movies	50
Zipf parameter	0.7
Frames per movie	400 (~2 minutes each movie)
Frame (Content) size	0.1 MB (~3MB per second of play)
Playback Buffer	30 MB (~300 content items)
Router Content Store	40 MB (~400 content items)
Encoding Restriction	*Encoding-groups* of size 10
Link Speeds	100 Mbps

Table 1: Summary of Emulation parameters

To evaluate the performance of our caching strategy, we compare the *distribution of the pauses* at the client, where the pause for each frame represents the time for which the client had to wait before being able to play the corresponding frame. We avoid a cold start, i.e we run the emulation until the router content stores have reached their capacity before beginning our measurements.

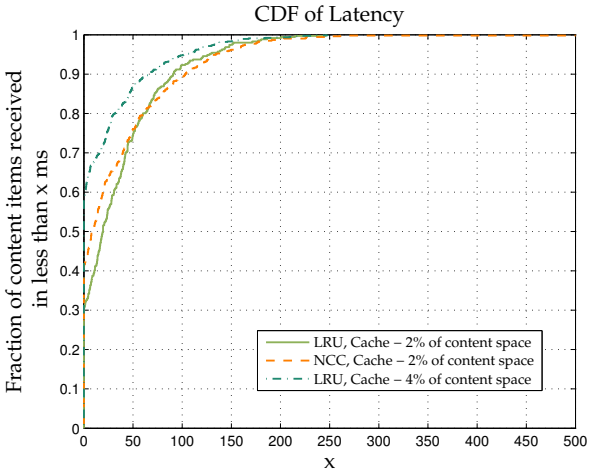

Fig. 6: **Distribution of the pause** at the client. Emulation parameters are as shown in Table. The router cache size is set to 2% of the content universe (400 content items)

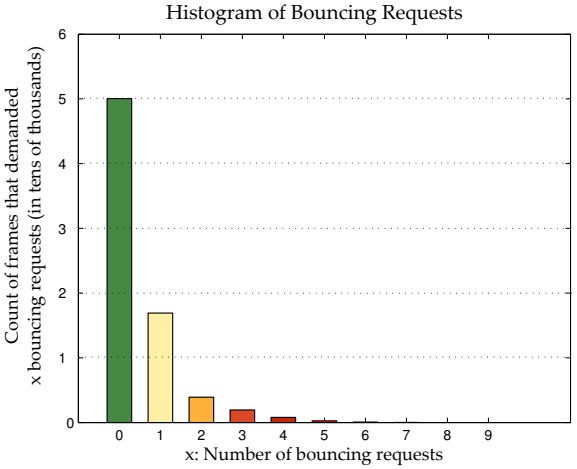

Fig. 7: **Histogram of bouncing requests**: Significant number of requests that generate a large number of bouncing requests

6.3 Observations

Figure 6 shows the distribution of the pauses at the clients with and without network coded caching for the set of parameters mentioned in Table 1, and for a link speed of 100 Mbps. The average latency at the client with LRU caching is 25.75 ms, whereas with network coded caching the average is 23.87 ms. The standard deviations of the average latency calculated over 5 runs are 2.2 ms and 3.16 ms respectively. Overall, this corresponds to an average improvement of 7.9% in the average latency observed. This implies that *on an average*, while the system could serve at $1000/25.75 = 38.82$ frames *per second* with ordinary caching, it can now serve at 41.89 frames *per second* with network coded caching. A closer examination of the distribution reveals that the number of frames (content items) with smaller pause values increases with network coded caching. But it also reveals that there are some frames that exhibit large pause values with network coded caching. Without network coded caching, the maximum pause exhibited by any frame is just over 250 ms, whereas with network coded caching, this maximum is over 500 ms. This can be explained by the fact that certain frames generate more bouncing requests than others, with the ones generating larger bouncing requests contributing to an increased latency.

Figure 7 shows the histogram of the number of bouncing requests generated for each frame that is requested by the client. Note that the upper bound on the number of bouncing requests is determined by the encoding restriction enforced in the system. It can be seen that although the number of requests that generate smaller number of bouncing requests (1 or 2) is higher, there is still a significant number of requests that generate a larger number (≥ 3) of bouncing requests. This supports the fact that some frames exhibit large pauses. We attempt to address this issue by introducing an Edge-coding variant of NCC (See Section 6.5).

