Integrated Network Coding and Caching in Information-Centric Networks

Abhiram Ravi[1,3], Parmesh Ramanathan[2] and Krishna M. Sivalingam[1,3]
[1]Department of Computer Science and Engineering, Indian Institute of Technology Madras, Chennai, India
[2]University of Wisconsin-Madison, Madison, USA
[3]India-UK Advanced Technology Centre of Excellence in Next Generation Networks, Systems and Services (IU-ATC)
Email: abhiram@cse.iitm.ac.in, parmesh@ece.wisc.edu, skrishnam@cse.iitm.ac.in

***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 testbed demonstrating the performance of our algorithm on a real network are included in the paper. 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 throughput for a small computational overhead.**

I. 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 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 the context of downloading and streaming high-definition video on-demand, and has been substantiated by results obtained through experimentation on the GENI testbed. We demonstrate that our caching strategy can show significant improvements (around 24%) in latency measures when compared to the case without network coded caching.

II. 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.

A. 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. A wide variety of protocols exist for implementing this framework.

In this paper, we work in the context of Named Data Networking [4] 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

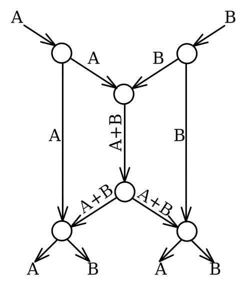


Figure 1: Network Coding on the Butterfly Network. Multicast is achieved at the maximum flow rate to the servers at both the clients.

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 server and increases the network's overall throughput. Our focus in this paper is on improving the *service* aspect of content caching.

B. 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 1). The problem is to multicast both packets A and B to the two bottom-most 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.

In the next section, we briefly discuss and compare ourselves to related work in the field.

III. 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 similar papers 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 CDNs) 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.

IV. Network Coded Caching

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

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). But in our proposed Network Coded Cache, when the cache reaches its storage limit, incoming data packets are linearly coded with the existing content items in the cache. Consider an example where $C = 2$, as demonstrated in Figure 2. 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.

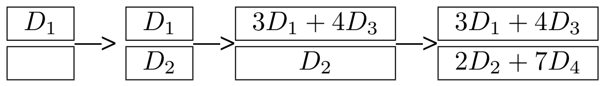


Figure 2: Content Store snapshots at an arbitrary router with a cache size of 2 content items. Content items are received in the sequence D_1 to D_4

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, in addition to 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 section.

V. 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.

A. Encoding at the Router

We first describe the encoding and request handling intelligence at the router.

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.

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 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.

B. 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.

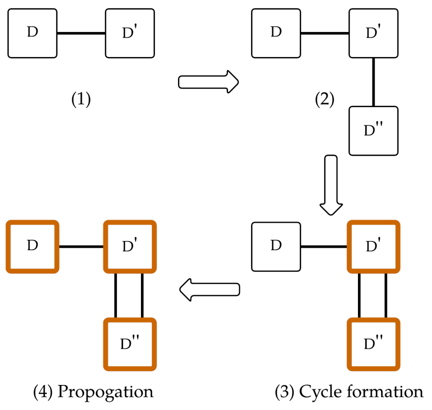


Figure 3: Visualization of the decoding process at the client. Cycles resolve dependencies. In this case, two coded data items that are different linear combinations of D' and D'' are received, thus forming a cycle and resolving the dependency. D is also resolved via a propogation of resolution. Thick edged boxes indicate resolved (decoded) content items.

Example: We illustrate the decoding process with an example, which is illustrated in Figure 3. 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.

C. 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.

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.

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.

VI. Discussion

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

A. 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.

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 paper. 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 only receive linearly independent data items for a given interest. This ensures that the cycles actually translate into a solvable set of linear equations.

