Quality of Experience from Cache Hierarchies:
Caching for Adaptive Streaming over Information-Centric Networks

by

Wenjie Li

A thesis submitted to the
School of Computing
in conformity with the requirements for
the degree of Doctor of Philosophy

Queen's University
Kingston, Ontario, Canada
May 2019

Copyright © Wenjie Li, 2019

Abstract

Video traffic is growing in dominance in today's Internet, prompting new challenges in timely delivery of video content. This demand motivates the development of Dynamic Adaptive Streaming over HTTP (DASH), as a de facto paradigm for streaming service. DASH attempts to maximize the video consumers' Quality of Experience (QoE) under varying network conditions. However, as an application-level solution, DASH is struggling to cope with both low-latency delivery and massive service requests. The emerging Information-Centric Networking (ICN) architecture, with its ubiquitous cache hierarchies, promises a scalable adaptive streaming service. However, current ICN caching capability is aimed at generic traffic, and lacks the optimization needed for video-specific applications. As the performance of existing caching schemes is discovered to decline as consumers dynamically select content encoded at different bitrates, DASH evidently clashes with caching hierarchies, which reinforces the need to explore novel caching schemes that can incorporate bitrate adaptation, as a controlling mechanism in DASH, to serve this dominating traffic.

In this dissertation, catering to adaptive video traffic is accomplished by caching mechanisms that emphasize various QoE measurements, such as video quality, bitrate oscillation, and rebuffering. Specifically, 1) the delivered video quality is enhanced by building a traffic model to estimate access delay under varying encoded bitrates.

This delay is further used to optimize cache placement that causes the highest video throughput. 2) Bitrate oscillations and playback rebuffering are reduced due to our findings where bitrates should be prioritized in cache hierarchies. This is manifested in the design of a cache partitioning mechanism that safeguards cache capacity for specific bitrates. 3) Building on the premises of cache partitioning, the underlying dependency between cache hierarchies and bitrate adaptation is addressed for caching decisions that rely on instant video statistics. That is, instead of focusing on currently popular bitrates, bitrate adaptation predicts preferable rates in the future, and guides cache decisions that optimize the long-term performance under dynamic video requests. Extensive simulations demonstrate the effectiveness of proposed approaches in achieving significant improvements to QoE. These results suggest that cache hierarchies can better serve adaptive streaming when caching schemes capture, understand, and react to bitrate adaptation.

Co-Authorship

Journal Articles:

1. W. Li, S. Oteafy, and H. S. Hassanein. Predictive Cache Partitioning for Adaptive Streaming over Information-Centric Networks. *in preparation.*

2. W. Li, S. Oteafy, M. Fayed and H. S. Hassanein. Quality of Experience from Cache Hierarchies: Keep Your Low-Bitrate Close, and High-Bitrate Closer. *IEEE/ACM Transactions on Networking*, under review.

3. W. Li, S. Oteafy, M. Fayed and H. S. Hassanein. A Cache-level Quality of Experience Metric to Characterize ICNs for Adaptive Streaming. *IEEE Communications Letter*, 23(2), 262-265.

4. W. Li, S. Oteafy, and H. S. Hassanein. Rate-Selective Caching for Adaptive Streaming over Information- Centric Networks. *IEEE Transactions on Computers*, 66(9), 1613-1628.

Conference Publications:

1. W. Li, S. Oteafy, and H. S. Hassanein. Performance Comparison of Transcoding and Bitrate-aware Caching in Adaptive Video Streaming. in proceedings of *2019 IEEE International Conference on Communications (ICC). IEEE.*

2. W. Li, S. Oteafy, M. Fayed and H. S. Hassanein. Bitrate Adaptation-aware Cache Partitioning for Video Streaming over Information-Centric Networks. in proceedings of *2018 IEEE Local Computer Networks (LCN) Conference. IEEE.*

3. W. Li, S. Oteafy, and H. S. Hassanein. On the Performance of Adaptive Video Caching over Information- Centric Networks. in proceedings of *2017 IEEE International Conference on Communications (ICC). IEEE.*

4. W. Li, S. Oteafy, and H. S. Hassanein. StreamCache: Popularity-based Caching for Adaptive Streaming over Information-Centric Networks. in proceedings of *2016 IEEE International Conference on Communications (ICC). IEEE.*

5. W. Li, S. Oteafy, and H. S. Hassanein. Dynamic adaptive streaming over popularity-driven caching in Information-Centric Networks. in proceedings of *2015 IEEE International Conference on Communications (ICC). IEEE.*

Acknowledgments

I am writing these words down when my journey as a graduate student comes to an end. Surprisingly, although I have rehearsed so many times in my mind how I would celebrate after my Ph.D. defense, when my supervisors finally shook my hand and said congratulations, I was not feeling the same thrill as what I had expected. For the past five years, research has always been the priority in my life, and when the work was finally completed, it seems like an important part withdrew from my body and leaves an empty hole in my soul. In fact, I feel no thrill, no sadness, no relief, I feel literally nothing. Fortunately, I have invaluable memories of the kindest people whom I would never forget for the rest of my life.

There is an old Chinese saying: "He who teaches me for one day is my father for life". My deep and sincere gratitude goes to my mentor Dr. Hossam Hassanein. Your enormous support was always my source of courage, like a flashlight which lights the path for me on this journey of exploration. Your diligence and dedication to work is my perfect example which motivates me to never stop pursuing the best of the best.

