PERVASIVE ROUTER CACHING STRATEGIES FOR STAGGERED VIDEO MULTICASTS

by

Arezou Pourmir

A thesis submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy
(Electrical Engineering)

at the
UNIVERSITY OF WISCONSIN–MADISON
2016

Date of the final oral examination: 06/27/16

The dissertation is approved by the following members of the Final Oral Committee:
Parameswaran Ramanathan, Professor, Electrical and Computer Engineering
Kewal K. Saluja, Professor, Electrical and Computer Engineering
Xinyu Zhang, Assistant Professor, Electrical and Computer Engineering
Azadeh Davoodi, Associate Professor, Electrical and Computer Engineering
Dan Negrut, Associate Professor, Mechanical Engineering

To my parents

ACKNOWLEDGMENTS

First, I would like to thank my advisor, Prof. Parmesh Ramanathan, for all his time and effort that he put during this journey, and his valuable input. For keeping me motivated with stimulating conversations about new ideas and research. This thesis would not have been possible without his guidance. In addition, I would like to thank the rest of my thesis committee: Prof. Kewal Saluja, Prof. Azadeh Davoodi, Prof. Xinyu Zhang, Prof. Dan Negrut, and Prof. Suman Banerjee, for their time and insightful comments and also some hard questions, which inspired me to widen my research from various perspectives, and look at some problems differently.

I am grateful to Prof. Mark Allie, the TA supervisor, for the opportunity to learn through teaching other students. During several years of being a TA, I learned so many precious lessons that helped me in different aspects of my life and work.

I would like to thank Micah Eldridge, my supervisor at Cellcom, from whom I learned different valuable skills; not only the technical knowledge, but his professional guidance. I would like to also thank Bob Sobieck, the engineering manager at Cellcom, for being flexible with my school schedule. Working at Cellcom helped me realize what I really wanted from my PhD program and my career, and as a result, I continued my PhD more determined than before.

My sincere thanks goes to my parents and my husband. With their constant support and encouragement, my parents made it much easier for me to reach my goals, and I am very grateful for

their support. My husband constant encouragement helped to remind me that all of the difficulties I faced were only steps that I needed to climb in order to reach my final goal, and always emphasized that I could do it.

There were many other individuals not mentioned above that directly and indirectly guided me along the way. Thanks again to all of those who helped, supported and encouraged me to reach my goals.

ABSTRACT

Video is the dominant content in the Internet. Rapidly growing demand for streamed video content and increasing demand for high quality videos, from high definition videos to 4K videos to 3D videos, put a huge burden on the streaming servers, and has the potential to create severe bandwidth bottlenecks in the Internet. This thesis exploits two emerging technologies to alleviate the bandwidth bottleneck problem. *Pervasive caching* takes advantage of the fact that, at any given time, some videos are extremely popular as compared to the others. Therefore, the number of distinct videos being streamed in the network is often a small fraction of the total number of videos available for streaming. *Software Defined Networking (SDN)* takes advantage of the separation of data and control plan, to enable experimentation and incremental deployment of new strategies. In this dissertation, SDN-enabled pervasive caching strategies are proposed. Since pervasive router caches are usually small, the focus of this thesis is on proposing novel strategies for long-term caching (with low update rate) and short-term caching (with high update rate) that decrease the bandwidth required for video delivery without much storage.

For long-term caching problem, a novel idea of storing cross encoded videos in the caches, similar to the idea of Network Coding (NC), is proposed. Our solution, called *CCIP (Caching and Coding using Integer Program)*, uses the statistical demand of the requests, and the delivery cost of videos from different routers in the network to the client, and calculates the caching solution that in addition to considering storing original contents, examines the possibility of storing cross encoded videos. The output of the CCIP model also includes the minimum cost delivery solution of each video for different clients. For large networks, clustering methods are used to address scalability issues in the CCIP method. Comparison of CCIP with the traditional solutions where

encoding is not used validates the advantage of the proposed idea. For short-term caching problem, and specifically in order to enhance the delivery of staggered video multicasts in the network, *Short-term caching of Staggered multicasts (SCS)* algorithm is introduced. SCS caching solves a knapsack 0-1 problem at each router to determine which portions of which streams should be stored in the router cache, in order to support the demand of another stream in the network. The complexity of SCS solution is shown to be small, while its performance is close to the optimal solution, with the objective of minimizing the average bandwidth requirement in a video streaming network. The comparison of SCS caching with the traditional caching methods, LRU and LFU, proves the superiority of the proposed method. The impact of jointly applying the proposed long-term and short-term caching approaches (CCIP and SCS) in a network are evaluated, and the efficient capacity assignment to each caching portion is discussed. In addition, it is explained how by deploying a caching aware routing, one can further improve the performance. The proposed solutions are evaluated for different demand statistics, with different user arrival behaviors. In addition, the impact of router caching capacity is evaluated. The evaluation of the proposed caching strategies in an SDN-enabled network shows that by taking advantage of the global view of the SDN controller and the idea of pervasive caching, even small router caches can significantly decrease the bandwidth demands of the network compared to the traditional caching algorithms.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

Video is the dominant content in the Internet. It is expected to account for about 80 percent of all United States Internet traffic by 2019 [17]. Real-Time Entertainment (streaming video and audio) traffic now accounts for over 70% of North American downstream traffic in the peak evening hours on fixed access networks. Five years ago it accounted for less than 35%. For example, in 2015, Netflix (37.1%), YouTube (17.9%), and Amazon Video (3.1%), were the top three sources of video traffic on fixed access networks in North America (Figure 1.1 shows this trend [1],). Netflix, the worlds leading Internet television network with over 81 million members, has reported the stream of more than 86,000 hours of TV shows and movies per minute [2]. This increase is due to growing demand for streaming stored video content and decreasing demand for broadcast telecasts. YouTube, in the category of user generated content (UGC) has over 1.3 billion users, and reported the number of people watching YouTube per day is up 40% y/y since March 2014 [3]. Meanwhile, the current popularity of High Definition (HD) video compared to Standard Definition (SD) and the certain path towards ultra HD and 3D is adding to the complexity of this transition. In order to support this expected quality, throughput in order of 100s of Mbps per request is required. This will bring a huge burden on the streaming servers, as well as the bandwidth demand on the network. In order to meet the requirement of this transition, network designers and content providers are focused on developing new approaches to satisfy the demand at the minimum cost.

Two most popular solutions to these problems are CDN (Content Delivery Network) and P2P (Peer-to-Peer). CDN is a distributed group of data centers, to enhance content delivery to the users.

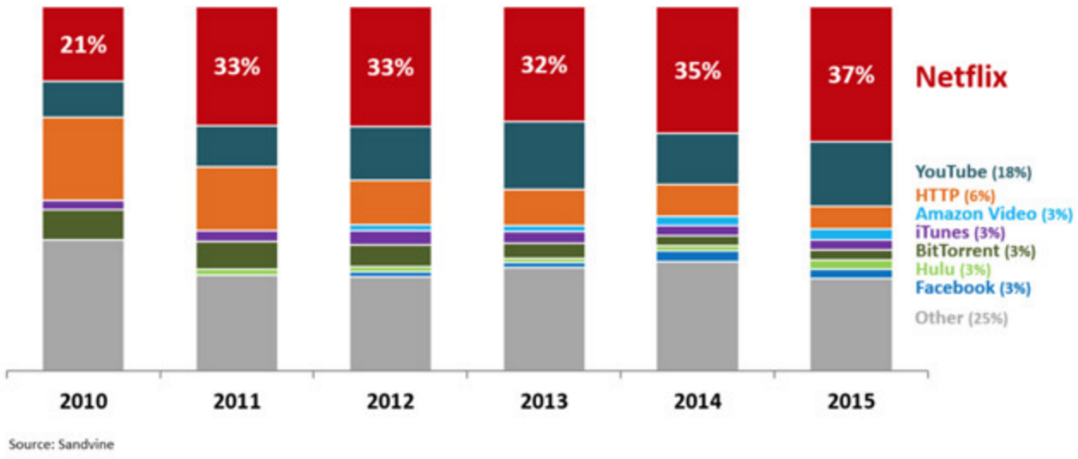


Figure 1.1: Share of peak download Internet traffic in North America

By copying content among multiple servers that are distributed geographically, and delivering the content from the servers closer to the clients, CDN provides high availability and high performance. CDNs play a very important role in todays networks, and the delivery of a large amount of content to a large number of users in the current Internet without CDN concept is not possible. The advantages of CDNs can be summarized as reducing original servers load, reducing the latency of content delivery, and increasing the throughput of the network. According to Ciscos VNI report [17], globally 57 percent of all Internet video traffic crossed content delivery networks in 2014, and it increases to 72 percent by 2019. However, CDN deployment and maintenance is very expensive, and so it is not an easily scalable solution to aforementioned problem. P2P networks, which is based on the concept of equal peers that can share content directly with each other, is a cheap solution compared to CDN. A group of peers can cooperatively provide the video request of another peer, each providing part of the request depending upon the content availability and their distance to the destination peer. P2P can provide scalability solution easier and cheaper than CDN, but the reliability and service availability might be an issue.

Another approach that can alleviate this growing demand for bandwidth, and the burden on streaming servers, is the concept of pervasive router caches. This is an emerging concept that proposes enabling caching, even small amount, at all the routers in the network. A router can

opportunistically store the forwarded contents and serve other requests for the same content from its cache. By caching commonly requested contents closer to the clients, the number of communication links involved in providing the requests of a client can be significantly reduced, and the amount of requests that should be handled directly from the server(s) will be reduced as well.

Video content placement and caching can be divided into two general groups: long-term caching [25, 82, 21, 52, 80, 26, 72] versus short-term caching [87, 59, 68, 71]. As is clear from the names, in long-term caching the update rate of the caches is low and contents will stay in the caches for a longer time compared to the short-term caching; for instance, a few hours compared to a few minutes or even seconds. In the long-term content placement algorithms, the long-term statistics of the contents is considered in order to decide what should be stored for the next few hours; For example, in [21] and [52], the authors are solving how to distribute the content in a VoD network, given the statistics of the requests, and the objective is to minimize the streaming rate from the servers. In contrast, in short-term content placement, the current requests are the base of the caching decisions, and the goal is to do the caching in order to help the existing requests. For example, in [31] the authors proposed a buffer management policy at the on-demand video servers to reduce the disk load. They consider the active flows at each moment, and based on them decide which parts of the videos should be buffered at the streaming servers.

In this thesis, we want to find caching solution for video streaming, both long and short caching, by taking advantage of the pervasive router caching, and using two concepts, network coding and Software Defined Networking (SDN), in order to improve the currently available solutions.

Our assumed video streaming network architecture is shown in Figure. 1.2. A content distribution network (CDN) provider distributes the video files in different geographical locations through multiple servers. Users access content from these CDN servers through a network of routers. All the routers are SDN-enabled, and have a logical connection with the SDN controller. Many, if not all, of these routers have a cache to facilitate a client's access to video content. We divide the caching capacity at each router to long-term cache and short-term cache, and the goal is to fill each portion appropriately, in order to minimize average rate on the links in the network. By utilizing the broader view at SDN controller, we propose SDN-enabled router caching strategies, for both

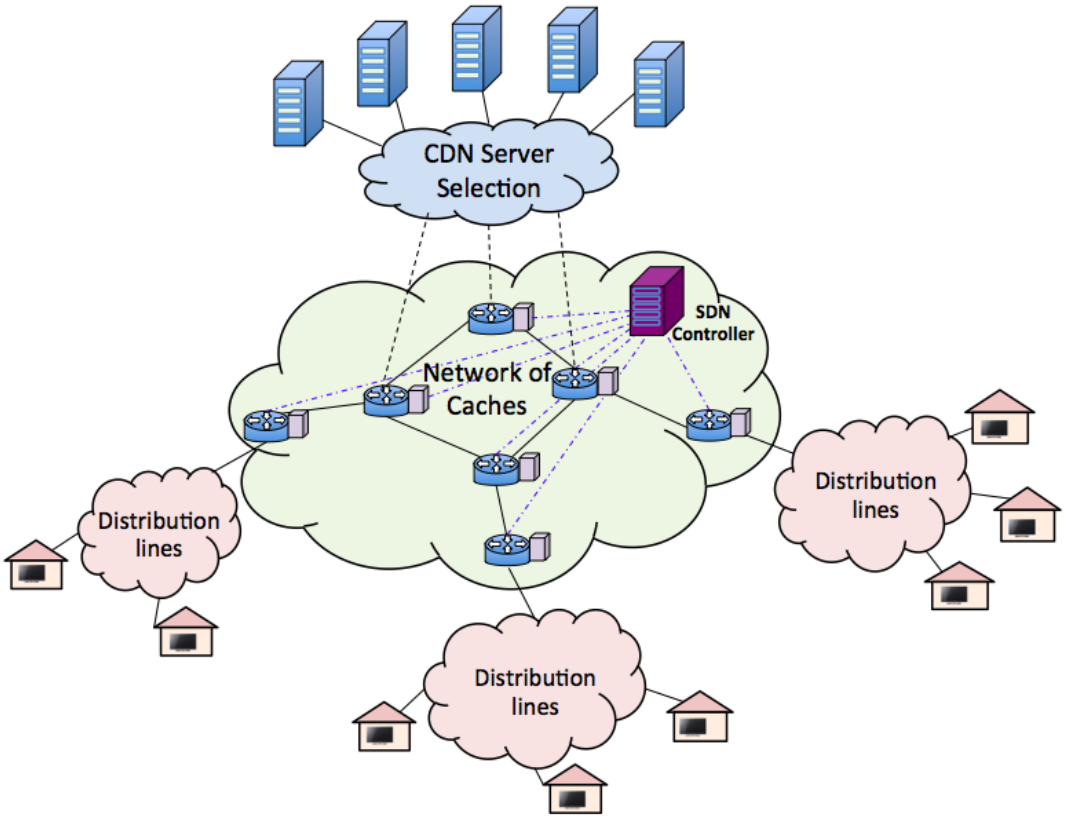


Figure 1.2: An example network of routers with caching capability

long caches and short caches, that significantly reduce the average network bandwidth needed to deliver the video streams to the respective clients.

Before describing our approaches to solve long and short caching at router caches, lets briefly review software defined networking and network coding.

1.1 Background

1.1.1 Software defined networking

Traditional networks have a static architecture, which makes it unsuitable for the dynamic nature and requirement of most todays networks. Making any kind of change in these static architectures, for example addition of a new device, is complex, time-consuming, and with the potential risk of service disruption. The reason is all the required changes and the network-wide policies

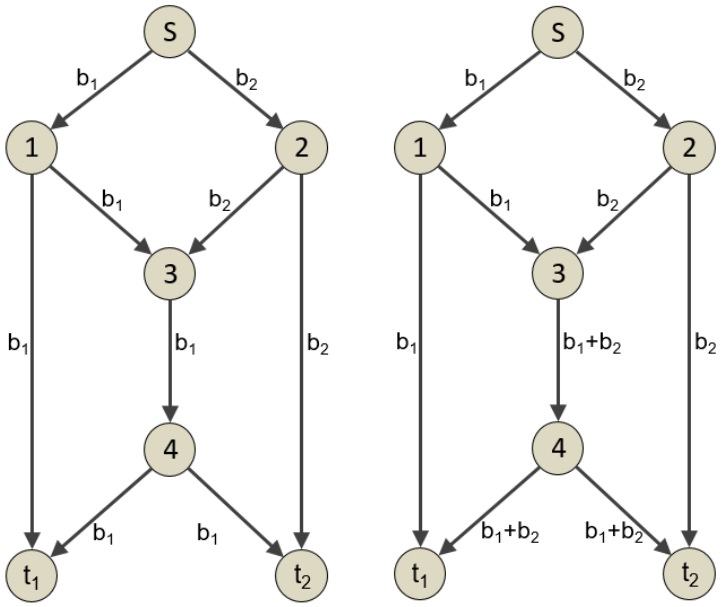
that need to be implemented because of those changes, need to be made manually, and because of lack of standard, they are mostly vendor dependent. This same issue also makes scaling of such networks a very complex and time-consuming task. In todays network, applications that requires large geographically distributed servers, with a very dynamic traffic and therefore resource requirements, call for a new networking paradigm.

Software Defined Networking (SDN) [16] is an emerging approach to solve these problems. By separating the control plane and the data plane, the SDN architecture brings increased flexibility to network design. SDN is manageable, adaptable and cost-effective solution to todays network requirement. By separating control layer and forwarding layer, network control can be programmed, and hence adjusted dynamically by the network administrators. In this architecture, a single controller usually manages the control plane of many routers. As a result, the controller has a more global perspective of the flows traversing a part of the network as compared to a traditional single-router perspective. The controller can take advantage of this global perspective to make more effective decisions on how the flows traverse the network. The controller can easily effect these decisions by making real-time changes to the routing rules in its routers.

One of the great advantages of SDN is it is gradually deployable, meaning that one can adopt SDN only on part of the network, without having to deploy it on all the network at the same time. For this reason, major service providers today are largely considering SDN as a solution to manage the ever growing traffic in their networks and trying to take advantage of this new networking paradigm to decrease the cost of their network maintenance. Internet content providers like Facebook and Google are already using SDN to improve hyperscale data center connectivity. AT&T publicly announced that by 2020, they plan to virtualize and control over 75 percent of their network using the software-defined architecture to meet the growing demands of data and video-hungry users [4]. In this thesis, we show that global perspective of a SDN controller can be exploited to make an efficient caching decision at each router and thereby dramatically reduce the bandwidth demands for video streaming in emerging Internet.

1.1.2 Network Coding

In their pioneering work, Ahlswede et al. [19] first showed that an idea called network coding (NC) could be used to improve the multicasting rate in multihop networks. In NC, intermediate relay nodes are required to strategically encode certain incoming packets before forwarding them to all the adjacent nodes in the multicasting tree. They showed that the minimum of the max flow rate from the source to the receivers can be achieved, if network coding being applied properly in the network. The idea of Network coding and how one can achieve higher rate in multicast scenarios can be explained by the famous butterfly example that was first introduced in [19] and is shown in Figure 1.3. There is a source node, S, and two receiver nodes, R1 and R2. The goal is to multicast data streams from the source to the receivers. Assume that all the links in the network have a unit capacity. It can be easily seen that the maximum flow rate that can be delivered from the source to either of the receiver nodes is 2. Now the question is how to achieve this delivery rate to both receivers at the same time in this network? For example, assume that b_1 and b_2 are the data streams that we want to multicast from the server to the receivers (Figure 1.3a). Without network coding, there is no way that both receivers can achieve their maximum flow rate of 2. The problem is the link from node 3 to node 4 is a bottleneck in this example, and only one data stream, b_1 or b_2, can be forwarded through that link at each moment. If b_1 is forwarded through the bottleneck link, t_2 will receive both b_1 and b_2, but t_1 can only receive b_1, twice. Same problem happens when b_2 is forwarded through the bottleneck link; only t_1 can receive both b_1 and b_2, but t_2 will receive only b_2, twice. Therefore, the average delivery rate for the case without network coding is 1.5, which is the maximum achievable rate. Figure 1.3a shows one such solution without network coding. However, one can take advantage of Network coding, and solve the bottleneck problem at the link between node 3 and 4, as shown in Figure 1.3b. After receiving streams b_1 and b_2 from nodes 1 and 2, instead of separately forwarding each stream on the bottleneck link 3-4, node 3 will linearly combine the two streams before forwarding them, and so we can deliver both streams to the receivers simultaneously. t_1 will receive stream b_1, and $b_1 + b_2$, from which it can reconstruct b_2 by having b_1. t_2 will receive stream b_2, and $b_1 + b_2$, from which it can reconstruct b_1



(a) Without network coding (b) With network coding

Figure 1.3: A one-source two-sink network with coding

by having b_2. Thus, by using network coding in this example, the maximum possible flow rate of 2 can be achieved.

In the general case, one can use a random linear network coding, first suggested by T. Ho et al. [44], where the encoded packets are the linear combinations of the original packets. Later, Chou et al. [30] showed that random linear coding is sufficient to reach the optimum NC performance. Random linear coding works as follows in a network: Consider a group of packets D_1, D_2, ..., D_n arrives at a router. The router generates a random linear encoded packet by associating a coefficient c_i to each packet D_i, where c_1, c_2, ...,c_n are randomly selected from a Galois Field of size T. The encoded packet is equal to $X = \sum_{i=1:n} c_i D_i$. The router sends the encoded packet, along with the coefficient vector $\overline{c}$. All nodes in the network will apply network coding on the received packets, encoded or non-encoded, before forwarding them. For example, assume a node receives a set of packets $(\overline{c}_1, X_1)$, , $(\overline{c}_m, X_m)$ on its incoming nodes, where X_j is packet j and $\overline{c}_j$ is the coefficient vector of packet j. The node randomly selects a new set of coefficient $\overline{k}$, and generates

a new encoded packet: $Y = \sum_{i=1:m} k_i X_i$. Then it sends Y along with the new coefficient array $w_i = \sum_{j=1:m} k_j c_{ij}$. At the receiver nodes, after receiving enough linearly independent encoded packets (along with their coefficient vector), the decoding can be simply done by solving a system of linear equations. By selecting the size of Galois Field large enough, the probability of having linear dependent coefficient vectors at the receivers, which will prevent the packets from being decoded, will be very small.

Network coding is mostly being applied at the delivery phase. But in this thesis we show that the same idea can be applied to the caching, by having advance knowledge of the contents statistics, and the network structure.

1.2 Thesis Contributions

As mentioned earlier in this chapter, lots of work have been done in the area of content placement, where the problem is modeled in different ways and different objectives have been considered. But the emerging area of pervasive router caches brings the new opportunity of having the network of these router caches, and video caching at that level in such a network has not been well studied. In addition, the move towards SDN and the capabilities that it brings to the network, add more potential to the router caching, that has not been fully studied. In this thesis, we want to look at video caching in such a network in order to help with efficient delivery of video streams. Our proposed solutions consist of both long-term and short-term caching. The router caches in these networks have small caching capacity, which needs to be carefully used for caching. We start by proposing a long-term caching algorithm that introduces the idea of network coding in caching. None of the previous work have studied the advantage of combining different contents before storing them in the caches, in the same way that routers in network coding combine different received packets before forwarding. Our solution is modeled as an integer linear program, which takes as input the probability distribution of the videos, as well as the cost of delivery of one unit of data from each router to the clients. By considering the possibility of storing cross encoded content, our proposed algorithm solves which content should be stored at each router, and whether it should be stored encoded or non-encoded, in order to minimize the average cost of video delivery. We

show that by considering storing cross encoded videos in the caches, we can decrease the cost of content delivery. In order to solve scalability problem, which is an issue for large networks, we partition the network to subsets and apply our method on each subset, in a systematic way. The performance of our method is simulated for a large network and we show that even by applying a simple partitioning solution, one can achieve a considerable gain from this method.

Filling the caches based on only the statistical information of the requests cannot fully utilize the caching capacity, because we cant take advantage of the dynamic nature of the requests and therefore the dynamic caching opportunities as a result. Short term caching, as explained before, should be considered along with long term caching, for maximum performance. Specially, in todays network, because of the vast popularity of Video on Demand(VoD), clients start videos at different times which put a large burden on the streaming server as well as the network. But it is commonly observed that at any given time, only a small set of videos is popular (see, e.g., [27], [28], [40]). Consequently, many clients are watching the same video at any given time, albeit with staggered start times. We introduce a new approach that substantially improves the delivery of staggered video multicasts, by taking advantage of the global view of the SDN controller in the network, as well as the distributed network of router caches. In our solution, the SDN controller carefully analyze the streams routing through different parts of the network, and try to find streams that can help other streams, by partial storing of some stream in some caches. By using its global view, and small computational complexity, SDN controller can find a very efficient caching solution, with a performance close to the optimum solution, and update the corresponding router caches accordingly. Through extensive simulations for different parameter values, we prove the effectiveness of our approach. We implemented our solution using the SDN capabilities in the National Science Foundations Global Environment for Network Innovation (GENI) experimental platform [5], to further prove the credibility of such a solution.

Our proposed long-term and short-term caching solutions should be jointly applied in a network for the maximum performance. Therefore, we also investigate how one can jointly use long and short caching at the router caches in the network. We analyze the impact of the cache size

assignment to long-term and short-term caching, and show that a correct division of caching capacity between them matters. In our short caching solution, we assumed that the routing is given, and caching decision is applied given the routing solutions for the current streams in the network. An interesting enhancement of this case is improving routing solution, such that it is aware of caching algorithms that are being applied on the network. We propose a caching aware routing that routes the new requests by considering the similar but time-shifted streams that might be streaming through the network, and trying to route the new request through similar and close paths, if possible, to enhance the caching opportunities. We show through simulation results that one can improve the caching gain by applying such a routing.

