# Revisiting Pervasive Content Caching in Information-Centric Networks

A Project Report

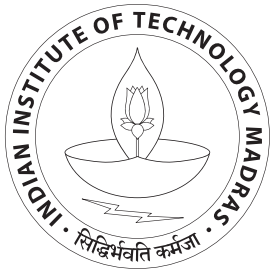
*submitted by*

**ABHIRAM RAVI**

*in partial fulfilment of the requirements*
*for the award of the degree of*

**MASTER OF TECHNOLOGY**
*and*
**BACHELOR OF TECHNOLOGY**

*under the guidance of*
**Dr. Krishna M. Sivalingam**
*and*
**Dr. Parmesh Ramanathan**




**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**
**INDIAN INSTITUTE OF TECHNOLOGY MADRAS.**
**May 2015**

# THESIS CERTIFICATE

This is to certify that the thesis entitled **Revisiting Pervasive Content Caching in Information-Centric Networks**, submitted by **Abhiram Ravi**, to the Indian Institute of Technology, Madras, for the award of the degree of **Bachelor of Technology** and **Master of Technology**, is a bona fide record of the research work carried out by him under my supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.


**Krishna M. Sivalingam**
Research Guide
Professor
Dept. of Computer Science and Engineering
IIT-Madras, 600 036

**Parmesh Ramanathan**
Research Guide
Professor
Dept. of Electrical and Computer Engineering
University of Wisconsin-Madison, 53706


Place: Chennai, India
Date: .

Place: Madison, WI, USA
Date: .

# ACKNOWLEDGEMENTS

Words cannot do justice in expressing my gratitude towards my advisors Prof. Parmesh Ramanathan and Prof. Krishna Sivalingam.

Prof. Parmesh has been a guiding light to me in the past two years. My research interests in computer networking were instigated by all the cool things he exposed me to during my internship. Be it through technical discussions about the future of the Internet or through conversations about life in general, he pushed my intellectual skills to the edge of my limits and beyond, and helped me gain tremendous confidence in my own abilities.

Prof. Krishna has been an inspiration to me, both as an advisor and as a teacher. The amount of support that he extended towards me pursuing my interests and his constant guidance in the whole process, is something that I have been extremely fortunate to have. His warmth, his wisdom and his remarkable ability to ensure perfection in everything he is a part of; is something that I am strongly trying to emulate.

I am forever indebted to both of them for everything I have learnt from them and for their constant support towards this project.

As I write this acknowledgement, I realize my days left in IIT as a student are numbered. The years that I've spent on this campus have in no way been enough to learn everything that it could teach me. I would like to thank everyone who has been part of this journey – my classmates, professors, seniors, juniors, and hostel wing-mates. I am ever grateful to them for the technical and soft skills I've gained during my time here.

I would also like to thank the Indo-UK Advanced Technology Center (IU-ATC) for their support towards this project. Finally, I would like to thank my parents and my younger brother for their unconditional love and encouragement.

# 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. A major challenge is to effectively use these caches to improve content recovery at the client. This work addresses this challenge and is motivated by the recent significant growth in Internet video traffic; Fraction of video traffic is predicted to exceed 70% by 2017. Our contribution is two-fold.

In the first part, we 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 this report. 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.

In the second part, we extend prefetching models in computer processors to an arbitrary network of caches and propose a prefetching oracle. The oracle exploits both content popularity *across* users (spatial information) and temporal predictability of *each* user's content access (temporal information) to reduce latency by placing the *right content* at the *right router cache* at the *right time*. An Integer Linear Programming (ILP) formulation of the oracle's challenge is first presented. A heuristic to solve this ILP without significant computational complexity is also presented. Simulation results show that the available bandwidth in the network is intelligently exploited to achieve significant reductions in access latency, and that small levels of temporal lookahead of user requests are sufficient to achieve a large chunk of this gain.

Overall, we demonstrate that the potential performance benefits of pervasive router level caching are significant and that it must be an integral part of any Information-Centric Network architecture.

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# CHAPTER 1

# Introduction

There is a growing consensus that the *host-centric* architecture of current Internet must evolve to a new *information-centric* architecture to better support emerging applications [20]. For instance, leading projects in the United States National Science Foundation's Future Internet Design (FIND) program (e.g., Named Data Networking [41], eXpressive Internet Architecture [19], MobilityFirst [28]) are based on this premise. In an information-centric architecture, content is treated as a *primitive* entity at the network layer. Applications send out interests for information they desire. Within the network, these interests are matched against published content from different providers. When a match is detected, the information is *diffused* over the network to the applications. The challenge is to diffuse the information in such a way that: (i) the network resources are used effectively, and (ii) application's latency to access the information is small. In this work, 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 [1].

## 1.1 Introduction to Information-Centric Networks

In the contemporary design of the Internet, nodes on the network are labelled 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 labelling 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 work, we work in the context of Named Data Networking [41] protocols for the ICN framework. Interest and data forwarding mechanisms are adapted from the NDN forwarding algorithm [40]. 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.

## 1.2 Caching in Information-Centric Networks

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 travelling in the network, and any redundancy, if detected, is removed [7], or duplicate packets are compressed in order to account for loss protection [18].
*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 travelling 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 work is on improving the *service* aspect of content caching.

## 1.3 Proposed Approaches

In this work, we present two approaches to effectively utilize the pervasive router-level caches in Information-Centric Networks to improve latency and rate measures of content delivery at the client.

### 1.3.1 Network Coded Caching

We propose a caching scheme called Network Coded Caching that integrates network coding and caching. This scheme is an algorithm for cache replacement at the intermediate routers in the network and incorporates principles from Network Coding. Network Coding is a novel technique that is used to maximize information flow in a network, typically in multicast video streaming. A primer on Network Coding is presented in Chapter 2. Network Coded Caching makes better use of the distributed storage that is available on the network and caches content more effectively. We evaluate the algorithm on GENI, an ultra-fast network testbed built across several universities in the United States. We integrate our algorithms into a Named Data Network that we built from scratch for GENI and evaluate the performance of our scheme in two different traffic pattern scenarios (i) Video-on-Demand (VoD), where users stream videos in a YouTube-like video browsing pattern, and (ii) Zipf-based web traffic, where users request for websites in a Web-browsing-like request pattern. We present the results of experiments that evaluate the performance of our proposed scheme against ordinary caching schemes and observe significant gains in latency and content delivery rates. We also present the headers and the additional logical components that are necessary for Information-Centric Networks to support Network Coded Caching.

### 1.3.2 Prefetching in a Network of Caches

We propose to introduce the notion of content prefetching at the ICN routers. We extend prefetching models typically used in computer processors to an arbitrary network of caches. With the aim of setting a benchmark for the best achievable latency and rates when prefetching is incorporated, we propose a prefetching oracle. This oracle has complete access to the future request of the clients on the network. The oracle exploits spatial information i.e content popularity across users and temporal information i.e temporal predictability of every client's content access pattern, and minimizes the latency incurred in serving these requests. It does so by attempting to place the right content at the right router cache at the right time. We formulate the problem that the oracle solves as an Integer Linear Program. We observe that the optimization problem is too huge to be solved by an off-the-shelf solver for reasonably sized network topolo-

gies and universe of content items. We go ahead to present a heuristic to solve the optimization problem without significant computational complexity. Through simulation, we show that the prefetching oracle is capable of intelligently exploiting the available bandwidth in the system to achieve drastic reductions in content access latency. We also observe that it is sufficient to have small levels of temporal lookahead of the future requests of the clients to achieve a major portion of this performance gain.

In a nutshell, we propose caching schemes that unleash the potential benefits of pervasive caching in information-centric networks and present a thorough performance evaluation of these schemes.

## 1.4 Organization of the Report

The rest of the report is organized as follows.

- **Chapter 2** presents the principles of Information-Centric Networking, a primer on Network Coding, a summary of related work to the contributions of this thesis, and a brief overview of the GENI network testbed that is used for performance evaluation.
- **Chapter 3** presents the first improvement that we propose towards effective content caching, which is the integration of network coding and pervasive caching. The chapter also discusses in detail the performance evaluation of the proposed scheme.
- **Chapter 4** presents the second improvement, which is to introduce the notion of content prefetching for an arbitrary network of caches. The chapter completely describes the prefetching oracle that we propose, and discusses the environment and results of the simulation that employs our proposed heuristic solver.
- **Chapter 5** presents the conclusion and discusses directions for future work.

# CHAPTER 2

# Background and Related Work

In this chapter, we set the context for the core contributions of our work. We first present a high-level overview of Information-Centric Networking (ICN) and its architecture, along with the motivation for introducing such an architecture and a summary of the expected benefits of deploying such an architecture on real networks. Second, we present a quick primer on Network Coding, its various applications across literature and some of the papers in network coding related to our contributions. Third, we present related work in the context of prefetching, both in the computer architecture context and in the networking context. Fourth, we present a quick overview of GENI, the networking testbed in the United States that is used as an evaluation platform for our proposed schemes.

## 2.1 A New Way to look at Networking

When Alexander Graham Bell first invented the telephone in the 19th century, he had only one thing in mind – to let two people talk over a wire. He built a system that can achieve this functionality, but the system was restricted to voice-based communication. It was not long before this was commercialized by AT&T, which built the first telephone system based on Bell's ideas. Soon, data communication was built on top of the system that Bell built. Once scientists and engineers figured out how to reliably transmit a bit over a wire, exchanging any kind of information became easy on the same platform. For two users that weren't directly connected by a wire but by a sequence of wires, a dedicated *path* of wires was allotted to every communicating pair, and multiple communication channels were time-multiplexed or frequency-multiplexed over the same underlying wires. This system is commonly referred to as *circuit switching* and comes with its own disadvantages. Particularly notable are the costs of setting up this dedicated channel, and its poor resilience to failures of network components.

Over time, the need for a large number of small-information exchanges made circuit switching unsuitable for communication, since the overhead of the setup was typically more than that of the actual information exchange. Also, a key property of this system was that data had no identity. If a data item travelled several hops and a link failed, nothing could be done to get the same data to the destination. The reason for this behaviour is that the data has no information within itself as to where it is coming from or where it is going to.

One of the pioneering papers that revolutionized data communication as the world then knew it was [9], written in 1964 by P. Baran. The paper described how one can use the then existing physical telephone network to achieve distributed communication via distributed routing techniques. The idea was to extend the ideas of a RAID-like redundancy model to networking: Instead of depending on the reliability of specific components on the network, the idea was to re-design the system to make use of what resources were *available* at the moment. The fact that path existence between nodes is a transitive property was the key notion that was exploited to achieve this functionality. The philosophy of the technique is similar to how the postal system worked in the day. In this framework, every node in the system is labelled uniquely (with an Internet Protocol (IP) address). With these labels fixed, every packet that is to be sent on the network is stamped with the label of the destination. When a node in the system receives a packet, if the destination label on the packet corresponded to its own label, then the destination has been reached and the packet is processed by the node. If not, the node then *intelligently forwards* the packet to one of its neighbouring nodes. This forwarding decision is what defines the routing mechanism of the system, and is handled using routing tables. This decision making process is crucial since it determines the path that each packet takes to get to the destination, and hence the efficiency of the system. A naive way to set up these routing tables is to configure them statically. Further improvements looked at distributed ways to build routing tables – and this includes distributed routing protocols like Open Shortest Path First (OSPF) and Routing Information Protocol (RIP). These protocols work on the principle of heartbeat packets, where each node periodically exchanges information that it has gained with its neighbouring nodes. Over time, every node in the system will have enough information to perform efficient routing. In this framework, data has *some* identity, since the data speaks a bit about itself – the label of the source of the data and the label of the destination to where it is headed. If something goes wrong in the network i.e node failure/link failure, the intermediate nodes can use this information to try and get the data back in the right direction.

This was termed *Packet switching* or *Data Networking*, the technology that powers most of the internet that we know today, and it works really well. Now on a packet-by-packet basis, the system can choose alternate paths on failure of network components, and reliability only increases exponentially with the size of the system. There is no overhead of setting up calls, like in circuit-switched networks, which allows the system to work at high efficiency at

any bandwidth. The Transmission Control Protocol (TCP) was designed as a transport-layer supplement to IP, and together, known as the *Internet Protocol suite*, the system guaranteed a reliable transfer of data packets from one node to any other node on the network.

When TCP/IP was created, machines were huge devices and there were very few of them. There was not a lot of information to be exchanged. The situation in today's internet is the exact opposite. There are several devices, which are much smaller and more mobile, and the amount of information to be exchanged is a lot more. This was not what the current design of the Internet was envisioned for. The Internet also doesn't *like* things that move i.e mobility was not a serious consideration decades back when TCP/IP was crafted. Over the past three decades, the world has changed significantly, and there have been several issues that have come up with the way the Internet operates. Many of these issues have been fixed with major workarounds, while they should have instead been inherently solved by the network. The problem itself, that the Internet aims to solve, has changed over the years, creating a need for a fundamentally better architecture.

One of the biggest assumptions that was made during the design of the Internet was that the data communication would still be about *conversations* between pairs of computers. This fundamental assumption has several limitations. For example, suppose there were a thousand people who wanted to watch the live India vs. Pakistan cricket match in IIT Madras on YouTube. There are essentially a thousand HTTP requests to the same video going out of the campus. Ideally, we would expect one copy of the video stream to be brought from YouTube to the campus server, which is then multicasted to all the users requesting the video stream. But with the current internet design, a thousand copies of the same video would be brought all the way from YouTube since the video stream is still over two-way conversations. There is no inherent way for the network to realize that the thousand users are requesting for the exact same video. The reason for this behaviour is that the video information is at the application layer, and the network routers operate at the network layer, with no faithful way for these router to understand application-level information. A workaround to this problem is to sniff into application-level data via *Proxy Caching* at intermediate nodes, a technique that works specific to the application and *caches* popular content that users request at proxy servers. The OSI stack was originally intended for email-like data communication. With over 90% of the IP traffic today being media content, it is about time that we took a step and addressed networking issues at an architectural level instead of devising complicated protocols over the existing stack. One of the pioneering efforts towards a clean-slate model of the Internet is to treat content as a primitive entity and to shift from a host-centric to a content-centric framework.

## 2.2 The Information-Centric Network Architecture

Information-Centric Networking is a new way to look at Computer Networking, which challenges the fundamental assumption of today's Internet – that of building data communication over a conversation model. ICN envisions a better Internet in which the intent of the user is to retrieve some content rather than to speak to a particular computer on the network. Instead of labelling computers on the network with IP addresses, ICN labels *content* with *names*. The users presents the *name* of the content that she desires to the network, and the network figures out where the content is and brings it back to her. A lot more intelligence is pushed into the network to enable this functionality. The user doesn't know where the received content came from, and doesn't care, as long as she is guaranteed that what she got is what she asked for, and that it is secure and reliable. The key property of this system is that data has complete identity. *Content* is the primitive entity in the ICN framework; Every piece of content is named, independent and self-certifying.

Information-Centric Networking'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/linkinpark/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. There are primarily two data structures at each node that help achieve this functionality: (i) The Pending Interest Table (PIT), and (ii) The Forwarding Information Base (FIB)

- **Pending Interest Table**: The Pending Interest Table at each node keeps track of the interfaces from which each of the incoming interests arrived. Interests to the same content can arrive from multiple users and from multiple interfaces. Once the Data packet for the

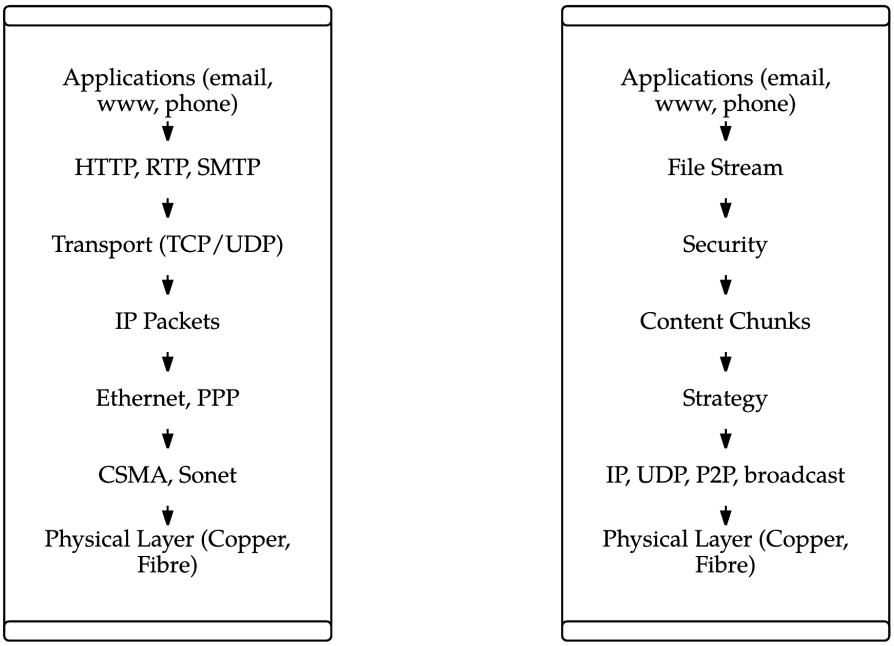



**Figure 2.1: Information-Centric Networking**: The Protocol Stack

corresponding interest is received, the PIT is consulted and the data packet is forwarded back along all the interfaces stored for the interest in the PIT. The PIT is essentially responsible for handling *Content Routing* in the system.