Figure 8 shows the effect of the routers' cache size on the achievable average content delivery rate for both LRU and NCC caching schemes. Encoding sets of size 10 are enforced. The standard deviations of the average rate observed are higher for Network Coded Caching

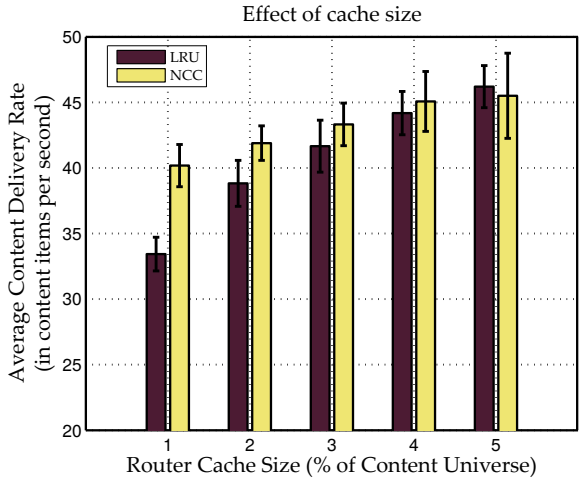

Fig. 8: **Average rate vs. Router cache size** - The two sided error bars represent the 95% confidence interval of the average rate, based on the experiments conducted.

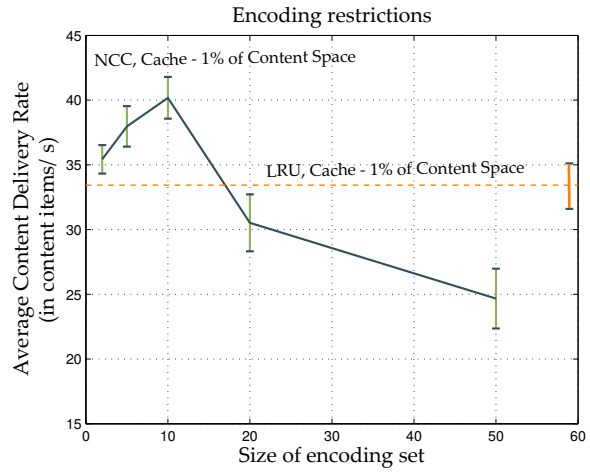

Fig. 9: **Average rate vs. Encoding Set size** - The two sided error bars represent the 95% confidence interval of the average rate, based on the experiments conducted.

when compared to LRU caching, indicating that the performance is more variable in the former.

Confidence Intervals: The 95% confidence interval for one population mean is given by the formula $x \pm s.t^*_{n-1}/\sqrt{n}$, where n is the number of samples, x is the sample mean, s is the sample standard deviation and t^*_{n-1} is the critical t^* value with $n-1$ degrees of freedom under a 95% confidence. We work with the t-distribution tables since it accounts for higher error when the number of samples is lower. We plot the confidence intervals as error bars in the relevant plots to study the statistical significance/insignificance of the gains presented in the plots.

The error bars in Figure 8 indicate the 95% confidence intervals of the mean, with 5 runs i.e a sample size of 5. We observe that NCC shows higher gains in content delivery rate for smaller router cache sizes. Observing the 95% confidence intervals, the gains are statistically significant for smaller cache sizes. For larger cache sizes, network coding does not provide substantial benefit, and also performs worse than LRU in some cases. There is a significant overlap in the confidence intervals of the mean for the higher cache sizes. A two-sample unpaired t-test shows that the gains are statistically insignificant for the higher cache sizes in the plot ($p \geq 0.05$, where p is the two-tailed P-value for the null hypothesis, which claims that the underlying distribution of the means of LRU and NCC are the same).