Many thanks also go to another friend and advisor Dr. Sharief Oteafy. Without you, none of my current achievements would be possible. It is from you that I learned how to be a professional researcher. Although you never keep your schedule on time LOL, I will absolutely miss those days when I hid outside your office, hoping to

ambush you for a meaningful discussion.

Special thanks must be given to Mrs. Basia Palmer. When I look back on this journey, I realize you are the first person I met in the school. Through these years, you have been so reliable and trustful, a person whom I can share little secrets with. Thank you for all the kindest support not only helping me through my research but also my life in this beautiful city.

How would I ever forget those closest friends from the Telecommunications Research Lab (TRL)? I thank God who arranged the kindest and smartest people to surround me. Thank you Ramy Atawia for bringing so many smiles in my life. I will pray for you and Mariam to enjoy the happiest life in Ottawa. Same blessing goes to Hesham Farahat. I appreciate so much your endless encouragement during my job hunting. Your excellence is unbeatable and I am sure of great success in your career. I would also thank Adel Ibrahim, Abdalla Abdelrahman, Amir Ibrahim, Ashraf Alkhresheh, and Galal Hassan for your company which made my life colorful in the lab. I would like to send my best wishes to Faria Khandaker, Sara Elsayed and Mary Zarif Riad, I have confidence that you all will be able to conqueror any difficulty in your future research.

For the past five years. I have been blessed to meet the best friends in my life. Heng Li, Yu Zhao, Ke Xu, Dayuan Wang, thanks for giving me this opportunity to know you and for bringing me so much joy. I cannot imagine how I can ever say goodbye. Sometimes you never know the value of a moment until it becomes a memory, and I would like to promise that I will never lose those memories.

I also want to say how I appreciate the thesis committee: Prof. Juergen Dingel, Prof. Ali Etemad, Prof. Alfredo Grieco, and Prof. Ahmed Hassan. Thank you very

much for your constructive feedback and support. Your comments provided me many insightful suggestions on my thesis work and future directions.

Last but not least, I would never be able to make this achievement without the love from my family. Moving to a different country is always difficult. Your love gives me hope and strength that supports me through many freezing nights in Canadian winter. I would also like to say to my other half, that although you have not shown up yet, I wish you are just right there waiting for me on the next journey I take. I am looking forward to the next challenge and I promise to live a meaningful life.

Statement of Originality

I hereby certify that this Ph.D. thesis is original and that all ideas and inventions attributed to others have been properly referenced.

Contents

List of Tables

List of Figures

List of Acronyms

CATT Cache Aware Target idenTification.

CBC Centrality Based Caching.

CCN Content-Centric Network.

CDN Content Delivery Network.

CE2 Cache Everything Everywhere.

CINC Cooperative In-Network Caching.

DASH Dynamic Adaptive Streaming over HTTP.

ICN Information-Centric Network.

LCD Leave Copy Down.

LFU Least Frequently Used.

LRU Least Recently Used.

MARL Multi-Agent Reinforcement Learning.

MCD Move Copy Down.

MDP Markov Decision Process.

MILP Mixed Integer Linear Programming.

MOS Mean Opinion Score.

MPD Media Presentation Description.

NDN Named Data Networking.

OSPFN Open Shortest Path First routing scheme for Named-data.

ProbCache Caching with Probability.

QoE Quality of Experience.

QoS Quality of Service.

Chapter 1

Introduction

The rapid evolution of Internet-based video dissemination is mandating novel paradigms to scale with the volume of traffic and heterogeneity of users. By 2021, video content is projected to dominate over 80% of global Internet traffic [10]. This massive amount of data has motivated many developments in application-layer standards, yielding Dynamic Adaptive Streaming over HTTP (DASH) [57] as a leading contender in delivering video content over unstable channel conditions. However, researchers have long argued that improvements are needed across the Internet protocol stack, to cope with this evolution [47, 18].

DASH has played a major role in improving video delivery services. However, it is intrinsically an application-layer solution, and thus cannot resolve the underlying network scalability problem or ease bottlenecks caused by massive video traffic. A recent focus has shifted towards Information-Centric Network (ICN) [24], as a promising solution from a network architecture perspective. ICN is regarded as a next-generation Internet, which builds upon a *Publish-Subscribe* model [1], enabling content-host decoupling, dynamic request forwarding, and adaptive in-network caching. These features make video streaming a scalable service in ICNs. There have been increasing

calls [64, 33] to adopt dynamic adaptive streaming as an essential component in this future Internet paradigm.

In principle, DASH provides a time-shift control on media requests in reaction to varying bandwidth conditions experienced by each user. At its core, DASH adopts three fundamental features in its operation: video content is first partitioned into equal duration segments, all segments are encoded at multiple bitrates, and an adaptive control algorithm is applied from the consumer side to request the highest possible quality given estimates of real-time network bandwidth.

Among all ICN features, in-network caching received the most attention. ICN exploits caching as a networking primitive by equipping each router with caching capacity, instead of pre-designated, sparse and static surrogate servers. In the context of video streaming, in-network caching can significantly leverage users' Quality of Experience (QoE) [45, 40]. As a result, higher resolution video content, which may grow difficult to retrieve without caching, will effectively become accessible.

Many research efforts have investigated ICN caching [69]. However, catering to adaptive streaming in ICNs presents significant challenges. Although popularity-based caching is known to improve the performance in terms of average cache hit ratio or reduce overall access delay [24, 9, 49], ICN caching research seldom sought to optimize QoE indicators, which essentially reflect consumer-side satisfaction, as a core objective in adaptive streaming applications.