- **Forwarding Information Base**: The Forwarding Information Base handles the forwarding of interest packets based on the content name being requested. The FIB is populated by a routing protocol that is name-based, forming the core of the interest forwarding strategy of the system. Every interest packet that arrives at the node is either served by the node if it contains a copy of the content being requested, or forwarded along a subset of the outgoing interfaces of the node; information about this subset is maintained in the FIB and is dependent on the forwarding strategy that is deployed at the node.

Transport layer functionalities of the OSI architecture are split across the Strategy layer and the Application layer in the ICN model. Essentially, the network provides an unreliable delivery of Interest and Data packets. If either the Interest or the Data packet is dropped , it is upto the requester to re-issue the interest after say, a time-out that is application specific. Senders, being stateless, only match names and serve content. The receiver drives the transmission of data and the rate of delivery. This is done by letting the Forwarding Information Base decide when to re-transmit interests. The FIB and PIT work together to identify if some interest has timed out

and retransmit automatically without any extra effort at the network layer.

In the transport layer of the current internet design, an issue is that there are end-to-end connections. TCP is capable of controlling delivery rates at the ends of the connection, but has no control over the intermediate nodes in the path of the connection. It is also unaware of the other connections that share the same underlying network resources. To ensure that the network resources are effectively utilized and that congestion is avoided, TCP implements congestion control algorithms. These algorithms listen to the network for feedback on packet drops, and then reconfigure parameters (sending rate, congestion window size etc.) with the aim of improving the overall throughput. In the ICN framework, there is no concept of an end-to-end connection, and hence there is no need for explicit congestion control protocols. If the load is high at a particular router, packets are simply dropped. These local control decisions translate to a global scale, and implicitly control congestion on the entire network. Flow balance is maintained at every hop and there is no uncontrolled intermediate node to worry about.

In a nutshell, the key design principle of ICN is to set a strong foundation for effective content dissemination at the scale of the Internet. There are several benefits in this framework of Dissemination-based networking that drive the recently gained popularity of this idea in literature. Some of the crucial benefits that it enables are listed below.

- **Redundancy elimination**: The names assigned to content are transparent at the Network Layer. This makes it easy for every router in the system to *understand* the content item being requested, and also if two users on the network have issued interests for the same content item. Let us reconsider the Live stream example in IIT Madras, where a thousand people want to watch the same India vs. Pakistan cricket match on YouTube. In this framework, the intermediate routers would now be able to identify that multiple users are requesting for the same content item. This information is collected and maintained at the Pending Interest Tables and fewer (if not only one) copies of the video stream are brought from the YouTube servers to the campus server, which is then disseminated to the users requesting the stream. This not only reduces the load on the server, but also substantially reduces congestion and the overall bandwidth usage.

- **Caching**: Improvements to latency and content delivery rate that Information-Centric Networks offer are primarily enabled via this benefit. Since data is independent and self-certified, it can be served from anywhere on the network. Routers can keep a local copy of popular content items (especially Video content, which makes up over 90% of the Internet traffic today) in a dedicated *Content Store*, which is a data structure at every router to maintain cached content items. If interests to the same content arrive from different users at the same node, but close in time, the locally cached copy of the data can be served instead of having to go all the way to the content source. This is not possible

in the current Internet since the content information is at the application layer and the routers work at the network layer. With ICN, pervasive in-network caching is enabled, with every router made capable of keeping copies of content items for serving future requests.

- **Mobility**: In the current Internet design, if a user streams a video while connected to an LTE network, and then shifts base stations or to a WiFi network, the IP address of the user's device would change. This would break the stream of content delivery and the video server would need to be informed about this change in order to re-route the stream to the new IP address. This issue arises in any push-based delivery mechanism, where senders are not stateless. Since ICN is pull-based and nodes in the system do not have any identity, there is no overhead or issue with moving across sub-networks: The path of breadcrumbs that is followed by the data will now change at the least common ancestor of the sub-networks across which the mobility shift happened, making the overhead insignificant.

Given this context, the feasibility of ICN deployment is still a topic of heavy debate [26], with the cost of its potential benefits being put into question. Several methods have been proposed to deploy ICNs as cost-effectively as possible, and many others to incorporate their benefits incrementally without obliterating the existing IP infrastructure (eg., [23, 39] and more recently, [12] and [38]). To illustrate, [36] claims a 20% average reduction in the packet path lengths in terms of hop count by enabling pervasive content caching at routers, along with a significant benefit compared to that of edge caching. But recently, [16] showed that the benefits of caching beyond the edge were insignificant and instead proposed an incrementally deployable architecture to achieve the same(almost) benefits with "less pain".

Our focus in this work is to address issues in the redundancy elimination and caching aspect of ICN's benefits. We provide schemes and mechanisms that attempt to unleash the hidden powers of the pervasive router-caching framework that ICN enables.

## 2.3 Network Coding and Related Work

*Network coding* is a novel idea in which the nodes in the network do not merely store-and-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 2.2). The problem is to multicast

both packets A and B to the two right-most nodes shown in the figure. Without network coding, a rate of at most $1.5$ packets per time unit can be achieved (See [30]) 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 [30] is typically implemented to achieve scalable network coding at gigabit speeds. The paper presents a polynomial time code construction scheme to achieve maximum flow multicast rates. The key challenge in Network Coding is to ensure that the encoded packets that are received are linearly independent. To ensure linear independence, the paper takes an optimistic approach and performs coding with coefficients that are randomly chosen from a finite field. The paper then proposes a polynomial time algorithm to test linear dependence quickly. It is also proven that the probability that random coding will result in linearly dependent codes is significantly low. We also employ random linear network coding in the Network Coded Caching scheme that we present in Chapter 3.

Network Coding as a technique as also been applied to distributed storage systems to increase reliability (content recovery upon failure) by adding redundancy. For example, [14] proposes regenerating codes that allow the system to reconstruct lost packets upon failure of nodes. The paper shows that a trade-off is achieved between the amount of redundancy that is introduced in the system and the bandwidth needed to restore the lost information upon failure. This notion is similar to performing Erasure Coding for repair within a single node, but generalized to a recovery scheme that operates across a network.

Several coding schemes have been proposed to achieve maximum flow rates in a variety of scenarios that *resemble* multicast. For example, [33] deals with heterogeneous receivers (each with a different maximum flow rate to the content source). The paper proposes an algorithm – *Linear Information Flow*, to achieve maximum-flow content delivery rate to all the receivers in a multicast multi-rate video streaming scenario that employs layered source coding i.e quality enhancement layers for video.

Network coding has also been used to address long term latency issues in content access. [27] proposes a content placement strategy for servers Content Delivery Network (CDN) in order to minimize the long term content access latency of the users in the system. The paper assumes a stochastic model for user request patterns and performs an optimization around this model. The idea of formulating the problems we address in this paper as optimization problems was inspired from the philosophy of this paper.

We employ several of the principles in the above papers and introduce them in the pervasive caching framework that Information-centric networks enable. Our proposed scheme, Network Coded Caching, is described in Chapter 3.

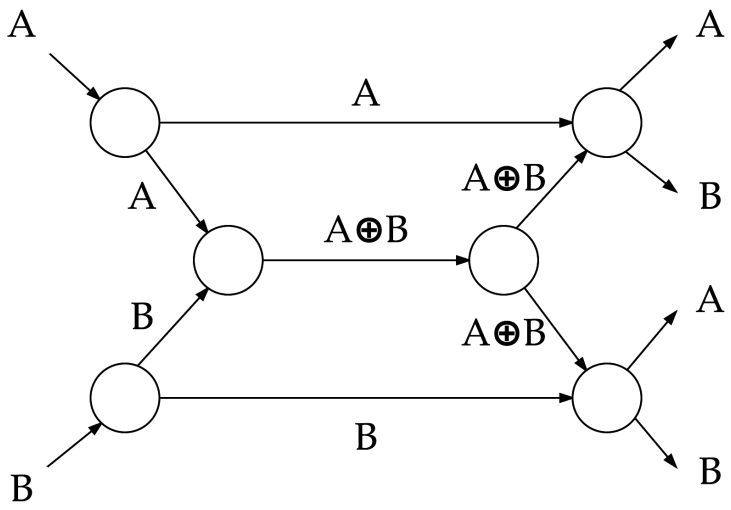



**Figure 2.2: Network Coding on the Butterfly Network**: Multicast is achieved at the maximum flow rate to the servers at both the clients.

## 2.4 Prefetching and Related Work

In this section, we review related work in the field of processor prefetching and prefetching in the content of web traffic, the motivating factors for introducing prefetching in the framework of Information-centric Networking.

Prefetching is a thoroughly studied problem in the context of computer processors. Models to predict the memory addresses that the processor would generate in the near future have been extensively studied in addition to the architecture needed to bring the elements from memory to the caches ahead of time. [22] is one of the pioneering papers that built a markov model based on the processor's memory request patterns. The paper proposes a hardware interface between the low latency chip caches and the high-latency main/secondary memory. The interfaces predicts the future requests of the processor by building a markov model and brings them to the on-chip caches ahead of time.

With the advent of multi-core processors equipped with dedicated caches for every core, the problem of effective prefetching became a significant challenge. The architecture community began exploring co-operating techniques for prefetching. For example, [15] proposes a technique to deploy multiple prefetchers for each of the processors and then co-ordinate them using synchronization protocols. The co-ordination is to manage the aggressiveness of each of the prefetchers to minimize bus interference with the main memory.

Caching and prefetching in computer processors has also received substantial attention from the theoretical perspective. [6] describes the theoretical foundations for a combined prefetching

and caching strategy. The paper assumes complete knowledge of the future requests of the processor and finds the minimum time needed to serve these requests in this framework. The paper formulates an integer program and proposes an approximation algorithm to solve the optimization. The work sets a theoretical bound on the achievable latency improvements, a bound that processors strive to achieve in practice.

Prefetching has also gained significant attention in the content of computer networking. Most of current literature in improving content delivery focuses on either of two aspects. The first set of methods deals with complete knowledge about the future requests (predictive information) of the users in the system, which is common in the context of video streaming, and focuses on trying to optimize the *rate* at which content can be delivered to a particular user/set of users. (for example, in the multicast scenario [33]). These techniques can be implemented in the ICN context using push-based overlay mechanisms. For example, [24] establishes a push tree using *persistent* interests. The second set of methods deals with a system that assumes complete information about the content popularity in the system, typically an Independent Reference Model (IRM) based zipfian workloads eg., [11, 16]), and are less concerned about the sequence of interests (temporal correlation) of a single user. The focus of these techniques is to optimally allocate caches or distribute content across servers, typically in Content Delivery Networks, to bring down the average latency of the system. Although some authors have looked at temporal variations of content popularity [35], the models are still not accurate enough in time to allow exploiting temporal correlation in any way to specifically reduce the latency experienced by an *individual* client. Our intention is to exploit *both* these ideas to design an integrated caching/prefetching strategy for the ICN framework. The motivation behind such an approach is to lead towards a generalized framework that can handle request stochastics of any form in the system.

Several approaches to *static* or *long term* prefetching of content have already been proposed (eg., [8, 34]) based on content popularity distributions, especially in the Content Distribution Network (CDN) context. Most of these approaches deal with optimal content placement across servers, which typically have large storage capacities, in the long term and only take into consideration what we define as *spatial locality* (content popularity across nearby users). In contrast, we design dynamic data prefetchers for generalized topologies consisting of a large number of typically small-sized router caches while *also* exploiting any predictive information about the clients' interests in the *near* future. Our overall intention is to have the right content, at the right place, at the right time, to minimize the request latencies that the users experience. Although existing literature has addressed the problems [25] and has presented issues [13] while prefetching in the *world wide web* context, we show that, in the ICN setting, pervasive caching can be thoroughly exploited to reduce latency via a cache-bandwidth trade-off. Details of the prefetching oracle that we propose in this work are presented in Chapter 4

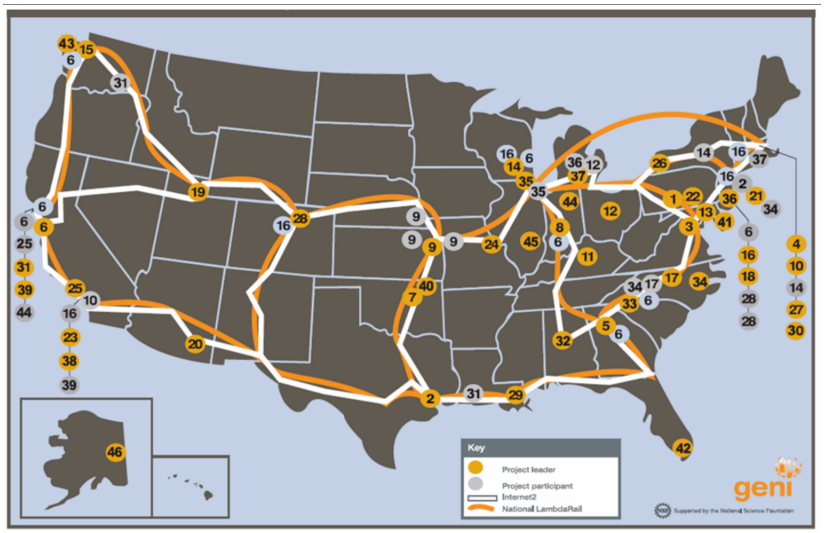



**Figure 2.3: GENI Physical Network**: (Map credits: http://www.geni.net). The physical network across the United States that GENI spawns user-specified topologies over.

## 2.5 Global Enterprise for Network Innovations (GENI)

In this section, we present a brief overview of GENI, the NSF-funded nation-wide Network testbed in the United States, the platform we use for evaluating our proposed schemes at the scale of the Internet.

Large-scale networks are hard to model owing to their complex behaviour. Most mechanisms and algorithms that are proposed for large-scale deployment over the Internet hence need to be empirically evaluated at this scale in order to demonstrate the true performance of the proposed schemes.