Figure 1.4 shows the blocks of the proposed pervasive router caching strategies for staggered video multicasts, and the connections between them. These blocks will be run and controlled by the SDN controller. We also show the inputs of each block. CCIP block on left, which solves the long-term caching based on network coding, will be run periodically by the SDN controller. The scheduling for running CCIP module depends on the network statistical behavior. In more static network, the network manager can setup the period to once a day, but in more dynamic networks, the network might require a more often update rate. The inputs of CCIP blocks are shown as demand statistics, delivery cost, cache size at routers. The output of CCIP block will determine what should be stored in the long-term caching portions of the caches. Then the next block, cache-aware routing, will be run by the SDN controller, to solve the routing given the knowledge of CCIP caching decisions, in addition to the list of inputs shown on top of the block: new requests, existing requests, and existing routes. The cache-aware routing block will be triggered upon every new request arrival, and the output will determine the selected route for the new request, in addition to the long cache (including the source) selected as the provider of the new request. Then SDN controller will run the SCS module, which solves short-term caching, given the calculated route of the new request(s), in addition to the other set of inputs shown on top of the block: new requests, existing requests, and existing routes of those requests. Similar to the cache-aware routing, the SCS module will be triggered upon every new request arrival. The output of the SCS block will

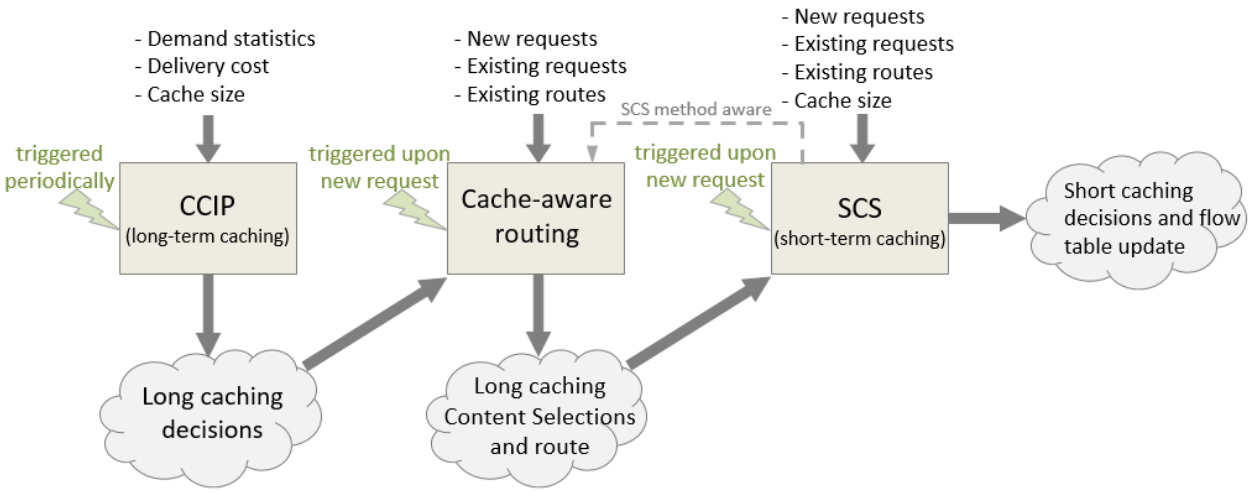


Figure 1.4: Proposed pervasive router caching strategies for staggered video multicasts

determine the short-term caching decisions, and how the flow tables at the routers need to be updated accordingly.

The contributions of this dissertation are summarized as follows:

1. The advantage of the pervasive router caches in an SDN-enabled network, to enhance staggered video multicasts has been shown for the first time in this work.

2. As a long-term caching approach, we generalize the network coding from delivery phase to caching, and propose the idea of storing the cross encoded videos in the router caches with our CCIP solution. We show how such a method can help in reducing the delivery cost in the network, especially considering often small capacity of router caches.

3. As a short-term caching approach for the problem of staggered video multicasts, we introduce our SCS algorithm with a performance close to the optimum model, but at much lower complexity. We show that our solution significantly reduces the bandwidth requirement in the network even with small router level caches, and largely outperforms traditional caching methods, Least Recently Used (LRU) and Least Frequently Used (LFU).

4. Analysis of joint long and short caching at small router caches for video streaming application in SDN-enabled networks, and the trade-offs to achieve the highest benefits, are for the first time studied in this work. We also proposed a routing algorithm aware of both long-term and short-term caching in the network, which further enhances the caching performance.

1.3 Thesis Organization

The rest of this document is organized as follows: First in chapter 2 we describe the general network conditions and assumptions considered in the later chapters of this thesis. In chapter 3 we review the related work, and clarify the differences of this work with the previous related work. We cover the related work in the area of caching and content placement by dividing them to two general areas of long caching and short caching. In chapter 4, we introduce the idea of storing cross-encoded videos in the router caches. We show that by considering the statistics of the requests, and the delivery cost of a unit of data from the routers to the clients, one can apply network coding idea in storage phase, rather than the well-studied delivery phase, and decrease the delivery cost. In chapter 5 we introduce our solution for short-term caching of staggered video multicasts. We also modeled the optimum solution as an integer program, and show that the performance of our proposed solution is close to the optimum solution, but with much less complexity. We also show the implementation of the proposed algorithm on GENI. Later in chapter 6, we explain how one should take advantage of both long and short caching of videos at router caches, by applying both methods introduced in the previous chapters in a network. We divide the caching capacity at the routers, and assign part of it to long cache and the rest to short cache. We also introduce a routing algorithm that is aware of long caching and short caching in the network, in order to benefit the most from our proposed caching solutions. The goal of our routing algorithm is to route the requests such that our caching methods can take the most advantage of the distributed caches in the network. We conclude this document in chapter 7 and explain the future work in order to enhance the proposed solutions in this work.

Chapter 2

Network Model and Assumptions

In this chapter, we want to describe the network model and assumptions that are considered in this thesis in different chapters. We start with the envisioned network architecture and explain the general network assumptions. Figure 2.1 shows a network of router caches which is the focus of this thesis. The caching capacity of the routers is divided to long-term and short-term caching, shown in the magnified part in Figure 2.1. In chapter 4 where we propose our novel long-term caching solution, we consider only the long-term caching portion of the router caches. Similarly, in chapter 5 where we propose our short-term caching algorithm, we only consider the short-term caching part of the router caches. At the end in chapter 6, which covers joint long and short caching, both parts of the caching capacities at routers are considered jointly.

The rest of the assumptions and models are categorized in small subsections as follows for easy reference.

2.1 SDN controller components

In the previous chapter, we mentioned that we are assuming the network we are considering is SDN enabled, and we want to take advantage of the global view of the SDN controller to improve video streaming in such a network. The SDN controller, with logical connection to all SDN-enabled routers as shown in Figure 2.1, is responsible for routing and caching decisions at all routers. We consider four main blocks at the controller in this thesis (see Figure 2.2): network topology database, flow table database, caching table database, and computing center. Not all the parts are used by both caching modules, but some are used only by the long-term caching module,

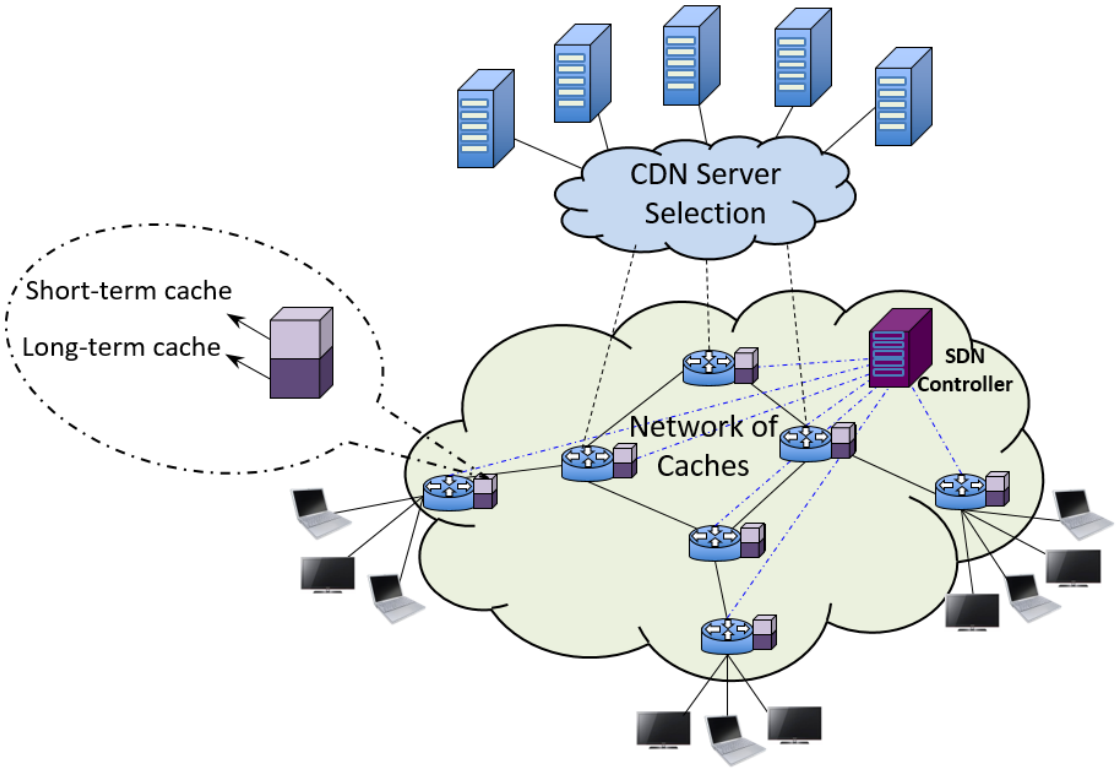


Figure 2.1: Network model where router caches capacity is divided to short-term and long-term portions

and some are used by only short-term caching module, and some blocks are used by both modules. Network topology database is responsible for keeping track of any change in the topology; for example, if a link is broken, the topology database should be able to quickly update the topology, so that routing and caching module (short-term caching) will be updated accordingly. Flow table database follows all the ongoing forwarding decisions that have been made per flow; Caching table database keeps record of the cached contents at all the routers. Computing center comprised of three main modules: network statistics, routing module, and caching module. The caching modules consist of our proposed long-term caching and short-term caching algorithms, which we will describe them in details in the later chapters. The network statistics module calculates some statistical information of the network, specifically the demand statistics of the clients (popularity of the videos), and the average delivery cost of a unit of data from different routers to the clients.

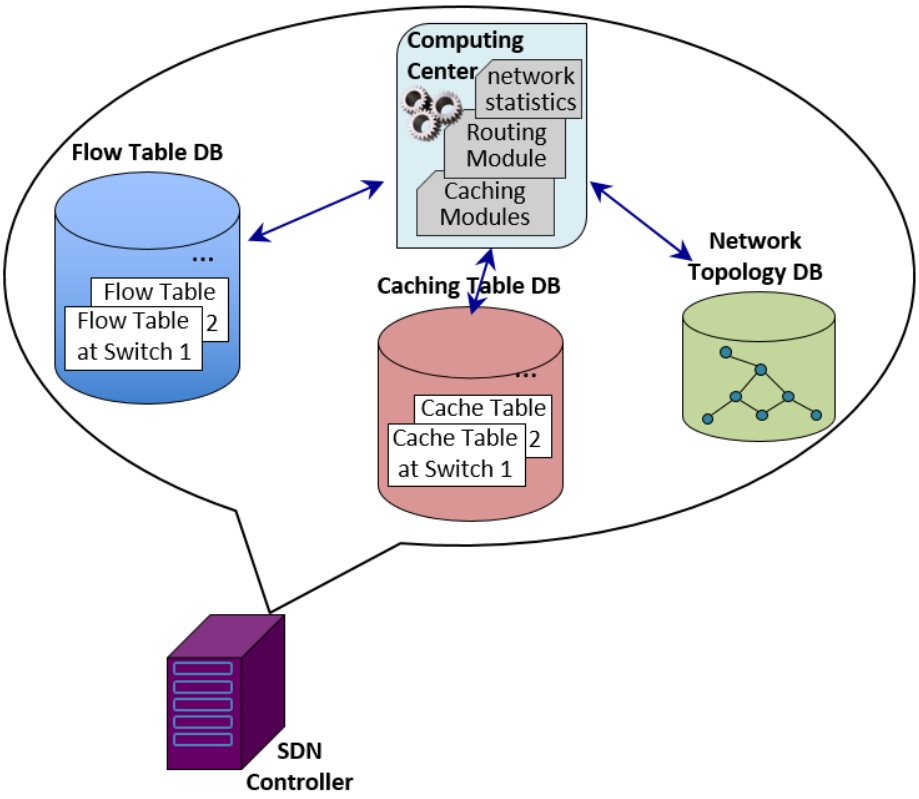


Figure 2.2: SDN controller components adopted in different parts of the thesis

SDN controller uses this statistical information to solve our long-term caching solution (CCIP algorithm). The network statistics information will be only used by the long-term caching algorithm but not short-term caching module. In this thesis, we will not describe how one should calculate the network statistics, and we assume that the output of this block is known and given to us.

Given the network statistics, the controller runs the long-term caching module (CCIP algorithm) and fills the long-term cache portion of the router caches accordingly. The update of the long-term caches happens once every few hours, or even less frequently, e.g. once a day, and only if the network statistics change. The SDN controller will update the long-term cache contents only during the times when the network is not too busy, in order to make sure that delivering the videos from the content servers to the caches will not put an intolerable load on the network. The caching table database will keep the long-term caching decisions as a result of CCIP caching module. It will be also continuously updated by the caching decisions made by the short-term caching

module. The caching table database and flow table database will be used only by our proposed short-term caching algorithm (SCS), and not our CCIP long-term caching module.

2.2 Video statistics and delivery assumption and model

The video request distribution is assumed to follow Zipf function, since multiple studies, have observed long tail properties in the popularity of videos, [28], [40]. We study different Zipf exponent for the request distributions in the simulations. We assume that all videos have same length and rate, and each video is divided into chunks of equal sizes, e.g., chunks of size 10 seconds, and the clients make requests for chunks. Specifically, we assume that all videos are divided into 100 chunks, and clients start by requesting the first chunk of a video, randomly selected according to the Zipfian distribution of a given exponent.

2.3 Client arrival model and behavior

We assume the clients randomly arrive according to a Poisson arrival of a specific rate, at different points in the network, and request a movie. We study different Poisson arrival rate for the clients in the simulations. Requests are in units of chunks, and after a client received a requested chunk, it will request the next chunk and so on. We assume that all clients start from the beginning of the movies, watch until the end of the movie, and then leave.

2.4 Router Cache assumption

We assume the caching capacity of the routers is divided to long and short caching, and the assigned size to each part remains the same during the simulation time. We study different assignments to each part in the joint long and short caching chapter. The cache sizes are given in units of chunks; for instance, caching capacity of 100 chunks means capacity to store one video, since we are assuming that each video consists of 100 chunks.

Chapter 3

Background and Related Work

Much research has been done to solve video distribution and caching problem, in order to reduce the large burden on the streaming servers, as well as the network links bandwidth requirements. In this chapter, we cover major related areas to the problem of video delivery and caching in the networks.

3.1 Content Delivery/Distribution Network (CDN)

Content Distribution/Delivery Network (CDN) is a popular approach to this problem. CDNs improve network performance by distributing the contents among the servers located at different locations, which are called surrogate servers. CDNs play a very important role in todays networks, and the delivery of the large amount of content to a large number of users in the current Internet without CDN concept is not possible. The advantages of CDNs can be summarized as reducing original servers load, reducing the latency of content delivery, and increasing the throughput of the network.

There is a lot of work on improving the content placement and performance of CDN networks. For example, DiPalantino et al. [33] and Jiang et al. [51] show the advantage of cooperation between the content providers (CDNs) and Internet service providers to jointly solve the content distribution and traffic engineering problems. Xie at al. [42], they modeled the joint problem as a linear model and compared it with the case when content distribution and traffic engineering is being done separately. A simple tree hierarchy of caches is proposed in [25] in order to decrease the load on the CDN servers. Minimizing load imbalance at the servers while maximizing the system

throughput is done by Zhou et al., [92]. Authors in [82] suggested a simple strategy to distribute the contents at the clients' gateways, in order to build a supervised peer-to-peer network to save energy consumption of the data centers. In [50], a largely distributed group of small storage at users homes is considered as last-mile CDN servers, and a scalable method to manage content replication and request routing is proposed. Applegate et al. [21], propose a Mixed Integer Program (MIP) formulation to find the optimum content placement at distributed servers in a large scale VoD system. In [52], authors are considering a similar problem, but fractional storage approach was proposed which simplified the solution. Unlike [21] that stores the whole video in a cache, or does not store it at all, their method allows storing fractions of the videos. Qiu et al. [73] focused on minimizing the overall cost by finding the optimum positions of the web servers in the network. Request routing is another problem in CDNs that has been studied in several research such as [20], [22] and [58], in order to find the best replicated server to direct a user request towards it. A survey of CDN solutions with coverage of different aspects of the problems is done in [70]. More information about the characteristics of the CDNs can be found in the research about two well studied CDNs, YouTube [18] and Akamai [69] CDNs.

3.2 Peer to Peer (P2P) networks

Another approach to the problem of video delivery to a large number of users is the idea of Peer to Peer (P2P) networks, which eliminates the need for dedicated servers. P2P networks are consist of equally privileged participants (peers), who are both supplier and consumer of the resources [75]. P2P networks [81] are designed for the direct sharing of computer resources rather than requiring any intermediate and/or central authority [70]. An example of such system is BitTorrent [47], which is a popular P2P replication application.

Many researchers studied different aspects and problems is such networks. For example, Tan and Massoulie [80] studied the problem of optimal content placement in P2P networks while maximizing the utilization of peer uplink bandwidth resources. Boufkhad et al. [26] formulate the problem to maximize the number of videos that can be served by a collection of peers. To avoid network congestion and to improve performance, P2P techniques can be used to build an adaptive

CDN, where content storage and workload from streaming server, network, and storage resources are offloaded to the end-users' workstations, e.g. [91]. P2P networks offer cheaper solution compared to CDN to the scalability issue in content distribution networks. But P2P network have their own problems. Content copyright and security, as well as reliability and stability are very important concerns in P2P networks. In addition, Quality of Service(QoS) is best-effort and cannot be controlled, [60].

3.3 Software Defined Networking (SDN)

SDN is a promising approach to manage complex networks. Through separation of control plane and data plane, SDN is offering a great flexibility in controlling the network from a controller. OpenFlow [64],[15] is todays dominant protocol in SDN networks to define the communications standard between the switches and the controller. A survey on SDN and OpenFlow, and their applications can be found in [45].

In the area of Video Streaming in SDN, there is some research that have been done in the past few years; for example, the authors in [34] proposed an OpenFlow-based video delivery scheme with Quality of Service (QoS) support for scalable video streaming. They consider two QoS levels and solve optimization of dynamic QoS routing as a constrained shortest path problem, by treating the base layer of scalable encoded video as a level-1 QoS flow, while the enhancement layers are treated as level-2 QoS or best-effort flows. In [65] the authors try to enhance the Quality of Experience by proposing an SDN application designed to monitor network conditions of streaming flow in real time, and dynamically change routing paths using multi-protocol label switching (MPLS) traffic engineering to provide reliable video watching experience. Jarschel et al. [49] have shown SDN-based application-aware networking for YouTube video streaming. They have conducted a performance test of several path selection mechanisms such as round-robin, bandwidth-based, deep packet inspection-based and application-aware in an SDN-enabled network. In [36] they present OpenCache as a configurable and transparent in-network caching service that aims to improve the Video on Demand (VoD) distribution efficiency by caching video assets as close to the

end-user as possible. By leveraging SDN, OpenCache benefits last mile environments by improving network utilization and increasing the QoE for the end-user. However, they do not propose any caching method and assumes very simple methods caching decision strategy. Some other related researches considering QoS in video streaming by utilizing the advantage that SDN brings to network are [46], [78], [86], [36], [65].

In the area of Information Centric Network (ICN) or Content Centric Network (CCN), which proposes a shift from host-centric networking to content-centric networking [55],[48], the topic of pervasive caching is a hot ongoing research topic and there are several new proposals on how to implement it in a network. For example, some groups of research are considering the implementation of ICN or CCN in SDN enabled networks, and taking advantage of the flexibilities that SDN provides, for example [83], [74], [35], [79], [29]. However, none of these works describe a caching strategy designed for SDN-enabled networks. One can take advantage of ideas in the these papers to efficiently implement the caching strategies in SDN networks.

3.4 Network Coding

In their pioneering work, Ahlswede et al. [19] first showed that an idea called Network Coding (NC) could be used to achieve optimal multicasting rate in multihop networks. In NC, intermediate relay nodes are required to strategically encode certain incoming packets before forwarding them to all the adjacent nodes in the multicasting tree. T. Ho et al. [44] suggested using random linear coding for this purpose, where the encoded packets are the linear combinations of the original packets with the coefficients randomly selected from a finite field. Later it was shown that random linear coding [30], or even sparse linear NC [61], is sufficient to reach the optimum NC performance. The following subsection reviews the related researches in the area of content delivery and caching using NC.

3.4.1 Network Coding Applications in Content Delivery and Caching

NC can be applied in different forms to help in content delivery. The original form of NC uses encoding in delivery phase and sends the encoded packet to multiple users, who can decode that

because of what they already have in their caches. There is also another way that NC can help in content delivery. Instead of storing the original packets, we can encode those packets multiple times with different coefficients, and store the encoded packets in the servers or peers and transmit them when requested.

Network coding can enhance the performance of distributed storage systems and peer-to-peer (P2P) networks by simplifying the block scheduling required in these networks, e.g. [39],[23],[62], [37],[53],[85]. An example of this idea is suggested by Gkantsidis et al. [39], using a network-coding based method for distributing large files efficiently in P2P content distribution networks. In their algorithm, each file is divided into many small blocks so that when a client requests the file, the server/peer sends one or more random linear combination of the blocks (applying random linear coding) to the client. The client recovers the file upon receiving sufficient number of independent encoded blocks. [37], [38] provides positive simulation evidence in the benefits of using NC in P2P networks. It has been shown that the throughput gain provided by network coding can be more than 2-3 times better than transmitting unencoded blocks. Also Wang et al. [85] did a reality check on the performance of the P2P with network coding in their testbed called Lava, and showed that network coding makes it possible to perform streaming with a finer granularity, which reduces bandwidth usage and improves resilience to network dynamics. In [57] the authors show that NC can improve caching efficiency besides network performance. Network coding is well considered in the previous research for video streaming application in different scenarios, e.g. [54], [77], [76], [84], [66]. Reference [32] did an overview of the researches on applying NC on distributed storage systems. Maddah et. al [63], [67] propose a general solution to the caching problem using network coding and showed that their solution is within a constant factor of the optimum solution. However, they do not consider any cooperation between the caches (each client is getting its requested data from its cache, or otherwise from the server), which distinguishes from the problem considered as a part of this thesis.

3.5 Long-term and short-term caching

Research in the area of video content placement and caching is divided to two general groups of long-term versus short-term content placement. In long-term caching the update rate of the caches is low and contents will stay in the caches for a longer time compared to the short-term caching; for instance, a few hours compared to a few minutes or even seconds. Many of the works in the CDN and P2P networks are in the category of long-term caching, where the data placement are not based on the current requests, but rather the statistical information of the demands. Most research described in the CDN and P2P and network coding sections falls in the category of long-term caching. So here we continue with the description of the short-term caching research.

On short-term caching, and specifically for the staggered video multicasts, there is a lot of research on time-shifted TV (TSTV); and the majority of them focus on the cache management at the server to decrease the servers I/O bandwidth. For example, [93] is looking at the problem of load balancing of the multimedia servers for the case of TSTV. In [31], authors proposed a new buffer management policy at the on-demand video servers to reduce the disk load. They consider the active flows at each moment, similar to our idea in this work, and based on them decide which parts of the videos should be buffered. Authors in [89] claim to have a more efficient piece-wise patching solution, compared to the ones previously proposed, to the TSTV problem in Hybrid Fiber Coaxial (HFC) network. This body of work is different from the problem in this thesis: we are not considering a single server cache, but a network of small caches at each router. There is some related research in TSTV where they consider the distributed caches in the network; for example, [56] recommends a method to spread the chunks of the videos in the network, in order to reduce the total requests that will be received by the server. The solution is not based on the active requests at each moment; so it is not a suitable solution for small router level caches in video on demand networks. In [87] they are suggesting an adaptive size caching window at the access network closer to the clients to help in reducing the server load for TSTV. They are assuming a very simple structure for the network, and their solution works only for TSTV when there is a main stream for each channel. Another approach to TSTV is peer-to-peer structure where a peer is a connected

device, and peers use the locally buffered portion of the videos at other peers. For example [59] uses multiple interval graph structure to track the stored portions of videos that are stored locally at the device, in order to determine which peer can provide a requested chunk by a new client. In [68] and as an extension to their peer-to-peer multicast protocol (SPPM), the authors added a simple solution to handle time-shifted requests in a TV broadcast. The video source maintains the estimated latest time-stamp of video contained in different peers buffers, and for each new arrival, will find the set of peers that can provide the contents and selects the best among them.

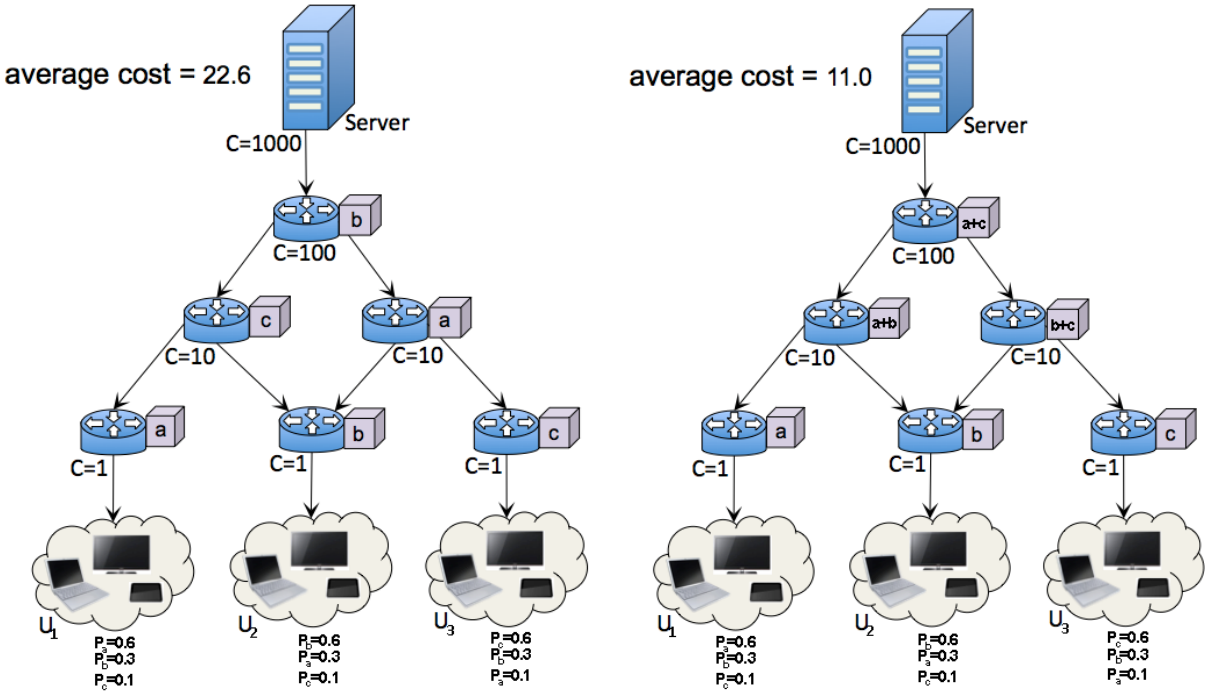
In contrast to these works, our proposed short-term caching approach in this thesis is an SDN-enabled caching that takes advantage of network of small router caches to help with efficient delivery of staggered video multicasts in Video on Demand networks (VoD). Using the global view of the SDN controller, our caching algorithm determines the cases that a flow can help another flow by caching the lag between them.