Understanding and optimizing QoE performance demands unique ICN caching design. For instance, the interplay between cache placement and consumer-side bitrate adaptation introduces 'oscillation dynamics' [64], which overshoots what generic

caching schemes could handle. Intrinsically, this oscillation is linked to inaccurate estimates caused by the ever-changing network conditions, that occur with intermittent cache hits and misses. A 'good' caching mechanism for adaptive streaming must capture the impact from bitrate adaptation, and accordingly optimize its decisions that adapt to dynamic video requests in the long term.

In this thesis, we demonstrate how our designs tackle the impact from bitrate adaptation and optimize for various QoE measurements, such as video quality, bitrate oscillation, and rebuffering. Extensive experiments demonstrate the achievements of our approaches in QoE, which reinforces the motivations and importance of this research.

1.1 Research Statement

We believe that:

> ICN's in-network caches will improve QoE when caching decisions capture and react to bitrate adaptation.

1.2 Thesis Contributions

The major contributions of this thesis are listed as follows:

1. *Video Throughput Improvement.* We propose a benchmark *DaCPlace* and a heuristic *StreamCache* scheme to improve average video throughput received by consumers. As video quality has the highest correlation with overall QoE [14] and bitrate adaptation decides video quality based on input throughput, we thus choose to optimize video throughput. DaCPlace is accomplished by building an

adaptive video traffic model to analyze the queueing delay differed by encoded bitrates.

2. *Bitrate Oscillation Reduction.* We design a benchmark *RippleClassic* and a heuristic *RippleFinder* scheme to reduce rapid bitrate oscillation caused by intermittent cache hits and misses. RippleClassic and RippleFinder leverage a novel idea called *cache partitioning* where cache capacity of each router is divided and safeguarded for a certain bitrate. These two schemes are designed under the guidance of a caching principle *RippleCache* where high-bitrate content is placed at the network edge and low-bitrate content is pushed into the network core. Our results demonstrate that cache partitioning schemes deliver content that suffers less oscillation and rebuffering, as well as the highest levels of video quality, indicating overall improvements to QoE.

3. *Predictive Cache Partitioning.* We investigate the dependency between cache placement and bitrate adaptation when caching decisions only rely on instant video statistics and result in degrading performance. In remedy, we propose *PredictiveRipple* which applies bitrate adaptation prediction to optimize the cache partitioning performance in the long term. PredictiveRipple is accomplished by a Multi-Agent Reinforcement Learning (MARL) modelling, where the $\mathcal{Q}$-learning [55] is the driven approach that guides the cache partition adjustments. Our results show PredictiveRipple overcomes the overfitting problem which is rooted in popularity-based caching schemes, and further enhances consumers' QoE.

1.3 Thesis Organization

In this chapter, we present the motivation of the primary research problem, and highlight the major contributions towards bitrate-adaptation-aware caching. The rest of this thesis is organized as follows.

Chapter 2 introduces background and related work. We review the fundamental concepts of DASH and directions of ICN caching studies. At the end of this chapter, we explain popular caching paradigms for adaptive streaming and conduct preliminary experiments to give a comprehensive comparison on QoE performance.

Chapter 3 first highlights the difference between this thesis and previous work. We then describe the system settings, based on which the benchmark solution DaCPlace is formulated. DaCPlace optimizes the delivered video quality by reducing the video access delay per bit. A low-overhead scheme StreamCache mimics the execution of DaCPlace and reduces the complexity by online measurement.

Chapter 4 highlights our novel cache partitioning concept. We start with the reasons for cache partitioning, followed by a RippleCache principle that guides how to partition. To validate RippleCache claims, we construct two separate implementations. The first is RippleClassic, which is a benchmark solution that optimizes content placement by maximizing a measure for cache hierarchies shown to have high correlation with QoE. The second is a lighter-weight RippleFinder, with distributed execution for application in large-scale systems.

Chapter 5 builds on the cache partitioning design and presents the PredictiveRipple system for bitrate adaptation prediction. PredictiveRipple uses a MARL framework to derive cache partition adjustments. We resolve coordination and exploration issues in MARL which enables distributed and efficient *online* training.

Chapter 6 presents a summary of the topics addressed in this thesis and discusses future research directions.

Chapter 2

Background and Overview

In the first part of this chapter, we elaborate on the DASH protocol and its fundamental bitrate adaptation control algorithm. In the second part of this chapter, we explain the specifics of ICN architectures and give an overview of literature in ICN caching studies.

2.1 Dynamic Adaptive Streaming

In dynamic adaptive streaming, video files are encoded in different bitrate levels by producers. These files are chopped into segments with equal duration. On the user end, each video player would request video segments, under a chosen bitrate, based on its implementation of a bitrate adaptation control algorithm. This choice can change bitrates over time, in response to varying link conditions in order to retrieve video content with the best possible quality [33].

As shown in Figure 2.1, if the user's request requires bandwidth which exceeds the amount that current network can provide, dynamic adaptive streaming changes the video quality on-the-fly to lower quality automatically. When the link condition improves, DASH would recommend that the current request switches to higher quality.