In order to provide Networking researchers with an experimentation base to test new networking protocols, architectures and algorithms at the scale of the Internet, GENI (Global Enterprise for Network Innovations) [10], a federated testbed for innovative network experiments, was founded and funded by the National Science Foundation (NSF). GENI allows experimenters to specify network topologies with precisely defined parameters (Nodes, layer 2 connections across nodes, compute resource specifications at each node, Operating systems on each node etc.). GENI then spawns a real instance of the specified topology on the physical network that the GENI Network Virtualizer operates on. The physical network is an ultra-fast network of computers across several universities in the United States (See Figure 2.3). The Network Virtualizer maps the requested topology to *slices* of real nodes, links and network resources that are currently available on the physical network. It spawns the requested topology over these resources and abstracts out the underlying virtualization process from the user. Once spawned, GENI provides the user with SSH access to the nodes. The user can then proceed to

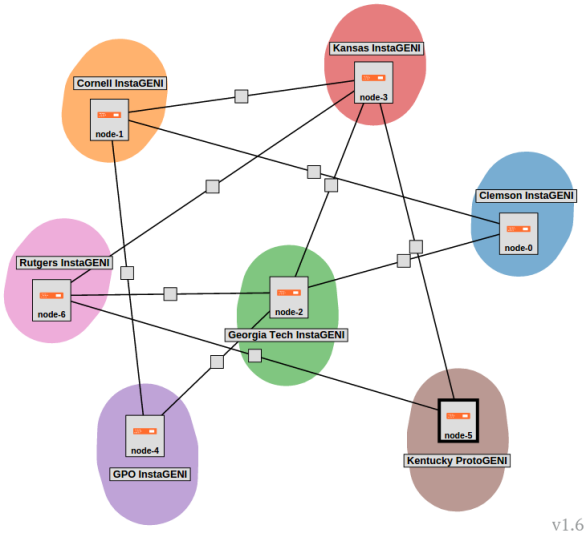



**Figure 2.4: SliceJacks Screenshot**: A sample 7-node stitched topology with real PC instances spawned on GENI across multiple aggregates.

load his protocol programs into the nodes, which are Layer-2 connected. This allows experimenters to run their own protocols at Layer 3 and above. GENI enables deep programmability by allowing the experiments to control the behaviour of not only the end hosts but also the routers and switches in the core of the network. GENI also comes with several measurement and instrumentation tools that simplify topology specification (SliceJacks GUI) and the collection and analysis of experimental results.

The topology that is requested is written in a specific XML format called the Request Specification (RSpec) format. The RSpec file describes the nodes in the topology, along with the number of interfaces at each node, layer 2 connections to other nodes and the OS/programs to be installed in these nodes. GENI spawns the topology on the physical network based on the parameters in the RSpec file. The spawned topology that the user observes and has access to, is termed a *Slice*. One of the key properties of GENI is the isolation of slices. Although multiple slices may share the same physical network, the experimenters will not experience the network effects of other slices that share the same physical resources. The GENI Network Virtualizer divides compute/network resources into isolated *slivers* and allocates these slivers to the slices created. A *Project*, led by a Project lead, typically manages a collection of slices related to a particular experimental goal. The manager of the physical network that lends networking resources for spawning the topology is called an *Aggregate Manager*. The GENI portal [3] allows the user to upload the RSpec files and to obtain the information and commands necessary to access the nodes of the spawned topology. Each aggregate can roughly support upto 100-200 real PC instances and upto around a thousand virtual machine node instances.

In order to evaluate the Network Coded Caching scheme that is proposed in Chapter 3, we

first implement a complete Information-Centric Network and deploy it on the GENI testbed. We then proceed to compare the performance of our algorithm against other caching algorithms on this framework. Details of the implementation and results obtained are presented in 3.4.

# CHAPTER 3

# Integrated Network Coding and Caching

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.

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, [31] and [33] have focused on improving multicast streaming by achieving maximum flow rates with network coding. On the other hand, [27] 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 work, 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 above to develop a ubiquitous network coded caching framework for ICNs. 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

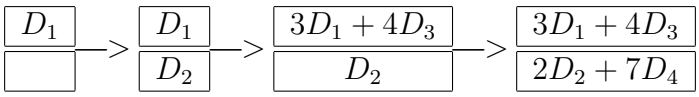



**Figure 3.1: Content Store snapshots** - At an arbitrary router with a cache size of 2 content items. Content items are received in the sequence $D_1$ to $D_4$

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.

## 3.1 Network Coded Caching

In this section, we describe our proposed network coding integration for content caching in ICNs. A primer on Network Coding is presented in Chapter 2.

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. Consider an example where $C = 2$, as demonstrated in Figure 3.1. 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 $D_1$or $D_2$ 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 section.

## 3.2 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.

### 3.2.1 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.

#### 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.

#### 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.

### 3.2.2 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.

#### Example

We illustrate the decoding process with an example, which is illustrated in Figure 3.2. 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.

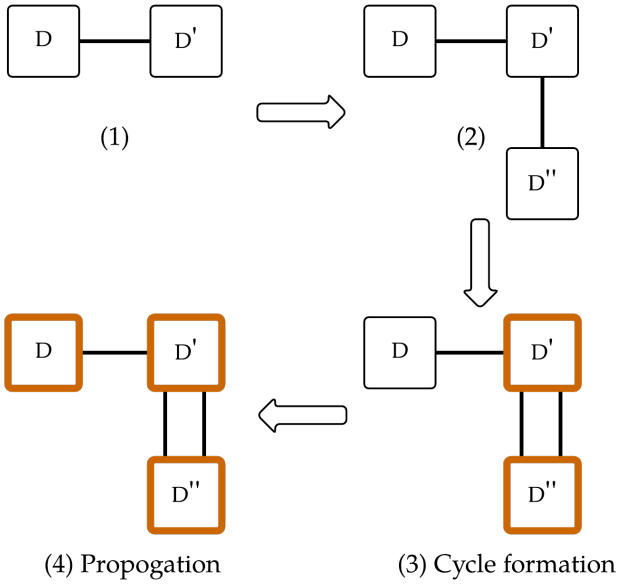



**Figure 3.2: Visualization of the decoding process** at the client. Cycles resolve dependencies. In this case, two coded data items that are different linear combinations of $D'$ and $D''$ are received, thus forming a cycle and resolving the dependency. $D$ is also resolved via a propagation of resolution. Thick edged boxes indicate resolved (decoded) content items.

### 3.2.3 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.

#### 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.

#### 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.

## 3.3 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.

### 3.3.1 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.

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

### 3.3.2 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.

### 3.3.3 Visualizing the Performance gain

Consider the network shown in Figure 3.3. (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 parallel. This demonstrates how network coded caching can take advantage of multi-path routing to improve throughput.

## 3.4 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.

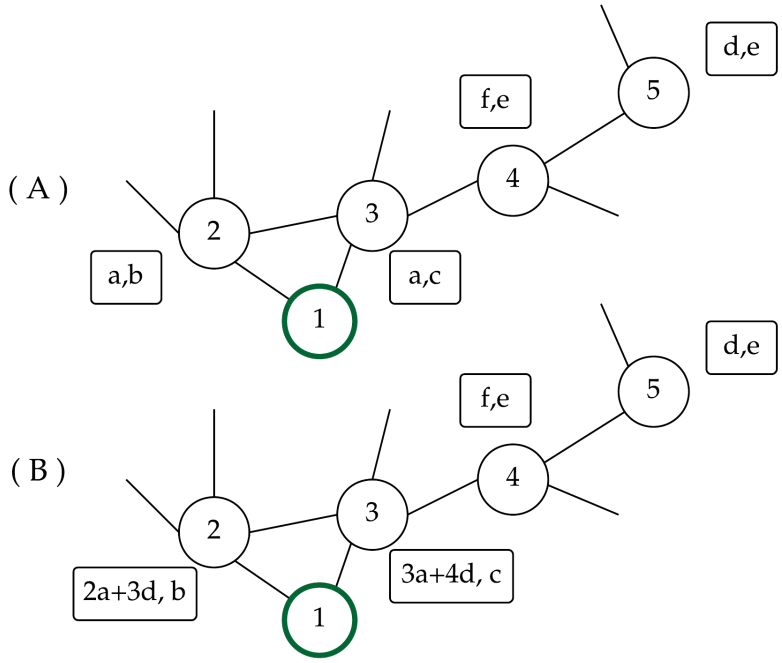



**Figure 3.3: Visualizing the potential gains** of network coded caching. Content store snapshots for the corresponding nodes are shown in rounded rectangles.

### 3.4.1 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 [10]. 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 [40]. We use flooding as the forwarding *strategy* at the routers, instead of the red, green, yellow port adaptive forwarding discussed in [40]. 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 [4]. 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.

| 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 3.1:** Summary of Emulation parameters

### 3.4.2 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 3.1 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.

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.

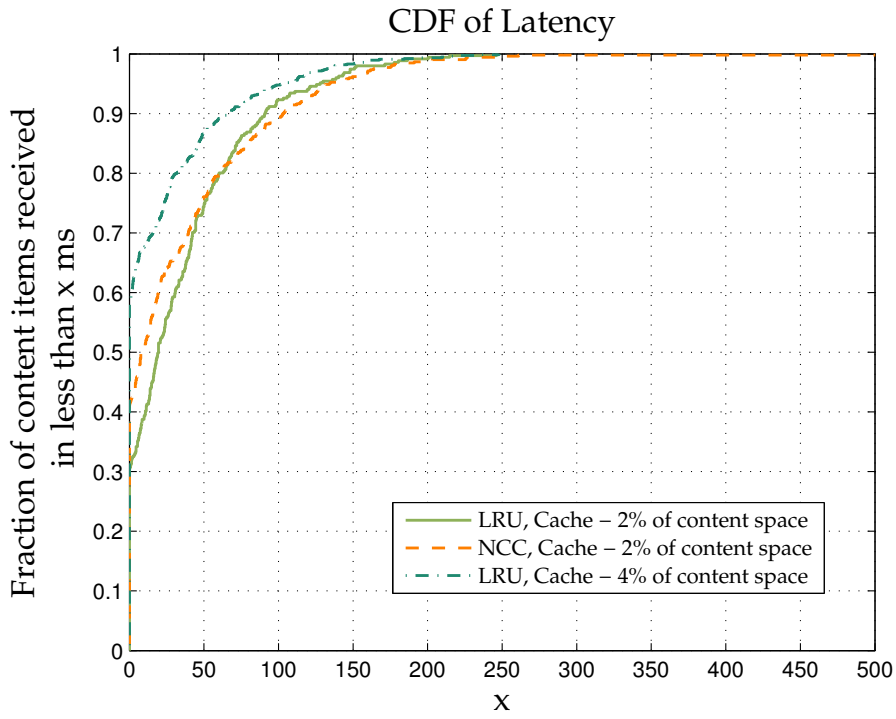



**Figure 3.4: Distribution of the pause** at the client. Emulation parameters are as shown in Table. The router cache size is set to 2% of the content universe (400 content items)

### 3.4.3 Observations

Figure 3.4 shows the distribution of the pauses at the clients with and without network coded caching for the set of parameters mentioned in Table 3.1, 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 3.5 shows the histogram of the number of bouncing requests generated for each

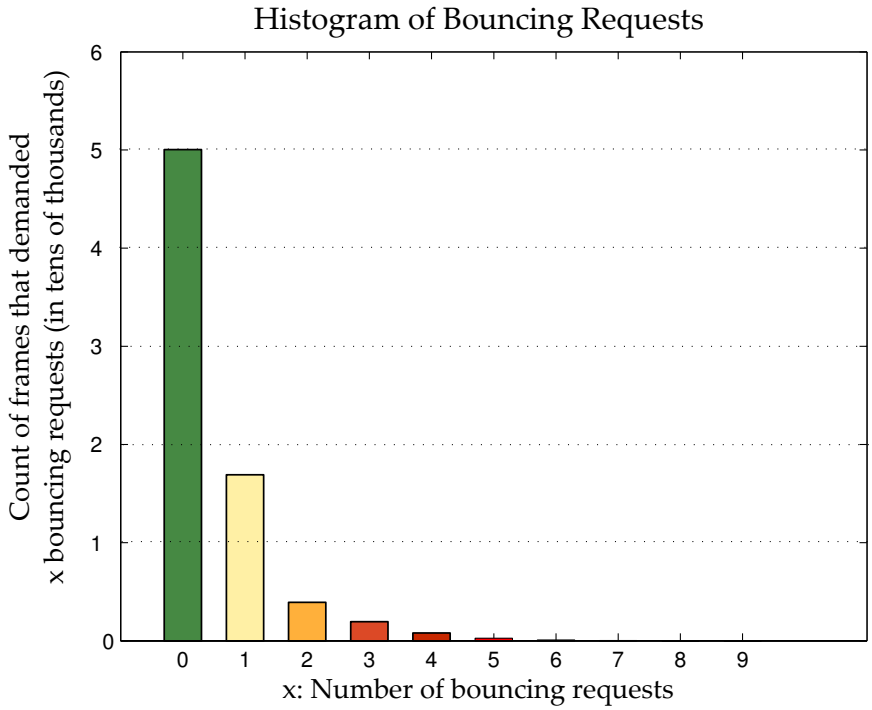



**Figure 3.5: Histogram of bouncing requests**: Significant number of requests that generate a large number of bouncing requests

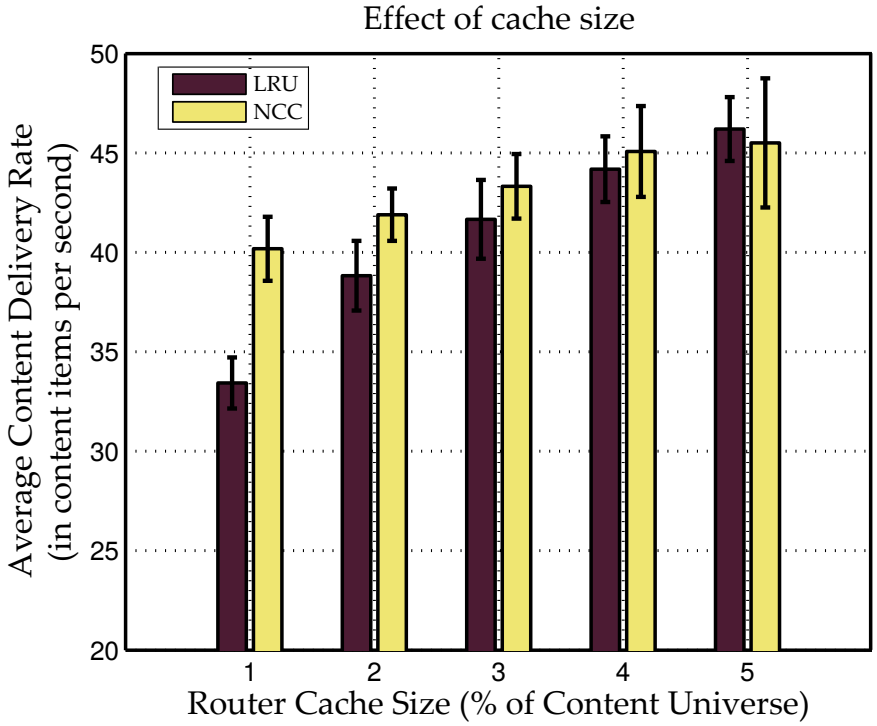



**Figure 3.6: Average rate vs. Router cache size** - The two sided error bars represent the 95% confidence interval of the average rate, based on the experiments conducted.

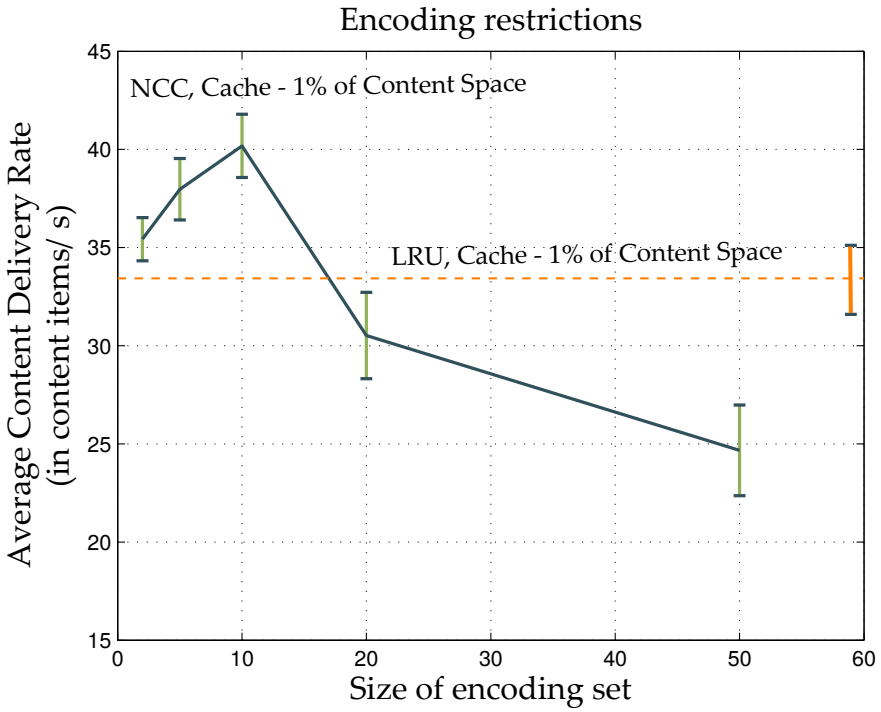



**Figure 3.7: Average rate vs. Encoding Set size** - The two sided error bars represent the 95% confidence interval of the average rate, based on the experiments conducted.