Chapter 4

Storing Cross-Encoded Video Contents at Router Caches

In the introduction and model description chapters, we explained about long-term caching as one type of content placement which considers the statistics of the demands and tries to fill the caches accordingly, in order to enhance the average performance of the network. In this chapter, we will focus on the portions of the caches at the routers designated to long-term caching.

The goal here is to improve the state-of-the-art by taking advantage of a concept called *network coding* [19], and was described briefly in the introduction chapter. Although papers in literature have proposed using network coding to enhance video streaming performance in a variety of ways, this chapter demonstrates that there are additional opportunities for exploiting network coding to further improve performance. Specifically, in literature, they divide video content into smaller chunks and network encode the different chunks prior storing or caching, e.g. [39]. However, none of the existing work network encode chunks from more than one video prior to caching. We show that network coding across videos can significantly improve performance when multiple users are likely to access the same set of videos. A simple example to illustrate the idea is depicted in Figure 4.1, which shows a video distribution network to three users. The users are most likely to access one of three videos labeled a, b, and c. The likelihood of accessing the three videos by these users is also in the figure. For instance, U_1 has a 60%, 30%, and 10% probability of accessing videos a, b, and c, respectively. The figure shows a network of router level caches on the paths from a video server to users U_1, U_2, and U_3. Assume the video server is large enough to hold all three videos, however, the router level caches can only hold one video. The cost of accessing the videos from each of the router-level caches is also shown in the figure; they are assumed to be 1, 10, 100, or 1000, respectively. Figure 4.1a shows conventional content placement in the



(a) Optimum Content placement with No Coding

(b) Optimum Content placement with Coding

Figure 4.1: An example to show network coding across videos can be helpful in caching

router level caches, i.e., when network coding across videos is not exploited. The average cost of accessing the three videos for U_1, U_2, and U_3, are 31.6, 4.6, and 31.6, respectively, with the overall average cost of 22.6. Figure 4.1b shows content placement where some caches contain network coding across videos. The average cost of accessing the three videos in this case for U_1, U_2, and U_3 are 14, 5, and 14 respectively, with the overall cost of 11. Thus, the benefit of network coding across videos is a reduction in expected cost from 22.6 to 11, when averaged over three users. The benefit, of course, depends on the relative cost of accessing the different caches, the probability of accessing each content by different users, and the placement of content in each cache. Given the costs of accessing the caches and the probability of accessing each content by different users, the challenge is to determine content placement that reduces the expected cost of access.

For each client, there is a certain cost to access the contents from a router level cache or from the CDN servers. The cost may be used to model access latency, bandwidth constraint, opportunity

cost of storing a video, number of hops to the clients, etc. The challenge is to determine what content should router level caches contain and when.

4.1 Problem Statement and Assumptions

The problem of optimizing video delivery networks can be modeled in many different ways, considering the different objectives one can target. An overview of these optimization models is given in chapter 3. In this chapter, the problem is modeled as an optimization problem, with the objective of minimizing the average cost of the video delivery in the network.

We are assuming that the network is SDN-enabled, and the SDN controller is responsible for routing and caching decisions in the network, and it will update the flow tables of the routers accordingly. In Figure 2.2 we showed the SDN controller modules responsible for the long-term caching. We showed that there is a module at the controller which is responsible for calculating the statistical information of the network, specifically the delivery costs of a unit of data from the routers in the network to the clients, and the popularity of the different videos in the network. Given this statistical information, and the long-term caching capacity at each of the routers in the network, the SDN controller runs our long-term caching algorithm which we will explain in this chapter. Then the controller fills the long caching portions of the routers caches according to the result of the caching algorithm. The SDN controller will schedule this data movement when the network is not busy, in order to prevent any unbearable load on the network. After filling the long-term caches, the controller will update the flow tables of the routers. The videos are divided into chunks of a specific size, e.g. 10 seconds, and we consider chunks as smallest unit of video that can be stored.

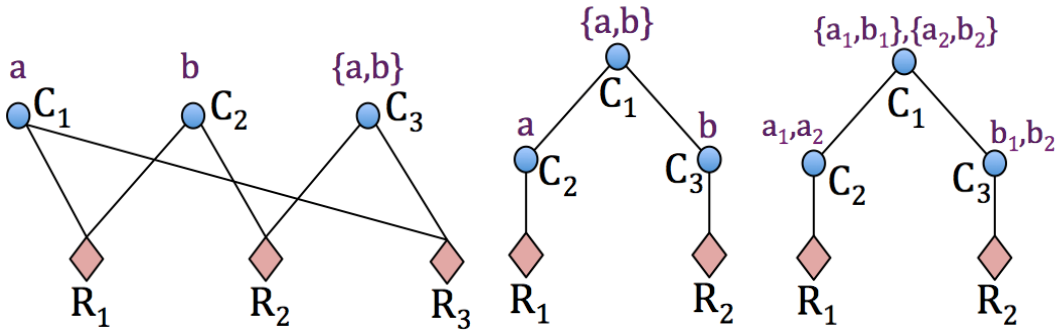
In the following sub-section, our idea of caching across videos and storing encoded chunks of different videos, and how it can help, is further explained.

4.1.1 Why Caching Encoded Data?

At the beginning of this chapter, we showed the usefulness of encoding across videos before caching, in an example. In this section we bring more examples to make the idea more clear by pointing out some specific cases when storing cross-encoded content can be helpful.

In the network shown in Figure 4.2a, assume that clients 1, 2 and 3 (R_1, R_2 and R_3) are interested in the same set of contents, statistically. In other words, if we sort the probabilities of the most frequently requested data for these clients in three arrays, the arrays are the same. Assume that the storage capacity of all three caches in this example, C_1, C_2 and C_3, is one data unit. A line between a cache and a client shown in the figure means that there is a connection, direct or indirect, between the cache and the client, such that the client can retrieve a data from that cache. In part (a) of this example, at each transmission time, each client can get at most two different data from the two caches it is connected to. Also, the costs of all the connections/links in this example are the same. Without storing encoded data, the best we can do is to give two clients their two most requested data, by storing the most probable data in two of the caches, and the second most probable one in the other cache. For example if a and b are the first and second most probable data respectively, then storing a at C_1 and C_3, and storing b at C_2 gives R_1 and R_2 access to both a and b, but R_3 would have access to only a, and needs to get b from the server, at a higher cost. But after considering the possibility of storing encoded data in the caches, by storing a at C_1 and b at C_2 and a randomly linear encoded version of a and b (shown as $\{a, b\}$) at C_3, all three clients will have access to both a and b. R_1 gets a and b from C_1 and C_2 respectively. R_2 gets b from C_2 and decodes a by getting $\{a, b\}$ from C_3 and b from C_2. R_3 has a connection to C_1, so it can get a from it, and decodes b by getting a, b from C_3 and a from C_1. $\{a, b\}$ means any random linear combination of a and b, $k_1a + k_2b$, where k_1 and k_2 are some coefficients from Galois field(t); the larger the t, the higher the probability of linear independency between different random linear combinations of a and b, which becomes important in implementations of network coding in practice.

Figure 4.2b shows another example where storing an encoded data can be beneficial. Assume that R_1 and R_2 are interested in the same set of data; a and b are the most frequently requested data for both. But the first most probable data for R_1 is a, but for R_2 it is b. The size of all three caches,



(a) (b) (c)

Figure 4.2: Examples of structures in which network coding across files can be helpful

C_1 , C_2 and C_3, is 1. The cost of getting a data unit from C_1 is more than the cost of getting a data unit from the closer caches, C_2 and C_3, for R_1 and R_2 respectively. Also it is assumed that C_2 and C_3 cannot request any data from each other, but only From C_1. Considering all these assumptions, the best way of storing data in the caches is to store a in C_2, b in C_3 and $\{a, b\}$ in C_1. R_1 and R_2 will get a and b from C_2 and C_3, respectively. They both get $\{a, b\}$ from C_1, and decode it given a and b, to recover b and a respectively.

It was mentioned before that encoding chunks of different files can improve the performance. In the first two examples in Figure 4.2 we worked with the whole content and did not consider the chunks. The idea is similar for chunks, but to make it more clear it is explained in a separate example. In Figure 4.2c, it is shown how one can take advantage of combining chunks of different files, and if it is not allowed, the performance will be degraded. Assume file a is divided into two chunks a_1 and a_2, and file b is divided into b_1 and b_2. Also, assume we have the same assumptions as part (b) of Figure 4.2. The optimum solution in this case is to store a_1 and a_2 in C_2, b_1 and b_2 in C_3, and $\{a_1, b_1\}$ and $\{a_2, b_2\}$ in C_1. If combining chunks of different files is not allowed, then it is clear that the solution will not be optimum.

4.2 Minimizing Cost in Content Placement Problem

In this section, we propose a novel solution to the data placement problem, that instead of only considering storing the original videos, it also considers the possibility of cross encoding some videos in some caches if it is helpful, as was shown in few examples earlier in this chapter. We start by modeling the whole problem as a binary linear model, where the objective is minimizing the average cost of the video delivery to the clients. We call this solution *Caching and Coding based on Integer Programming (CCIP)*. The output of CCIP determines what should be stored in each cache, and how they should be stored, assuming that the caches can store encoded or non-encoded (original) chunks of data. It also tells where each client should retrieve each content from. CCIP method finds the best content placement solution that minimize the average cost, under the given cost and demand statistics conditions. We also model the same problem where only original content can be stored in the caches for comparison purpose, and we call it *CIP, Caching based on Integer Programming*. CIP (with no encoding), clarifies the importance of encoding across the videos in the caches. By comparing the performance of CCIP with CIP, one can say how helpful caching encoded data is. In order to scale our CCIP solution to larger networks, we extend CCIP to *Clustered Caching and Coding based on Integer Programming (C-CCIP)*. We also describe an algorithm for comparison purpose with the CCIP for large networks, which is based on Greedy algorithm and does not consider caching encoded data, in order to show the advantage of storing cross encoded data in large networks.

4.2.1 Formulating the problem as a binary integer linear program

In this section we introduce our proposed caching solution, Caching and Coding based on Integer Programming (CCIP), which considers not only storing original chunks of videos, but also cross encoded chunks of different videos as well. The solution is modeled as a binary integer linear program, with the objective of minimizing the average delivery cost. The inputs of our integer program are the average delivery costs from each router in the network to the clients, the probability array of different videos or in other words the demand statistics of the clients, and also

the cache sizes. The output of our integer program determines what should be stored at each cache, and where each client should retrieve each video from, in order to minimize the average cost of delivering the requested videos by clients.

CCIP objective function is described in equation (4.1), followed by the list of constraints. Before explaining the model, lets define the parameters and variables of our model. We show the total number of videos by K, total number of caches by J, and total number of clients by I. Size of cache j is shown by s_j. The request probability of video k by client i is shown by p_{ik}. The cost of retrieving one unit of content from cache j by client i is shown by c_{ij}. The model has four sets of binary variables, x_{kj} , z_{kj} , $y_{kk'j}$ and u_{ikj}. x_{kj} is 1 if video k is stored in cache j, 0 otherwise. z_{kj} is 1 if video k is stored as an encoded chunk in cache j, 0 otherwise. If $y_{kk'j} = 1$ it means video k and k are encoded together in cache j. u_{ikj} tells which cache(s) client i should receive video k from . Table 4.1 summarizes the list of the parameters and variables in our model, with a short description of each. Each k in our model can be one movie, or a chunk of a movie. We assume that the size of each video k is one unit, and we measure the caches capacity in the same unit.

The binary integer program for the case with coding, CCIP algorithm, is as follows:

Maximize $\sum_{i=1:I} \sum_{k=1:K} \sum_{j=1:J+1} p_{ik} c_{ij} u_{ikj}$ (4.1)

Subject to:

1. $z_{kj} \leq x_{kj}$ $\quad \forall k \in \{1, ..., K\},\ \forall j \in \{1, ..., J\}$
2. $\sum_{k'=1:K} y_{kk'j} \geq z_{kj}$ $\quad \forall k \in \{1, ..., K\}, \forall j \in \{1, ..., J\}$
3. $\sum_{k=1:K} x_{kj} - 0.5 z_{kj} \leq s_j$ $\quad \forall j \in \{1, ..., J\}$
4. $\sum_{j=1:J} u_{ikj} \leq 2$ $\quad \forall i \in \{1, ..., I\}, \forall k \in \{1, ..., K\}$
5. $\sum_{j=1:J+1} u_{ikj} \geq 1$ $\quad \forall i \in \{1, ..., I\}, \forall k \in \{1, ..., K\}$
6. $\sum_{i=1:I} u_{ikj} \geq x_{kj}$ $\quad \forall k \in \{1, ..., K\}, \forall j \in \{1, ..., J+1\}$
7. $\sum_{j'=1:J+1} u_{ikj'} \leq 3 - u_{ikj} - x_{kj} + z_{kj}$ $\quad \forall i \in \{1, ..., I\}, \forall k \in \{1, ..., K\}, \forall j \in \{1, ..., J\}$
8. $x_{kj} \geq 1 + u_{ikj} - \sum_{j'=1:J+1} u_{ikj'}$ $\quad \forall i \in \{1, ..., I\}, \forall k \in \{1, ..., K\}, \forall j \in \{1, ..., J\}$
9. $z_{kj} + z_{kj'} \leq 3 - u_{ikj} - u_{ikj'}$

   $- z_{kj} - z_{kj'} \leq 1 - u_{ikj} - u_{ikj'}$

   $x_{kj} + x_{kj'} \leq 3 - u_{ikj} - u_{ikj'}$

$$-x_{kj} - x_{kj'} \leq 1 - u_{ikj} - u_{ikj'} \qquad \forall i \in \{1,...,I\}, \forall k \in \{1,...,K\}, \forall j, j' \in \{1,...,J\}$$

10. $x_{k'j'} \geq y_{kk'j} + u_{ikj} + u_{ikj'} - 2$

$$z_{k'j'} \leq 3 - y_{kk'j} - u_{ikj} - u_{ikj'} \qquad \forall i \in \{1,...,I\}, \forall k, k' \in \{1,...,K\}, \forall j, j' \in \{1,...,J\}$$

11. $\sum_{j'=1:J} u_{ikj'} \geq 2 - 2(2 - y_{kk'j} - u_{ikj})$

$$\sum_{j'=1:J} u_{ik'j'} \geq 2 - 2(2 - y_{kk'j} - u_{ik'j}) \qquad \forall i \in \{1,...,I\}, \forall k, k' \in \{1,...,K\}, \forall j \in \{1,...,J\}$$

$x_{kj} \in \{0,1\}; x_{k(J+1)} = 1$

$z_{kj} \in \{0,1\}; z_{k(J+1)} = 0$

$u_{ikj} \in \{0,1\}$

$y_{kk'j} \in \{0,1\}; y_{kkj} = 0; y_{kk'(J+1)} = 0; y_{kk'j} = y_{k'kj}$

Note 1: The unit of cache size and video size are the same.

Note 2: CDN server is considered as $(J+1)^{th}$ cache to simplify the formula, and all videos are stored in the CDN server.

Here is a brief explanation of the constraints of our CCIP optimization formula :

Constraint 1 ensures that video k is stored in cache j, if it is encoded there. Constraint 2 refers to the definition of $y_{kk'j}$ and checks that if video k and k' are encoded together in cache j, then the coding indicators for video k and k' in cache j are 1. Constraint 3 captures the capacity limit of each cache. Constraint 4 limits encoding to at most 2 chunks of videos. This constraint is added in order to limit the complexity of the model, and simplify the decoding process. Constraint 5 ensures that for each client i and video k, content delivery is assigned to at least one cache. Constraint 6 checks that each stored video, encoded or non-coded, is used by at least one client. Constraint 7 confirms that if video k is encoded in cache j, then there is at least one video k' which is coded with k. Constraint 8 assures if client i gets video k from cache j non-coded, then video k is in cache j. The set of constraints in 9 ensure that if client i uses caches j and j' to decode video k, then video k is only in one cache, and it is encoded with another video in that cache. The pairs of constraints in 10 verify that if client i decodes video k from caches j and j', and k and k' are combined in cache j, then k' should be stored in cache j', non-coded. Finally constraint 11 ensures

Table 4.1: CCIP Parameters Definition

I	Total number of the clients
K	Total number of the videos
J	Total number of caches
s_j	Size of cache j
p_{ik}	Probability that client i asks for video k
c_{ij}	Cost for client i to get one video unit from cache j
x_{kj}	Storage variable; 1 if video k is stored in cache j; 0 otherwise
z_{kj}	Coding indicator variable; 1 if video k is stored coded with another video in cache j; 0 otherwise
u_{ikj}	Delivery variable; 1 if client i gets video k from cache j; 0 otherwise.
$y_{kk'j}$	Coding variable; 1 if k is coded with k' and stored in cache j; 0 otherwise. $y_{kkj} = 0$

that if k and k' are encoded in cache j, and client i asks for video k from cache j, another cache is needed to decode it as well.

The integer program for the case without network coding, CIP algorithm, is as follows:

Minimize $\sum_{i=1:I} \sum_{k=1:K} \sum_{j=1:J+1} p_{ik} c_{ij} u_{ikj}$ (4.2)

Subject to:

1. $\sum_{k=1:K} x_{kj} = s_j$ $\quad \forall j \in \{1, ..., J\}$
2. $\sum_{j=1:J+1} u_{ikj} = 1$ $\quad \forall i \in \{1, ..., I\}, \forall k \in \{1, ..., K\}$
3. $\sum_{i=1:I} u_{ikj} \geq x_{kj}$ $\quad \forall j \in \{1, ..., J\}, \forall k \in \{1, ..., K\}$
4. $u_{ikj} \leq x_{kj}$ $\quad \forall i \in \{1, ..., I\}, \forall k \in \{1, ..., K\}, \forall j \in \{1, ..., J\}$

$x_{kj} \in \{0, 1\}; x_{k(J+1)} = 1$ $\quad \forall k \in \{1, ..., K\}, j \in \{1, ..., J\}$

$u_{ikj} \in \{0, 1\}$ $\quad \forall i \in \{1, ..., I\}, k \in \{1, ..., K\}, j \in \{1, ..., J\}$

For the definition of the parameters and variables in equation (4.2), please refer to table 4.1. In equation(4.2) there is no coding variable z_{kj} and $y_{kk'j}$ any more, because there is no encoding in this case. Constraint 1 captures the capacity limit of each cache j. Constraint 2 ensures that for each client and each video content, there is exactly one optimum location that the client should get that content from. Constraint 3 reflects the fact that if a video is stored in a cache, there should be at least one client who will get that video from that cache. And finally constraint 4 is a direct result of the definition of the variables x_{kj} and u_{ikj}. A client can retrieve a video from a cache, only if that video is stored in that cache. Both optimization problems in equation (4.1) and (4.2) are binary integer linear programs and Gurobi optimizer [41] is used to solve them.

4.2.2 Clustered CCIP (C-CCIP)

The CCIP model explained in the previous section provides the optimum solution to our problem of caching cross encoded videos, but it is not possible to apply it on large networks because of its computational complexity. In order to scale our CCIP solution to larger networks, clustered CCIP (C-CCIP), is proposed. In C-CCIP we partition the network to smaller networks and apply CCIP solution on these small networks separately. The order in which these clusters should be

solved matters, because the solutions of the clusters closer to the clients are the input for the farther ones. The details of the C-CCIP method are explained in the following steps, assuming that the network is a Directed Acyclic Graph (DAG):

1. Partition the network to some clusters with the following conditions: (i) Each cluster should be as large as possible to obtain the most gain from encoding, but small enough for CCIP algorithm. (ii) Each cluster should be a connected network. (iii) If there is a link from cluster A to cluster B, all the vertices in cluster B should have higher index than any vertices in cluster A, assuming the topological ordering in DAG, and assuming that the vertices (routers) connected to the clients are the ones with highest indices.

   Apply steps 2, 3 and 4 on all the clusters, in order of their indices, starting from the ones closet to the clients

2. Apply CCIP algorithm on the current lowest layer cluster. Lowest layer clusters are those farthest from the CDN server and closest to the clients.

3. Calculate the remained probability array of the solved cluster, considering the stored data in the caches of that cluster. To calculate the remained probability array, delete those data that can be provided by the current cluster for each client, from the probability array of that client. (A later example will make this step more clear.)

4. Consider each lower layer cluster as a single client for the higher layer cache(s) connected to it. Add the remained probability array of the lower layer cluster(s), to the probability array of the clients of the cache(s) in the higher layer connected to that lower cluster. Then normalize the result, so the sum equals 1.

To make the procedure more clear, consider Figure 4.3, which is a network of router caches and clients. This is a DAG network, and caches are shown by C and clients are shown by R, with different indices. The Cache at the top layer of the network, C_{41} in this case, is the one closest to the CDN server and has the smallest index in the DAG. Caches in the lowest layers are those

farthest from the server and closest to the clients, with largest indices in the DAG. C-CCIP steps for Figure 4.3 network are as follows:

1. First partition the network to clusters, considering the conditions mentioned before. An example of cluster selection is shown in Figure 4.3b.

2. Apply CCIP algorithm on clusters G_1, G_2 and G_3, the clusters with largest index nodes. None of these clusters need any information from any other clusters, because there is no cluster with larger index nodes connected to them.

3. Consider cluster G_1. Assume that the probability arrays of the most requested videos for R_{11}, R_{12}, R_{13} and R_{21} are P_1, P_2, P_3 and P_4 respectively. To calculate the remained probability array of G_1, delete those videos in the probability arrays of the clients in cluster G_1 that will be provided to each client by G_1, and sum up the probabilities of the same videos in all arrays. This is the remained probability array of G_1. For example, assume that the probability arrays of R_{11}, R_{12}, R_{13} and R_{21}, for data set a, b, c, d, e, are $P_1 = [0.3, 0.1, 0.2, 0.25, 0.15]$, $P_2 = [0.4, 0.15, 0.1, 0.25, 0.2]$, $P_3 = [0.1, 0.3, 0.2, 0.15, 0.25]$ and $P_4 = [0.2, 0.1, 0.2, 0.25, 0.25]$, respectively. Assume the available videos to R_{11}, R_{12}, R_{13} and R_{21}, from the caches in G_1, are a, d, a, d, a, b, e and a, e respectively. To calculate the new probability arrays, first delete the videos available to each of the client from G_1, from their probability arrays. So we will have: $P'_1 = [0, 0.1, 0.2, 0, 0.15]$, $P'_2 = [0, 0.15, 0.1, 0, 0.2]$, $P'_3 = [0, 0, 0.2, 0.15, 0]$ and $P'_4 = [0, 0.1, 0.2, 0.25, 0]$. In the next step, we will add up P'_1, P'_2, P'_3 and P'_4 by summing up the probabilities of the same videos. Let's show the result by P_{G_1}; $P_{G_1} = [0, 0.35, 0.7, 0.4, 0.35]$ in this case. Calculate the remained probability array of G_2 and G_3 in the same way. Notice that the sum of the elements in the remained probability array, e.g. P_{G_1}, is not necessarily 1.

4. Add the remained probability array of G_i, to the probability of the direct client (or other indirect clients) of the cache G_i is connected to; then normalize the result so the sum equals 1. In our example, the cache that G_1 is connected to is C_{31} and its direct client is R_{31}. C_{31} is connected to cluster G_2 as well. Assume that the probability array of R_{31} is $P_4 =$

$[0.35, 0.2, 0.1, 0.25, 0.1]$. Also assume the remained probability array of P_{G_2} is the same as P_{G_1} in our example. Then, normalized($P_{G_1} + P_{G_2} + P_4$) = normalized([0.35, 0.90, 1.50, 1.05, 0.80]) = [0.076, 0.196, 0.326, 0.228, 0.174]. Calculate the new probability that cache C_{32} sees in the same way.

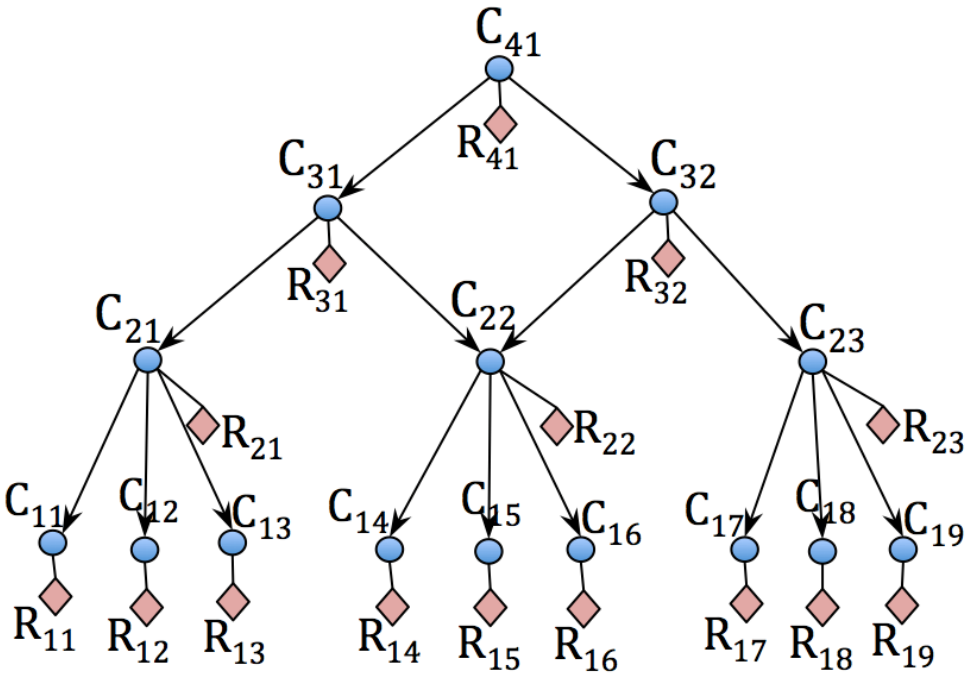
5. Use the new calculated probability arrays and apply CCIP method on cluster G_4.