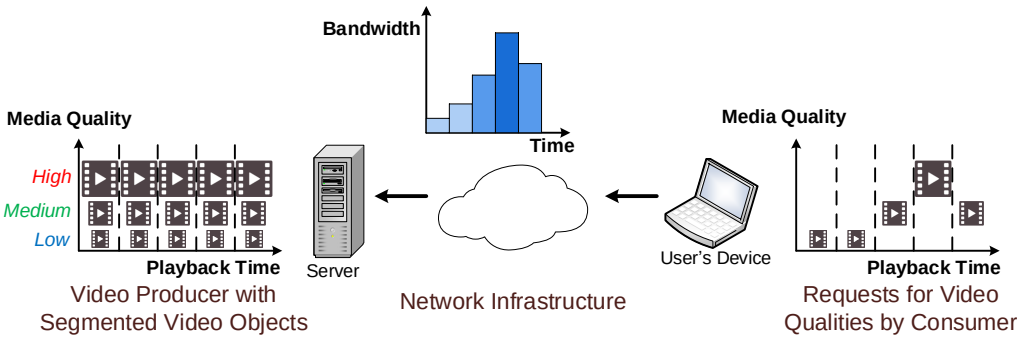


Figure 2.1: Concept of dynamic adaptive streaming

2.1.1 Bitrate Adaptation Control

DASH is driven by the adaptation engine, as shown in Figure 2.2. In this engine, an adaptation control logic makes bitrate choices based on measurements of the most recent video segments. The purpose of this adaptation is to minimize the impact of network bandwidth variations and improve users' QoE.

Bitrate adaptations are designed to handle network bandwidth fluctuations, caused by traffic congestion, simultaneous requests, and so on. In the context of ICN, the hierarchical caching structure is another source of bandwidth fluctuation since the cache hit and miss on consecutive video requests would result in significantly different access delay and throughput. This severe fluctuation is more likely to make the requested bitrates switch up and down. Recent studies on bitrate adaptation tackled this problem by seeking a balance between bitrate switch efficiency and stability.

Two types of approaches, throughput-based and buffer-based [31], are widely applied to achieve this balance of efficiency and stability. Throughput-based methods, such as [42, 26], have smooth adaptation. Typically, extra conditions have to be

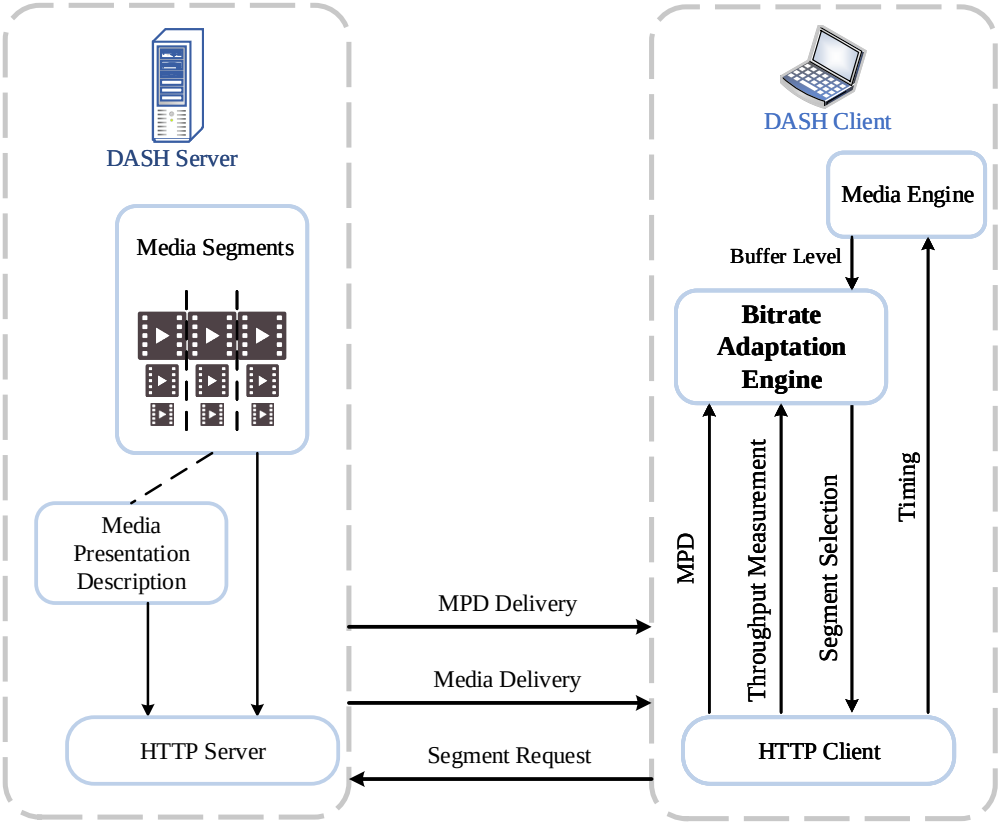


Figure 2.2: Process of dynamic adaptive streaming over HTTP

satisfied before any bitrate switch occurs. For instance, in FESTIVE [26], a different bitrate would be selected if it yields a higher combined efficiency and stability score than the current bitrate. Note that bitrate adaptation can only occur between the next higher or lower video quality level so that users would not notice a sudden video quality change.

Buffer-based adaptation methods [23, 58, 13] utilize the buffer occupancy, which is the current data size in the video playback buffer, as a indicator that guides bitrate selection. This type of approach is designed based on an argument that the buffer occupancy captures the relationship between requested video quality and network

bandwidth. For instance, BBA [23] makes different adaptation decisions at different buffer levels. When the video playback buffer is close to empty, the bitrate adaptation attempts to be conservative and prefers low quality video content. Conversely when the buffer is close to full, the bitrate adaptation would then become aggressive and prefer high quality content.