Figure 9 shows the impact of the encoding restriction on the performance of Network Coded Caching. Average rate values are reported for different cardinalities of the encoding set. We group consecutive frames into sets of this cardinality and enforce a constraint that only frames within each set can be coded with each other. Two sided error bars representing the 95% confidence interval of the mean are also plotted. We observe that an encoding set size of 10 shows the highest rate gain when compared to LRU caching. Smaller sizes do show performance improvements, but do not leverage the total potential of the system, whereas higher sizes result in a larger number of bouncing requests and degrade system performance. An encoding restriction of size 10 best captures the trade-off between the overhead of bouncing requests and the gains of decoding multiple consecutive frames at the same time and buffering them.

6.4 Performance on Zipf-based Web browsing

We also evaluate our algorithm in a scenario where users are browsing webpages. Each user independently samples webpage requests from a zipf popularity distribution over websites. The requests are issued in sequence and are blocking in nature - i.e., a user issues the i-th request only after the (i-1)-th request has been served. For each user, we generate a synthetic trace of 10^4 web requests, each drawn from a zipf distribution over 10^3 content items with a zipf parameter of 0.7. The average size of a content item (a webpage) is set to 1.6 MB for the emulation [15].

We plot the cumulative distribution of the latency incurred in serving these requests under different caching scenarios. Figure 10 shows the comparative performance of Network coded caching against ordinary LRU caching

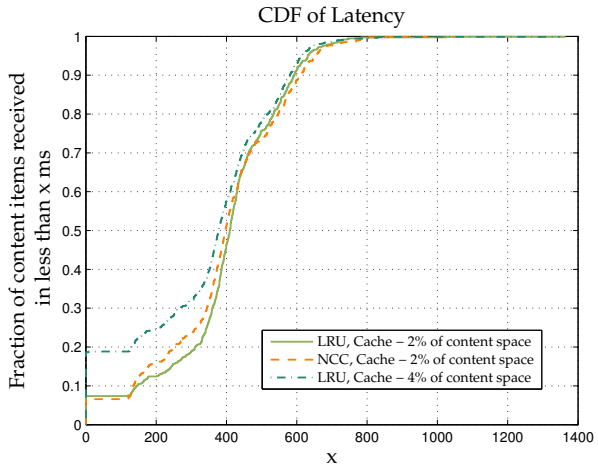

Fig. 10: **CDF of Latency** for Zipf-based web browsing

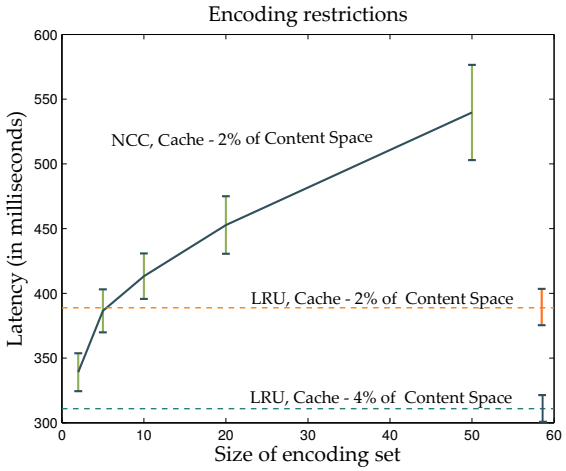

Fig. 11: **Average Latency vs. Encoding restriction** - Two-sided error bars represent the 95% confidence interval of the mean, based on the experiments conducted.

(1) with the same router cache size, and (2) with double the router cache size, for encoding sets of size 2. The cache size at the routers is set to some percentage of the total number of objects in the universe of content items. The number of content items served with zero-latency (a hit in the client's cache) roughly doubles for the LRU caching scheme when the cache size doubles. When compared to LRU, NCC shows an increased number of content items that are served with low latency. But it also shows an increased the number of content items served with very high latency. The average latency in the LRU caching scheme is roughly 380 ms, whereas with NCC, it is brought down to 364 ms, an overall 4.2% improvement.