Although deciding how to partition the network is not always straightforward and it can be sometimes challenging, but the selected clusters will be valid while the network structure remains the same. In addition, small changes to the network structure would affect only a few clusters close to the change point, and not the others. So the cluster decision complexity can be considered as a one-time complexity and wont be a problem.

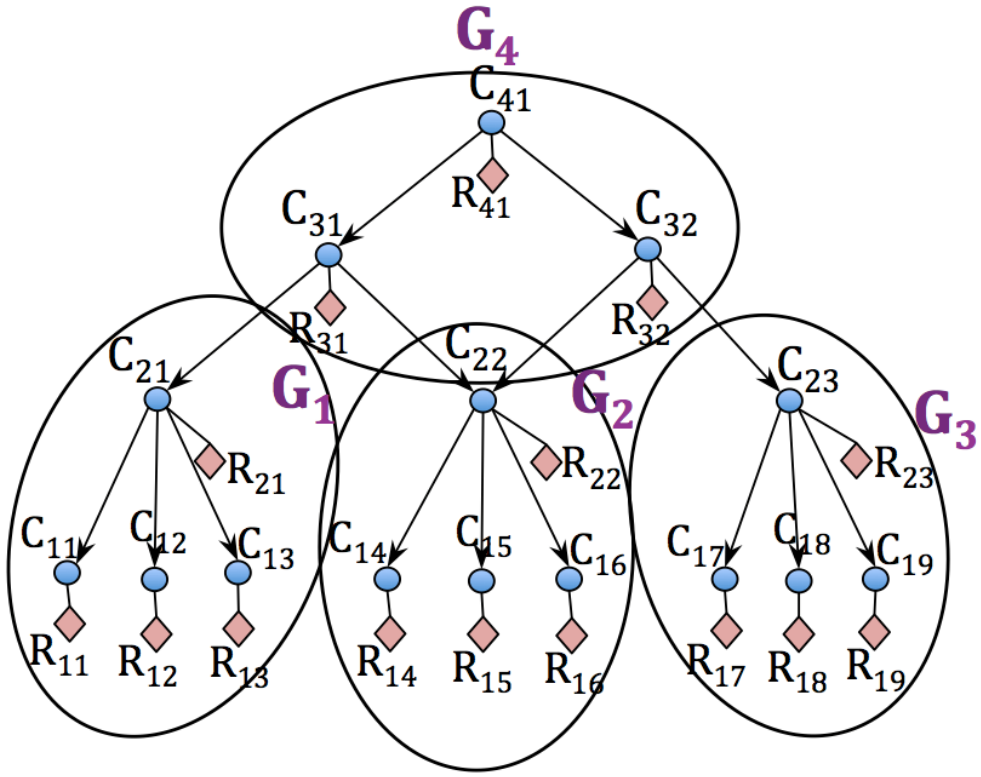
4.2.3 Greedy Caching (GC)

In this section we describe a caching approach, based on greedy algorithm, which we use as a baseline of comparison with our proposed caching solution, CCIP and C-CCIP, explained before. This greedy based approach, like any Greedy algorithms, finds a locally optimum solution at each step [24]. In this algorithm, in order to fill the caches, one should consider the caches in the topological ordering of DAG, starting from the ones closest to the clients and with largest indices in the DAG. For the caches which are connected to the clients, it decides what to store in each of those caches to minimize the average cost, given the demand statistics of the clients connected to each (local statistics that each cache sees). For farther caches considering the topological ordering, one should decide what to store in the caches to minimize the average cost, given the demand statistics of each caches client, and the remained probability arrays of lower clients, calculated by deleting the stored video in the lower layer caches connected to it. The caches client probability array and the remained array should be summed and normalized to 1.

When a cache is connected to multiple caches from the layers above, its probability array will be considered in calculating the probability array of the most demanded data for both of those



(a) A network of router caches and clients



(b) Partitioning the network in part (a) to clusters in C-CCIP

Figure 4.3: Clustered CCIP example

caches. For example, nodes C_{31} and C_{32} in Figure 4.3 which are connected to C_{22}, will both consider C_{22} remained probability when calculating their new probability array.

To better understand how GC algorithm works, we explain its behavior in the example in Figure 4.3: The lowest layer caches, C_{1j}s, will look at the probability array of the client(s) connected to each of them. For example C_{11} only considers the average probability array of R_{11}, and stores the most probable videos of this client(s) until it is completely filled up. All the other C_{1j} caches will do the same and only consider the probability array of R_{1j}. Assume the probabilities of the clients are the same as the example for C-CCIP. So $P_1 = [0.3, 0.1, 0.2, 0.25, 0.15]$, $P_2 = [0.4, 0.15, 0.1, 0.25, 0.2]$ and $P_3 = [0.1, 0.3, 0.2, 0.15, 0.25]$ for video set a, b, c, d, e for R_{11}, R_{12} and R_{13}, respectively. So the cached video in C_{11} , C_{12} and C_{13} would be a, a and b respectively. GC algorithm then considers the next largest index caches considering the topological ordering in DAG (in this example C_{21}, C_{22} and C_{23}.) Consider C_{21}: It is connected to C_{11}, C_{12} and C_{13} which have larger indices and were filled up earlier. Update each probability array P_j, by deleting those videos already stored in C_{1j}. So we will have $P'_1 = [0, 0.1, 0.2, 0.25, 0.15]$, $P'_2 = [0, 0.15, 0.1, 0.25, 0.2]$ and $P'_3 = [0.1, 0, 0.2, 0.15, 0.25]$, where P'_j s are the updated probability arrays. C_{21} will sum up these new probability arrays P'_1, P'_2, and P'_3 in addition to the probability array of its direct client, R_{21}, $P_4 = [0.2, 0.1, 0.2, 0.25, 0.25]$, and then normalize it to sum to 1. This new average probability for C_{21} is equal to normalized($[0.3, 0.35, 0.7, 0.9, 0.85]$) = $[0.097, 0.113, 0.226, 0.290, 0.274]$. C_{21} stores the most important video in this new probability array, which is d. C_{22} and C_{23} will do similar calculations to decide what to cache. Apply the same method for all the caches starting from those with larger index, and update the probabilities according to the previously cached data in the lower caches with larger indices. To update the probability array of C_{31} and C_{32} after filling the lower layer caches, the updated probability of C_{22} will be considered in both of them.

The GC method is much less complex compared to our proposed C-CCIP solution explained earlier, but it may result in a bad performance. For example, assume the following conditions in Figure 4.4:

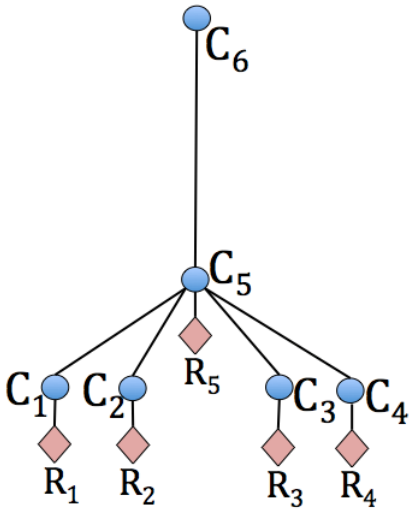


Figure 4.4: An example where GC algorithm results in very poor performance

(i) Each cache has a storage capacity of one unit; (ii) All clients shown as R_i have the same probability array of $[0.200, 0.199, 0.199, 0.199, 0.199]$ for their first five most requested videos which are a, b, c, d and e, respectively; (iii) R_1, R_2, R_3, R_4 and R_5 can receive their video from any cache, but the cost of receiving it from C_6 is much higher than the other caches for all clients.

By applying our GC algorithm, C_1, C_2, C_3, C_4 and C_5 all store a in their caches. But the better solution is to keep a, b, c, d and e at the closer caches, C_1, C_2, C_3, C_4 and C_5 and they should not all store the same video.

4.3 Simulation Results

First, we want to examine how much storing encoded chunks of different videos can be helpful in caching. We applied the proposed CCIP solution on some randomly generated networks with the properties mentioned later in this section, and took average on all of them to see how much gain can be achieved by storing encoded data in the caches, whenever helpful. The results of CCIP algorithm, which includes encoding, is compared to CIP, which is the optimum case when coding is not allowed.

In table 4.2, different network sizes are tested. If we show the size of a network as a pair of (# clients, # caches), the five examined networks have the following sizes: (3,4), (4,5), (5,5), (5,6) and (5,7). All caches are assumed to have 1 unit storage capacity. The connections in each case are made randomly to generate different chances to show the real usefulness of network coding. The connection probability between each receiver and each cache is considered as 60% in these randomly generated networks. The minimum and maximum of the delivery costs from each of the routers to the clients in the simulation are 0.05 and 1.00 respectively and the cost of requesting one data unit from the source is assumed to be 1.00. The demand probability arrays for the clients in this example are assumed to be the same, with a distribution generated randomly on the video set. The size of the video set that needs to be considered in our model is equal to the sum of the size of all caches, shown by K in our formulations, and they are the first K most probable videos in the whole video set. Considering any value larger than that for K will not make any difference in the result.

The comparison in table 4.2 shows an average gain of 16.1% for the largest network can be achieved if we consider storing encoded chunks in the caches, compared to the case without encoding. In general, and with the same assumption for this example, larger networks should have larger gain, because we will see the gain of small networks, plus having more opportunities to take advantage of encoding.

Another parameter that affects the performance of our algorithms is the demand probability array. The authors in [27] have shown that the popularity of WWW documents generally have a behavior similar to Zipf law, i.e., the relative access frequency for a document is inversely proportional to the rank of that document. The model of content popularity is very important and

Table 4.2: Gain of storing coded data in five different network sizes with random structures

Network size	**(3,4)**	**(4,5)**	**(5,5)**	**(5,6)**	**(5,7)**
Average Gain from Coding	1.6%	4.7%	5.5%	15.8%	16.1%
Percentage of the networks with gain	8.0%	24.0%	34.0%	82.0%	84.0%

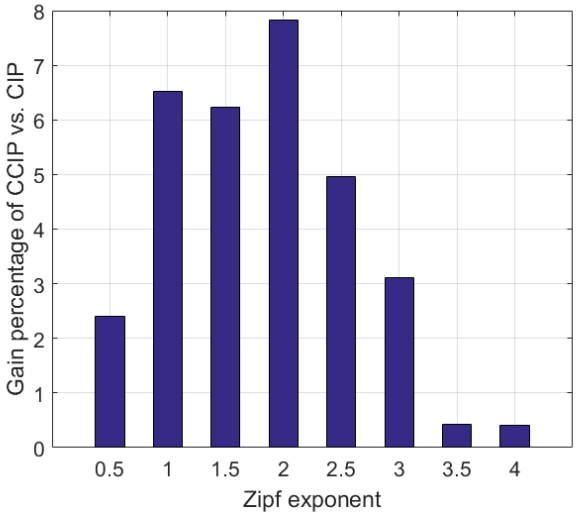


Figure 4.5: Percentage gain of CCIP vs. CIP for different exponents of Zipf distribution function

the impact of different models such as Zipf and other heavy-tail distributions is studied in several papers, such as [88], [43]. Multiple studies, e.g. [28], [40] have observed long tail properties in the popularity of videos as well. Therefore, we evaluate the performance of our algorithms with Zipf distribution assumption for the videos. In Figure 4.5 we compare the gain from caching encoded contents in a network of size (4,5) with Zipf distribution of different exponents. Each point in the plot is the average of 50 different structures (different connections and link costs, generated randomly.)

The simulation result shows the maximum gain of 7.82% for Zipf exponent of 2. By increasing or decreasing the exponent from 2, the gain decreases. For small exponents the distribution gets closer to a uniform distribution, and so the chance of taking advantage of coding is mostly from the structure, similar to the example in Figure 4.2a, and not the different interests of clients. For large exponents, only a few of the videos becomes important for all clients and so coding different videos cant help much.

We also introduced C-CCIP as a solution to the scalability problem with CCIP in large networks, and we explained GC algorithm for comparison purposed of C-CCIP. To evaluate the performance of C-CCIP method compared to GC, we apply them to the network in Figure 4.3, but

we are assuming that only the caches in the lowest layer have a direct client attached to them. For C-CCIP case, the subnets are the same as those in Figure 4.3b. There are four layers of connections in the network; we assume that the cost of the lowest layer links (those connected directly to the clients), is 0.05, and the cost of those on higher layers is 0.25, 0.5, and 0.75, respectively. The cost of receiving one data unit from the source is 1. Assume all caches have storage capacity of 1 unit.

The probability distribution of clients demands is a Zipf function with exponent 2, but each client has a random assignment of the probabilities on the contents. In our example, the probability array of clients R_{11}, R_{12} and R_{13} are $P_1 = [0.7024, 0.1756, 0.0780, 0.0439]$, $P_2 = [0.1756, 0.7024, 0.0780, 0.0439]$, $P_3 = [0.7024, 0.1756, 0.0780, 0.0439]$ on video set a, b, c, d, which is a Zipf distribution with exponent 2. R_{14}, R_{15} and R_{16} have the same probability distribution, P_1, P_2 and P_3 respectively, but on video set a, d, e, f. The probability array of R_{17}, R_{18} and R_{19} are also P_1, P_2 and P_3 respectively, but on video set g, h, i, j. We applied C-CCIP (with encoding) and GC algorithm (no encoding) on this network. The average cost of content delivery in our example network with GC solution is 0.61 and with C-CCIP solution is 0.27! C-CCIP improves the performance considerably compared to GC. The results of this example is summarized in table 4.3. CCIP algorithm clearly has the best performance among the proposed methods. But its complexity can be too much for large networks. C-CCIP seems to have a good performance-complexity trade-off.

Table 4.3: Gain of C-CCIP vs. GC for the network in Figure 4.3

Caching method	GC	C-CCIP
Average video delivery cost	0.61	0.27

4.3.1 Large-scale simulations

In this section, we simulate the performance of the proposed caching method in a larger network. In the model chapter we explained the details of the large scale simulations, and how the random DAGs are generated. We run the C-CCIP method on these random DAGs, and show the

Table 4.4: Cluster size impact in the gain of C-CCIP

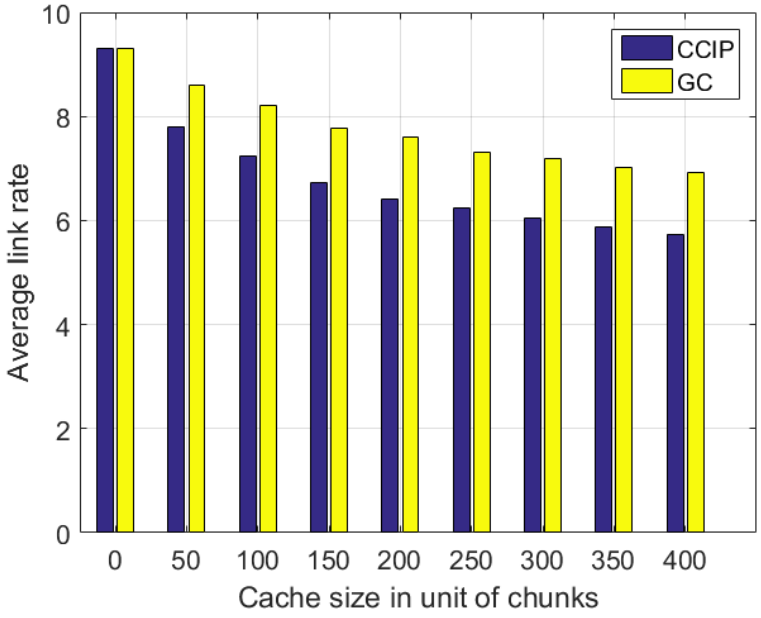
	1 cluster with 30 nodes	3 clusters each with 10 nodes	6 clusters each with 5 nodes
Cache size = 100 chunks	7.2	7.45	7.49
Cache size = 200 chunks	6.4	6.51	6.60

average results and compare it with the result from GC algorithm. For the simulations in this section, the delivery cost of one unit of data from a router to a client is set equal to the minimum hop distance between them. The reason behind this choice of cost is that in the next chapter where we propose our short-term caching solution, we will work with the objective of minimizing the average link rate in the network. Therefore, by setting up the cost equal to minimum hop distance, the objective in these simulations will match with the objective in later chapters. It becomes specifically important in chapter 6 where we study joint long and short-term caching, and the joint objective is to minimize the average link rate in the network.

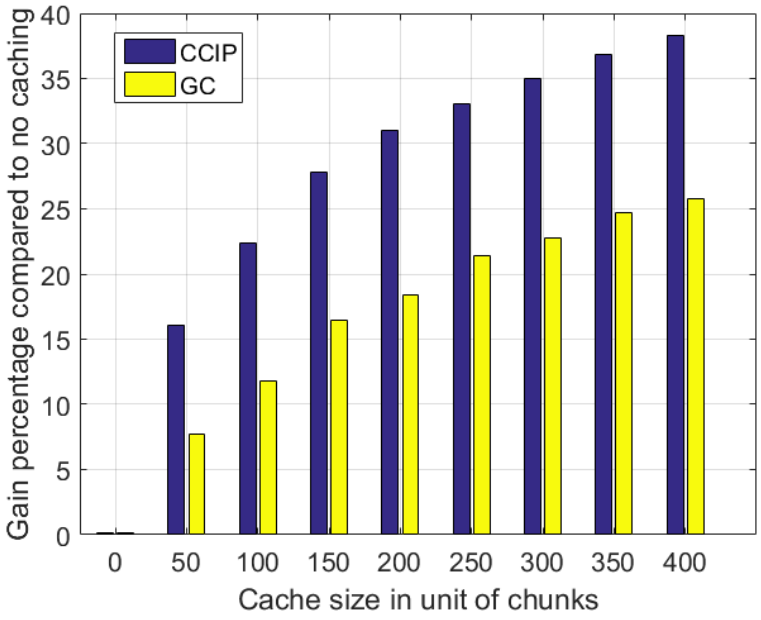
First, lets look at the impact of cluster size on the gain of C-CCIP. In table z we show the average link rate for three different cluster sizes, 1 cluster of size 30 nodes, 3 clusters each of size 10 nodes, and 6 clusters each with 5 nodes. The Zipf exponent is 0.9, number of videos is 10000, and two different values of cache sizes, 100 and 200 chunks (1 and 2 videos) is considered. We used a simple intuitive method to do the clustering by considering the topological ordering in DAGs and grouping the nodes with close orders in one cluster. Assuming that the CDN server (source) is labeled as the first node 1, and the rest according to their topological connections, we consider nodes 1, 2, ..., 10 in one cluster, nodes 11, 12, ..., 20 in another cluster, and nodes 21, 22, ..., 30 in another one. This simple clustering approach is selected since for random generated DAGs it is hard to control the clustering of each DAG separately. But given a specific network structure, we should be able to do a better clustering, and since it is a one-time complexity, it won't be a problem.

The result in table 4.4 shows only a small performance degradation as a result of having more clusters of smaller sizes. It means that even with our simple clustering solution, we are still seeing a performance close to the optimum solution. Therefore, we can conclude that C-CCIP clustering approach is an appropriate solution to the scalability issue of CCIP algorithm. In the rest of the simulations in this section, for each randomly generated DAG, we consider 3 clusters of size 10 nodes, using the same method explained before.

In Figure 4.6a, we show the delivery cost of C-CCIP and GC when Zipf exponent is 0.9, but cache size is a variable. Figure 4.6b shows the gain compared to no caching for each case. As expected, we can see that the impact of cache size on the performance decreases, by increase in the cache size. C-CCIP method is showing a considerable gain compared to GC algorithm. For example, at cache size of 100, the gain of C-CCIP and GC are 22.5% and 12.1%, respectively. The gain at cache size 200 chunks for C-CCIP and GC are 31.9% and 18.1% respectively.



(a) Average video delivery cost



(b) Average gain compared to no caching

Figure 4.6: Clustered CCIP vs GC algorithm

Chapter 5

SDN-Enabled Caching Algorithm for Staggered Multicasts

The focus of this chapter is the short-term caching part of the router caches, which was shown in Figure 2.1 in chapter 2 Specifically, in this chapter we would like to find an affective router caching algorithm for staggered video multicasts [71]. We start by describing the problem with an example, and then explain our algorithm, SCS caching. After that we show through simulations and implementation in GENI the effectiveness of our approach, and its superiority compared to traditional caching algorithms, LRU and LFU. The simulation results for large DAG networks are added after initial investigation and proof of concept for smaller networks.

5.1 Problem Definition and Assumptions

In Figure 2.1 in chapter 2 we showed the network of router caches. We divided the router caches to long-term caching and short-term caching portions, where in this chapter we will focus on the long-term caching part of the router caches in the network. We also explained that there is a SDN controller, with logical connection to all SDN-enabled routers, is responsible for routing and caching decisions at all routers. The different parts of the SDN controller that are assumed in this thesis were shown in the problem model in chapter 2 in Figure 2.2. Here, in order to clarify and focus only on the parts related to short-term caching, we want to explain the parts of the SDN controller that are related and required for the short-term caching decisions at router caches. Figure 5.1 shows the blocks of SDN controller that are being used for short-term caching in this chapter. It consists of four main parts: network topology database, flow table database, caching table database, and computing center. Network topology database is responsible for keeping track

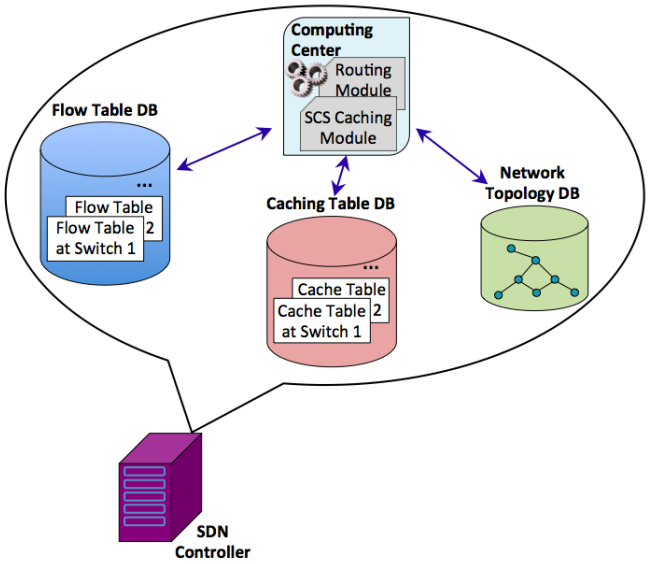


Figure 5.1: SDN controller components for short-term caching

of any change in the topology; for example, if a link is broken, the topology database should be able to quickly update the topology, so that routing and caching module will be updated accordingly. Flow table database keeps track of all the ongoing forwarding decisions that have been made per flow; Caching table database keeps track of the ongoing caching decisions at all the routers. In this chapter, we focus on the SCS caching module at the SDN controller. This module runs the proposed algorithm. We assume that the caching decisions are made after the SDN controller has made its routing decisions. Please notice that in Figure 2.2 in the model chapter, we had a block for network statistics calculations, which is not shown in Figure 5.1. The reason is the statistical information is only being used for long-term caching algorithm ,and as you will see in this chapter, short-term caching solution is only based on the current flows in the network. The problem we are trying to solve in this chapter and the intuition behind it is as follows: considering the dominance of video content in the network, and the Zipf probability of the videos, many clients are watching the time-shifted version of same video at any given time. Our goal is to find an efficient router caching algorithm that uses this property to reduce links bandwidth usage in the network.

A set of staggered requests for two different videos is shown in Figure. 5.2. To simplify the figure, we only showed one CDN server in the figure. But our model considers the general case

when requested videos of different clients may come from different servers. Assume that all videos have same length and rate, and each video is divided into chunks of equal sizes, e.g., chunks of size 10 seconds, and the clients make requests for chunks. The cache sizes are also given in units of chunks; for instance in this example each router has a caching capacity of 20 chunks, while each video consists of 100 chunks. A client requesting for chunk j of video i is shown as (i, j). The dashed lines show the routing decisions for each flow. Given the routes of each flow, the SDN controller decides what should be stored in each router cache to minimize the average link rate in the network.

An optimum caching solution (formulated later in this chapter) that results in the minimum average link rates is shown in Figure. 5.2. Stream of client i, called flow i, is shown as f_i, and the notation $f_i \rightarrow f_k$ indicates that flow i services flow k at a cache by storing an appropriate number of latest chunks of f_i and meeting the delivery needs of f_k from its cache, thereby eliminating the need for forwarding those contents from the source. For instance, at router cache C_3, $f_{10} \rightarrow f_8$ means that f_{10}, which is currently at chunk 30 of video 1, can service f_8, which is currently at chunk 27 of the same video, by caching the three most recent chunks of f_{10}. By doing so, the rate on the link between C_6 and C_3 decreases by 25%. Other caching decisions are also shown in the figure. This solution results in 50% saving in the bandwidth requirement of the link between the server and C_7.

SDN controller makes a new caching decision when the conditions in the network change either due to arrival or termination of a flow or due to changes in the routes of existing flows. The changes in the caching decisions are implemented over the control plane of the network.