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 ($\geq 3$) 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 3.4.5).

Figure 3.6 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 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 3.6 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 3.7 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.

### 3.4.4 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 [2].

We plot the cumulative distribution of the latency incurred in serving these requests under different caching scenarios. Figure 3.8 shows the comparative performance of Network coded caching against ordinary LRU caching (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

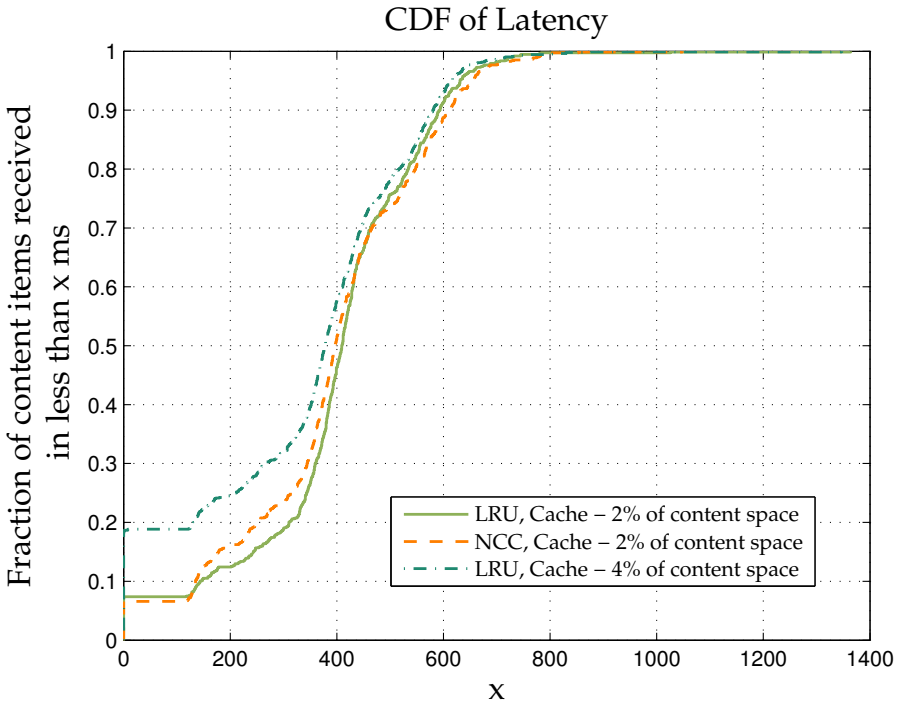



**Figure 3.8: CDF of Latency** for Zipf-based web browsing

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.

Figure 3.9 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 3.10 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

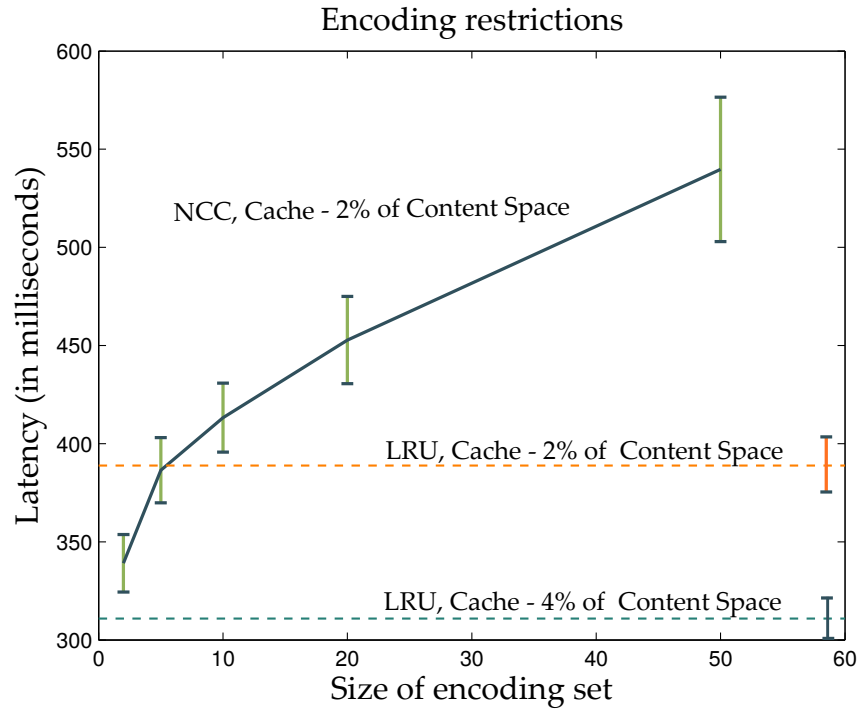



**Figure 3.9: Average Latency vs. Encoding restriction** - Two-sided error bars represent the 95% confidence interval of the mean, based on the experiments conducted.

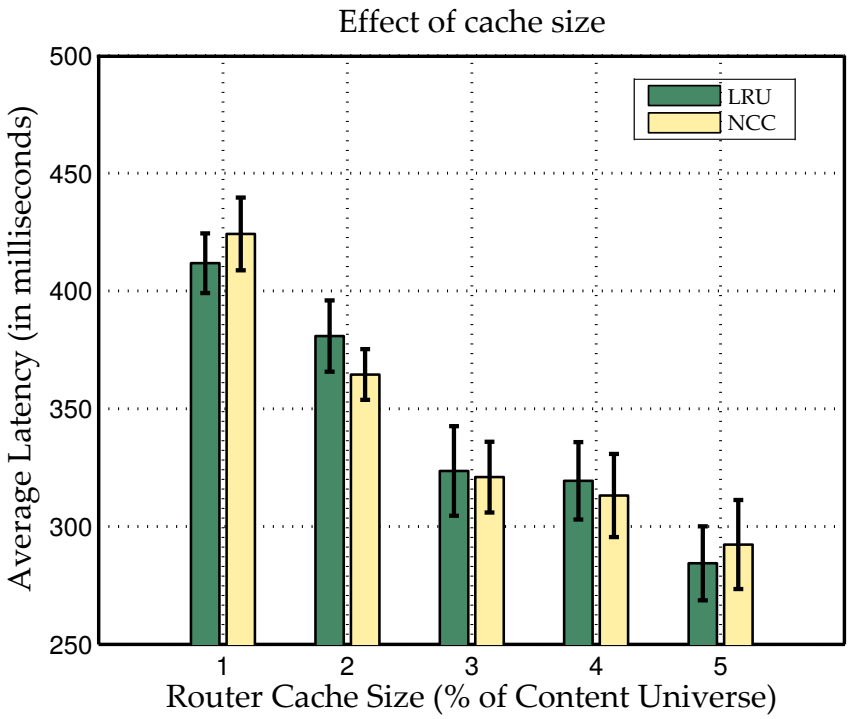



**Figure 3.10: Average Latency vs. Router cache size** - The two-sided error bars represent the standard deviation of the average latency calculated over 5 runs.

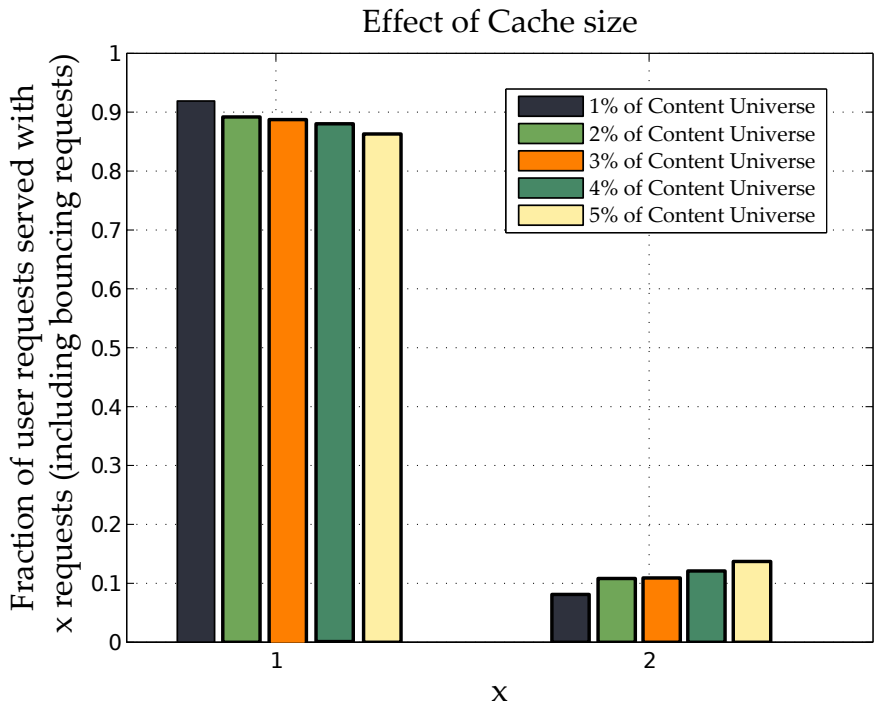



**Figure 3.11: Bouncing requests vs. Cache size**: Zipf-based web traffic.

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 performs 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 3.11 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.

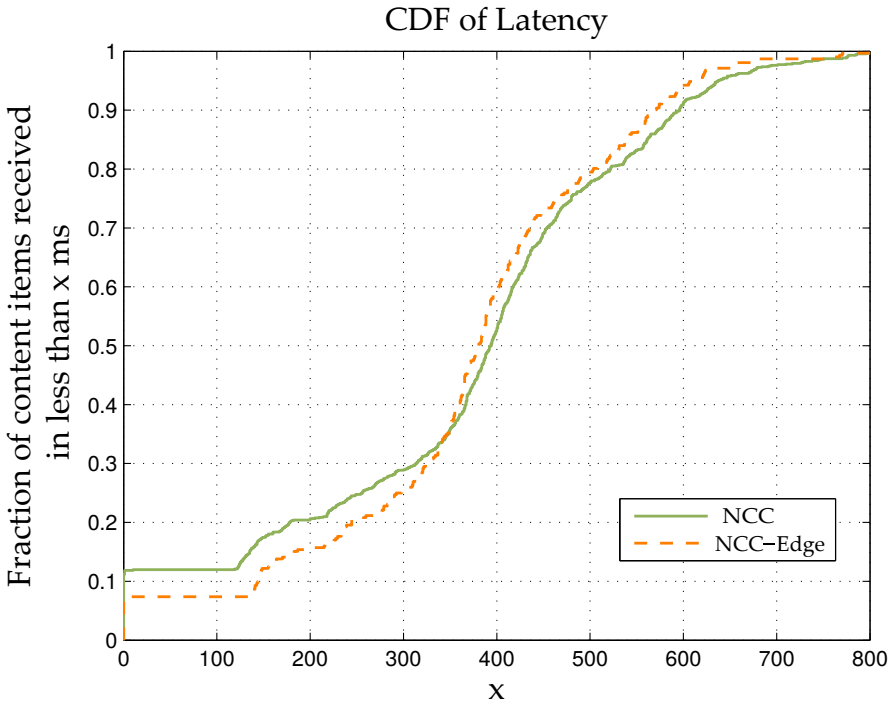



**Figure 3.12: Edge Coding - CDF of Latency** for Zipf-based web traffic

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).

### 3.4.5 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 atleast 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. 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 3.12 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

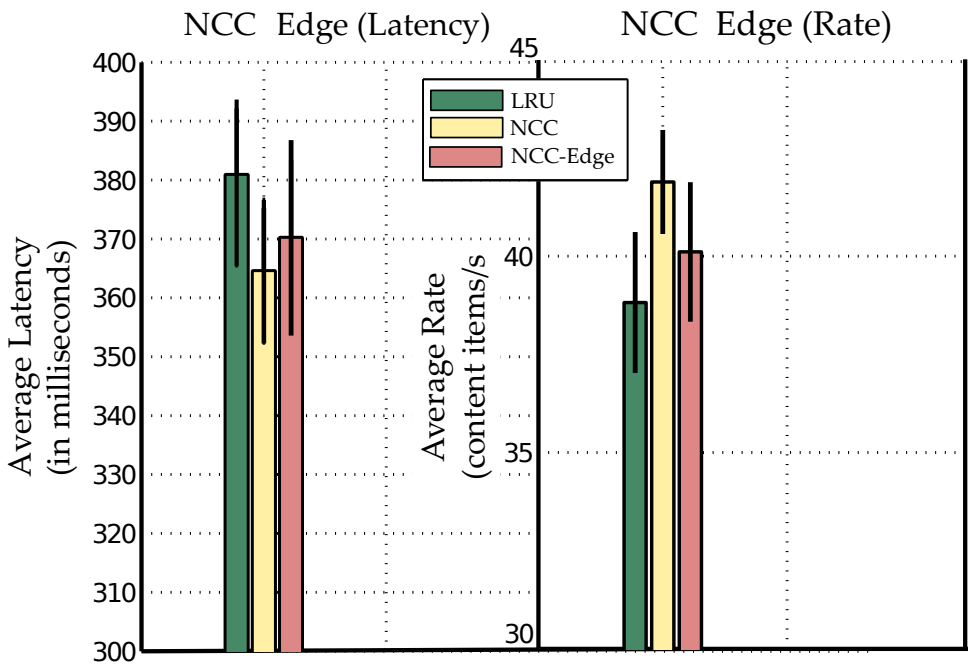



**Figure 3.13: Edge Coding Performance comparison** with NCC and LRU: (left) latency in zipf-based web traffic and (right) content delivery rate in VoD

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 3.13 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 LRU might not be statistically significant (two tailed $p$-values $= 0.41$ for latency and $= 0.19$ for rate). Nevertheless, 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 is *sufficient* to achieve a good fraction of the gain that pervasive-coding has to offer.

## 3.5 Summary

In this chapter, 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 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.

# CHAPTER 4

# Prefetching Oracles for Pervasive Caching

In this chapter, we propose and evaluate a router-level content *prefetching and caching* strategy to address the challenge of effective content caching and delivery in ICNs.

The goal of router-level prefetching and caching is to exploit the available bandwidth in the network to place content items in the right caches *at the right time* so that applications can access them with minimum latency. There is extensive research in literature on content distribution networks that have addressed the problem of effective application-level content caching. They typically assume stochastic knowledge of content popularity among users (e.g., Zipf distribution with known parameters [16]) and then strive to prefetch and place contents near edge networks so that application requests from end hosts do not have to travel far to reach the content they desire. This body of literature typically assumes that the application-level caches are servers with large storage capacities and with ability to simultaneously provide contents for a very large number of users [8, 34]. In contrast, our work focuses on router-level caching at the edge networks.

Specifically, in this work, the *source* of the content is assumed to be an application-level cache of a content distribution network. Within the edge network, each router is assumed to have a content store to cache items that are likely to be needed in the near future. Note that, router caches are usually much smaller compared to server storage capacities typically assumed in application-level caches. Of course, one major issue is to predict who will the need a particular content and when. This work, however, does not focus on this issue. Instead it focuses on identifying the network layer transactions needed to place the content items at the right cache at the right time and for the right duration. It assumes perfect knowledge of applications' immediate future interests. The rationale is to characterize the potential benefits of router-level prefetching and caching and the key factors that determine these benefits so that one can design the right protocols for the information-centric Internet architecture.

Similar to this work, video streaming research in literature also usually assume full knowledge of future application requests for content. For example, there is extensive work on optimizing the delivery rate of video streaming in a multicast scenario where a potentially large number of users are simultaneously watching the same video. This synchrony in access among all the users plays a key role in the solution [24] and these solutions are not effective in situations where the access to the same content is not synchronized. The oracle in this paper does not require synchrony in content access among users. It can, however, take advantage of the synchrony that may exist between the users.