2.1.2 Standardized QoE Metrics

QoE combines users' experience with non-network-specific parameters, such as glitches, artifacts, and excessive waiting times [54]. However, another important concept, Quality of Service (QoS) is represented by network-specific parameters, such as throughput, packet drop rate, and latency. Obviously, QoS problems imply QoE problems. The quantitative relationship between QoE and QoS could be found in [17], where QoE and QoS parameters are connected via an exponential relationship.

For adaptive video streaming applications, the DASH Industry Forum [56] has published a set of QoE standard metrics. Several representative metrics adopted in this thesis are listed below.

- **Rebuffer Count**: the number of video playback freezing that has occurred during a given time window (e.g., within a entire video session).
- **Rebuffer Percentage**: the time spent in a playback freezing state over a given time interval.
- **Average Video Quality**: the average video bitrate requested by the consumer.
- **Bitrate Switch Count**: the number of times the requested bitrate changed in a given time interval.
- **Bitrate Switch Rate**: the rate of bitrate switches in a given time interval.

2.2 ICN Architectures and Components

In recent years, an increasing concern for the scalability and efficiency of the Internet has resulted in the emergence of ICN. The core premise is adapting to content/information, and catering to both consumers and producers, rather than adopting the client-server approach, which requires the establishment of end-to-end connection, following the host-centric architecture.

ICN differs significantly from the current Internet in the following aspects [1, 65]:

• **Location Decoupling**. ICN is built upon the *Publish-Subscribe* model [1]. The information source advertises its content through *Publishing* to the network and users retrieve what they need via *Subscribing*. There is no need to know the exact destination of the information (i.e., IP address) since the routing of request and data are autonomous in ICN. Publishers do not need to know how many subscribers are interested in the content, which guarantees the space decoupling between consumers and producers in ICN.

• **Naming**. Information in ICN is identified by its unique 'Name' rather than being bound to a known location. Requests have no explicit destinations to retrieve the corresponding data. Instead, the best available data source is automatically chosen by ICN. Naming is designed at the network layer, which enables ICN to adjust routing hop-by-hop and exploit in-network storage optimally. In the literature there are several naming schemes. A name of content could be flat or hierarchical, readable or un-readable. A name could look like an HTTP request in a hierarchical design or merely a string of seemingly meaningless numbers.

• **Routing**. ICN routing can be generally categorized into two types. In the first type, routing depends on name resolution. Similar to the Domain Name System

(DNS) service of the current Internet, ICN name resolution matches an information name to a producer that can supply that information. To help routing a request, the name resolution service creates a source-route, or installs routing states on nodes. In the second type, each ICN node adds or edits its own routing table based on the received cache status from other routers in the neighbourhood. This type of routing adds more information sources in addition to producers, and improves the utilization of in-network caching.

- **Ubiquitous Caching**. ICN exploits caching as a networking primitive. Each router is equipped with cache storage. This storage contains recent data which are transmitted through the router and would be refreshed based on a predefined or application-specific cache policy. ICN can be considered as a distributed cache architecture.

The ICN architecture has gained attention from both academia and industry. There are several on-going projects which implement the basic principles of ICN, such as Publish-Subscribe Internet Routing Paradigm (PSIRP) [19], Network of Information (NetInf) [12], Data-Oriented Network Architecture (DONA) [29], Content-Centric Network (CCN) [25] and Named Data Networking (NDN) architecture [68]. Among these projects, the CCN architecture proposed by Jacobson et al. [25] stands out as a prominent architecture that facilitates rapid benchmarking and insightful performance analysis. The proposed schemes are designed and evaluated on the NDN architecture, and can be adapted to work under any other ICN design.

2.3 ICN Caching Research

Ubiquitous caching [69] is a fundamental component of ICN, and could effectively reduce redundant data traffic generated by duplicated requests. Thanks to the content and location decoupling made by the naming mechanism of ICN, users' requests can be satisfied by not only the producer but also the cache of any intermediate routers. As a result, ICN can alleviate the pressure on the network architecture from the rapid growth of data traffic.

However, ICN caching capability differs from traditional web caching, which brings new challenges. In this section, we elaborate on the differences between caching architectures, and expand on two main research topics of ICN caching: Cache Decision Policy (Section 2.3.2) and Cache Capacity Allocation (Section 2.3.3).

2.3.1 ICN Caching vs. Web Caching

The web caching problem has been extensively studied in the 1990s, and most of the research conducted was to improve the web content availability. However, as the Internet transforms from a host-centric to a content-centric network, significant differences arise between existing web caching and ICN caching [69]. These differences are summarized in Table 2.1.

ICN caching is designed to provide better information accessibility than web caching, in terms of ubiquity, granularity, and transparency. Compared to web caching, ICN caching allows each node in the network to cache information, instead of designated nodes as surrogate servers in web caching. Besides, ICN caching provides a finer caching granularity where content is divided into chunks and a portion of content can be cached as needed.

Table 2.1: Differences between ICN caching and web caching

Features	ICN Caching	Web Caching
Ubiquity	Any router in the network can be cache node. Caching points are not fixed.	Cache locations are predetermined and optimized based on the type of topology.
Granularity	Content is divided into chunks. Caching nodes selectively store pieces of content, which results in finer granularity.	Web content must be cached as an entire object.
Transparency	Caching decisions are made upon content names, which is independent of applications.	Caching is not a transparent service and targets a particular traffic class/application in a specific domain. Copies in different domains are logically separated.

2.3.2 Cache Decision Policy

Current research on cache decision policy needs to answer two fundamental questions, first *what to cache* or which content should be stored in cache, and second *where to cache* or which router should be selected to cache a particular content.