B. Visualizing the Performance gain

Consider the network shown in Figure 4. (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

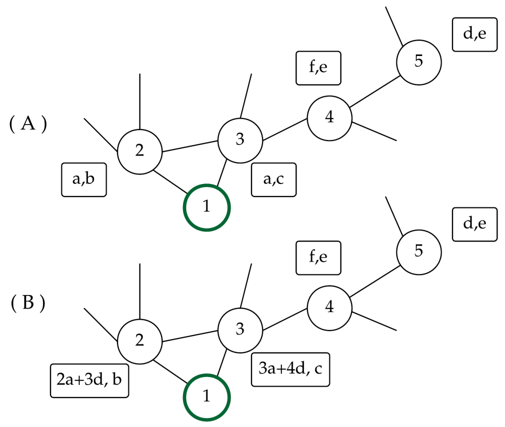


Figure 4: Visualizing the potential gains of network coded caching. Content store snapshots for the corresponding nodes are shown in rounded rectangles.

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

C. Theoretical Limits

Our goal is to show that, with the same cache size, *network coded caching* performs better than the ordinary LRU caching scheme. The performance of network coded caching with router cache capacity C cannot exceed the performance of the same system with network coded caching disabled but with router cache capacity $n \times C$, where n is maximum number of unique packets that are linearly coded at a cache entry. This is true because in the latter, we could store all the content items stored in the former without performing any encoding. This would eliminate the need for any decoding at the client, thus reducing the latency penalties of bouncing requests. This sets a theoretical limit on the performance gain that network coded caching can demonstrate.

VII. 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 NDN framework on GENI, and our algorithms on top of it, we carry out a performance evaluation comparing *latency* 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 network.

A. ICN 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 a real network at a large scale. For our emulation, we use nodes provided by Emulab [12].

We implement ICN (i.e the NDN framework, caching and forwarding algorithms) as an application level overlay over TCP/IP sockets. Using NS-3 scripts, we spawn desired topologies on the testbed (with real nodes), and 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. The content space is a set of movies, and each movie is associated with a sequence of content items (frames).

B. Emulation Environment

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. At each server, we store real video files for a subset of the content space, such that all servers together span the universe of content items. 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 I 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 (VoD) scenario implemented over an information-centric network.

Parameter	**Value**
Topology	20-node Wisconsin topology :- 2 servers, 4 clients
Number of Movies	16
Zipf parameter	0.7
Frames per movie	1000 (~5 minutes each movie)
Frame (Content) size	0.1 MB (~3MB per second of play)
Playback Buffer	30 MB (~300 content items)
Router Content Store	60 MB (~600 content items)
Encoding Restriction	*Encoding-groups* of size 10
Link Speeds	100 Mbps or 64 Mbps
Emulation time	1 hour
Cache replacement	LRU replacement

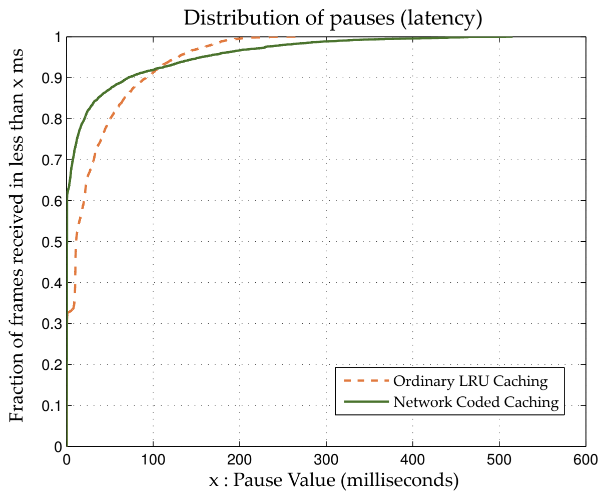
Table I: Summary of Emulation parameters

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.