Redoing the example with LRU caching strategy at the router caches, we observe that only one flow in the example, namely f_8 at C_3, is helped by another flow f_{10}. All the other flows must come directly from the server. Consequently, instead of the 50% bandwidth saving of the optimal caching scheme shown above, LRU has only a 10% saving. The smaller reduction for LRU is no surprise. LRU treats all the most recently requested chunks equally and it does not exploit the knowledge of flows currently in the network. In contrast, the optimal strategy and our proposed strategy cleverly select the streams to be cached, resulting in more reduction in the average link rate. In the

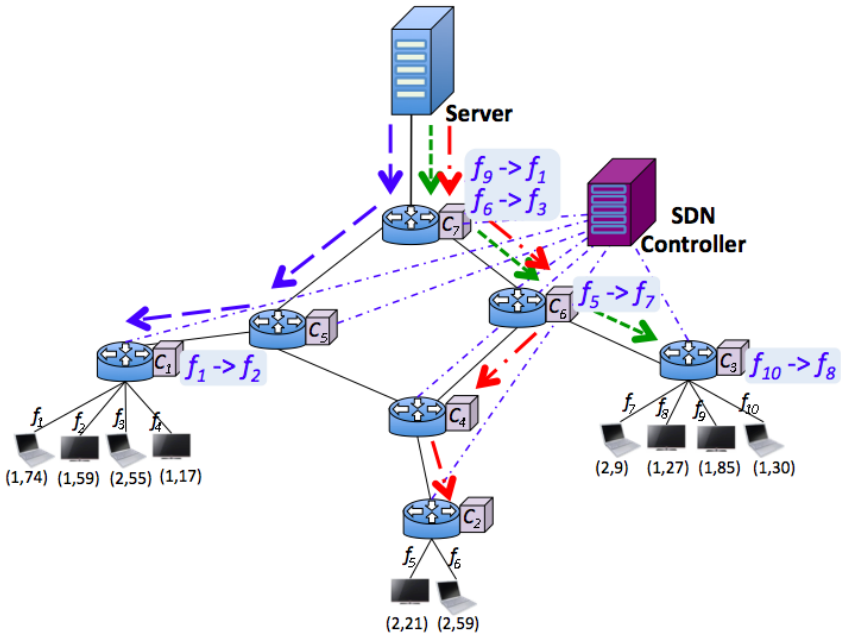


Figure 5.2: An example of staggered requests of videos and caching

evaluation section, we will show that our caching strategy largely outperforms traditional caching methods, LRU and LFU, for staggered video multicasts in a network of router caches.

5.2 SCS Caching Algorithm

In this section, we propose a router caching strategy, called *SDN-enabled Caching of Staggered videos (SCS)*. SCS takes advantage of the global view that SDN controller provides. However, it can be implemented by the SDN controller in real-time. The lower runtime complexity for SCS is achieved by individually solving a simple weighted Knapsack problem at each router. In particular, SDN controller first solves the caching for the routers closer to the clients, and then solves the farther caches. In farther caches, the SDN controller considers the chunks that are already stored in the caches closer to the clients, to prevent redundant caching. As a result of this global view of the SDN controller, the caching performance is significantly improved, while the runtime complexity remains close to that of a completely distributed solution. An empirical evaluation shows that its performance is also very close to that of an optimal algorithm.

Let $\mathcal{L}$ be the set of all links in the network. Let $R_{(u,v)}$ denote the rate of link that connects router u to router v. Then, the objective at router v is to minimize the rate of the incoming links to v, $Minimize \sum_{u:(u,v)\in\mathcal{L}} R_{(u,v)}$, which is equivalent to maximizing the cache hit rate at v. Let $b_i(t)$ denote the chunk number requested by flow f_i at time t. Define a set of binary variables x^r_{ik}, to indicate which flows are stored in each router cache;. For example, $x^r_{ik} = 1$ means f_i services f_k by caching the last $(b_i(t) - b_k(t))$ chunks of f_i in cache of router r. The SDN controller solves the following optimization problem, equation(5.1), for each router.

$$\textbf{Maximize} \quad \sum_{i,k\in V_r} x^r_{ik}$$

$$\textbf{Subject to:}$$

$$\sum_{i,k\in V_r} (b_i(t) - b_k(t))x^r_{ik} \leq S_r, \tag{5.1}$$

where S_r is the storage capacity of cache at router r and V_r is the set of flows forwarded through router r, after considering caching decisions. That is, if a flow is cached at a router closer to the client than r, it will not be included in V_r. Clearly, x^r_{ik} needs to be defined only if f_i and f_k correspond to the same video, and if stream f_i is ahead of f_k, because otherwise the value of x^r_{ik} will be 0.

Optimization problem (5.1) is a *0-1 Knapsack problem*, where the values of items are the same, and the size of each item, x^r_{ik}, is $b_i(t) - b_k(t)$. The optimum solution to this Knapsack problem is to select the smallest items first, until the capacity constraint prevents additional selection. To solve Equation(5.1), first sort the streams for each video according to their chunk number, and then calculate the distance between the chunk numbers of successive streams in each group. The list of calculated distances is the same as the list of weights in Knapsack problem. Second, start with the smallest distances to fill the cache. It can be easily shown that the complexity of solving SCS caching is $O(|\mathcal{C}||\mathcal{F}|log(|\mathcal{F}|)$, where $|\mathcal{C}|$ is the number of routers with cache, and $|\mathcal{F}|$ is the total number of ongoing streams at the time of new request arrival in the network. Here we assumed that every time the SCS module is called, the sorting is being done from scratch. One can improve the performance by keeping a sorted list of the distance between the flows who have that node on

their path towards the source. If a new flow is routed through the node, few additional entries have to be inserted into a sorted list, at complexity of $log(n)$, where n is total number of flows through that node. Removing a flow from a sorted list also has complexity of $log(n)$. This decreases the complexity of SCS module to $O(|\mathcal{C}|log(|\mathcal{F}|)$.

The following example illustrates how SCS module at SDN controller works:

Example: Figure 5.3a is a snapshot of a network with a set of staggered requests for two different videos; similar to the example in Section 5.1. The routing for the current flows are shown as dashed lines. The caching decisions in SCS for this snapshot are shown next to each cache. For example, at C_1 we have $f_1 \rightarrow f_2$ which means the last 15 chunks of f_1, (1,74), is being cached at C_1 to help f_2, (1,59), so that f_2 can come directly from C_1. SDN controller keeps all these caching decisions in its caching table database.

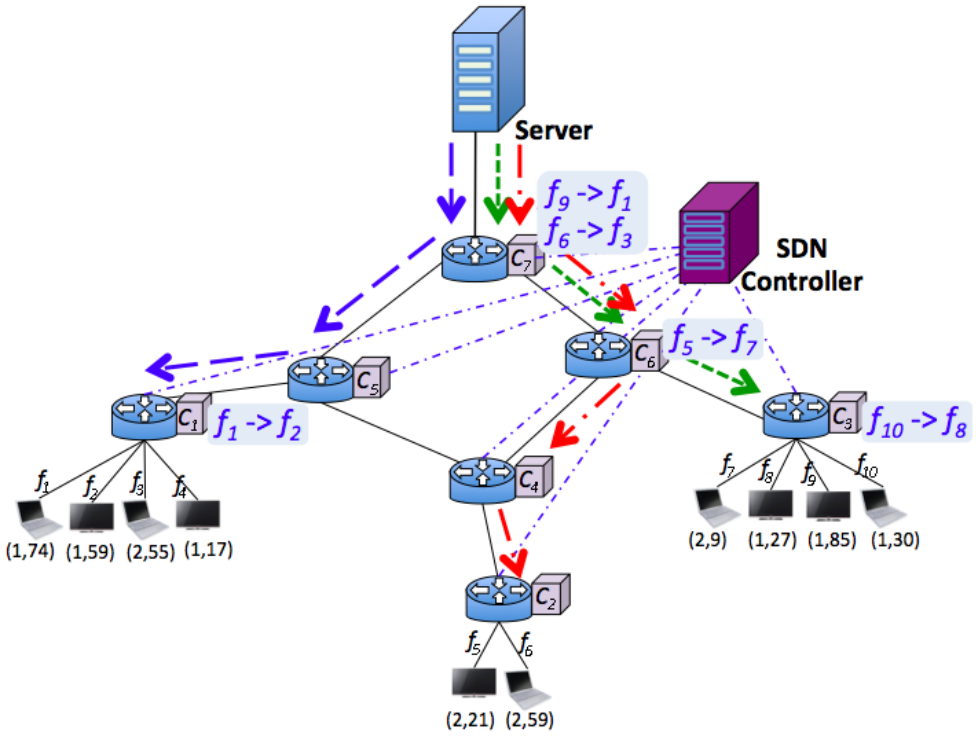
One time step (one chunk) after the one in Figure 5.3a, a new request f_{11}, asking for (1,1), which means the first chunk of video 1, arrives at C_1. The following are all the steps that will be taken by the SDN controller to update the caching solution to accommodate this new request (We call the routers by their cache label for simplicity):

1. Since there is no a matching entry for f_{11} in the flow table at C_1, it sends the request to the SDN controller.

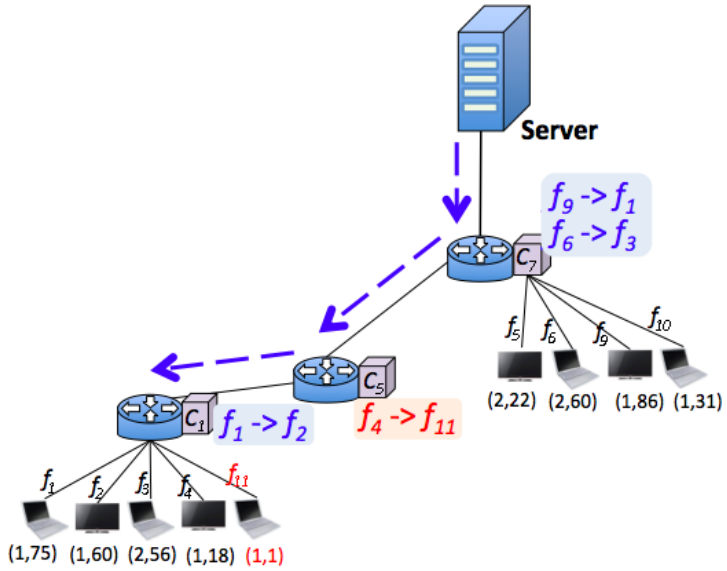
2. The controller calculates a route for f_{11} from the source to C_1, namely $Source \rightarrow C_7 \rightarrow C_5 \rightarrow C_1$. It sends the new decision to C_1, C_5 and C_7 and new entires are added to corresponding flow tables.

3. The controller considers the subset consists of only those routers which are part of the route assigned to the new flow, C_1, C_5 and C_7, and all the flows that reaches these routers, after removing those flows that are cached in the other routers with larger indices, considering the topological ordering in DAG (Figure 5.3b); then it applies the SCS module on the subset (steps 4 to 6).

   SCS algorithm:



(a) An example of staggered requests of videos and caching



(b) The subset of our example with the corresponding routers and flows

Figure 5.3: SCS algorithm in an example

4. From the network topology database at the SDN controller, the SCS module identifies the router with largest index (closer to clients) in the DAG: C_1 in Figure 5.3b. Then it applies Knapsack problem on C_1 as follows: The list of requested chunks at C_1 is $\{(1, 75), (1, 60), (2, 56), (1, 18), (1, 1)\}$. Calculating the distances between the requested chunks for each video, we have two lists of distances, $\{17, 42, 15\}$ and $\{\}$, for video 1 and video 2, respectively. The distance list for video 2 is empty since it is currently being requested by only one client. Now consider both lists of distances together, and sort them in ascending order: $\{15, 17, 42\}$. Since the cache capacity is 20 chunks, after storing the smallest distance, 15, corresponding to $f_1 \rightarrow f_2$, there will not be enough space for the next smallest distance, 17. So the caching decision at C_1 remains as before.

5. Next, SCS module considers the router with the second largest index in the subset in Figure 5.3b: C_5 in this case. Then it applies Knapsack problem on C_5 after removing the flows already cached in the previous step(s). The details of the caching decision are not mentioned here but it is similar to step 4. There is one option for caching; $f_4 \rightarrow f_{11}$, so that will be the new caching decision at C_5. The controller forwards the new caching decision to C_5.

6. SCS module repeats step 5 on the rest of the routers in the subset, starting from those with largest indices: only C_7 left in this example. Considering all the flows reach C_7 from the left side in Figure 5.3b, plus all those on the right side, our caching options are: $f_6 \rightarrow f_3$, $f_9 \rightarrow f_1$ and $f_{10} \rightarrow f_4$, which correspond to distance of 4, 11 and 13, respectively. Starting from the smallest distance until the cache is full, the caching decision at C_7 will be similar to the previous caching decision, $f_6 \rightarrow f_3$ and $f_9 \rightarrow f_1$. Because the cache capacity is 20 chunks, $f_{10} \rightarrow f_4$ will not be cached at C_7.

The SCS caching module is summarized in Table 5.1. One can improve the performance by solving the network jointly, instead of solving separately for each router. This joint optimization can be formulated as a integer linear program, as shown in the Appendix. The disadvantage of the joint optimization is complexity. The SCS algorithm described above has much less computational

Table 5.1: Complete Caching Procedure for Directed Acyclic Graphs

For each new request arrival:

1. When a router receives a new request and there is no matching entry for that flow, it forwards it to the SDN controller.

2. The controller runs the routing module to calculate the routing for the new request(s).

3. The controller considers the subset consists of all the routers on the route that is assigned to the new flow, and considers all the flows that are forwarded through these routers, after removing those flows that are cached in the other routers. Then it runs the SCS module (below) for this subset and the set of flows passing through it.

   SCS Algorithm:

4. Apply Knapsack problem in (5.1) at the router with the largest index in the subset, considering the topological ordering of the nodes in the DAG, and assuming that the server nodes have the smallest index.

5. Consider the router with the 2nd largest index in topological ordering. Apply Knapsack after removing the flows already cached in the previous step(s).

6. Continue step 5 with the other routers in the subnet, starting from those with the largest indices (closer to the clients).*

*At each router, if there are multiple caching choices with the same performance, those with highest similarity to the previous caching decision at that router will be selected.

complexity than the optimal solution and can be easily implemented at SDN controller and run in real-time, while the performance will be close to optimum, as we will see in the simulations.

5.3 Optimum Integer Linear Model

In this section we model the caching decision problem as an integer linear optimization. The objective is to minimize the average rates of all the links in the network. Let $\mathcal{L}$ be the set of all links in the network. Also let the rate of link that connects router u to router v be $R_{(u,v)}$. Then, the objective can be formulated as: $Minimize \frac{1}{|\mathcal{L}|} \sum_{(u,v)\in\mathcal{L}} R_{(u,v)}$.

Define a set of binary variables x_{ik}^r, to indicate which flows are stored in each router cache. Specifically, $x_{ik}^r = 1$ means f_i services f_k by caching the last $b_i(t) - b_k(t)$ chunks of f_i in cache of router r, where $b_i(t)$ and $b_k(t)$ are the chunk numbers requested by f_i and f_k at time t, respectively. Clearly, x_{ik}^r needs to be defined only if f_i and f_k correspond to the same video. Then, $R_{(u,v)}$ can be written for all $(u, v) \in \mathcal{L}$ as:

$$\begin{aligned} R_{(u,v)} &= |U_{(u,v)}| - \sum_{r\in C_{(u,v)}} \sum_{k\in U_{(u,v)}} x_{ik}^r \\ &\quad - \sum_{r\in C_{(u,v)} \bigcap 1stHopCaches} \sum_{F_r^i \in U_{(u,v)}} |F_r^i| - 1 \\ &\quad - \sum_{r\in C_{(u,v)} \bigcap CachesFartherThan1hop} \sum_i n_i^r \end{aligned} \tag{5.2}$$

Equation (5.2) assumes that the rate of each flow is 1 unit. The first term is the total number of flows forwarded through link (u, v) based on the routing decisions. The second term subtracts the flows that do not have to be forwarded through link (u, v) because of the caching decisions on routers from the link to the clients. The third and fourth terms account for the reduction in the rate on link (u, v) due to multicasting, i.e., one can share the forwarding of chunks to flows that are synchronous to each other. In these two terms, *1stHopCaches* is a subset of routers that are directly connected to at least one client and *CachesFartherThan1hop* are subset of routers not in *1stHopCaches*. The reason that the synchronized flows that go through the routers one hop away from the clients should be counted separately from the synchronized requests that goes through the farther routers, is that those synchronized flows that pass through the routers one hop away from the clients can be always treated as one flow, but this is not the case for the routers farther than one hop from the clients; in this case it depends on the flows stored in the caches closer to the clients.

Below we show the integer linear optimization formulation of the caching decision problem. The first constraint models the capacity limit of the cache at router r. The second constraint defines an auxiliary variable needed in the objective. The third constraint is added for simplicity of presentation. One can eliminate the third constraint and only define x_{ik}^r for i, k, and r satisfying the conditions listed in this constraint.

Maximize $\sum_{(u,v)\in\mathcal{L}} \sum_{r\in C_{(u,v)}} x_{ik}^r$ (5.3)

$+ \sum_{(u,v)\in\mathcal{L}} \sum_{r\in C_{(u,v)} \bigcap CachesFartherThan1hop} \sum_i (n_i^r - 1)$

Subject to:

1. $\sum_{i,k\in U_{(u,v)}} (b_i(t) - b_k(t)) x_{ik}^r \leq S_r \qquad \forall r$
2. $n_r^i = max(1, |G_r^i|) \qquad \forall r, \forall i$
3. $x_{ik}^r = 0 \quad$ if $\quad v_i \neq v_k$ or $b_i(t) \leq b_k(t)$ or $\{i,k\} \notin V_r$

Table 5.2 summarizes the definition of all the parameters in our model.

Our optimization model is not imposing any constraint on the links rate, but rate constraint could be easily included using the definition of the rate in equation (5.2).

The main limitation of this optimum formulation is that it is computationally expensive. They cannot be easily solved in real-time on an SDN-controller.

5.4 SCS Implementation in GENI Platform

We demonstrate the proposed caching strategy using the SDN capabilities in GENI experimental platform [5]. As mentioned earlier, GENI provides an OpenFlow testbed for evaluating SDN applications. In our GENI experiment, all the routers in the network are OpenFlow enabled [15], and communicate with a central Controller. Our controller is Floodlight [6] and OpenFlow routers are Open vSwitch (OVS) [7]. Each router maintains a flow table, and performs packet lookup and forwarding. When a router receives a packet, it will be compared against its flow table. Upon finding a matching entry in the flow table, the router performs all the corresponding actions to that entry. If there is no match, the router will forward the packet to the controller. The controller will

then inform that router (and maybe other routers) how to handle that packet and the similar ones, by adding or removing some flow entries.

Each new request requires an exchange of few control messages before the client receives the first chuck of the video. Specifically, when a new request arrives for the first time at an OpenFlow router, a message is sent to the controller. The controller forwards this message to the Caching Module and response containing the caching decisions is sent back to the controller. The controller then updates the flow tables in the routers with changes in the caching decision. If the latency in

Table 5.2: Parameters Definition for optimum staggered multicasts model

$\mathcal{L}$	Set of all links in the network
$R_{(u,v)}$	Forwarding rate from router u to router v as a result of caching decisions
S_r	Storage capacity of cache at router r
v_i	Type of video f_i is asking for
$b_i(t)$	Chunk # f_i is requesting for at time t
$C_{(u,v)}$	Set of all routers involved in the routes of flows in $U_{(u,v)}$, from router v to the clients
$U_{(u,v)}$	Set of all flows whose route includes the link between router u and v
V_r	Set of all flows whose route includes router r after considering caching decisions
F_r^i	Set of all flows whose route includes router r and synchronized to flow i
G_r^i	Set of all incoming links into router r through which chunks for flows in F_r^i reach router r (removing those flows that are not being forwarded because of the caching decisions)
x_{ik}^r	Binary variable; 1 if f_i helps f_k by storing $b_i(t)$ to $b_k(t)$ chunks at cache of router r, 0 otherwise
n_r^i	Auxiliary integer variable; $n_r^i = 1$ if $\|G_r^i\| = 0$, o.w., $n_r^i = \|G_r^i\|$

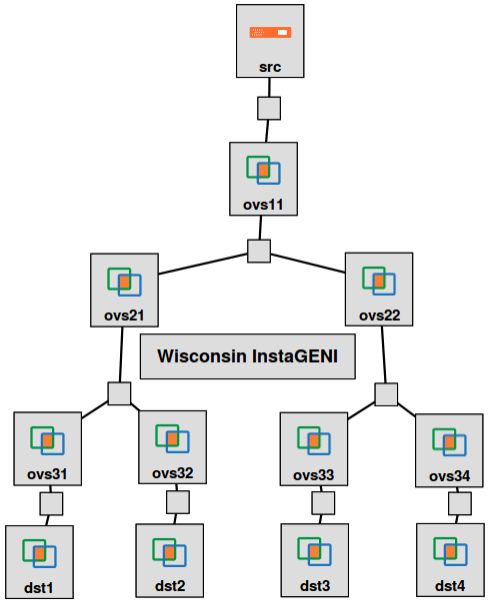


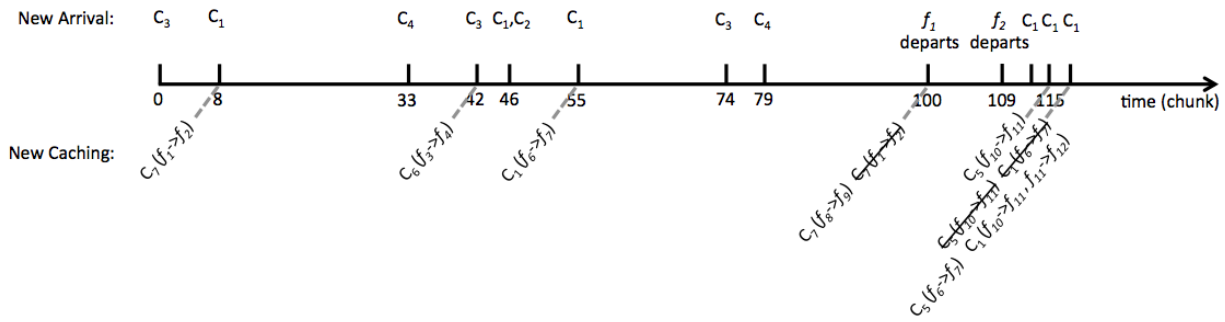
Figure 5.4: GENI setup in our example

delivering a control message is in the order of few milliseconds, then this entire exchange adds at most few tens of millisecond of latency to the start of the video. There are no control overheads after the video streaming commences.

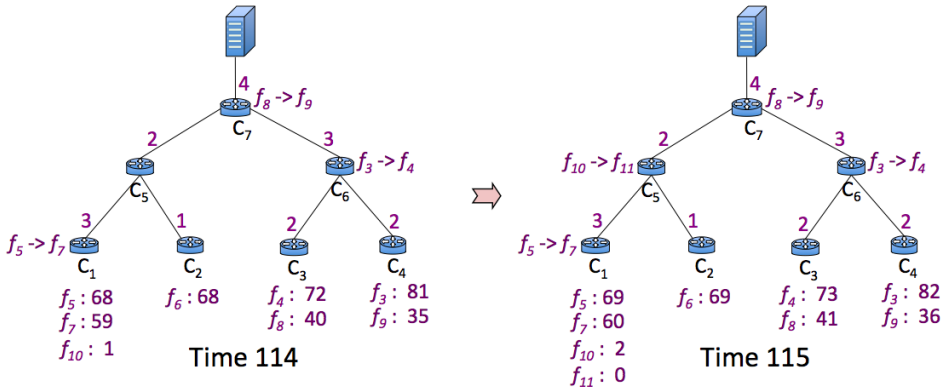
Our setup is shown in Figure. 5.4. It consists of a *raw PC* (GENI term for a hardware PC as opposed to a VM), used as a source for the videos. All other network components are Virtual Machines (VMs), which runs a kernel module of the OVS, which basically enables the host machine to behave as an OpenFlow enabled router. In the experiment, all the nodes in the network were situated at the Wisconsin InstaGeni site. All the OVS nodes have caching capacity of 10 chunks in this example. There are four destination nodes in Figure 5.4. Different users will be distinguished by different port numbers at each of these destination nodes.

Figure 5.5a shows the arrival of new clients at different times and different points in the network into this experimental setup. The time unit is T_{ch}, duration of one chunk. In this example, all the requests are for a single video, and all clients start from the beginning of the video. The new caching decisions that have been made at different times are included in the picture. When a client

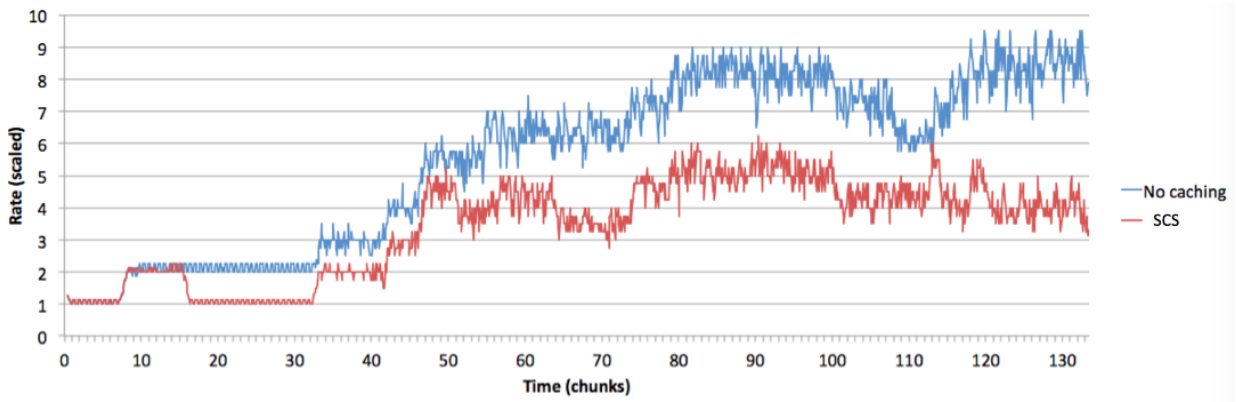
has a new video request, the lack of a corresponding flow entry in the routers results in the request



(a)



(b)



(c)

Figure 5.5: The experimental example on GENI

packet being sent to the controller. The controller then recognizes that it is a new request packet and forwards it to the ”Cache module” which runs SCS algorithm. Then the controller forwards the new flow rule and caching rule for the new request to the corresponding routers, along with the possible updates for some of the current flows in the network, as a result of the caching algorithm.