Based on whether routing and caching work in a collaborative manner, caching decision policies can be categorized into two types, *on-path* caching and *off-path* caching. In on-path caching, the choice of routing path is decided by name resolution service only, and would not take the caching status into consideration. As a result, only the caching storage located on each routing path can be utilized. Off-path caching by comparison, depends on cache discovery: each node would detect the cache status from its neighborhood/nearby nodes, and later build dynamic routing paths to these routers.

Although on-path caching requires no coordination with routing, nodes along the

default routing path still need to advertise the cache status to facilitate the caching decision. This advertisement will effectively reduce the policy design complexity. Under the umbrella of on-path and off-path caching, the policies can be further categorized into *cooperative* and *non-cooperative* caching. Depending on whether there exists message exchange on cache status, cooperative caching policies can be further classified into *explicit* and *implicit* cooperation. As their names imply, explicit caching policies require advertising the cache status whenever the cache holds or evicts a certain content; implicit caching on the contrary requires no cache status exchange but usually needs extra operations to infer the transition of states in the cache.

On-Path Caching

- **CE2**. Cache Everything Everywhere (CE2) [25] is a widely applied policy, which is also the default caching policy in CCN/NDN. Contents can be cached on any router along the routing path, in order to minimize the upstream bandwidth demand and downstream latency. Accordingly, CE2 is a non-cooperative caching decision policy, and is prone to causing serious cache redundancy since every cache on a certain routing path holds a copy of requested content, and content will always be served from caches in downstream routers when there is a request, which makes the copies on upstream nodes redundant.
- **ProbCache**. Caching with Probability (ProbCache) improves the cache efficiency and hit ratio. ProbCache [49] determines different caching probability p to ICN routers, based on the length of a routing path. In particular, each request adds a *Time Since Inception* (TSI) field to its packet header. The TSI value increases by one as a certain router relays this request to the next hop. When this request reaches the

content producer, TSI then represents the length of a routing path. A *Time Since Birth* (TSB) field, along with the TSI retrieved from the request, is added to the header of a data packet by the producer. The TSB value also increases by one at each hop when the data packet is delivered back to the consumer. Any router on the routing path then uses $\frac{TSB}{TSI}$ to derive the caching probability.

ProbCache assigns a different caching probability to routers according to their locations on the routing path. As to a router near the consumer, the TSB value would be close to TSI which results in a higher caching probability close to one. As to routers near the content producer, caching probability would be close to zero. A small probability makes sense since there is no need to cache around the producer. In addition, ProbCache takes the different cache sizes of routers into consideration. The caching probability will also be adjusted according to the remaining cache on the data delivery path. As each router participates in changing TSI and TSB, we consider ProbCache to be an implicit decision policy.

• **LCD and MCD**. Leave Copy Down (LCD) [32] is another approach which improves the performance of CE2, not by assigning caching probability but by restricting which routers can cache content. In LCD, a caching suggestion bit is added to the data packet header, and will be turned on once a request for this data arrives at the producer or any cache node. A router on the data delivery path can cache the data packet only if the suggestion bit is on, and the bit will be switched off after the corresponding data is cached to prevent additional caching on downstream nodes. Therefore, every time there is a cache hit, the content can move one hop towards the consumer. Popular content thus can be stored at the network edge after several retrievals while less popular content is prevented from occupying cache space.

However, when moving towards the edge, duplicate copies of content are left on the upstream routers which results in cache redundancy. Move Copy Down (MCD) [32] works in a similar way to LCD and eliminates the on-path redundancy by deleting the cached copy when data is moved downstream. Both LCD and MCD are implicit schemes as the caching suggestion is transmitted piggyback through data packets.

- **WAVE**. Cho et al. proposed WAVE [9], which utilizes the suggestion flag bit to facilitate caching decisions. As shown in Table 2.1, ICN caching is performed on a finer granularity, which is a significant difference compared to traditional web caching. WAVE highlights this feature of ICN caching by considering the inter-chunk relationship. The number of chunks in a content object, where the caching suggestion bits should be switched on, is determined based on a *Chunk Marking Window* (CMW), where the size of the window increases exponentially according to the request frequency of this content. For example, when a certain content object is requested the first time, CMW is set to 1 by the content producer, which means only the first chunk in the object would be moved one-hop towards the consumer. When the content producer receives the second request for the same object, CMW is set to two (suppose the base is two), which represents that the second and third chunks would be moved from content producer to the cache in the next hop (if the third request is received, CMW is then set to four). As WAVE is inherited from LCD, WAVE is also an implicit caching decision policy.
- **CBC**. Chai et al. proposed Centrality Based Caching (CBC) [8] based on empirical results that random caching at a single router along the content delivery path can achieve almost the same performance as ubiquitous caching when LRU eviction is applied. Thus, CBC solves the problem of choosing cache locations, which exploits

the topological feature to ensure high cache hit ratios. CBC measures the *centrality betweenness* of each router, and chooses nodes with the highest betweenness value to cache content.

The rationale behind such design is that a node with high centrality betweenness, is more likely to get a cache hit. CBC computes the centrality betweenness offline and attaches the largest betweenness value among routers on the routing path to the header of the request. The corresponding data packet will copy this value to its header and a router will cache the data only if the centrality betweenness of this router matches the attached one in the packet header. Since CBC relies on this betweenness value to coordinate caching, CBC is regarded as an implicit policy.