Specifically, our prefetching oracle unleashes the potential of pervasive router-level caching by exploiting both spatial information (content popularity across users) and temporal information in the interest sequence of each user. The oracle's strategy is first formulated as an Integer Linear Program (ILP). We also propose a heuristic to solve the optimization problem. Although sub-optimal, the heuristic is computationally inexpensive and still shows the tremendous potential router-level prefetching in information-centric architecture. For several different edge network topologies, the improvement in latency relative to conventional content access scheme is over 50%. Also, as expected, the improvement increases with increase in temporal prediction interval. Finally, the distribution of the latency to access different contents shows that the oracle is able to place the right contents at the right place, at the right time.

Clearly the use of spatial and temporal localities in prefetching and caching has been well-explored in the computer architecture community [6, 21, 22, 37]. The approach in this work extends the principles and notions from the architecture community to an *arbitrary* network of caching elements with multiple request generators and content sources. The number of request generators and sources is much larger in our context than the common case in the computer architecture area.

The rest of this chapter is organized as follows. The system model is described in Section 4.1. An Integer Linear Programming formulation of the oracle's strategy is presented in Section 4.2 and a heuristic to efficiently solve this optimization formulation is proposed in Section 4.3. The results of an empirical evaluation of the oracle's performance is presented in Section 4.5. The chapter concludes with Section 4.6.

## 4.1 Prefetching in a Network of Caches

In this section, we generalize the notions of prefetching to an arbitrary network of caches and set the system model and foundations for the problem that we address in this work.

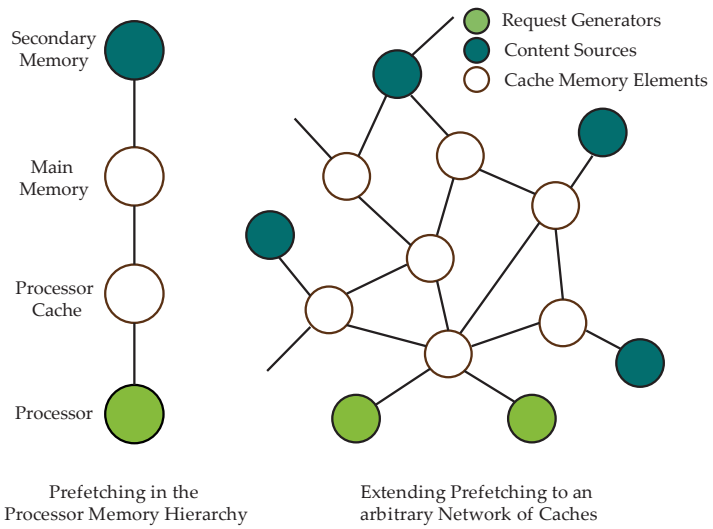



**Figure 4.1: Network of Caches** Framework.

### 4.1.1 Generalizing the Prefetching framework

In the processor setting, prefetching is typically performed from off-chip to on-chip memories, or from the secondary memory to the main memory. Each processor core generates a sequence of requests, and the goal of the system is to reduce the latency of serving these requested items from memory. Requests that *hit* in the on-chip cache are typically served with latencies that are orders of magnitude lesser than those served off-chip. In the single processor setting, we can view the system as a linear graph (see Figure 4.1), and in the multiprocessor setting as a hierarchy of memory elements. In our setting, the multiprocessor hierarchy generalizes to arbitrary networks of memory (see Figure 4.1). The nodes of the graph represent memory elements, and edges represent direct connectivity. Each edge is associated with a latency value for unit data transfer. This setup can be naturally viewed from the ICN perspective where we have a network of router *content stores.* Note that there can be multiple clients and content origin servers that can lie *anywhere* in the network. If we assume that the routers of the system can generate requests in addition to forwarding them, i.e., behave like clients in some sense, we have a framework where routers can *prefetch* content for a nearby client.

Evaluation methods for prefetchers also extend directly from the processor context, with *Prefetch coverage*, *prefetch accuracy*, and *timeliness* [22] being the terms in concern. The three traditional questions regarding *what* to prefetch, *when* to prefetch, and *where* to place the prefetched content also apply in this extended context. Our goal in this work is to set up an *oracle* solution that answers these three questions, as a benchmark for designing distributed prefetching policies.

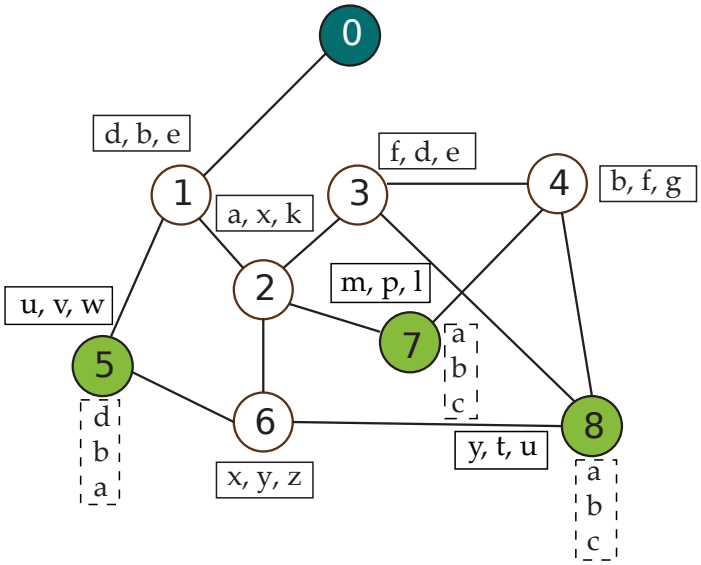



**Figure 4.2: Benefits of Prefetching** :- Request queues at the clients are shown in dotted boxes, along with the snapshots of the content stores at the routers in solid boxes.

### 4.1.2 Model and Definitions

#### System Model

Extending the processor model used by [6], we make the following simplifying assumptions about the network in consideration. Let $G = (V, E)$ be a *simple unit capacity* network. We assume that :-

- Each content item $\alpha \in \zeta$ is of unit size.
- Each client $c \in C$ in the system is associated with a interest sequence $\mathbf{r}_c = (r_{c,1}, r_{c,2}, ..., r_{c,R})$ of length $R$, where each request item $r_{c,i} \in \zeta$ . The interests are serialized and issued in sequence i.e., if $r_{c,i}$ has been issued, then $\forall j < i$, $r_{c,j}$ must have been served successfully. Since each client has a small buffer, the requested content need not arrive in serial order. However, each client maintains a window of size $w$ and it will not issue a request for a new content that is $w$ steps ahead of any current pending requests.
- Each node $i \in C \cup R$ is associated with a content-store capacity $L_i$. Each server node $s \in S$ is the origin server for some subset of $\zeta$.

**Benefits of Prefetching**

Figure 4.2 illustrates a simple case showing how prefetching can be used to exploit both spatial and predictive information via the availability of spare bandwidth in the system to reduce latency at the clients. Consider the given network where node $0$ is the origin server, and nodes $5$, $7$ and $8$ are clients. The request queues and content store snapshots at a certain time are as shown. Since the nearest replica of $a$ for node $8$ is at node $2$, we need at least two hops to bring it to $8$. Observe that node $4$ has content $b$ readily available for node $8$, and sends it across right away. The choice now remains whether to bring $a$ from node $2$ via node $3$ or via node $6$. Now, by exploiting *spatial information*, i.e., the fact that node $5$ also needs $a$ in the *near* future, we would prefer to bring $a$ via node $6$, so that a copy of it can be retained[1] for service to node $5$. By exploiting predictive information i.e the fact that node $7$ needs $c$ in the near future, we could bring $c$ from the origin server to node $1$. Note that even though we have these choices here, they are necessarily because of the availability of spare bandwidth along those links. Though it is not always the case that this bandwidth will be available for our exploitation, applying these prefetching techniques will in essence *create* spare bandwidth along several links for future requests. For example, since now $c$ has already been placed at node $1$, we have spare bandwidth along the $0-1$ link when $c$ is actually needed by node $7$. Note that, this example has not worked out the full details for the given snapshot, but has rather motivated the need for executing these *key* decisions to reduce latency.

Given this setup, we now begin to establish some basic definitions to ease our understanding of the system.

**Definition 1. (Transaction)** A *transaction* $\tau_{e,t,\alpha,\beta}$ is an indicator variable that represents the transfer of a single content item $\alpha$ across an edge $e = (e_s, e_f)$ in the network.
It is associated with four indexing parameters :- The directed edge $e$ along which the transfer happens, the time slot $t$ of the transfer, the content $\alpha$ being transferred, and the content $\beta$ being replaced at $e_f$.

With just this notion, in the next section, we go ahead to formalize the arguments presented in the sample case by presenting a centralized optimization formulation to the problem.

## 4.2 The Oracle Problem Formulation

In this section, we formulate the centralized prefetching problem that the system is trying to solve, as an *Integer Linear Program*. We assume a complete-information scenario i.e., *oracle* access to the request queues of all the clients and to the snapshots of the system. The

[1] If $a$ was needed by 5 only in the *far* future, it may not be advantageous to do so.

| Notation | Explanation |
|---|---|
| $G = (V, E)$ | A unit capacity network. |
| $e = (e_s, e_f)$ | A unit capacity edge in the network |
| $C$, $N$, $S \subseteq V$ | Set of Client, Router, and Server nodes respectively. |
| $L_i$ | Content-store capacity at node $i$ as number of items. |
| $\zeta$ | (Finite) Space of content items. |
| $r_{c,i}$ | The $i^{th}$ request of client $c$. |
| $R$ | Length of the request sequence of each client. |
| $\tau_{et\alpha\beta}$ | Indicator variable for transaction along edge $e$ at time $t$. delivering $\alpha$ and replacing $\beta$. |
| $d_{i,j}$ | Shortest path distance between nodes $i$ and $j$. |
| $d_c(\alpha)$ | Distance of client $c$ to the nearest replica of content $\alpha$. |
| $i_\alpha(c)$ | First index (starting from 0) of content $\alpha$ in $c$'s request queue. |
| $n_c(\alpha)$ | Set of nodes containing the nearest replica of content $\alpha$ for client $c$. |
| $C_{\alpha,t} \subseteq C$ | Set of clients for whom $\alpha$ is in the request queue at time $t$. |
| $C_{\alpha,t,k} \subseteq C$ | Set of clients for whom $\alpha$ is among the first $k$ items in its request queue at time $t$. |
| $y_{i,t,\alpha}$ | Indicator variable representing if content store of node $i$ has $\alpha$ at time $t$. |
| $\mu_{c,i,t}$ | Indicator variable representing if $r_{c,i}$ was served at time $t$. |
| $w$ | Maximum number of pending requests at a client |

**Table 4.1:** Summary of Notation

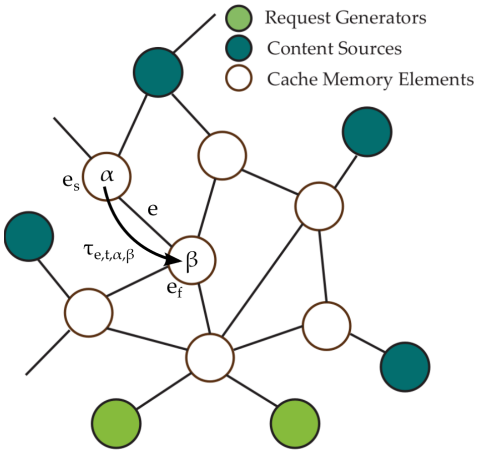



**Figure 4.3: Transaction** :- The transaction $\tau_{e,t,\alpha,\beta}$ in action.

problem now reduces to finding the best *transaction scheduling* scheme. Note that this is a completely deterministic scenario, and there are no stochastics involved. The intention is to set a benchmark for the best achievable latencies, given exact spatial information and temporal predictive information. The goal is to minimize the time required to serve a set of $R$ requests at each client, subject to the constraints regarding the feasibility of transaction sets and servicing conditions of client requests. Assuming that the maximum latency for a single request in the system is upper bounded by some $U$, we work over a finite time horizon $T = R \times U$, which is the maximum possible time that the system can take to successfully serve the requests of all the clients. Let $\Gamma$ be the time taken to successfully serve every client's requests. Please refer to Table I for the summary of notation used. We now have,

**Minimize** $\Gamma$

**Subject to**

1. $\tau_{e,t,\alpha,\beta} \leq y_{e_s,t,\alpha}$ --→ $\forall t \; \forall e \; \forall \alpha \; \forall \beta$
   Source node of the transaction contains $\alpha$ at time $t$.
   **Explanation**: If $\tau_{e,t,\alpha,\beta}$ is 1, then $y_{e_s,t,\alpha}$ is forced to 1. This enforces the constraint that if the transaction happens, then the source node contains the content item being transferred.

2. $y_{e_f,t+1,\alpha} \geq \tau_{e,t,\alpha,\beta}$ --→ $\forall t = 0$ to $T-1 \; \forall e \; \forall \alpha \; \forall \beta$
   Destination node of the transaction contains $\alpha$ at time $t+1$.
   **Explanation**: If $\tau_{e,t,\alpha,\beta}$ is 1, then $y_{e_f,t+1,\alpha}$ is forced to 1. This enforces the constraint that if the transaction happens, then the destination node contains the content item that the transaction transfers.

3. $y_{e_f,t+1,\beta} \leq (1 - \tau_{e,t,\alpha,\beta})$ --→ $\forall t = 0$ to $T-1 \; \forall e \; \forall \alpha \; \forall \beta$
   Destination node of the transaction doesn't contain $\beta$ at time $t$
   **Explanation**: If $\tau_{e,t,\alpha,\beta}$ is 1, then $y_{e_f,t+1,\beta}$ is forced to 0. This enforces the constraint that if the transaction happens, then the destination node does not contain the content item that the transaction replaced.

4. $y_{e_f,t+1,z} \leq y_{v,t,z} + \sum_{\alpha} \sum_{e} \tau_{e,t,z,\alpha} \; \forall t = 0$ to $T-1 \; \forall e_f \in V$
   If a cache doesn't have an element at time $t$, the element cannot exist in the cache at time $t+1$ unless there is a transaction that brings it in.
   **Explanation**: If a node doesn't have $z$ at time $t$, and if there are no transactions that bring $z$ in at time $t$, then the RHS is 0. This forces the LHS, $y_{e_f,t+1,z}$, to 0.

5. $y_{e_f,t+1,z} \geq y_{v,t,z} - \sum_{\alpha} \sum_{e} \tau_{e,t,\alpha,z} \; \forall t = 0$ to $T-1 \; \forall e_f \in V$
   If a cache has an element at time $t$, the element cannot leave the cache at time $t+1$ unless

there is a transaction that replaces it.
**Explanation**: If a node has $z$ at time $t$, and if there are no transactions that replace $z$ at time $t$, then the RHS is 1. This forces the LHS, $y_{e_f,t+1,z}$, to 1.

6. $\sum_\alpha y_{e_s t\alpha} \leq L_{e_s}$ --→ $\forall t\ \forall e_s \in V \setminus S$
Constraint on content store capacity.
**Explanation**: The count of all content items in the cache should be lesser than or equal to the capacity of the cache.

7. $\sum_\alpha \sum_\beta \tau_{e,t,\alpha,\beta} \leq 1$ --→ $\forall t\ \forall e$
At most one transaction on a given edge $e$ (unit capacity) and time $t$.
**Explanation**: The count of transactions that happen at time $t$ over the same edge $e$ should be less than or equal to 1.

8. $\mu_{c,i,t} \leq y_{c,t,r_{c,i}}$ --→ $\forall c\ \forall i\ \forall t$
A request $r_{c,i}$ can be served at time $t$ only if the item is in $c$'s content store.
**Explanation**: If $\mu_{c,i,t}$ is 1, then $y_{c,t,r_{c,i}}$ is also forced to 1. This enforces the constraint that if a request is served at time $t$, then the content item served is in the content store of the client that generated the request.

9. $\sum_{i=1}^{R} \mu_{c,i,t} \leq w$ --→ $\forall c\ \forall t$
At most $w$ content items can be consumed by a client at a given time $t$, since there are at most $w$ requests pending.
**Explanation**: For every time $t$, the count of the content items served cannot exceed the window of the client.

10. $\sum_{t=1}^{T} \mu_{c,i,t} = 1$ --→ $\forall c\ \forall i$
A request $r_{c,i}$ is served exactly once.
**Explanation**: We sum over all time $t$ and ensure that every request is served exactly once by forcing this summation to be exactly 1.