When a new caching decision is made at a cache, e.g. $f_i \rightarrow f_k$ at cache C_j, the data is not available to f_k right away, since C_j can cache only the future packets of f_i, after receiving the new caching decision. In other words, f_k can get the packets from C_j only after $b_i(t) - b_k(t)$ chunks. Until the data is available at the assigned cache, it comes from the source node. We will see this transient effect later, in the example in Figure 5.5.

Figure 5.5b shows a point in time, when a new caching decision is made after the arrival of a new request. The chunk number being requested by each flow is shown next to the edge routers. Figure 5.5c of the plot shows the rate of the top link that connects C_7 to the source for two cases: when no caching is applied, and when SCS is implemented. The rates are scaled, so that the rate of one video stream is 1. At time 115, we expect the rates for no-caching and SCS to be 8 and 4, respectively. As shown in Figure 5.5c, the measured rates on GENI implementation is close to these expected rates. The SCS curve at this point is a little higher than what we expect, due to the transient effect explained before. This transient effect can also be seen clearly on the plot when there are only few clients at earlier times. For example at time 8, a new client arrives, but the rate on the link connecting to C_7 does not increase in the long run due to caching decision. However, since C_7 can only cache forwarded packets after a caching decision has been made, the first eight chunks must be delivered from the source before the effect of caching decision comes into play.

To evaluate the effectiveness of the proposed SCS algorithm, the simulation results for different cases and the comparison with LRU and LFU caching are presented in the following section. We also compare the SCS performance with the optimum solution for further evaluation.

5.5 Simulation Results

In GENI, it is difficult to evaluate the performance of SCS on a large number of topologies. Evaluation on each topology involves reserving resources, booting up the nodes, installing software, running the experiment, downloading the performance measurements, and releasing the resources.

To evaluate SCS on many different topologies, we conducted experiments on a simulator written in Python. The simulator implements SCS, LRU, and LFU caching strategies. The inputs to the simulator are: (i) the number of nodes, (ii) constraints on topology such as bounds on degree of the nodes, (iii) arrival rate of requests from clients, and (iv) number of videos and parameters modeling the selection of videos by clients. The simulator generates a random topology and then generates a sequence of video requests based on a Poisson arrival process and Zpif distribution. The Poisson arrival rate is denoted by λ and the Zipf distribution that models the user preference among the set of available videos is assumed to have an exponent of 0.9. A single video is assumed to be comprised of 100 chunks. For simplicity, within the simulator, a unit time corresponds to the time required to watch one chunk of a video. The simulator maintains the rate on each link in the network for all time instances and outputs the average link bandwidth required in the network at the end of the experiment. When a request for a new video arrives, the simulator invokes the actions undertaken by the SDN controller and then implements the new caching decisions, if any, at the corresponding nodes in accordance with the algorithm being simulated.

We start with the performance comparison of our SCS algorithm with the optimum model, and LRU and LFU. Since the complexity of solving the optimum model is high, to facilitate the analysis, we assume the total number of videos is 20, and we apply it on the same network as in GENI implementation, Figure 5.4. Then later we will focus on large Directed Acyclic Graph (DAG) networks, and we consider much larger number of videos.

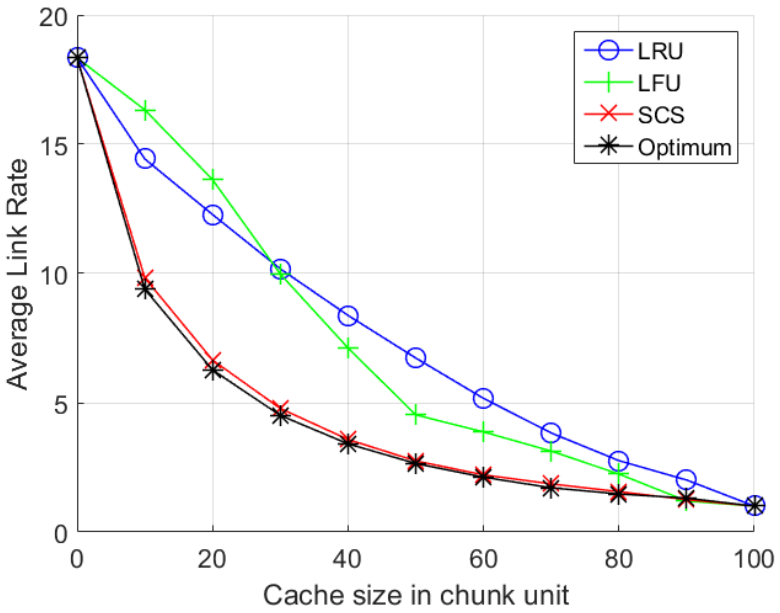
5.5.1 Comparison of SCS with Optimum model, LRU and LFU

In this section, we evaluate the performance of our algorithm, and compare it with the optimum model, LRU and LFU caching strategies. The optimum solution, which is modeled as an integer linear program earlier, is solved using Gurobi [41]. As mentioned earlier, to deal with the complexity of optimum model, we work on the network in Figure 5.4, and assume total number of videos is 20.

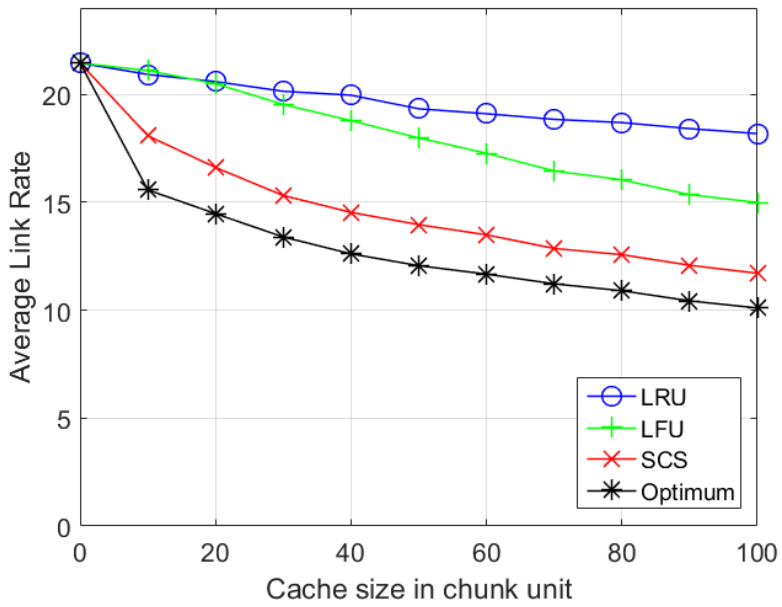
We start with the results for single-video case, when all the clients requests are for a single video, because it is easier to understand; then we will focus on the real case of multi-video simulations.

Figure 5.6 compares the performance of SCS, LRU and LFU caching strategies, as a function of cache size. Figure 5.6 also includes the results for the optimum solution obtained by solving the integer linear program described earlier. The cache size varies from 0 to 100 chunks and is plotted on the x-axis. All caches have the same storage capacity. The y-axis shows the average traffic rate of all the links in the network for the four different strategies. In Figure 5.6a, all clients request the same video, whereas in Figure 5.6b, the clients request one of 20 possible videos based on Zipf distribution with exponent 0.9. The mean arrival time between two consecutive video requests is 1 chunk, i.e. there are on average 100 users watching videos at any given time instant, since the length of each video is 100 chunks, and we assume each client starts from the beginning of the video and watches until the end of the video.

Figure 5.6 shows the performance of SCS is very close to the optimum in both single-video and multi-video cases. In single-video case (Figure 5.6a), the maximum difference in average link rate between the optimum solution and SCS algorithm occurs when cache capacity is 10 chunks, where the average link rate is 9.83 and 9.39 for SCS and optimum case, respectively. The maximum difference in this case is less than 4.5%. In the multi-video case (Figure 5.6b), the difference between the performance of SCS and optimum is larger. The maximum difference happens again at cache size 10 chunks, where the average link rate of SCS and optimum are 18.08 and 15.57, respectively. The maximum difference is about 13.9% in this case. From Figure 5.6 we can see that the average rate of SCS and optimum model decreases rapidly with increases in the caching



(a) Average rate for single-video case, $\lambda = 1$



(b) Average rate for multi-video case, $\lambda = 1$

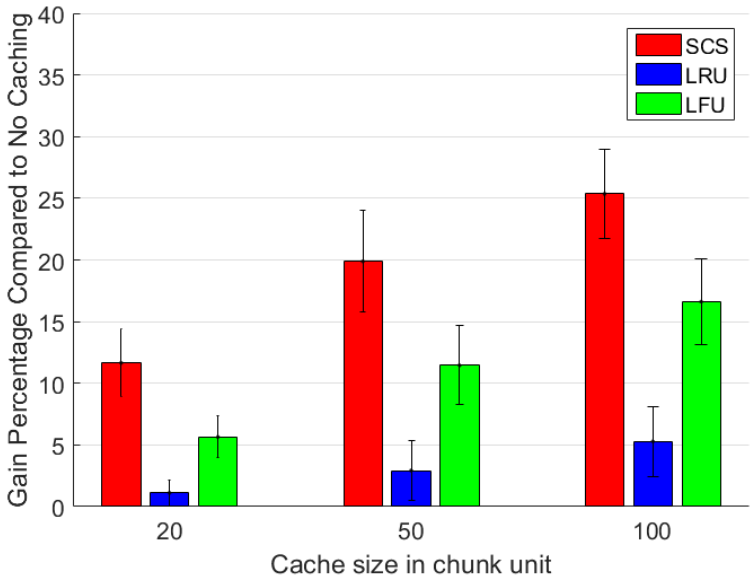
Figure 5.6: Average link rate of SCS algorithm compared to optimum, LRU and LFU

capacity, meaning even small cache can help a lot. For example, at 10 chunk storage capacity in Figure 5.6a, the average links rate is 9.83, down from 18.34 at cache capacity 0, which is almost a 50% decrease in the rate. On the other hand, the rate decrease behavior of LRU and LFU is much slower. Since we are focusing on small router level caching capacity, the rapid rate decrease behavior of SCS and optimum model is very important for us.

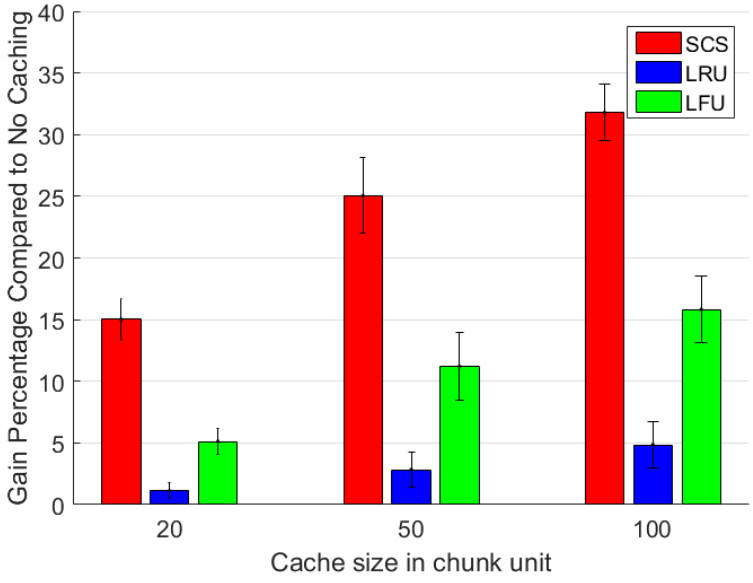
In multi-video simulations, cache capacity of 100 chunks can be considered as the storage capacity for storing the most popular video, which in our case has probability 0.24, with 20 videos and Zipf(0.9) probability distribution. Therefore, in Figure 5.6b, the point with cache capacity 100, can be compared with the capacity 24 in Figure 5.6a; at these points both methods are capable of storing 24% of the requested chunks, on average. Comparison of the gain of SCS compared to no caching at 24 chunk caching capacity shows that in single-video case, the average gain is approximately 68% in Figure 5.6a, while it is approximately 28% at the equivalent point, capacity 100, in Figure 5.6b. The reason for this difference is clear: in multi-video case, only the streams from the same video can help each other, while this restriction does not exist in single-video scenario.

5.5.2 Large Scale Simulations in Directed Acyclic Graphs

In this section we focus on large DAG networks, and we analyze the performance of SCS algorithm and compare it with LRU and LFU caching strategies in such networks. The video catalog size is 10,000 in this case, and has Zipf distribution with exponent 0.9. We generate random DAG networks of size 30 vertices. These vertices are our routers and we assume all of them have the same caching capacity. Out of these 30 routers, we consider 7 of them to be the edge routers connected directly to the clients. The random DAGs are generated such that each vertex is connected to 2 to 6 vertices selected randomly from the 9 closest vertices, considering the topological order in the DAG. For each new request arrival, we randomly select a route from all the possible routes from the source to the new client, which are at most two hops farther than the shortest route(s). In this way, we improve the chance that the router vertices in the random DAG are being utilized evenly.



(a) $\lambda = 1$



(b) $\lambda = 2$

Figure 5.7: Rate gain of SCS, LRU and LFU vs. no caching for two different values of λ, along with 90% confidence interval

Figure 5.7 shows the average gain of SCS, LRU and LFU, for three different values of cache sizes, 20, 50 and 100 chunks. Two different values of clients arrival rate is analyzed in part (a) and (b) of Figure 5.7. Each bar in this plot is the average of at least 70 randomly generated DAGs. The 90% confidence intervals are also shown in the plots. The gain is calculated as follows:

$$\text{gain} = \frac{\text{Rate of no caching} - \text{Rate with caching}}{\text{Rate of no caching}} * 100 \tag{5.3}$$

For $\lambda = 2$, Figure 5.7b, shows that the gain of SCS algorithm is larger compared to $\lambda = 1$, Figure 5.7a, for the same caching capacity. The reason is obvious: for larger λ, the gaps between the requests are smaller on average, so with the same caching capacity the gain of SCS will be higher. This is not the case for LFU and LRU. The gain of LFU in Figure 5.7a and Figure 5.7b shows almost no change for different values of λ, while the gain of LRU sees even a little decrease on gain by increase in the arrival rate. These observations match our expectations. LFU works based on the frequency of the requests, and changing the arrival rate without changing the request statistics will have no impact on the LFU average gain. In LRU, by increase in the clients arrival rate, at each point in time there will be more requests. Since LRU treats all the requests the same and stores the most recently requested chunks among them, the chance that some of these can help others will decrease by increase in the arrival rate.

Figure 5.8 shows the histogram of SCS gain compared to no-caching for one case in Figure 5.7: $\lambda = 1$ and cache size 50 chunks. We can see how the gain values are distributed for different DAGs. It shows the results for 83 topologies, and the mean value is 20% and all gains are greater than 15%. We also showed 90% confidence interval in this plot.

In these simulations, for each new request, on average there were less than 5 routers in the subset that would have potentially new caching decisions. The number of routers with some changes in their flow table, whether addition of new caching decisions or removal of old ones, is equal to the number of control messages need to be sent from the controller. Therefore, on average, there will be less than 5 control messages sent from the controller to the corresponding routers per new request arrival.

In Figure 5.7, the caching capacity for each single router is shown. But even considering the sum of the caching capacity in the whole network, the value is still very small compared to the

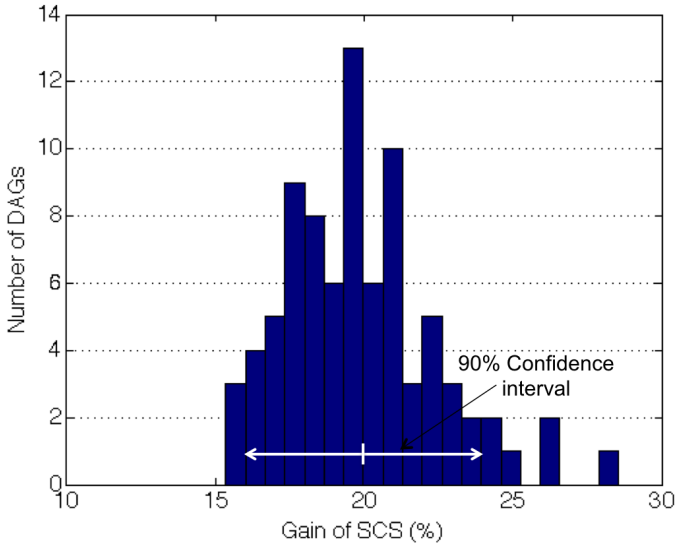


Figure 5.8: Histogram of SCS gain: $\lambda = 1$, cache size = 50 chunks

video catalog size. For cache sizes of 20, 50 and 100 chunks, the ratio of the total caching capacity to the catalog size in our simulations is 0.0006, 0.0015 and 0.0030 respectively. The gain achieved from SCS caching for such small caching capacities is a great advantage.

Chapter 6

Joint Long-term and Short-term Caching

In chapter 1 (introduction), we described long-term and short-term caching as two generic ways of in-network data storage. On one hand, the purpose of long-term caching is to take advantage of the long-term statistics of the request, and store the most frequently requested contents closer the clients. On the other hand, short-term caching serves a different purpose: examining the current requests and exploring opportunities of caching. In chapter 4, we proposed CCIP algorithm, a caching algorithm for long-term caching of content. We showed that CCIP improves the performance by using the information about the long-term statistics of the requests, and taking advantage of encoding across different videos. In chapter 5, we introduced SCS caching algorithm as a short-term caching method, which looks for the opportunities where storing a portion of a stream can help other stream(s). Long-term caching and short-term caching target the same objective in different ways.

Another major difference between long-term caching and short-term caching, which is clear from their names, is that long-term caches content will remain in the caches for longer time, since long-caching is interested in the long-term statistics of the contents and nothing will change until the long-term statistics of the requests change. On the other hand, since short-term caching looks for caching opportunities given the current flows in the network, the changes in the caching decision is much more frequent compared to long-term caches. Because of the nature of long and short caching, and the differences between them, we expect to see a better performance by taking advantage of both. In other words, we expect a better caching performance when the cache size is shared between long and short caching, compared to when assigning all the cache capacity to short-term or long-term cache. In this chapter, we explore the case when both long-term and

short-term caching solutions are being applied on a network. We will discuss the impact of joint CCIP and SCS caching solutions, and show that the gain depends on the ratio of the caching size assigned to long-term and short-term solutions.

Another interesting topic that we will discuss in this chapter is joint routing and caching. The CCIP solution in chapter 4 not only tells us what to store at each cache, but where each client should retrieve each video from. In chapter 5 where we proposed SCS as a short-term caching solution for staggered video streaming, the assumption was that the caching algorithm is being applied after the routing is solved. In this chapter, we introduce a heuristic routing algorithm, which considers cached contents from CCIP solution, as well as the possibility of providing the current request from the short-term caches.

6.1 Joint CCIP and SCS caching

In this chapter, we explore the advantages of CCIP and SCS caching in a network. We assume that the capacity of router caches in the network are divided between long-term and short-term caching, and the assigned caching size to each part remains the same during the simulations. Since the update rate of long-caching is much lower compared to short-caching, we assume long-caching solution (output of CCIP), does not change during a simulation experiment. We first run CCIP, and fill the long-cache portion of the caches; then apply the short-term caching method to fill the short-cache portion of the caches.

The output of CCIP solution includes what should be stored at each cache, and where should each client get each chunk of its video. One of the inputs to the CCIP was the cost of content delivery from each cache to each client. The costs of delivery from a cache to a client in CCIP are set equal to the shortest distance from that cache to that client. When a new request arrives, CCIP solution specifies which node should provide the data to the client. Then we select a random path that is at most two hops longer than the shortest path from that node to the client. After selecting the route, SCS caching is used to determine whether the content should be cached further closer to the client.

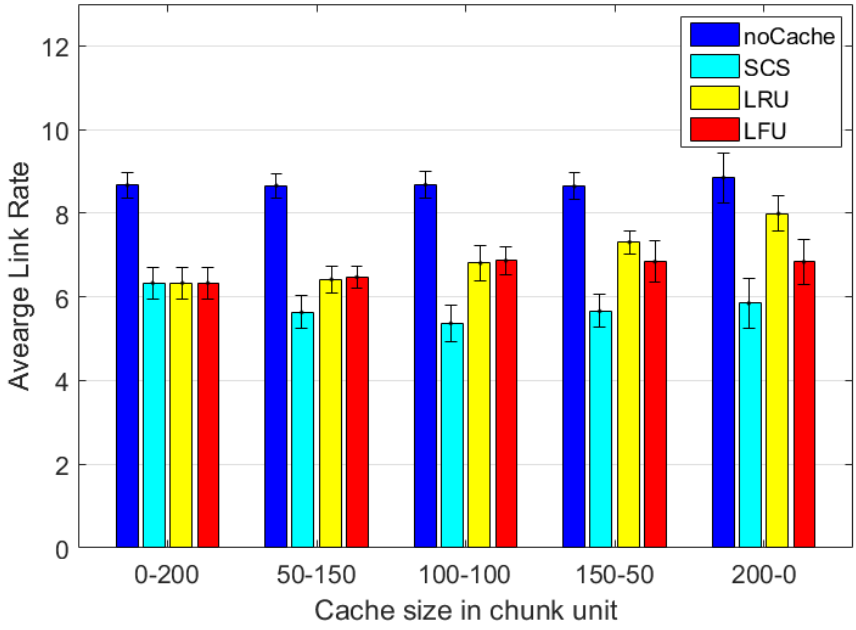
If CCIP content delivery solution includes encoded data, a random route at most two hops longer than the shortest path is selected from each of the involved caches. Then, SCS solution is used for one of these two contents; If SCS caching results in storing the new request in a short cache on that route, then that short cache will be the delivery point and we are done with the routing and caching decision. If SCS caching does not result in storing the new request in any short cache on the route, then SCS solution is investigated on the other route. Algorithm 1 shows the pseudo code of joint CCIP and SCS caching.

Figure 6.1 shows the performance of CCIP combined with SCS, LRU or LFU, for different ratios of short-term to long-term caching capacity. The total caching capacity at each router in Figure 6.1 is 200 chunks. x axis shows different assignment of short-term and long-term cache size. The two cases on left and right side show the extreme cases when all the caching capacity is assigned to long-term caching and short-term caching, respectively. If we consider the pair of (short-term cache size)-(long-term cache size), then the plot considers 0-200, 50-150, 100-100, 150-50 and 200-0 cache size assignments. Figure 6.1a shows the average link rate in the network, while Figure 6.1b shows the average gain compared to no caching case. The definition of gain is same as the previous chapters. Each bar is the average of at least 70 randomly generated DAGs.

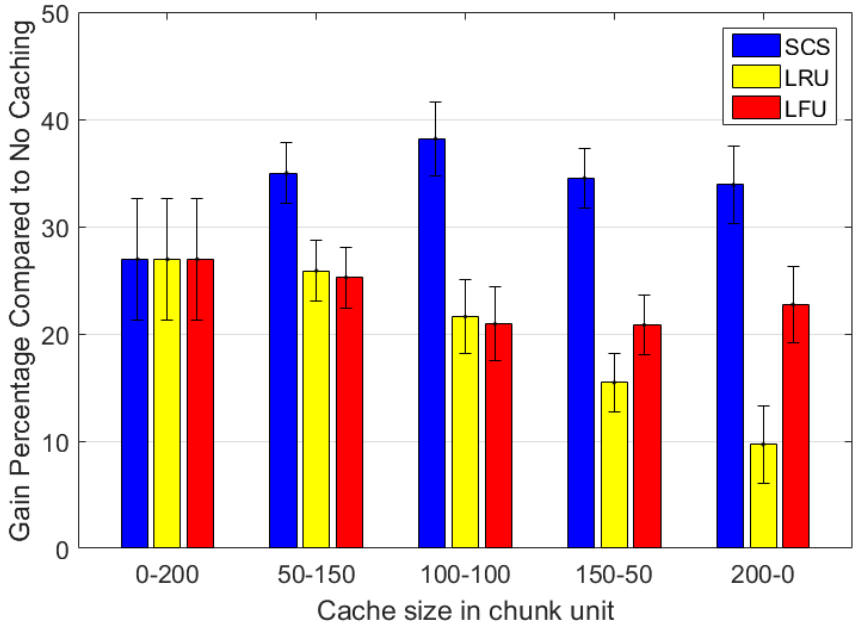
The random DAGs are generated in the same way as explained in the previous chapter. Moving from the start point on the x axis, 0-200 point, towards the end of x axis, 200-0 point, (increasing the short cache size and decreasing long cache size), there is an increase and then decrease in the gain of joint CCIP and SCS caching. For example, the gain increases from 27% at 0-200 to about 35% and 38% at 50-150 and 100-100, respectively. But further increase in the short-term cache size and decrease in the long-term cache size decreases gain, to about 34% at 200-0. This trend can be explained as follows: Long-term caching often does not help in the situations where some flows are close in staggering and can easily help each other using SCS caching, if they are not already stored in long-term caches. But SCS can easily help with those situations. However, SCS caching opportunities saturates beyond certain cache size. The saturation point depends on the arrival rate of the requests and Zipf exponent of the video requests distribution. For long-term caching, similar situation happens: depending on the exponent of Zipf distribution, the gain saturates beyond certain

Algorithm 1 joint CCIP and SCS caching

CCIP-module() /* SDN controller first runs CCIP module*/
SDN controller fills the long cache portion at each router accordingly
for each new request arrival **do**
  $i \leftarrow$ new client
  $k \leftarrow$ requested video by i
  $c \leftarrow$ suggested node(s) to deliver video k to client i from CCIP solution
  if $|c| = 1$ **then**
    $l \leftarrow$ length of shortest path from c to i
    $r \leftarrow$ select a random route from c to i, which is at most 2 hops larger than l
    SCS(subset from c to i on route r)
    if SCS caching does not results in storing the new request **then**
      Deliver the request from c
    end if
  else
    $\{r_1, r_2\} \leftarrow$ Select a random route from each router in c, $\{c_j, c_{j'}\}$, to i
    /* r_1 and r_2 are the random selected routes from c_j and $c_{j'}$ to i, respectively */
    SCS(subnet from c_j to i on the selected route r_1)
    if SCS caching does not results in storing the new request **then**
      SCS(subnet from $c_{j'}$ to i on the selected route r_2)
      if SCS caching does not results in storing the new request **then**
        Deliver the request from c_j and $c_{j'}$
      end if
    end if
  end if
end for



(a) Average link rate



(b) Average gain

Figure 6.1: Joint short and long caching analysis when total cache size is fixed

cache size. The initial increase in the average gain is because the gain from increased cache size for SCS is larger than the decrease in gain due to smaller cache size for CCIP. Eventually, as SCS saturates, the decrease in gain from CCIP becomes larger and the overall gain starts to drop. From Figure 6.1, one can conclude that with a small cache size at the routers, e.g. 200 chunks in this case, assigning all the cache to long-cache or short-cache is not a good idea. One should instead divide the total cache size between long and short cache. For instance, in this specific case, the best assignment is the equal division of cache size between long-term and short-term caches, 100-100.