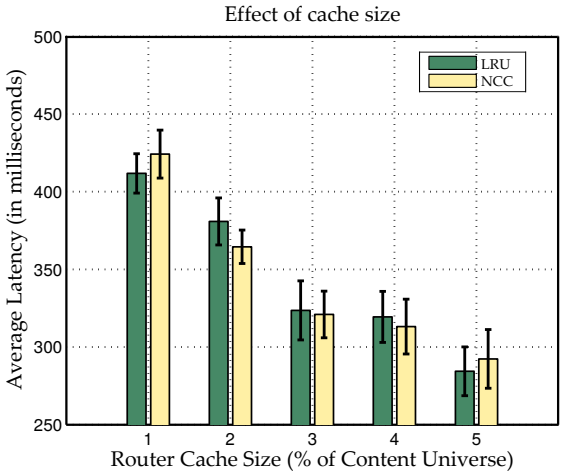

Fig. 12: **Average Latency vs. Router cache size** - The two-sided error bars represent the standard deviation of the average latency calculated over 5 runs.

Figure 11 shows the impact of encoding restrictions on the performance of Network Coded Caching. Average latency values are reported for different cardinalities of the encoding set used. Also included are two-sided error bars that represent the 95% confidence interval of the average latency, for a sample of 5 runs with different synthetic traces. The router cache size is set to 2% of the content universe. We observe that maximum latency gains are observed for an encoding set of size 2. As the cardinality of the encoding set increases, the number of bouncing requests increases. This increases the network load, thus increasing the observed latencies. For very high cardinalities of the encoding set, Network coded caching degrades performance significantly. We observe that an encoding set cardinality of 2 captures the right trade-off between network load and content reachability, providing a latency gain of 4.2 %. The confidence intervals of LRU and NCC with an encoding set cardinality of size 2 for a router cache size of 2% of the content universe show no overlap. This indicates that the average gain reported for this encoding restriction is statistically significant.

Figure 12 shows the effect of router cache size on the latency values observed for both LRU and NCC caching schemes. Average latency values are reported along with two-sided error bars that represent the 95% confidence interval of the average latency observed for a sample of 5 runs, each with a different synthetic trace. Router cache sizes are varied in percentages of the universe of content items, similar to [16]. Encoding sets of size 2 are enforced. We observe that NCC shows the highest gain in terms of average latency for a router cache size of 2% of the content universe. Nevertheless, it per-

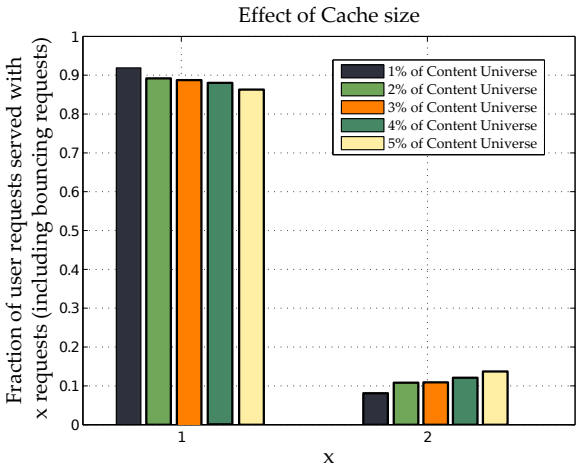

Fig. 13: **Bouncing requests vs. Cache size**: Zipf-based web traffic.