11. $\sum_{t=1}^{k-1} \mu_{c,i-1,t} \geq \mu_{c,i,k}$ --→ $\forall c\ \forall i = 2$ to $R$
If request $r_{c,i}$ is served, then $\forall j < i$, $r_{c,j}$ must have been served.
**Explanation**: This is the sequencing constraint. This ensures the ordering in the requests that are served. If some request is served at time $k$ i.e the RHS is 1, then for the previous request, we sum over all time from 0 to $k-1$ and ensure that this previous request was served i.e the LHS is forced to 1. By transitivity, all the requests previous to the immediately previous request are also served before the request at hand.

12. $t \times \mu_{c,i,t} \leq \Gamma$ --→ $\forall c\ \forall i\ \forall t = 1$ to $T$
Every request is served before $\Gamma$ time.
**Explanation**: If a request was served at time $t$, then $\mu_{c,i,t}$ is 1, and the constraint reduces

to $t \leq \Gamma$, which enforces the constraint on the time $t$ of the content service to be less than or equal to $\Gamma$, the value that the optimization tries to minimize.

The solution to the above optimization forms a time sequence of transaction sets. Executing these sets at the corresponding time steps would result in the client requests being served within the optimal objective time $\Gamma$ that is being solved for.

*Remark*: If the value for $T$ is a known upper bound of the optimal value of the objective in the above formulation, then the problem is always feasible. Intuitively, this is enforced by letting the time horizon $T = R \times U$, since in the worst case, each request of each client is served in at most $U$ time units. Another approach is to first use the heuristic described in Section 4.3 to find an upper bound for the optimal objective value and set it equal to $T$.

To test the computational feasibility of the proposed optimization, we feed the optimization problem to the Gurobi Integer Program Solver [17] through the AMPL libraries. The solver took about one day to solve the optimization problem on a 32-core machine with 64 GB of RAM (part of the Dell Computing Cluster, DON Lab, IIT Madras), for a topology of 10 nodes and 15 content items, for a request length $R$ of 10 requests. The solver takes an indefinite amount of time for the same setup with 20 content items (The time horizon $T$ that is set is obtained from the heuristic described in Section 4.3). This clearly indicates that although the above optimization problem has constraints and variables polynomial in the input parameters, it is computationally too expensive to directly solve the problem using an off-the-shelf solver for a reasonably sized topology.

Also note that if requests keep coming in at every time step, the optimization needs to be re-solved after every time step to maintain optimality.

## 4.3 Towards Heuristic Algorithms

Since optimally solving the *oracle problem* formulation is practically infeasible, we propose heuristic algorithms for transaction scheduling that nevertheless show drastic improvements in performance. We define a few more terms before formally describing our heuristic solver.

**Definition 2. (Progressive Transaction)** $\tau_{e,t,\alpha,\beta}$ is said to be a *progressive* (resp. *k-progressive*) transaction if $\exists c \in C_{\alpha,t}$(resp. $C_{\alpha,t,k}$) such that $d_{c,e_f} < d_c(\alpha)$ , i.e., upon this transaction, $n_c(\alpha) = \{e_f\}$ .

Essentially, for some client that is interested in obtaining $\alpha$, $e_f$ must become the only node containing the nearest replica of $\alpha$. A *regressive* transaction has $d_{c,e_f} \geq d_c(\alpha)\ \forall c \in C_{\alpha,t}$.

**Definition 3. (Dominating Transaction)** We say that $\tau_{e,t,\alpha,\beta}$ dominates $\tau_{e',t,\alpha,\beta'}$ if (i) $\forall c \in C_{\alpha,t}$ we have $d_{c,e_f} \leq d_{c,e'_f}$ and (ii) $\exists c \in C_{\alpha,t}$ such that $d_{c,e_f} < d_{c,e'_f}$ . For weak domination, (ii) need not hold, and for *strict* domination $d_{c,e_f} < d_{c,e'_f}$ $\forall c \in C_{\alpha,t}$. A *dominating* transaction is a transaction that is not dominated by any other transaction.

**Definition 4. (Conflicting Transactions)** We say that $\tau_{e,t,\alpha,\beta}$ conflicts with $\tau_{e',t,\alpha',\beta'}$ if *any* of the following holds :-

- $e$ and $e'$ are along the same edge.
- $e_f = e'_f$ and $\beta = \beta'$ :- Replacing the same item.
- $e_f = e'_s$ and $\beta = \alpha'$ :- Replacing an item that the other transaction is transferring.

Essentially, conflicting transactions cannot be performed together, since they violate the constraints imposed on the system.

**Definition 5. (Gravity)** The gravity $g(\tau)$is a value associated with every transaction[2]. The notion of gravity is used while making cache replacement decisions i.e., $\tau = 1 \Rightarrow g(\tau) > 0$, and for resolving scheduling conflicts which we shall see later. Intuitively, gravity represents the attractive force generated by the system to replace content $\beta$ with content $\alpha$ at $e_f$. It can take several functional forms.

**Definition 6. (Feasibility)** A set of transactions spanning a finite time interval $t = [0, T]$ is said to be *feasible* if and only if it satisfies constraints (1) to (5) in the Oracle Problem Formulation. It can be seen that two *conflicting* transactions cannot be part of the same feasible set.

## 4.4 Proposed Heuristic Framework

In this section, we present heuristic algorithms to efficiently find solutions to the centralized optimization problem. Instead of solving for sets of transactions across time units simultaneously, we take a *greedy* approach and make transaction scheduling decisions for the immediately next time step, given a current snapshot of the system. Results show that this greedy approach itself provides significant performance gain.

The basic idea behind the heuristic is, for every time step, to greedily choose certain transactions. The overall template of the heuristic is as follows. Given a snapshot of the system, we (i) prune all irrelevant (*regressive*) transactions (ii) order the relevant (*progressive*) transactions in decreasing order of importance (*gravity*) and (iii) iteratively choose *dominating* transactions

[2]We drop the indices on $\tau$ for convenience

with highest gravity that do not *conflict* with previously chosen transactions, until no other transaction can be chosen. This idea is reflected in the generic heuristic template shown in Algorithm 1. We now look at specific instances of the template.

**Algorithm 1** Heuristic Template

```
1:  procedure GREEDYSOLVER
2:      while there is a pending request do
3:          pTrans ← Get all k-progressive transactions
4:          sortedPTrans ← Sort pTrans in decreasing
                           order of gravity
5:          chosenTrans ← Empty List
6:          for t in sortedPTrans do
7:              if some t' in chosenTrans conflicts with t
                 or dominates t then
8:                  continue
9:              chosenTrans.add (t)
10:         Execute-all (chosenTrans)
11:         Serve & update request-queues of clients
```

## Discounted K-Lookahead

We define the gravity of a transaction as follows

$$g(\tau) = h_{e_f,t}(\alpha) - h_{e_f,t}(\beta)$$

$$h_{n,t}(\alpha) = \sum_{c \in C_{\alpha,t,k}} f_{n,t,c}(\alpha)$$

$$f_{n,t,c}(\alpha) = \begin{cases} \gamma^{i_\alpha(c)} \cdot \dfrac{1}{(i_\alpha(c)+1)(d_{c,n}+1)} & \textit{if } d_c(\alpha) > d_{c,n} \\ 0 & \textit{otherwise} \end{cases}$$

where , $0 \leq \gamma \leq 1$

The *pull* $h_{e_f,t}(\alpha)$ represents the priority level of content $\alpha$ at node $n$, and the *push* $h_{e_f,t}(\beta)$ represents same for content $\beta$. Each of them is a sum over the individual pulls $f_{n,t,c}(\alpha)$'s (or pushes) from every client interested in the respective content items. The *gravity* value of a transaction represents the effective force with which content $\alpha$ is brought to replace $\beta$ at node

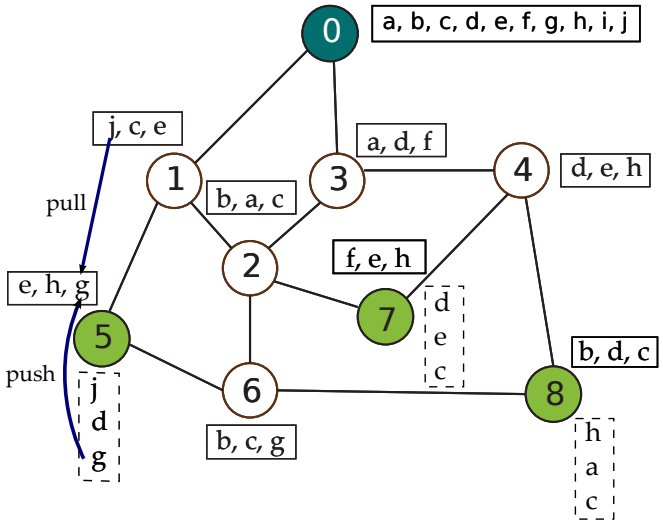



**Figure 4.4: Gravity** :- The push and pull exhibited by nodes in the system.

$n$. The heuristic solver only considers transactions with positive attraction i.e., *pull* $>$ *push* and hence, *gravity* $> 0$.

Figure 4.4 gives an intuitive picture of the push-pull mechanism that is proposed. Node $5$ is interested in obtaining content item $j$. Consider the transaction that brings in content item $j$ from node $1$ and replaces content item $g$ in the content store of node $5$. This transaction gets a pull from node $5$ since $j$ is immediately needed, but also gets a push from the same node since the replaced item $g$ is needed in the near future. The overall gravity of this transaction is the difference between the pull and the push. A transaction that brings in $j$ from node $1$ and replaces $e$ at node $5$ has a higher gravity since there is no push ($e$ is not needed in the near future).

We choose an appropriate functional form for gravity that captures the properties we desire. The design of the *pull* function reflects the following properties.

1. If both $\alpha$ and $\beta$ appear in the same indices at request queues of clients interested in them, then the client nearer to the node in concern exhibits a larger pull.

2. If a client interested in $\alpha$, and another in $\beta$, are equidistant from node $n$, then a larger pull value is exhibited on the content that appears at the earliest index in the request queues.

3. A node that already has a replica of $\alpha$ nearer to it than the node's distance to $n$ shows *no* pull to $\alpha$ at $n$.

4. $\gamma$ represents a decay of priority for content items that appear in higher indices. A $\gamma$ close to $0$ gives very low priority to content appearing in higher indices.

5. $k$ represents the *Lookahead* value of the heuristic, i.e., number of future requests in the client request queues that are considered.

Note that the given function is just one possible definition for gravity, and there are several alternate choices that one could choose to implement. Simulation results show that our heuristic shows drastic latency gains using this gravity function, and saturates at a lookahead value $H$, which is the maximum hop distance of any client to the source. We term this the *horizon lookahead* (LA-H).

## An Example

The following example demonstrates the functioning of the heuristic. Consider the snapshot of the system as shown in Figure 4.5. We now observe the decisions made by the heuristic at this snapshot. We assume that all content stores have a capacity of three content items, and we look ahead two requests into the future. Assume $\gamma = 0.25$.

- *Pruning* :- Node $5$ needs $j$. Since node $1$ is the only nearest node that contains $j$, the transaction from node $1$ to node $5$ *dominates* the transaction transferring $j$ from $0$ to $1$. Also, the latter is *regressive*. Hence it (the latter) is discarded from consideration. Similarly, the transaction from node $5$ to node $6$ transferring $h$ is discarded since it is not a progressive transaction (node $4$ already has $h$ ready for node $8$). Several transactions are eliminated in this fashion. This represents the pruning stage of the algorithm.

- *Conflict Resolution* :- Consider the transactions from node $1$ to node $5$ transferring $j$. We will now calculate the gravities of these transactions for the various possible items that $j$ can replace ($e, h,$ and $g$). Intuitively, we observe that since node $5$ needs $g$ in the near future, it should be less likely that $g$ is replaced. This intuition is captured as follows. The pull for $j$ at $5$ is 1. The push for $g$ is $(0.25)^2.1$. The push at $e$ because of node 7 (Only node 7 has $e$ in its request queue) is 0 because node 4 already has the nearest replica of $e$ for node 7. Similarly, the push from node 8 (Only node 8 has $h$ in its request queue) for $h$'s replacement is $0$ because node 4 already has a nearer replica. Hence, the gravity of the transaction that replaces $g$ (i.e., $0.9375$) is lesser than the gravity of the transaction replacing $e$ or $h$ (i.e., $1$). The tie between $e$ and $h$ is resolved randomly by the heuristic.

The above was just an example case to illustrate the behaviour of the heuristic. We now proceed to the evaluation of our heuristic solver.

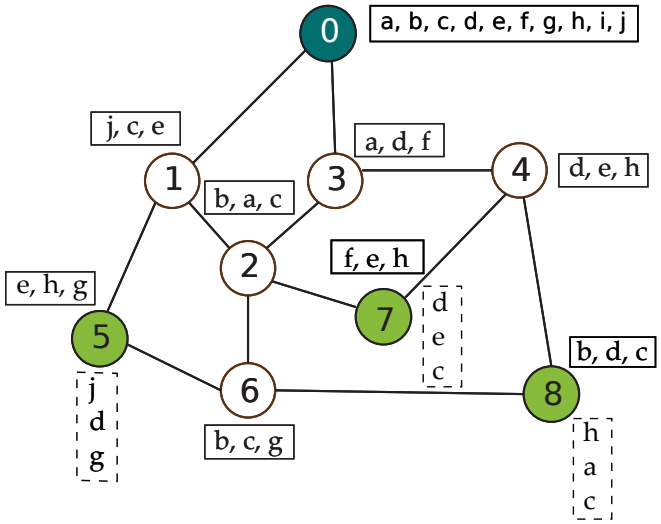



**Figure 4.5: System Snapshot**. Content stores are shown in solid boxes, and the request queues at the clients are shown in dotted boxes.

## 4.5 Evaluation

In this section, we evaluate the performance of our heuristic-based oracle on a variety of parameters. Overall, we observe a significant improvement ($\sim 50\%$) in access latencies for a wide range of topologies. In the multicast scenario, we also observe that the solution achieves a significant percentage ($\sim 80\%$) of the minimum of the maximum flow rates from the clients to the servers. We first describe the simulation environment, and then present several plots, studying the sensitivity of our algorithms to various input and algorithm parameters.

### 4.5.1 Simulation Environment

We use a custom simulator and evaluate our heuristic on sub networks of the University of Wisconsin-Madison topology [4], each having 20-25 nodes, and 30-35 edges. We also assume that the servers are at most 6-7 hops away from the client, which is a reasonable assumption at the edge of a content distribution network. We assume that all links have equal bandwidth (*unit* bandwidth, for convenience). Each router is allocated a content store budget of some $k\%$ of the total number of content items in the system, for varying values of $k$. There are several clients, and one or several *source servers* each one housing a subset of the universe of content items. We assume that the clients have buffers of size equal to or greater than their respective maximum flow values to the servers.

#### The Interest Oracle

The Interest Oracle represents complete-information access to the future interests that will be generated by every client (i.e., $|C| \times R$ Interest matrix $[r_{c,i}]$). We consider three different request patterns at the clients.

- *Zipf* :- First, we look at the scenario at one end of the spectrum where each client follows an *Independent Reference Model*, with a *Zipf* distribution over content items (Zipf parameter of 0.8).
- *Stream* :- Second, we look at the scenario at the other end of the spectrum where every client requests the exact same sequence of (distinct) content items, representing the multicast streaming scenario.
- *Stream-zipf (Hybrid)* :- Third, we consider a hybrid of both request patterns. We assume that there are $N$ movies in the system, each consisting of a sequence of $F$ distinct frames[3], where $F$ is uniformly distributed from 1 to $M$. Each client first chooses a movie from a Zipf distribution over the $N$ movies, and then issues interests sequentially for the frames corresponding to that movie. Once all the frames for the chosen movie are served, the client samples another movie from the zipf distribution and repeats the process. We only work with an $2N/M$ ratio of 1, the pattern that lies in the center of the request spectrum. $N \gg M$ represents *Zipf* at the limit, and $N \ll M$ represents *Stream* at the limit.

The motivation behind introducing the hybrid request pattern is to capture the intermediate region of the spectrum that lies between the assumptions of an independent reference model and that of a multicast streaming scenario. We assume oracle access to the interest matrices that represent each of these three patterns and analyze the performance of our heuristic. The metrics in concern are the *access latencies, network utilization* and the *rates* of content delivery. Except for the case where we analyze performance on the Stream pattern, we initialize the system with content distributed randomly across the routers in proportion to the content popularities before beginning our measurements.