Now examine the performance of joint CCIP with LRU or LFU from Figure 6.1. The best performance happens when all the 200 chunks of caching capacity is assigned to CCIP, and nothing to LRU and LFU. It means that the gain from LRU or LFU caching is not enough to compensate for the decrease in the gain due to smaller CCIP caching capacity. There is another interesting observation in Figure 6.1. In chapter 5, we observed that the caching gain for LFU was considerably better than LRU for small cache sizes. However, in Figure 6.1, for cache assignments of 50-150 and 100-100, the performance of CCIP with LRU is slightly better than CCIP with LFU. The reason behind this behavior can be explained as follows: the main concept behind both CCIP and LFU is to store the most frequently requested contents closer to the clients, while LRU stores the most recently requested contents. Therefore, when there is a slightly time-shifted request of another earlier request, LRU can help, even if those streams are not the most common streams. But LFU on the other hand is closer to how CCIP caching works, and since LFU caching is being applied after CCIP, the highest probability contents are already being stored by CCIP, and LFU works on lower probability contents, which have less impact on the average rate. In other words, since CCIP and LRU have different behaviors compared to CCIP and LFU, for larger CCIP caching capacity and smaller short caching capacity, the joint gain for LRU case will be larger than LFU. But by increase in the short caching capacity and decrease in the CCIP capacity and so its gain, the higher performance of LFU compared to LRU will become dominant.

In Figure 6.2, we show joint long-term and short-term caching when long-term cache size is a fixed value, and short-term cache capacity increases. Two fixed values of long-term cache size, 100 and 200 chunks, are considered in the plots. Unlike Figure 6.1, the total cache size in this

case is not a fixed number. We can see that for both values of long cache sizes, increasing short cache size more than 150 chunks has a small impact on the results and the improvement is small. In other words, the gain saturates at short-term cache size of 150 for both long-term cache sizes of 100 and 200 chunks, and more short-term caching capacity will not be much beneficial on average. In addition, this plot again shows that how by adding a small short-term caching and applying SCS on top of long-term caching, one can improve the network bandwidth usage.

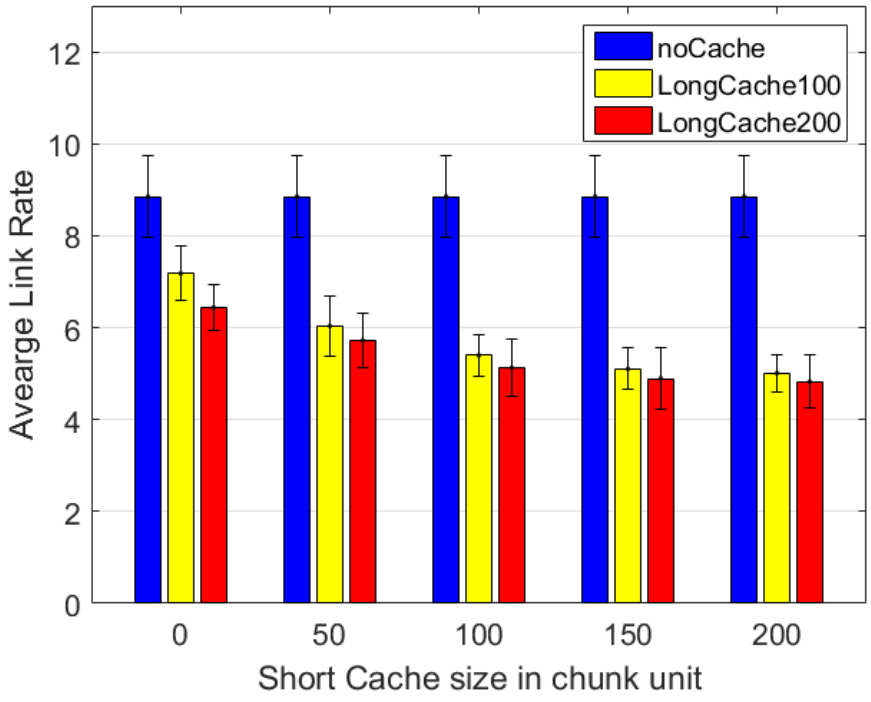
In Tables 6.1 and 6.2, the average link rate and gain for different values of long-term and short-term cache size is shown. One can use such a table and decide on the operation point, given the available caching capacity at each router. The 3D plots of rate and gain for the values in table 6.1 and 6.2 are shown in Figure 6.3.

Table 6.1: Average rate for different values of long and short cache size

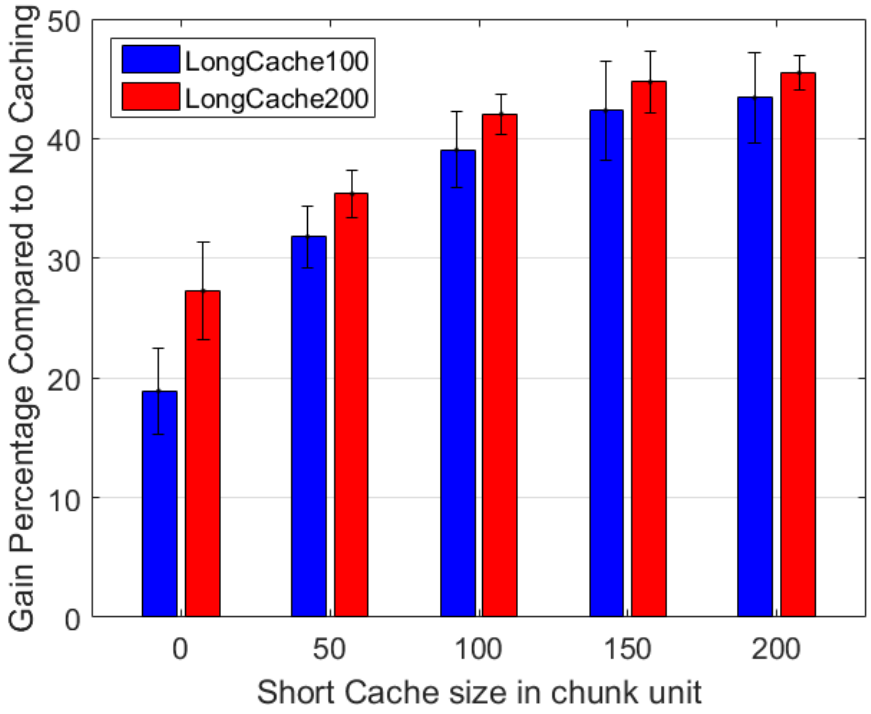
STC size / LTC size	0	50	100	150	200
0	8.12	6.55	6.00	5.71	5.55
50	7.35	6.15	5.64	5.51	5.30
100	7.18	6.04	5.40	5.11	5.01
150	6.40	5.62	5.22	5.00	4.88
200	6.44	5.72	5.13	4.89	4.82

6.2 Caching aware routing

The long-caching solution proposed in chapter 4 was a joint caching and routing method, where the solution also identifies lowest cost way of retrieving data from the caches for each client. In chapter 5, short-term caching solution assumes that the routing is given. In this chapter, we propose a routing method which is not only long-cache aware, but also considers the fact that SCS short-term caching will be applied on the network. We show that this joint long and short caching aware routing improves the network performance by decreasing the average link rate.

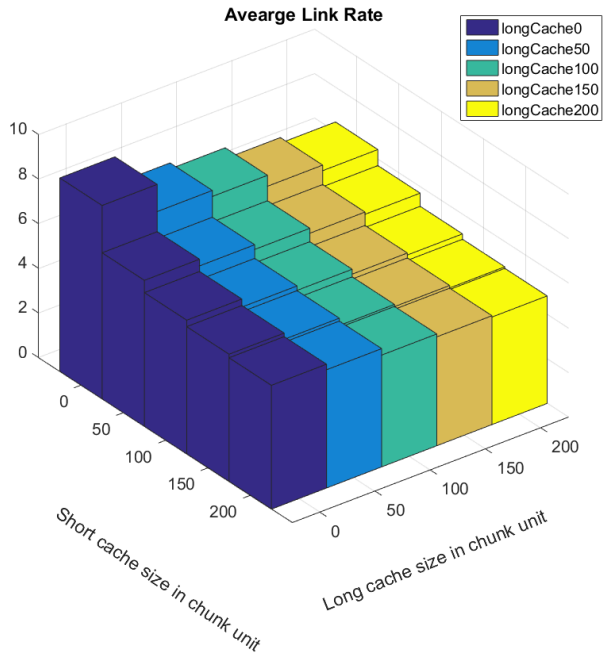


(a) Average link rate

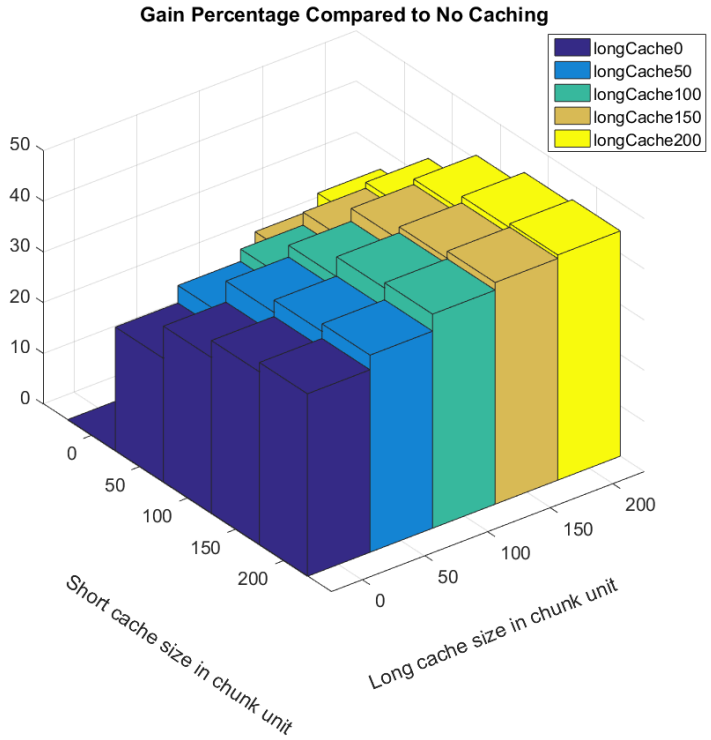


(b) Average gain

Figure 6.2: Joint short-term and long-term caching when long-term cache size is fixed



(a) Average link rate



(b) Average gain

Figure 6.3: Joint short and long caching analysis for fixed long cache size, and variable short cache size

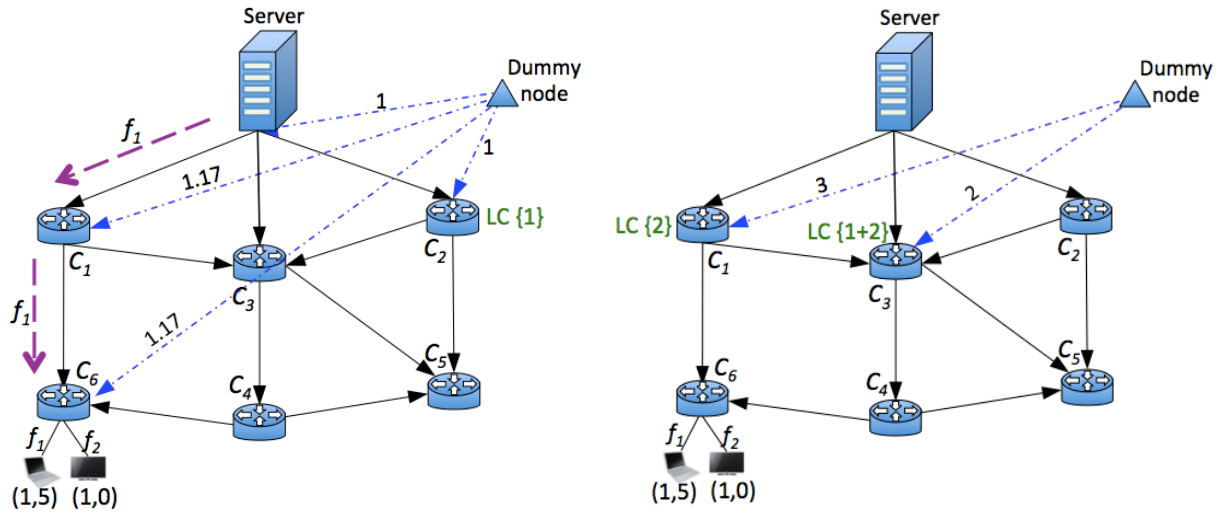
Table 6.2: Average gain for different values of long and short cache size

LTC size \ STC size	0	50	100	150	200
0	0	24.45	30.79	34.14	35.98
50	14.07	27.91	34.94	37.37	38.86
100	17.18	30.33	37.25	41.17	42.21
150	22.83	32.87	39.79	42.32	43.71
200	25.72	34.02	40.83	43.59	44.40

6.2.1 Joint Long and Short Cache aware Routing

The goal is to find a route with the knowledge of the CCIP caching solutions, and the fact that SCS short term caching will be applied on the network. The solution can be best explained with an example of a DAG network shown in Figure 6.4. Assume that the router caches have storage capacity of 100 chunks for short-term caching and 100 chunks for long-term caching.

Assume that C_2 router stores video 1 in its long cache, shown as LC$\{1\}$. Now consider two requests in the network, both asking for video 1. f_1 started 5 chunks before, and f_2 just arrived. Given the information about the long-term cache contents and the current flows going through the network, we want to find a route that can efficiently use the stored content at long-term caches, and also help SCS short-term caching solution to effectively takes advantage of staggered multicasts. In order to solve this problem, we introduce a dummy node, and draw a link from the dummy node to every router in the network that might be able to provide the newly arrived stream from its long-term or short-term cache. When the long-term caching solution from CCIP for a specific client and the current request is from one router, meaning that the content is not encoded in the router, router C_2 in this example, and draw a link from the dummy node to the router, and set the cost equal to 1 (one hop distance.) For short-term caches, the problem is not as easy as long-term caches, since SCS caching will be solved after routing is done. We use a simple trick to solve this problem,



(a) Cost of links from dummy node to the routers

(b) Cost of links from dummy node to those routers who can provide encoded data jointly

Figure 6.4: Joint short and long caching aware routing

considering the knowledge about how SCS caching works. For each router, if the long cache part of the router contains the data non-encoded, then do not consider any short caching possibility, since we do not want to store the same data twice. Now for the routers whose long cache do not contain the requested video, or contains the flow encoded, we consider all the flows that go through them, and see if a time-shifted version of similar video is passing through it. Assign a threshold to consider only flows that are not farther than the threshold from the new request. For instance, set the threshold to 30 chunks, meaning that if the closest time shifted stream is not farther than 30 chunks from the current request, then consider that stream as a potential candidate in SCS routing that can help the new request. In this example, f_1 is the time-shifted version of the same video as f_2, and has distance of 5 chunks, which is smaller than the threshold of 30 chunks. Then assign the cost from the dummy node to the router as $1 + (\text{distance of current request from the closest flow at that router})/(\text{distance threshold})$, where distance threshold is 30. In our example, the numerator is 5, and therefore the link cost is 1.17. This will be the cost of the links from the dummy node to

both C_1 and C_6 routers, since both of them are seeing f_1 passing through them. All the other links in the network have cost of 1.

When the long-term caching solution from CCIP for a specific client and the current request is from more than one router, meaning that the content is encoded at long cache, we calculate the link cost as follows: The solution is shown in the example in Figure 6.4b. Assume that C_1 and C_3 jointly provide video 1 from their CCIP long caching (C_1 stores video 2 and C_3 stores combination of v_1 and v_2, $v_1 + v_2$). Assign the cost from the dummy node to C_1 equal to $1 +$ (shortest path from C_3 to the new client), and the cost from the dummy node to C_3 equal to $1 +$ (shortest path from C_1 to the new client). So in this case, the cost of the links from dummy node to C_1 and C_3 will be 3 and 2, respectively. For the routers whose long-term caches have the data encoded, the flows going through the router will be considered as well. Assign the final cost of the link from the dummy node to those routers equal to the minimum of the calculated cost of delivery from long-term cache and calculated cost of possible delivery from short-term cache.

Then solve the routing consistent with the previous sections: randomly selecting a route which is at most 2 hops longer than the shortest path from the dummy node to the current client. In the example in Figure 6.4a, the shortest cost path is *dummy node* $\rightarrow C_6 \rightarrow$ *client*, with cost of 1.17, which is smaller than the cost of delivery from the long cache, which is 4, or delivery from *dummy node* $\rightarrow C_1 \rightarrow C_6 \rightarrow$ *client*, which is 2.17. Algorithm 2 shows the pseudo code of long and short cache aware routing.

If in the selected route the dummy node is connected to a long cache that contains the requested video, it means that the cost of delivery from the long cache was the lowest and there is no need to run SCS caching. If the dummy node is connected to the server, or a router which does not contain the requested video in its long cache, then we run SCS caching on the selected route, ignoring the dummy node. The simulation result using long and short cache aware routing are shown in Figures 6.5 and 6.6. Figure 6.5 shows the gain of joint SCS and CCIP caching when the total cache size is fixed at 200 chunks. The result from the previous simulations in Figure 6.1b where the routing was not aware of short caching is added on top of the new bars for comparison purpose. Figure 6.6 shows the gain of joint SCS and CCIP caching when the long cache size is fixed at 100

Algorithm 2 long and short cache aware routing

$Thr \leftarrow 30$ /* defined threshold */
for each new request arrival **do**
  $i \leftarrow$ new client
  $k \leftarrow$ requested video by i
  $c \leftarrow$ suggested node(s) to deliver video k to client i from CCIP solution
  $d \leftarrow$ dummy node
  /*first add a link from the dummy node to the long cache containing the video */
  if $|c| = 1$ **then**
    add $d \rightarrow c$ link
    cost($d \rightarrow c$) $\leftarrow 1$
  else
    /* Assume that $c = \{c_j, c_{j'}\}$ */
    cost(c_j) $\leftarrow 1+$(shortest path from $c_{j'}$ to i)
    cost($c_{j'}$) $\leftarrow 1+$(shortest path from c_j to i)
  end if
  /* Then add a link from the dummy node to short caches that can potentially provide the video*/
  for each router r the network **do**
    if r does not have k stored in its long cache, or it is stored encoded there **then**
      $F \leftarrow$ set of all flows that goes through r
      $f \leftarrow$ the closest flow in F for the same video k, requesting a chunk smaller than Thr
      if $|f| > 0$ **then** /* if the set is not empty */
        add $d \rightarrow r$ link
        cost($d \rightarrow r$) $\leftarrow 1+$ (chunk # f is asking for) $/Thr$
      end if
    end if
  end for
end for

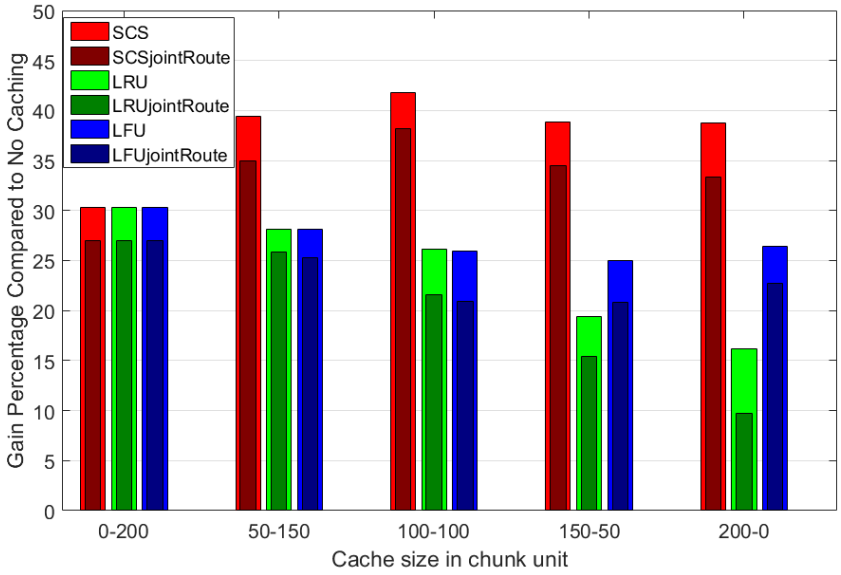


Figure 6.5: Routing aware of long-caching vs. routing aware of both long and short caching (fixed total cache size)

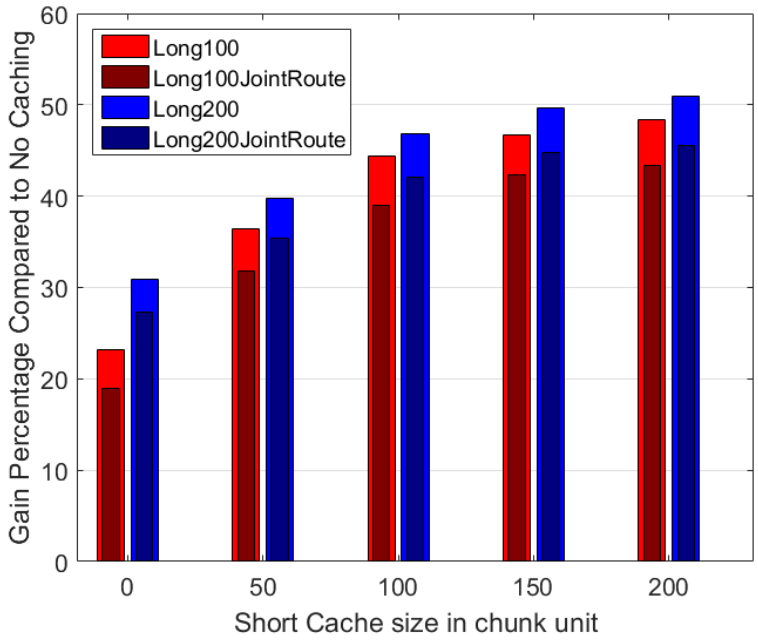


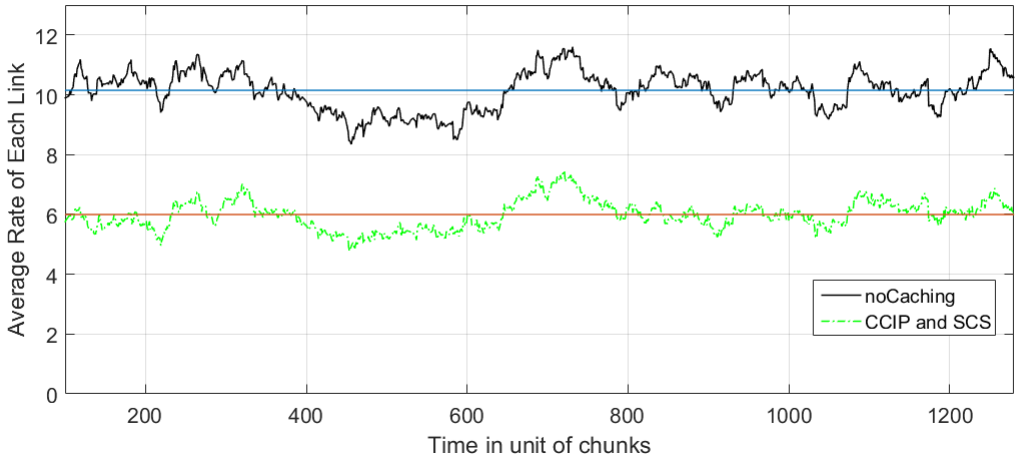
Figure 6.6: Routing aware of long-caching vs. routing aware of both long and short caching (fixed long cache size)

and 200 chunks. Previous simulation result from Figure 6.2b, where the routing was not aware of short caching, are added on top of the new bars for comparison purpose.

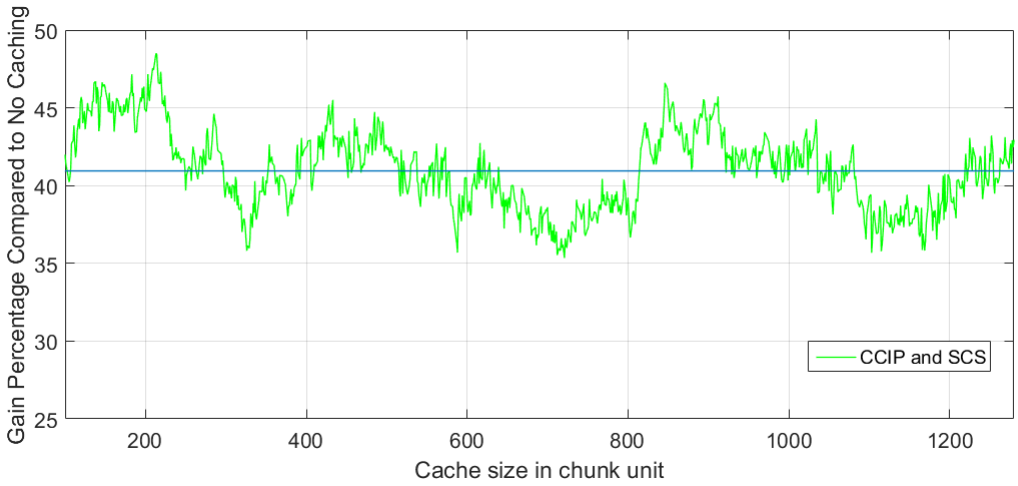
6.3 Time Analysis of the results

The error bars in the previous cases were showing the variations of the DAGs, but not necessarily showing the variance of gain at different times. Figure 6.7 shows the performance and gain variance as a function of time, for the 100-100 point: 100 chunks of CCIP caching capacity and 100 chunks of SCS caching capacity. Figure 6.7 shows the performance as a function of time for one randomly generated DAG. Figure 6.7a shows the average link rate of joint CCIP and SCS, when routing is aware of both CCIP and SCS caching. Also included in the figure is the rate of no caching case. Figure 6.7b shows the gain of joint CCIP and SCS compared to no caching case at different times of the same simulation.