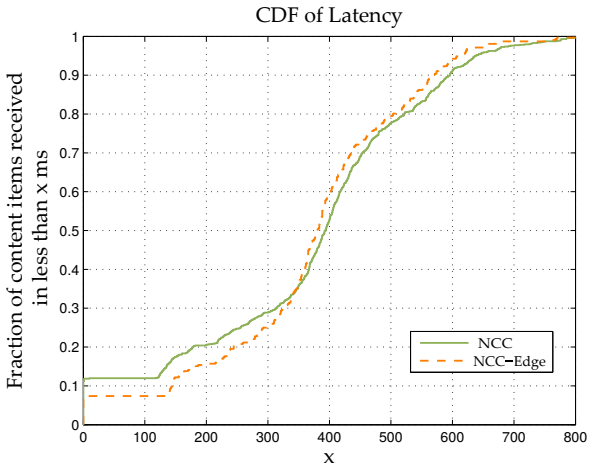

Fig. 14: **Edge Coding - CDF of Latency** for Zipf-based web traffic

forms worse than LRU caching when the cache sizes are both (1) very high ($\geq$5%) and (2) very low ($\leq$1%). For higher cache sizes, the benefits of network coding are lower since there is enough space in the router caches to store highly popular content as is, and coding would increase the latency overhead by causing bouncing requests. For very small cache sizes, the overall storage available on the network is lower and it is more probable that a bouncing request would require it to go all the way to the source, more so in this case since the encoding sets are small sets of size 2. The standard deviation of the average latency varies from 10 to 15 ms in both caching schemes, indicating that the performance itself varies significantly across different synthetic traces. The confidence intervals overlap in all cases. Two-sample unpaired t-tests indicate that the latency gain from NCC is statistically significant only for the case when the router cache size is 2% of the content universe ($p = 0.04$ for this case) .

Figure 13 plots the fraction of requests that were bouncing requests for different router cache sizes. We observe that the fraction of bouncing requests increases with an increasing cache size. With higher cache sizes, it is less likely that a request needs to go all the way to the source i.e. a cache hit for a coded content item at the intermediate nodes is more likely, thus resulting in a higher fraction of bouncing requests.

Overall, the above plots show that Network Coded Caching demonstrates statistically significant improvement in average latency (upto 4.2%) for web traffic when router caches are reasonably sized (2-4 % of content universe).

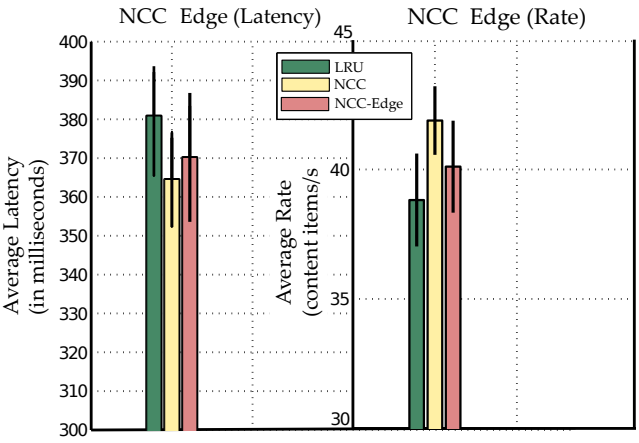

Fig. 15: **Edge Coding Performance comparison** with NCC and LRU: (left) latency in zipf-based web traffic and (right) content delivery rate in VoD

6.5 Edge Coding

We evaluate a variation of the Network Coded Caching scheme where coding is performed only on the edge of the network i.e on routers that are one hop away from at least one user. The rest of the network operates an opportunistic LRU caching scheme. The motivation is to bring down the latency overhead of bouncing requests. It is also more practically feasible to deploy and enable network coding at the edge of real networks. In this scheme, if a request travels more than a hop, it is guaranteed to be served with a non-coded content item. Network coded content does not travel beyond the edge of the network and the other intermediate LRU-based routers need to deal with only ordinary (non-coded)

content. This sets a better bound on the latency overhead of bouncing requests. It also reduces the load on the network by decreasing the number of content items that are exchanged on the network as a consequence of bouncing requests. Figure 14 shows the cumulative distribution of the latency for Zipf-based web traffic with NCC and its edge-coding variant (NCC-Edge) for an encoding restriction of size 2 and a router cache size of 2% of the content universe. The average latency for NCC-pervasive is 364 ms and that of NCC-Edge is 370 ms. The standard deviation of the average latency measured over 5 different runs is 13 ms and 12 ms respectively. The average latency without NCC is 380 ms. Although the average latency is observed to be higher for NCC-Edge, we observe that the number of requests served with higher latency values is lower for NCC-Edge. This is expected from the bound that edge coding sets on the latency overhead of bouncing requests.

Figure 15 shows the average performance of Edge Coding in comparison to Network Coded Caching and LRU caching schemes. Performance in VoD and in Zipf-based web traffic are presented. The error bars represent the 95% confidence interval of the mean, for a sample size of 5 runs. The overlaps are pretty significant and a two-sample unpaired t-test shows that the gains reported for NCC-Edge when compared to NCC might not be statistically significant. We observe that NCC-Edge, on an average, achieves a significant portion (40-50%) of the average gain that NCC-pervasive offers when compared to ordinary LRU caching. This indicates that edge-coding might be *sufficient* to achieve a good fraction of the gain that pervasive-coding has to offer.