Off-Path Caching

• **CINC**. Cooperative In-Network Caching (CINC) [41] reduces the cache redundancy by distributing the cached content based on a hash function. Each router is assigned a label which is calculated offline with the objective of minimizing the total distance to neighbourhood nodes of different labels. Data packets also carry a sequence number and can only be cached on routers where the router label matches the hashed sequence number. As to a certain router, it is expected that there is no duplicate content existing in the one-hop range. To implement this design, CINC adds a new *Collaborative Router Table* (CRT), which records the neighbourhood routers and corresponding labels. ICN routers would forward data packets to the appropriate neighbour for caching, according to CRT.

To utilize the cached content in the neighbourhood, CINC adds a second table

in each router called *Collaborative Content Store* (CCS), which records the neighbourhood routers, the cached names, and the output interfaces to reach the caches. Requests relayed from any router would refer to this table first instead of following the default routing path discovered by name resolution service. This explains why CINC is categorized under off-path caching.

In non-cooperative caching, the availability of information is discovered via name resolution only. In cooperative caching, the cache consistency problem arises due to the frequent insertion and deletion of content at different caches. CINC solves this consistency issue by inferring the caching status of neighbourhood routers and updates the CCS table based on the incoming requests. If a request comes from a neighbour node where the corresponding entry exists in CCS, it means the content has been evicted by the neighbourhood router.

- **Intra-AS**. The Intra-AS cache cooperation scheme [60] is similar to CINC in terms of neighbourhood collaboration. Requests are redirected to a neighbouring node to achieve better cache hit ratio. Intra-AS is different from CINC in two aspects. First, the collaboration between routers is made by exchanging the cache status summary. Second, the information availability is improved by tracking a *Minimum Dominating Set* (MDS), instead of applying the hash-based mapping. Only a subset of routers, which are reachable by any node in the network, would be selected to cache content.
- **CATT**. Cache Aware Target idenTification (CATT) [15] is a potential-based caching scheme. Each cache router would initially assign a *potential value* for content. This potential value is then broadcast inside the network, and will decrease as the hop distance from the origin router increases. This value represents the 'attraction' to requests. A route is selected by the node to forward a request, only if the route receives

the highest potential value. The evaluation results show that CATT can significantly reduce the access latency.

From above descriptions, the design of ICN caching decision policy usually combines features from different categories, such as off-path with cooperative caching. With support from the ICN architecture, the caching policy can be customized to adapt to different settings and network conditions.

2.3.3 Cache Capacity Allocation

Prior studies on cache capacity allocation attempted to answer how to allocate the cache capacity among ICN routers in order to reach the maximal cache hit ratio under limited cache budget. Some researchers advocate adjusting cache capacity dynamically in real time. Wang et al. [62] modelled the optimal cache allocation with 0-1 knapsack problem and solved it with dynamic programming. Their solution adjusts the cache capacity of routers based on the content popularity. They conclude that more cache space should be allocated on the edge when dealing with frequently requested content, while intermediate routers should cache more of the less popular content to reduce cache redundancy.

Another direction argues against ubiquitous caching in ICN. Fayazbakhsh et al. [16] claimed that caching at the edge of the network yields almost the same performance gain as ubiquitous caching. Their discovery was reinforced in [11], where an analytical model derived a similar result as [16]. However, these contributions overlook an important issue of content redundancy due to edge caching, especially with limited cache budgets and a large number of edge routers. Their performance was evaluated under the simplistic Least Recently Used (LRU) scheme, which differs significantly

from popularity-based and application-specific caching schemes.

2.4 Caching for Adaptive Streaming

The importance of in-network caching for adaptive streaming is well recognized. However, there is no consensus on how to manage these resources to maximize their utilization. To date, two main paradigms have been explored, one where caches are distributed across the entire network and one where caching nodes are only allocated at the network edge. Evidently, both models have their own merits. Ubiquitous caching on network intersections would effectively reduce traffic load in the core network, but would in turn be unable to fulfill requests closer to consumers. Edge caches, on the other hand, would result the least access delay, but at the same time result in a higher degree of cache redundancy and require more caching capacity.

Ubiquitous and edge caching are both adapted to cater to ever-expanding video streaming applications. With recent advancements in mobile/edge computing, computational resources are moving from the cloud to the network edge, which enables on-the-fly video transcoding at edge caches. For example, a representative transcoding paradigm [27] describes that network edge nodes may only cache the highest quality content and transcode requests to lower qualities when the demand on low bitrate content arrives. Ubiquitous caching, instead, relies on bitrate-aware caching schemes to enhance multi-bitrate streaming. For example, smooth playback can be achieved by safeguarding a network of caches for particular bitrates along each forwarding route [39]. However, deciding which one of these paradigms is superior in performance, has been a long-standing question.

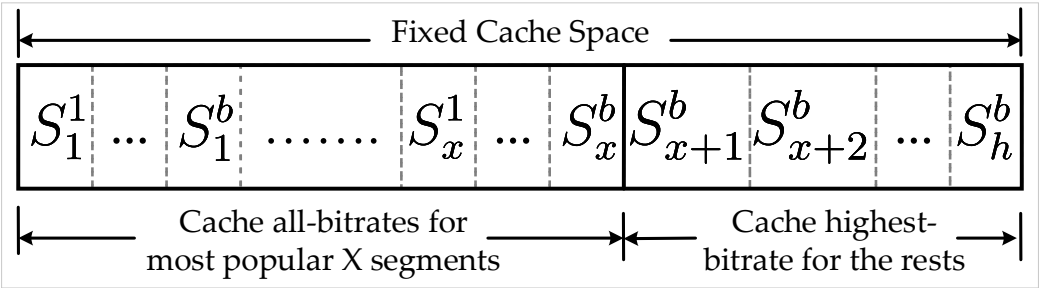


Figure 2.3: Illustration of a transcoding scheme. All-bitrate versions are cached for the most popular x video content, while only the highest version is stored for the rest.