### 4.5.2 Key Results

#### Latency Reduction

Figure 4.6 plots the average latency gains for our *Discounted Horizon-Lookahead* heuristic (latency in terms of hop count) as compared to latency to the sources for five different topologies.

[3]We assume that each frame corresponds to one content item

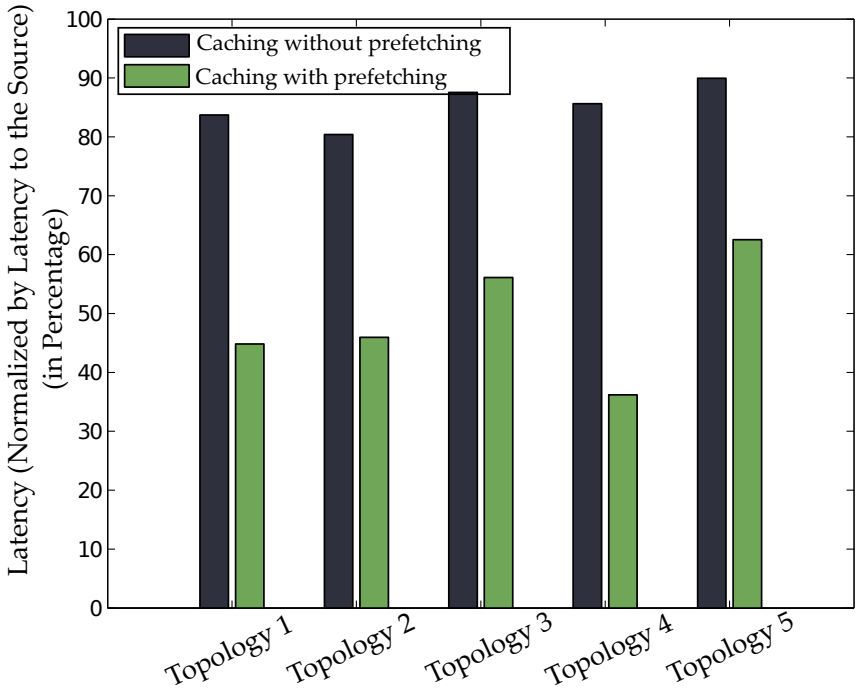



**Figure 4.6: Latency Reduction** :- Average query latency over clients across different topologies, normalized by the latency to the source, for *Stream-zipf*.

For these results, the client requests are based on *Stream-zipf*. The averages are taken over varying cache sizes ranging between 2–6% of the total number of content items in the request streams. For each topology, the left bar corresponds to the case with *no* prefetching while the right bar corresponds to the oracle with the *Discounted Horizon-Lookahead* heuristic. For each topology, note that, the gains due to prefetching in the oracle are significant. The reduction in latency for the left bar (i.e., no prefetching) is only due to exploitation of content popularity among clients (i.e., spatial information). However, the reduction in latency for the right bar (i.e., oracle) comes due to both spatial and temporal information (i.e., looking ahead in time). For instance, for Topology 1, the average latency is about 80% of the source latency when there is no prefetching, whereas it is about 40% of the source latency for the oracle. This is 50% improvement due to prefetching.

### Multicast Rate

Figure 4.7 plots the delivery rates for different levels of lookahead when the client access pattern corresponds to the *Stream* scenario. Recall that, *Stream* scenario corresponds to a multicast where all clients request the exact same set of distinct content items over time. To evaluate performance in this scenario, we consider the *rate* of content delivery as opposed to latency, and observe that the heuristic achieves $81.5\%$ of the minimum of the maximum flow over all the

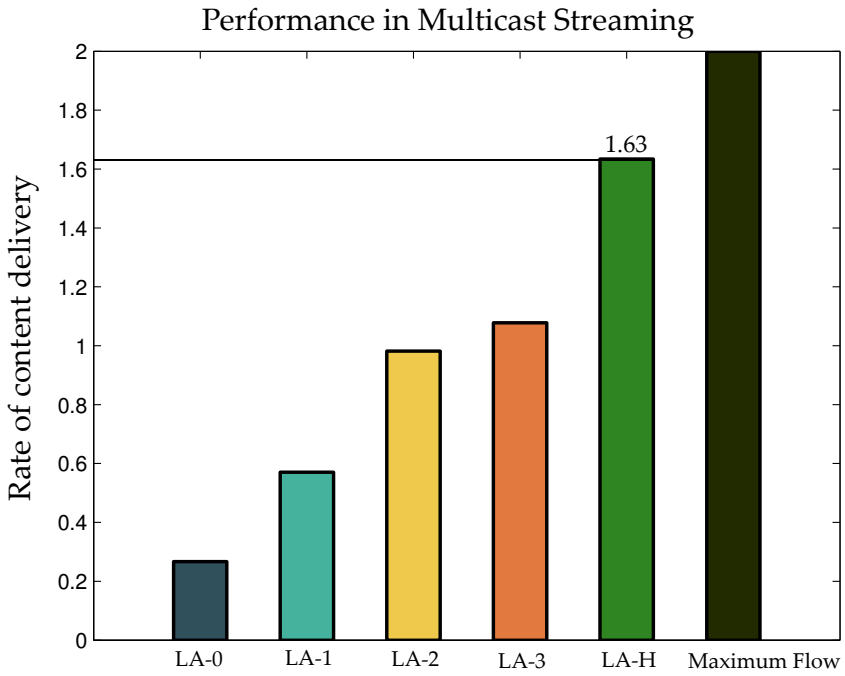


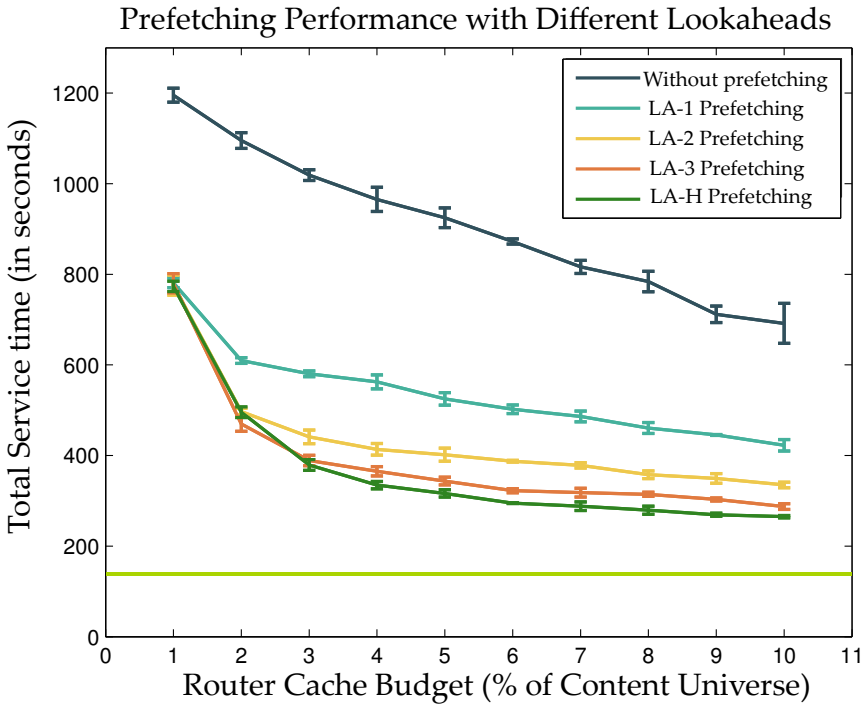

**Figure 4.7: Performance in Multicast Streaming** (*Stream* request pattern) :- Rate of content delivery for different lookaheads.

clients. Note that it may not be possible to achieve maximum flow without network coding [5, 32]. Since the oracle does not use network coding, we only observe that the heuristic is capable of achieving near-maximum flow rates when challenged with a multicast request pattern.

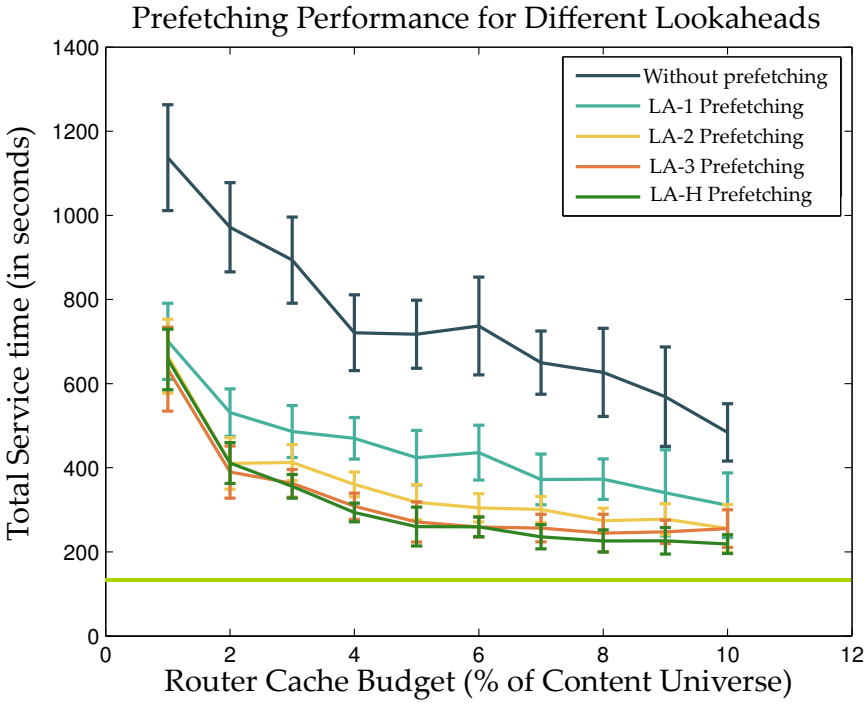
The figure also clearly shows the benefits of lookahead. The leftmost bar in the figure corresponds to the case of zero lookahead (i.e., no prefetching). The next three bars show the results for a lookahead of one, two, and three time steps respectively. The LA-H bar corresponds to the case where the lookahead is equal to the number of hops to the source. There are no advantages to looking ahead beyond the number of hops to the source. Hence, LA-H corresponds to the best performance from the oracle with respect to the levels of lookahead.

### 4.5.3 Sensitivity & Analysis

In the next set of results, we evaluate the sensitivity of oracle's performance to different input and algorithm parameters.




**(a) Zipf Request Pattern**




**(b) Stream-Zipf Request Pattern**

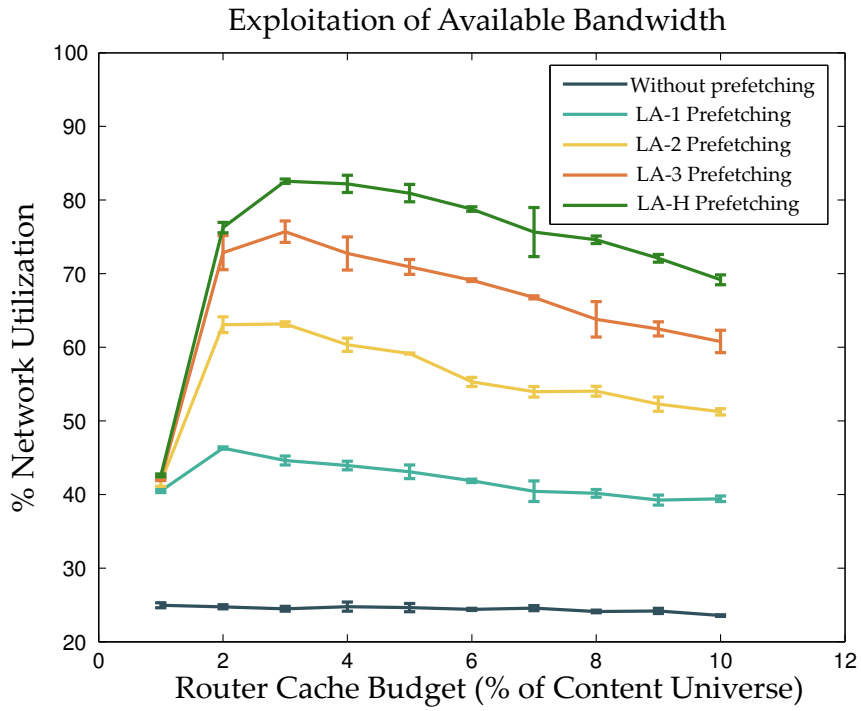
**Figure 4.8: Prefetching Performance for Different Lookaheads** on a single topology, with error bars. H=6

### Cache Budget and Lookahead

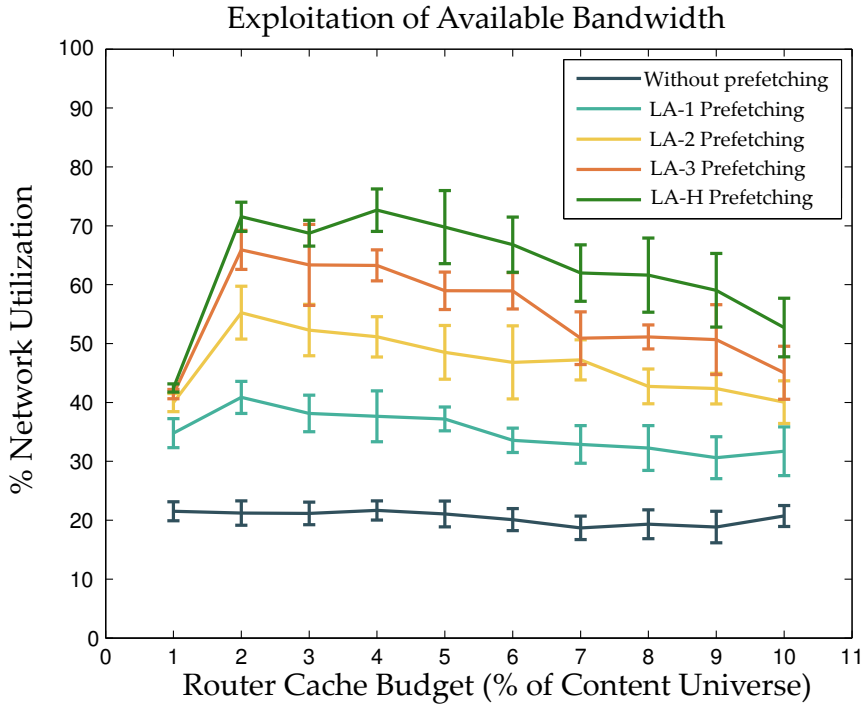
Figure 4.8 shows the variation in average total service time for each client to complete 2500 accesses as a function of cache size, for the *Zipf* and *Stream-zipf* patterns. Smaller total service times are better than larger total service times. There are five curves in the figure. The topmost curve corresponds to the case with only caching and no prefetching (i.e., lookahead is zero). The next three curves from top to bottom correspond to lookaheads of one, two, and three, respectively. The lowest curve corresponds to lookahead equal to the number of hops to the server (H = 6). The averaging is over many different access request sequences for length 2500 for each client. As expected, the average total service time decreases when the cache size increases. We also observe that a lookahead of just 2 or 3 time steps provides a significant reduction in total service time, and beyond that the each additional lookahead provides smaller and smaller reduction in total service times. This indicates that it is *enough* to look ahead a small time into the future to make significant performance improvements. The reference line represents the ideal latency if the cache at the edges was infinite, which is eventually reached by all the curves at $100\%$ cache budget. For a more realistic scenario, where the cache budget is $\sim 5\%$ of the content space, we see a clear improvement in performance for higher lookaheads in both *zipf* and *stream-zipf* request patterns. Error bars representing the standard deviation for 20 runs are also shown for each data point. Variance of data points for *Stream-zipf* is higher because of the varying levels of spatial and temporal properties in the interest matrices generated for different runs.