7 Conclusions and Future Work

In this paper, we have presented Network Coded Caching, an integrated network coding-and-caching strategy that better utilizes the distributed storage network that ICN enables. We have presented the logical components required at the routers and clients, and also the architectural changes in terms of header fields, necessary to support network coded caching. We have shown through emulation on the GENI testbed that our strategy can provide significant gains in latency measures and content delivery rates on a real network. We have also presented Edge-coding, a variant of Network Coded Caching and have shown that Edge-coding can achieve a significant percentage of the gains that pervasive coding has to offer, for a smaller cost of deployment.

Future work could focus on exploring co-ordinated approaches to network coded caching, where routers in the system exchange information to minimize redundancy. Another direction is to integrate forwarding strategies into the framework of caching schemes. It might also be worth studying the applicability of software defined networking in this context. A centralized and all-knowing controller could make caching decisions for every node on the network and communicate only the corresponding decisions to the nodes through an OpenFlow-like cache rule protocol. Centralized decisions in this context are more powerful and can result in content delivery at substantially higher efficiencies.

Acknowledgements We would like to thank the India-UK Advanced Technology Center of Excellence in Next Generation Networks, Systems and Services (IU-ATC) for their constant support towards this project.

References

1. G. Xylomenos, C. N. Ververidis, V. A. Siris, N. Fotiou, C. Tsilopoulos, X. Vasilakos, K. V. Katsaros, and G. C. Polyzos, "A Survey of Information-Centric Networking Research," *IEEE Communications Surveys & Tutorials*, pp. 1024–1049, 2013.
2. V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," CoNEXT, pp. 1–12, ACM, 2009.
3. "Cisco's visual networking index forecast," 2013. http://newsroom.cisco.com/release/1197391/.
4. L. Zhang et. al, "Named Data Networking (NDN) Project," tech. rep., University of California, Los Angeles, Oct 2010.
5. C. Yi, A. Afanasyev, L. Wang, B. Zhang, and L. Zhang, "Adaptive forwarding in named data networking," *SIGCOMM Comput. Commun. Rev.*, pp. 62–67, June 2012.
6. A. Anand, A. Gupta, A. Akella, S. Seshan, and S. Shenker, "Packet caches on routers: The implications of universal redundant traffic elimination," SIGCOMM, pp. 219–230, ACM, 2008.
7. D. Han, A. Anand, A. Akella, and S. Seshan, "Rpt: Re-architecting loss protection for content-aware networks," in *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI, USENIX, pp. 6–6, 2012.
8. P. Sanders, S. Egner, and L. Tolhuizen, "Polynomial time algorithms for network information flow," SPAA, pp. 286–294, 2003.
9. H. Seferoglu and A. Markopoulou, "Opportunistic network coding for video streaming over wireless," in *Packet Video*, pp. 191–200, IEEE, 2007.
10. N. Sundaram, P. Ramanathan, and S. Banerjee, "Multirate media stream using network coding," in *Proc. 43rd Annual Allerton Conference on Communication, Control, and Computing*, 2005.
11. A. Pourmir and P. Ramanathan, "Distributed caching and coding in VoD," in *Computer Communications Workshops (INFOCOM WKSHPS)*, IEEE, April 2014.
12. K. Nguyen, T. Nguyen, and S.-C. Cheung, "Video streaming with network coding," *J. Signal Process. Syst.*, pp. 319–333, June 2010.
13. M. Berman, J. S. Chase, L. Landweber, A. Nakao, M. Ott, D. Raychaudhuri, R. Ricci, and I. Seskar, "GENI: A federated testbed for innovative network experiments," *Com-*

puter Networks, 2014. Special issue on Future Internet Testbeds - Part I.
14. "University of Wisconsin at Madison Network Statistics," April 2015. http://stats.net.wisc.edu/newcore.html.
15. "Average web page breaks 1600k," July 2014. http://www.websiteoptimization.com/speed/tweak/average-web-page/.
16. 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," *SIGCOMM Comput. Commun. Rev.*, pp. 147–158, 2013.