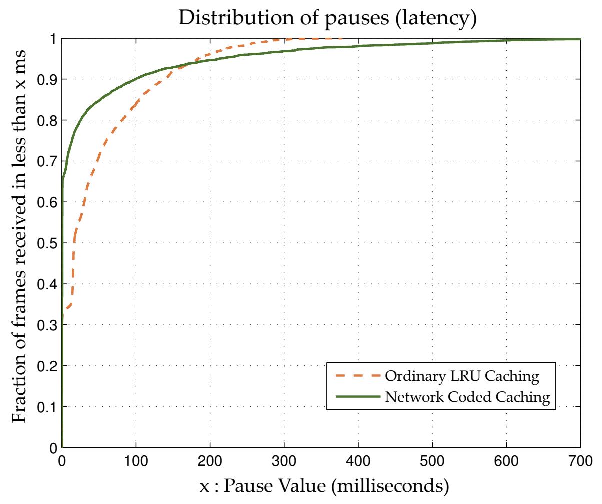
C. Results

Figure 5a shows the distribution of the pauses at the clients with and without network coded caching for the set of parameters mentioned in Table I, and for a link speed of 100 Mbps. The average pause (latency) at the client with ordinary caching is 30.64 ms, whereas with network coded caching the average is 23.94 ms. This corresponds to an improvement of 21.8% in the average latency. This implies that *on an average*, while the system could serve at $1000/32.65 = 32.63$ frames *per second* with ordinary caching, it can now serve at 41.77 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 5b shows the pause distributions for a slightly lesser link bandwidth (64 Mbps). The average pause at the client with ordinary caching in this case is 45.03 ms, while with network coded caching it reduces to 34.24 ms (24% improvement). Similar to Figure 5a, we observe that the system shows an increase in the number of requests with smaller pauses, along with some frames exhibiting larger pauses than those exhibited without network coded caching.



(a) Link Speed :- 100 Mbps



(b) Link Speed :- 64 Mbps

Figure 5: Distribution of the pause (latency) at the client. Emulation parameters are as shown in Table I. Graphs for two different link speed values are shown.

Figure 6 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.

VIII. 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 information-centric networks enable. 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 that our

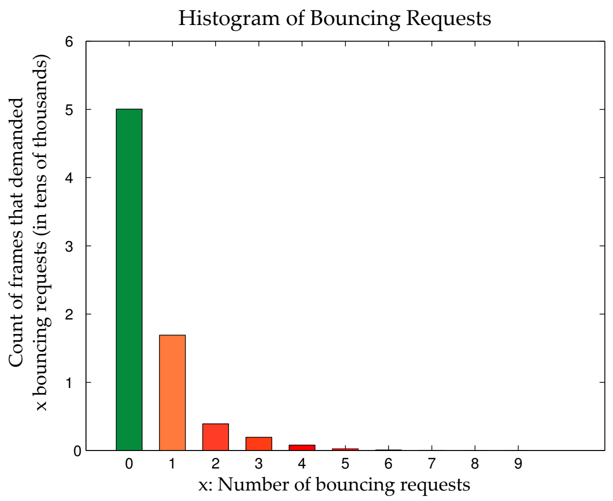


Figure 6: Distribution of bouncing requests

strategy can provide significant gains in latency measures on a real network.

Although significant improvements are observed in the average latency measures, we have observed that there are a few requests for which the latency is high, and attributed this observation to the fact that some requests generated large number of bouncing requests. One way to solve this would be to incorporate better encoding restrictions. Note that having stronger encoding restrictions implies lesser coding on the network, and thus lesser gains. In our current work, we are looking at better ways to handle this trade-off. We are also exploring ways to integrate forwarding strategies into this framework.

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*, 2013.

[2] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," CoNEXT, ACM, 2009.

[3] cisco forecast http://newsroom.cisco.com/release/1197391/

[4] L. Zhang et. al, "Named Data Networkng (NDN) Project," tech. rep., University of California, Los Angeles.

[5] A. Afanasyev and L. Wang, "Adaptive Forwarding in Named Data Networking," 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, ACM, 2008.

[7] D. Han, A. Anand, A. Akella, and S. Seshan, "Rpt: Re-architecting loss protection for content-aware networks," NSDI, USENIX, 2012.

[8] P. Sanders, S. Egner, and L. Tolhuizen, "Polynomial time algorithms for network information flow," SPAA, 2003.

[9] H. Seferoglu and A. Markopoulou, "Opportunistic network coding for video streaming over wireless," in *Packet Video*, 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, 2014.

[12] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar, "An integrated experimental environment for distributed systems and networks," USENIX Association, 2002.