### Network Utilization

One of the key features of prefetching is that it exploits the *available* spare bandwidth to carry out useful transactions. Figure 4.9 plots the average percentage of active links for different lookahead values. Here again, there are five curves in the figure corresponding to zero, one, two, three, and horizon lookaheads (starting from bottom to top). We see that higher lookaheads utilize the available bandwidth in the system to bring content closer to the clients and improve latency. Note that the baseline exploits $\sim 25\%$ of the available bandwidth of the system at any given time. Higher lookaheads exploit more available bandwidth. Note that, even for LA-H prefetching, there is still a significant gap of around $15 - 20\%$ to full utilization. This does not imply that our heuristic is sub-optimal. We observe from the link utilization data that the unused links were irrelevant in the context of prefetching (*Prefetch-dead* links :- Links that do not help in bringing content closer to any client). In fact, it is possible that the most optimal scheduling scheme would utilize lesser bandwidth and still provide better latencies! We observe that with increase in cache size, the network utilization initially increases. This is because very small caches do not have space for prefetched data i.e prefetching cannot take greater advantage of




**(a) Zipf Request Pattern**




**(b) Stream-Zipf Request Pattern**

**Figure 4.9: Network Utilization** (Exploitation of Available Bandwidth) **for Different Lookaheads** :- Average percentage of active links in the system.

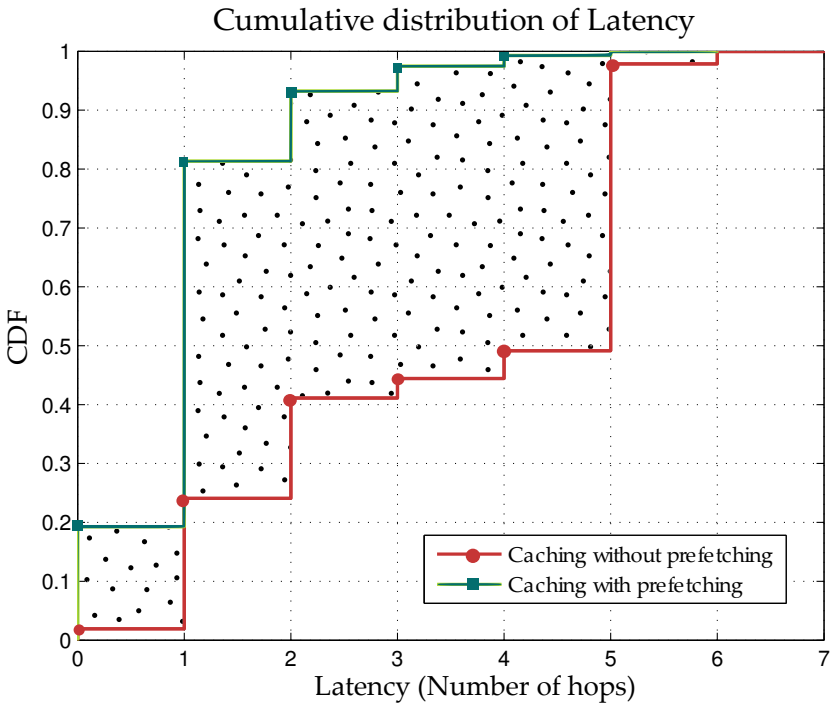



**Figure 4.10: Prefetch Quality**: CDF of Latency for router cache budget $k = 5\%$ .

the network due of storage constraints at the routers. Note that even in this scenario, although the effect of higher lookahead is little, prefetching can still take significant advantage of the network (at least 40% utilization, as opposed to the 25% baseline) to improve latencies. We also observe that the utilization gradually decreases, for higher cache sizes. This decrease represents the cache-bandwidth trade-off i.e., the system needs lower bandwidth resources for larger cache sizes to deliver at the same latency.

### Prefetch Quality

Figure 4.10 plots the cumulative distribution function of latency for the Stream-zipf request pattern. Higher fraction for the same latency indicates that smaller latencies are much more prevalent. The left curve in this figure corresponds to caching with prefetching with horizon lookahead while the right curve corresponds to the case of no prefetching (i.e., lookahead of zero). We see that with prefetching enabled, a large number of requests ($> 80\%$) are served within one hop. This graph is reflective of the *prefetch timeliness* metric, which indicates that when the content was needed, it was at most two hops away with a $90\%$ chance. We would also like to point out that the other commonly used evaluation metrics of cache performance, namely, Prefetch Coverage and Prefetch Accuracy are irrelevant here since we have assumed oracle access to the client requests.

## 4.6 Summary

Inspired by the prefetching model in processors and extending them to an arbitrary network of caches, we have presented a formulation to the prefetching oracle as a problem of transaction scheduling in the form of an Integer Linear Program. We have also proposed a feasible heuristic solver for the same, and shown that the heuristic solution can decrease latencies by more than $50\%$ in typical university topologies. We have demonstrated that small sized router caches, that lie in the heart of an Information-centric Network, can be used to leverage both spatial and temporal information about client requests and drastically improve access latency. In addition, we have also shown that small lookaheads into the future requests of the clients contribute to a significant portion of the gain. Overall, we have established a framework to study prefetching in the ICN context, and have paved the way for a thorough study of incomplete information & distributed prefetching algorithms by setting up a benchmark to compare against.

# CHAPTER 5

# Conclusions and Future Work

We have presented and evaluated two ways to improve content recovery in Information-Centric Networks. In the first method, we integrated network coding and caching into a unified framework. In the second method, we extended prefetching models in processors to an arbitrary network of caches and presented a prefetching oracle for the information-centric framework. This has paved the way for a thorough analysis of distributed prefetching policies.

Note that the oracle had two distinct advantages: (i) correct knowledge of future requests from all clients, and (ii) centralized decision-making with prefetching over the entire network. Future work must focus on relaxing both these requirements. That is, each router must execute prefetching in a distributed (but possibly coordinated) fashion. Furthermore, each router may have to predict future requests based on the request patterns it observes, and is hence prone to errors in prediction. An important contribution would be to demonstrate the practicality of the prefetching framework through implementation on the GENI testbed. This would be possible in a Software-defined networking framework where the controller is capable of issuing caching decisions to the nodes on the network. One direction to pursue would be to define custom OpenFlow protocol packets and actions to support caching decisions.

In current literature, caching strategies are viewed orthogonal to forwarding strategies. A direction of approach is to try and integrate forwarding strategies into the caching-prefetching framework. Another idea is to integrate network coding in the prefetching framework i.e to introduce the notion of Network Coded Prefetching. The framework should generalize network coding from the multicast scenario to an arbitrary on-demand user patterns. With information about the future requests of the clients, the oracle must be able to devise network coded transactions that can further improve the benefits of pervasive caches beyond ordinary prefetching. An optimization problem to model this scenario might be too huge to be solved even for very small topologies ($\approx$ 5 nodes). This calls for intelligent heuristics that can solve the optimization and still achieve a significant chunk of the gain that network coding achieves.

Another insight is to see if the prefetching optimization problem reduces to a simpler form for special cases of request patterns. For example, it would be worth it if we are able to reduce the optimization problem for a modified version of multicast, which we would like to call *Staggered* Streaming. In this request pattern, all users on the network request for the same video stream, but these requests are offset in time. User 1 might be on the 400th frame while User 2 is still on the 100th frame. If the optimization problem reduces to a simple enough form, it might become possible to exactly solve the problem and obtain the best possible content-delivery rate (maximum flow).

Currently, Software Defined Networking protocols assume IP based addressing of nodes and IP based forwarding while setting up flow rules in the switches on the network. A major challenge is to effectively support Software Defined Networks over an information-centric network architecture and vice-versa. [29] proposes techniques to support ICN using concepts from SDN. Another direction would be to embed the network intelligence that defines ICN into a centralized controller, so that forwarding and caching strategies are centralized instead of distributed.

Overall, Information-Centric Networks present a major challenge in terms of both proving their benefits and in feasibility of their deployment on real networks [26]. Addressing any of issues with respect to caching, scalability, mobility, security, and naming/name-management would be a significant step towards supporting the ICN revolution.

# CHAPTER 6

# Publications from this Work

[1] **(Conference Paper)** Abhiram Ravi, Parmesh Ramanathan, Krishna M. Sivalingam, Integrated Network Coding and Caching in Information-Centric Networks, pp 1-6, In Proceedings of IEEE Conference on Advanced Networks and Telecommunication Systems (IEEE ANTS), New Delhi, 2014.

[2] **(Journal Paper)** Abhiram Ravi, Parmesh Ramanathan, Krishna M. Sivalingam, Integrated Network Coding and Caching in Information-Centric Networks, Submitted to Springer Photonic Network Communications, PNET, 2015.

[3] **(Technical Report)** Abhiram Ravi, Parmesh Ramanathan, Krishna M. Sivalingam, Prefetching Oracles for Pervasive Caching in Information-Centric Networks,

# REFERENCES


[1] (2013). Cisco's visual networking index forecast. URL http://newsroom.cisco.com/release/1197391/.

[2] (2014). Average web page breaks 1600k. URL http://www.websiteoptimization.com/speed/tweak/average-web-page/.

[3] (2014). Geni portal. URL http://portal.geni.net/.

[4] (2015). University of wisconsin at madison network statistics. URL http://stats.net.wisc.edu/newcore.html.

[5] **Ahlswede, R.**, **C. Ning**, **S.-Y. R. Li**, and **R. W. Yeung** (2000). Network information flow. *IEEE Transactions on Information Theory*.

[6] **Albers, S.**, **N. Garg**, and **S. Leonardi**, Minimizing stall time in single and parallel disk systems. *In Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98. ACM, 1998.

[7] **Anand, A.**, **A. Gupta**, **A. Akella**, **S. Seshan**, and **S. Shenker**, Packet caches on routers: The implications of universal redundant traffic elimination. SIGCOMM. ACM, 2008.

[8] **Applegate, D.**, **A. Archer**, **V. Gopalakrishnan**, **S. Lee**, and **K. K. Ramakrishnan**, Optimal content placement for a large-scale vod system. *In Proceedings of the 6th International Conference*, Co-NEXT '10. ACM, 2010.

[9] **Baran, P.** (1964). On distributed communications networks. *Communications Systems, IEEE Transactions on*.

[10] **Berman, M.**, **J. S. Chase**, **L. Landweber**, **A. Nakao**, **M. Ott**, **D. Raychaudhuri**, **R. Ricci**, and **I. Seskar** (2014). Geni: A federated testbed for innovative network experiments. *Computer Networks*. Special issue on Future Internet Testbeds - Part I.

[11] **Breslau, L.**, **P. Cao**, **L. Fan**, **G. Phillips**, and **S. Shenker**, Web caching and zipf-like distributions: evidence and implications. *In INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*. 1999.

[12] **Chanda, A.** and **C. Westphal**, A content management layer for software-defined information centric networks. *In Proceedings of the 3rd ACM SIGCOMM Workshop on Information-centric Networking*, ICN '13. ACM, 2013.

[13] **Crovella, M.** and **P. Barford**, The network effects of prefetching. *In INFOCOM '98. Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*. 1998.

[14] **Dimakis, A.**, **P. Godfrey**, **Y. Wu**, **M. Wainwright**, and **K. Ramchandran** (2010). Network coding for distributed storage systems. *Information Theory, IEEE Transactions on*.

[15] **Ebrahimi, E.**, **O. Mutlu**, **C. J. Lee**, and **Y. N. Patt**, Coordinated control of multiple prefetchers in multi-core systems. *In Proceedings of the 42Nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 42. ACM, 2009.

[16] **Fayazbakhsh, S. K.**, **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. ACM, 2013.

[17] **Gurobi Optimization, I.** (2015). Gurobi optimizer reference manual. URL `http://www.gurobi.com`.

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

[19] **Han, D.**, **A. Anand**, **F. Dogar**, **B. Li**, **H. Lim**, **M. Machado**, **A. Mukundan**, **W. Wu**, **A. Akella**, **D. G. Andersen**, **J. W. Byers**, **S. Seshan**, and **P. Steenkiste**, XIA: Efficient supoprt for evolvable internetworking. *In Proceedings of NSDI*. 2012.

[20] **Jain, R.**, Internet 3.0: Ten problems with current internet architecture and solutions for the next generation. *In Proceedings of the 2006 IEEE Conference on Military Communications*, MILCOM'06. IEEE Press, 2006.

[21] **Jin, W.**, **R. Barve**, and **K. Trivedi**, A simple characterization of provably efficient prefetching algorithms. *In Dependable Systems and Networks, 2002. DSN 2002. Proceedings. International Conference on*. 2002.

[22] **Joseph, D.** and **D. Grunwald**, Prefetching using markov predictors. *In Proceedings of the 24th Annual International Symposium on Computer Architecture*, ISCA '97. ACM, 1997.

[23] **Katsaros, K.**, **C. Stais**, **G. Xylomenos**, and **G. Polyzos**, On the incremental deployment of overlay information centric networks. *In Future Network and Mobile Summit, 2010*. 2010.

[24] **Kim, K.**, **S. Choi**, **S. Kim**, and **B.-h. Roh**, A push-enabling scheme for live streaming system in content-centric networking. *In Proceedings of the 2013 Workshop on Student Workhop*, CoNEXT Student Workhop '13. ACM, 2013.

[25] **Padmanabhan, V. N.** and **J. C. Mogul** (1996). Using predictive prefetching to improve world wide web latency. *SIGCOMM Computer Communications Review*.

[26] **Perino, D.** and **M. Varvello**, A reality check for content centric networking. *In Proceedings of the ACM SIGCOMM Workshop on Information-centric Networking*, ICN '11. ACM, 2011.

[27] **Pourmir, A.** and **P. Ramanathan**, Distributed caching and coding in vod. *In Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2014.

[28] **Raychaudhuri, D.**, **K. Nagaraja**, and **A. Venkataramani** (2012). MobilityFirst: A Robust and Trustworthy Mobility-Centric Architecture for the Future Internet. *SIGMOBILE Mobile Computing and Communication Review (M2CR)*.

[29] **Salsano, S.**, **N. Blefari-Melazzi**, **A. Detti**, **G. Morabito**, and **L. Veltri** (2013). Information centric networking over sdn and openflow: Architectural aspects and experiments on the ofelia testbed. *Computer Networks*.

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

[31] **Seferoglu, H.** and **A. Markopoulou**, Opportunistic network coding for video streaming over wireless. *In Packet Video*. IEEE, 2007.

[32] **Sundaram, N.** and **P. Ramanathan**, A distributed bandwidth partitioning scheme for concurrent network-coded multicast sessions. *In Proceedings of the IEEE Conference on Global Telecommunications*. IEEE Press, 2009.

[33] **Sundaram, N.**, **P. Ramanathan**, and **S. Banerjee**, Multirate media stream using network coding. *In Proc. 43rd Annual Allerton Conference on Communication, Control, and Computing*. 2005.

[34] **Tan, B.** and **L. Massoulié** (2013). Optimal content placement for peer-to-peer video-on-demand systems. *IEEE/ACM Transactions on Networking*.

[35] **Traverso, S.**, **M. Ahmed**, **M. Garetto**, **P. Giaccone**, **E. Leonardi**, and **S. Niccolini** (2013). Temporal locality in today's content caching: Why it matters and how to model it. *SIGCOMM Computer Communications Review*.

[36] **Tyson, G.**, **S. Kaune**, **S. Miles**, **Y. El-khatib**, **A. Mauthe**, and **A. Taweel**, A trace-driven analysis of caching in content-centric networks. *In ICCCN*. 2012.

[37] **Vanderwiel, S. P.** and **D. J. Lilja** (2000). Data prefetch mechanisms. *ACM Computing Surveys*.

[38] **Veltri, L.**, **G. Morabito**, **S. Salsano**, **N. Blefari-Melazzi**, and **A. Detti**, Supporting information-centric functionality in software defined networks. *In ICC*. IEEE, 2012.

[39] **Wang, Y.**, **K. Lee**, **B. Venkataraman**, **R. L. Shamanna**, and **I. Rhee**, A step toward practical deployment of the content-centric networking architecture. *In Proceedings of The ACM CoNEXT Student Workshop*, CoNEXT '11 Student. ACM, 2011.

[40] **Yi, C.**, **A. Afanasyev**, **L. Wang**, **B. Zhang**, and **L. Zhang** (2012). Adaptive forwarding in named data networking. *SIGCOMM Computer Communication Review*.

[41] **Zhang et. al, L.** (2010). Named Data Networkng (NDN) Project. Technical report, University of California, Los Angeles.