The mean value of the rate for no caching and joint CCIP and SCS is 10.15 and 6.00 respectively. The mean gain is 40.94%, and the variance of the gain is 10.38. The variance of the rate for no caching case is 3.14, while it is 1.11 with CCIP and SCS. From these plots we can see that the gain does not vary much as a function of time. For example in Figure 6.7b the gain always stays above 35%.



(a) Link Rate



(b) Gain Compared to NoCaching

Figure 6.7: Deviation of the gain at different times, noCaching, long-aware, long and short aware routing

Chapter 7

Summary and Future Work

7.1 Summary of Thesis Contributions

In this dissertation we introduced pervasive router caching strategies in SDN-enabled network, where we used the global view of the SDN controller in order to take the most advantage of the small router caches capacities. We addressed the problem of high network bandwidth demand in staggered video multicasts, which is an extremely important problem considering the popularity of VoD and fast growing demand of high quality video delivery, which may cause bandwidth bottleneck in the Internet. We divided the caching capacity at each router into two portions: long-term and short-term caching. In long-term caching, we look at the statistics of the clients demands and the delivery cost of a unit of data from different routers in the network to the clients, and make caching decisions accordingly. Our proposed solution, CCIP, proposes the idea of using network coding at the storage level in a novel way: in addition to the previously proposed idea of storing encoded chunks of same videos, we developed and showed the advantage of storing cross-encoded chunks of different videos. The scalability issue with the proposed CCIP solution is solved by the network clustering solution, called C-CCIP. The advantage of the proposed method is shown in a large network and fro different demand and network parameters.

The other portion of the router cache capacity is assigned to short-term caching, where unlike long-term caching, the statistics of the demands are not the base of the caching decisions. Instead, we look at the existing streams in the network, and decide how much of a stream should be stored in different caches, in order to provide the request of the other streams. We showed that the problem can be modeled as a 0-1 knapsack problem at each router cache, given the information

about the flows passing through that routing. The simulation and implementation results showed the effectiveness of our short-term caching algorithm, called SCS, compared to the traditional LRU and LFU caching. We showed that by applying our SCS caching solution, even small router caches significantly reduce the bandwidth requirement in the network. The performance of SCS is compared with the optimum model, and we showed that SCS performance is close to the optimum solution, but it has a much lower cost compared to the optimum integer model, which makes it a very effective and practical caching solution.

After introducing our CCIP and SCS approaches as long-term and short-term caching solutions, respectively, it is important to show the result when they are jointly applied in a network. We showed the impact of different assignment of caching capacity at a router to short-term and long-term caching. We introduced a routing method, which is aware of the cached video chunks as a result of the CCIP method. In addition, our cache-aware routing strategy is designed such that the selected routes will benefit SCS caching, which will be applied after a route is selected for the new request. The simulation results show the advantage of our proposed cache-aware routing.

7.2 Future Work

7.2.1 Extending the proposed methods and assumptions

In the proposed caching methods, we had a set of assumptions for the videos, clients and the network, in addition to some constraints that we considered in our methods. Here we will point out some possible improvements to the proposed approaches in this dissertation.

In our CCIP caching approach, our integer linear model only considers the possibility of encoding chunks of two different videos, in order to keep the complexity of the model low. Although the case of considering only encoding two videos improves the performance, but there is more space for improvement. Therefore, one can research how to extend the CCIP model to the case where chunks of more than two videos can be combined, and at the same time controlling the complexity of the model.

We assumed that all videos have the same rate. In network, different requests for the same video will be provided at different rates, depending on the network condition on the path of each request, as well as the device the video is streaming on. Therefore, an interesting step to extend the proposed methods is considering different video rate requirements in the network, and how CCIP and SCS should be adjusted to work in general scenario.

Adaptive caching capacity assignment to the long-term and short-term caching at each router is another possible future study. In this project, the assigned capacity to each section was decided in advance and would remain unchanged during the network simulation time. However, one can improve the network performance by adaptively adjusting the caching capacity to each portion, depending on the demand statistics situation; for example, in a network with more dynamic requests statistics, it might be more efficient to assign a larger portion of the available cache size to the short-term caching, in order to manage the sudden increase and decrease in the popularity of some videos. In addition, in the simulations of in this project, it was assumed that the demand statistics remains the same during the network simulation time. Thus, along with the adaptive caching capacity assignment, considering the case when the demand statistics changes is another possible insightful analysis.

7.2.2 Caching for staggered video multicasts in mobile networks

In mobile networks, users are expected to move from one cell to another in the network, and the original end router providing the content to a user might not stay the same during the streaming time of a video. By increasing interest and rapid move seen during the past few years towards the autonomous cars (also known as self-driving cars) in the industry, the bandwidth demand in mobile networks will experience a sharp increase; The drivers will be added to the streaming users, since there won't be any need for paying attention to the road.

Analysis of the proposed caching methods in this thesis for a mobile network, where users movement result in the change of the original selected route during the streaming time, is an interesting future work idea. In SCS caching algorithm, we assume that the location of the clients in the network and the selected routes will not change often during the streaming time of the requested

video. Applying SCS caching might result in poor performance in such networks, considering changes in the caching that can result in the degradation of performance.

7.2.3 Caching in staggered video multicasts in emerging network technologies

The study of staggered video multicasting, the application considered in this thesis, using pervasive router caching in emerging network technologies is another possible future work. For instance, Information Centric Network (ICN), which proposes a shift from *host-centric* networking to *Information Centric* or *Content-centric* networking [55],[48], has been a hot trending topic in the past few years, and the focus of several projects, such as [14], [55], [13] [8],[9],[10],[11]. One of the main features of ICN networks is in-network pervasive caching [90], since the content will be the center of the communications, not a specific host.

Since the proposed CCIP and SCS caching are specifically focusing on pervasive router caches, the implementation of them in ICN networks is an interesting and related field of study. In addition, since we assume the network of our interest in this dissertation are SDN-enabled, which are believed and studied as quickest way of deploying ICN in the Internet [83], [74], [35], [79], [29], it makes the application of our proposed caching methods with ICN even more interesting.

7.2.4 Pervasive router caching in SDN-enabled network for emerging applications

There are a lot of emerging new applications in different industries, such as energy, transportation, medicine, education, public safety, etc, [12] that can take advantage of SDN and pervasive router caching. As an example, Virtual Reality (VR) is a promising area, with many applications of it already have been discussed and invested in industry. VR brings the possibility of virtual educations, virtual traveling, remote surgery, and of course great popularity in entertainment, and many more. The shift towards virtual reality can cause a huge bandwidth bottleneck that current network should be able to handle. Pervasive in-network caching has a great potential to be very helpful in such scenarios, and SDN brings an easier control and management of such networks.

In this thesis, we focused on video streaming application, in an SDN-enabled network with pervasive caching. Looking at the new emerging applications in such networks, and providing the appropriate support tools for them is an exciting possible future work, and probably a requirement for the broad adoption of those application.

LIST OF REFERENCES

[1] www.sandvine.com. Accessed: May 16th, 2016.

[2] http://ir.netflix.com. Accessed: May 16th, 2016.

[3] https://www.youtube.com/yt/press/statistics.html. Accessed: May 15th, 2016.

[4] http://about.att.com/innovation/sdn. Accessed: May 15th, 2016.

[5] https://www.geni.net/. Accessed: 2015.

[6] http://www.projectfloodlight.org/.

[7] http://openvswitch.org/.

[8] http://www.ccnx.org.

[9] http://www.sail-project.eu/.

[10] http://www.comet-project.org.

[11] http://www.fp7-pursuit.eu.

[12] https://www.us-ignite.org/apps/. Accessed: June 5th, 2016.

[13] *eXpressive Internet Architecture (XIA)*. https://www.cs.cmu.edu/~xia/.

[14] *Named Data Networking (NDN) project*. http://www.named-data.net/.

[15] *Open Networking Foundation (ONF)*. https://www.opennet-working.org.

[16] *Software-Defined Networking: The new norm for networks*. Open Network Foundation, April 2012.

[17] *Cisco Visual Networking Index: Forecast and Methodology, 2014–2019*. Cisco Visual Networking Index (VNI), 2015.

[18] V. K. Adhikari, S. Jain, Y. Chen, and Z. L. Zhang, "Vivisecting YouTube: An active measurement study," In *Proceedings of IEEE International Conference on Computer Communications (INFOCOM)*, pp. 2521–2525, March 2012.

[19] R. Ahlswede, N. Cai, S. R. Li, and R. W. Yeung, "Network information flow," *IEEE Transactions on Information Theory*, vol. 46, no. 4, pp. 1204–1216, July 2000.

[20] H. Alzoubi, S. Lee, M. Rabinovich, O. Spatscheck, and J. V. der Merwe, "Anycast CDNS revisited," In *Proceedings of ACM International Conference on World Wide Web*, pp. 277–286, 2008.

[21] D. Applegate, A. Archer, V. Gopalakrishnan, S. Lee, and K. K. Ramakrishnan, "Optimal content placement for a large-scale VoD system," In *Proceedings of ACM International Conference on Emerging Networking Experiments and Technologies (CoNEXT)*, pp. 1–12, 2010.

[22] A. Barbir, B. Cain, R. Nair, and O. Spatscheck. *Known Content Network (CN) Request-Routing Mechanisms*, RFC 3568 edition, 2003.

[23] D. Bickson and R. Borer, "The BitCod client: A BitTorrent clone using network coding," In *Proceedings of IEEE International Conference on Peer-to-Peer Computing (P2P)*, pp. 231–232, September 2007.

[24] P. E. Black. *Greedy algorithm*. Dictionary of Algorithms and Data Structures. U.S. National Institute of Standards and Technology (NIST).

[25] S. Borst, V. Gupta, and A. Walid, "Distributed caching algorithms for content distribution networks," In *Proceedings of IEEE International Conference on Computer Communications (INFOCOM)*, pp. 1–9, March 2010.

[26] Y. Boufkhad, F. Mathieu, F. De Montgolfier, D. Perino, and L. L. Viennot, "Achievable catalog size in peer-to-peer video-on-demand systems," In *Proceedings of International Workshop on Peer-to-Peer Systems*, February 2008.

[27] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and Zipf-like distributions: Evidence and implications," In *Proceedings of IEEE International Conference on Computer Communications (INFOCOM)*, volume 1, pp. 126–134, March 1999.

[28] M. Cha, H. Kwak, P. Rodriguez, Y.-Y. Ahn, and S. Moon, "I Tube, You Tube, Everybody Tubes: Analyzing the world's largest user generated content video system," In *Proceedings of Conference on Internet Measurement (IMC)*, pp. 1–14, 2007.

[29] A. Chanda and C. Westphal, "ContentFlow: Adding content primitives to software defined networks," In *Proceedings of Global Communications Conference (GLOBECOM)*, pp. 2132–2138, December 2013.

[30] P. Chou, Y. Wu, and K. Jain, "Practical network coding," In *Proceedings of Allerton Conference on Communication, Control and Computing*, 2003.

[31] A. Dan and D. Sitaram, "Buffer management policy for an on-demand video server," Research report RC 19347, IBM, 1993.

[32] A. G. Dimakis, K. Ramchandran, Y. Wu, and C. Suh, "A survey on network codes for distributed storage," *Proceedings of the IEEE*, vol. 99, no. 3, pp. 476–489, March 2011.

[33] D. DiPalantino and R. Johari, "Traffic engineering vs. content distribution: A game theoretic perspective," In *Proceedings of IEEE International Conference on Computer Communications (INFOCOM)*, pp. 540–548, April 2009.

[34] H. E. Egilmez, S. Civanlar, and A. M. Tekalp, "An optimization framework for QoS-enabled adaptive video streaming over OpenFlow networks," *IEEE Transactions on Multimedia*, vol. 15, no. 3, pp. 710–715, April 2013.

[35] P. Georgopoulos, M. Broadbent, A. Farshad, B. Plattner, and N. Race, "Using software defined networking to enhance the delivery of video-on-demand," *Elsevier Computer Communications*, vol. 69, no. C, pp. 79–87, September 2015.

[36] P. Georgopoulos, M. Broadbent, B. Plattner, and N. Race, "Cache as a service: Leveraging SDN to efficiently and transparently support video-on-demand on the last mile," In *Proceedings of International Conference on Computer Communication and Networks (ICCCN)*, pp. 1–9, August 2014.

[37] C. Gkantsidis, J. Miller, and P. Rodriguez, "Comprehensive view of a live network coding P2P system," In *Proceedings of ACM SIGCOMM Conference on Internet Measurement*, pp. 177–188, 2006.

[38] C. Gkantsidis, J. Miller, and P. Rodriguez, "Anatomy of a P2P content distribution system with network coding," In *Proceeding of International Workshop on Peer-to-Peer Systems (IPTPS)*, February 2006.

[39] C. Gkantsidis and P. Rodriguez, "Network coding for large scale content distribution," In *Proceedings of IEEE Annual Joint Conference of Computer and Communications Societies.*, volume 4, pp. 2235–2245, March 2005.

[40] L. Guo, E. Tan, S. Chen, Z. Xiao, and X. X. Zhang, "Does Internet media traffic really follow Zipf-like distribution?," *Proceedings of ACM SIGMETRICS Performance Evaluation Review (PER)*, vol. 35, no. 1, pp. 359–360, June 2007.

[41] Gurobi Optimization, Inc. *Gurobi Optimizer Reference Manual*, 2015.

[42] X. Haiyong, S. Guangyu, and W. Pengwei, "TECC: Towards collaborative in-network caching guided by traffic engineering," In *Proceedings of IEEE International Conference on Computer Communications (INFOCOM)*, pp. 2546–2550, March 2012.

[43] M. Hefeeda and O. Saleh, "Traffic modeling and proportional partial caching for peer-to-peer systems," *IEEE/ACM Transactions on Networking*, vol. 16, no. 6, pp. 1447–1460, December 2008.

[44] T. Ho, R. Koetter, M. Medard, D. R. Karger, and M. Effros, "The benefits of coding over routing in a randomized setting," In *Proceedings of IEEE International Symposium on Information Theory*, pp. 442–447, June 2003.

[45] F. Hu, Q. Hao, and K. Bao, "A survey on Software-Defined Network and OpenFlow: From concept to implementation," *IEEE Communications Surveys and Tutorials*, vol. 16, no. 4, pp. 2181–2206, 2014.

[46] H. O. II and A. Durresi, "Video over Software-Defined Networking (VSDN)," In *Proceedings of International Conference on Network-Based Information Systems*, pp. 44–51, September 2013.

[47] M. Izal, G. Urvoy-Keller, E. W. Biersack, P. A. Felber, A. A. Hamra, and L. Garces-Erice, "Dissecting BitTorrent: five months in a torrent's lifetime," In *Proceedings of Passive & Active Measurement Workshop, PAM*, April 2004.

[48] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," In *Proceedings of ACM International Conference on Emerging Networking Experiments and Technologies (CoNEXT)*, pp. 1–12, 2009.

[49] M. Jarschel, F. Wamser, T. Hohn, T. Zinner, and P. Tran-Gia, "SDN-based application-aware networking on the example of YouTube video streaming," In *Proceedings of European Workshop on Software Defined Networks*, pp. 87–92, October 2013.

[50] W. Jiang, S. Ioannidis, L. Massoulié, and F. Picconi, "Orchestrating massively distributed CDNs," In *Proceedings of ACM International Conference on Emerging Networking Experiments and Technologies (CoNEXT)*, pp. 133–144, 2012.

[51] W. Jiang, R.-Z. Shen, J. Rexford, and M. Chiang, "Cooperative content distribution and traffic engineering in an ISP network," *Proceedings of ACM SIGMETRICS Performance Evaluation Review (PER)*, vol. 37, no. 1, pp. 239–250, June 2009.

[52] L. Kangwook, H. Zhang, Z. Shao, M. Chen, A. Parekh, and K. Ramchandran, "An optimized distributed video-on-demand streaming system: Theory and design," In *Proceedings of Allerton Conference on Communication, Control, and Computing*, pp. 1347–1354, October 2012.

[53] Y. Kao, C. Lee, P. Wu, and H. Kao, "A network coding equivalent content distribution scheme for efficient peer-to-peer interactive VoD streaming," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 6, pp. 985–994, June 2012.

[54] Z. Kiraly and E. Kovacs, "A network coding algorithm for multi-layered video streaming," In *Proceedings of International Symposium on Networking Coding*, pp. 1–7, July 2011.

[55] T. Koponen, M. Chawla, B. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica, "A data-oriented (and beyond) network architecture," In *Proceedings of ACM conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, SIGCOMM*, pp. 181–192, 2007.

[56] Z. Li and G. Simon, "Time-shifted TV in content centric networks: The case for cooperative in-network caching," In *Proceedings of International Conference on Communications (ICC)*, pp. 1–6, June 2011.

[57] W. Liu, S. Yu, Y. Gao, and W. Wu, "Caching efficiency of information-centric networking," *IET Networks*, vol. 2, no. 2, pp. 53–62, June 2013.

[58] X. Liu, F. Dobrian, H. Milner, J. Jiang, V. Sekar, I. Stoica, and H. Zhang, "A case for a coordinated internet video control plane," In *Proceedings of the ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, SIGCOMM*, pp. 359–370, 2012.

[59] Y. Liu and G. Simon, "Peer-assisted time-shifted streaming systems: Design and promises," In *Proceedings of International Conference on Communications (ICC)*, pp. 1–5, June 2011.

[60] Z. Lu, Y. Wang, and Y. Yang, "An analysis and comparison of CDN-P2P-hybrid content delivery system and model," *Journal of Communications (JCM)*, vol. 7, no. 3, pp. 232–245, 2012.

[61] G. Ma, Y. Xu, M. Lin, and Y. Xuan, "A content distribution system based on sparse linear network coding," In *Proceedings of Workshop on Network Coding, Theory, and Applications (NetCod)*, pp. 1–6, 2007.

[62] G. Ma, Y. Xu, K. Ou, and W. Luo, "How can network coding help P2P content distribution?," In *Proceedings of International Conference on Communications (ICC)*, pp. 1–5, June 2009.

[63] M. Maddah-Ali and U. Niesen, "Decentralized coded caching attains order-optimal memory-rate tradeoff," *IEEE/ACM Transactions on Networking*, vol. 23, no. 4, pp. 1029–1040, August 2015.

[64] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling innovation in campus networks," *Computer Communication Review*, vol. 38, no. 2, pp. 69–74, March 2008.

[65] H. Nam, K. H. Kim, J. Y. Kim, and H. Schulzrinne, "Towards QoE-aware video streaming using SDN," In *Proceedings of Global Communications Conference (GLOBECOM)*, pp. 1317–1322, December 2014.

[66] K. Nguyen, T. Nguyen, and S. Cheung, "Video streaming with network coding," *Journal of Signal Processing Systems*, vol. 59, no. 3, pp. 319–333, June 2010.

[67] U. Niesen and M. Maddah-Ali, "Coded caching with nonuniform demands," In *Proceedings of IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 221–226, April 2014.

[68] J. Noh, P. Baccichet, F. Hartung, A. Mavlankar, and B. Girod, "Stanford peer-to-peer multicast (SPPM)-overview and recent extensions," In *Proceedings of Picture Coding Symposium*, pp. 1–4, 2009.

[69] E. Nygren, R. K. Sitaraman, and J. Sun, "The Akamai network: A platform for high-performance Internet applications," *Proceedings of ACM Operating Systems Review (SIGOPS)*, vol. 44, no. 3, pp. 2–19, August 2010.

[70] A. Pathan and R. Buyya. *A taxonomy and survey of content delivery networks*. Technical Report, GRIDS-TR-2007-4, Grid Computing and Distributed Systems Laboratory, The University of Melbourne, Australia, February 2007.

[71] A. Pourmir, A. Parakash, and P. Ramanathan, "Optimizing staggered multicasts in SDN-enabled network through distributed caching," *submitted for review*.

[72] A. Pourmir and P. Ramanathan, "Distributed caching and coding in VoD," In *Proceedings of INFOCOM Workshop on Communication and Networking for Video*, pp. 233–238, April 2014.

[73] L. Qiu, V. N. Padmanabhan, and G. M. Voelker, "On the placement of web server replicas," In *Proceedings of IEEE International Conference on Computer Communications (INFOCOM)*, volume 3, pp. 1587–1596, 2001.

[74] S. Salsano, N. Blefari-Melazzi, A. Detti, G. Morabito, and L. Veltri, "Information centric networking over SDN and OpenFlow: architectural aspects and experiments on the OFELIA testbed," *Elsevier Computer Networks*, vol. 57, no. 16, pp. 3207–3221, November 2013.

[75] R. Schollmeier, "A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications," In *Proceedings of International Conference on Peer-to-Peer Computing*, pp. 101–102, August 2001.

[76] H. Seferoglu and A. Markopoulou, "Opportunistic network coding for video streaming over wireless," In *Proceedings of IEEE Packet Video Conference*, pp. 191–200, November 2007.

[77] H. Seferoglu and A. Markopoulou, "Delay-optimized network coding for video streaming over wireless networks," In *Proceedings of International Conference on Communications (ICC)*, pp. 1–5, May 2010.

[78] S. Sharma, D. Staessens, D. Colle, D. Palma, J. Goncalves, M. Pickavet, L. Cordeiro, and P. Demeester, "Demonstrating resilient quality of service in software defined networking," In *Proceedings of IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 133–134, April 2014.

[79] D. Syrivelis, G. Parisis, D. Trossen, P. Flegkas, V. Sourlas, T. Korakis, and L. Tassiulas, "Pursuing a software defined information-centric network," In *Proceedings of European Workshop on Software Defined Networking (EWSDN)*, pp. 103–108, October 2012.

[80] B. Tan and L. L. Massoulié, "Brief announcement: adaptive content placement for peer-to-peer video-on-demand systems," In *Proceedings of Symposium on Principles of Distributed Computing (PODC)*, pp. 293–294, 2010.

[81] S. Theotokis and D. Spinellis, "A survey of peer-to-peer content distribution technologies," *ACM Computing Surveys (CSUR)*, vol. 36, no. 4, pp. 335–371, December 2004.

[82] V. Valancius, N. Laoutaris, L. Massoulié, C. Diot, and P. Rodriguez, "Greening the Internet with nano data centers," In *Proceedings of ACM International Conference on Emerging Networking Experiments and Technologies (CoNEXT)*, pp. 37–48, 2009.

[83] N. van Adrichem and F. Kuipers, "NDNFlow: software-defined named data networking," In *Proceedings of IEEE Conference on Network Softwarization (NetSoft)*, pp. 1–5, April 2015.

[84] H. Wang, J. Liang, and C. Kuo, "Overview of robust video streaming with network," In *Journal of Information Hiding and Multimedia Signal Processing*, volume 1, pp. 36–50, 2010.

[85] M. Wang and B. Li, "Network coding in live peer-to-peer streaming," *IEEE Transactions on Multimedia*, vol. 9, no. 8, pp. 1554–1567, December 2007.

[86] W. Wang, Q. Qi, X. Gong, Y. Hu, and X. Que, "Autonomic QoS management mechanism in software defined network," *China Communications Journal*, vol. 11, no. 7, pp. 13–23, July 2014.

[87] T. Wauters, W. Van de Meerssche, F. De Turck, B. Dhoedt, and P. Demeester, "Co-operative proxy caching algorithms for time-shifted IPTV services," In *Proceedings of Conference on Software Engineering and Advanced Application (SEAA)*, pp. 379–386, August 2006.

[88] A. Wolman, G. Voelker, N. Sharma, N. Cardwell, A. Karlin, and H. Levy, "On the scale and performance of cooperative web proxy caching," *ACM Symposium on Operating Systems Principles (SOSP)*, vol. 34, no. 2, pp. 16–31, April 1999.

[89] W. Xiang, G. Wu, Q. Ling, and L. Wang, "Piecewise patching for time-shifted TV over HFC networks," *IEEE Transactions on Consumer Electronics*, vol. 53, no. 3, pp. 891–897, August 2007.

[90] G. Xylomenos, C. N. Ververidis, V. A. Siris, N. Fotiou, C. Tsilopoulos, X. Vasilakos, K. V. Katsaros, and G. C. Polyzos, "A survey of information-centric networking research," *IEEE Communications Surveys Tutorials*, vol. 16, no. 2, pp. 1024–1049, February 2014.

[91] K. A. Y. Wah and R. Buyya, "Decentralized Media Streaming Infrastructure (DeMSI): An adaptive and high-performance peer-to-peer content delivery network," *Elsevier Journal of Systems Architecture*, vol. 52, no. 12, pp. 737–772, December 2006.

[92] X. Zhou and C. Xu, "Optimal video replication and placement on a cluster of video-on-demand servers," In *Proceedings of International Conference on Parallel Processing*, pp. 547–555, 2002.

[93] J. Zhuo, J. Li, G. Wu, and S. Xu, "Efficient cache placement scheme for clustered time-shifted TV servers," *IEEE Transactions on Consumer Electronics*, vol. 54, no. 4, pp. 1947–1955, November 2008.