2.4.1 Transcoding on Edge Cache

Grandl et al. [20] pointed out a core challenge of caching for adaptive streaming, whereby the same content encoded in different bitrates competes for limited caching storage. The potential of edge caching shown in previous studies [16, 11] motivated transcoding at the network edge to enhance adaptive streaming service. Specifically, Grandl et al. proposed *DASH-INC*, which only caches the highest bitrate of video on the network edge, arguing that requests for lower bitrates would be serviced via transcoding at each router. However, as transcoding has to be done almost on a per-request basis, one cannot simply assume that with the increasing demand for video content, in-node processing would be scalable for real-time requests.

Built on the prototype introduced in [20], Jin et al. proposed the adoption of a partial transcoding model [27] shown in Figure 2.3, as a hybrid compromise. In this model, each edge node selectively caches all bitrate versions for popular video content, and only keeps the highest quality for the rest (as long as the cache capacity allows it). As a result, transcoding only needs to be performed for unpopular content which reduces the in-node processing load. The partial transcoding ratio then becomes a critical parameter which decides how to divide the caching space for these two

different usages. This ratio is derived by minimizing the total cost of caching system.

There are three types of costs: *Storage*, *Transcoding* and *Bandwidth*. The storage cost is charged by allocating cache space to each edge node. The transcoding cost is incurred when the highest bitrate content is transcoded into any lower bitrate version. The bandwidth cost is triggered by retrieving content from a video producer when a cache miss occurs at the network edge. The optimal partial transcoding ratio then depends on the trade-off between these three types of costs. However, different cost models, which focus on various performance metrics, such as energy consumption, access delay, and throughput, would result in far different results. For example, CPU usage becomes a significant component of transcoding cost in [70]. Finding a generic cost model for transcoding is still an open issue, as it varies from the underlying hardware and topology.

2.4.2 Bitrate-Aware Ubiquitous Caching

With the advent of next-generation routers that are capable of caching, there are a number of networking paradigms considering a wider-adoption of caching at its routers. While ICNs have this feature inherent in its design, the expansion of adaptive Content Distribution Networks, with more agile models for allocating caching resources across networks, are opening up new frontiers in network-wide caching. Ubiquitous caching [24] is a fundamental feature of ICN. Due to the decoupling of content and location in ICN naming mechanisms, information is not bound to a particular host and can be retrieved from anywhere in the network.

The interaction between in-network caching and bitrate adaptation has attracted the attention of researchers, leading to studies on bitrate-aware caching schemes.

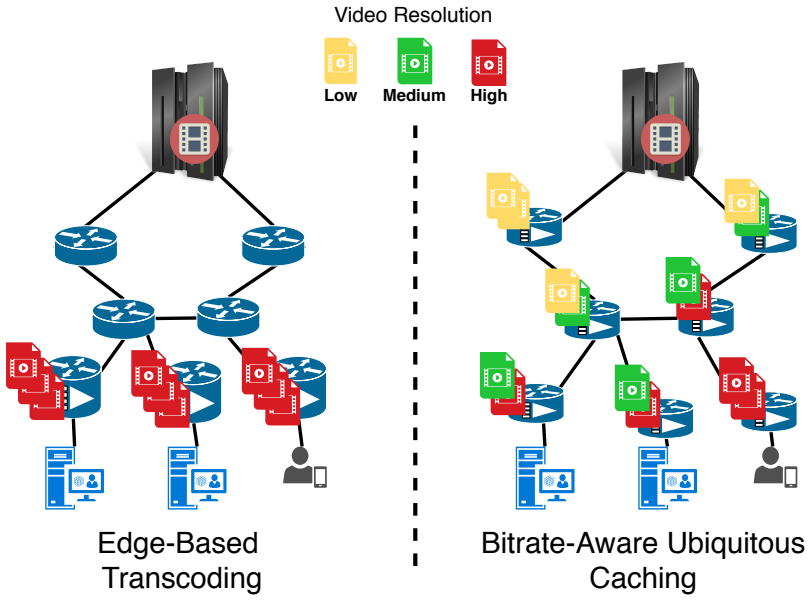


Figure 2.4: Contrasting the caching behavior of ubiquitous bitrate-aware caching vs. Edge-based caching with transcoding. In the latter paradigm, more caching resources need to be allocated at the edge to cater for storing most of the popular video segments at their highest bitrates.

Araldo et al. [3] attempted to build a more direct relationship between video quality and cache placement of multiple versions of video content. Lee et al. [36] extended the principal of value-based caching schemes and applied it over different layers of bitrates to increase cache reuse ratio. Kreuzberge et al. [30] solved the challenge of unfair bandwidth sharing and proposed a cache-aware traffic shaping policy for adaptive streaming.

2.4.3 Transcoding vs. Bitrate-Aware Caching

Although both transcoding and bitrate-aware ubiquitous caching are promising, their impact of each on consumers' QoE were never compared under realistic adaptive

streaming scenarios. It was our intuition, given the inherent scalability and adaptiveness to content, that ubiquitous in-network caching would outperform edge-based transcoding in most scenarios. After building our ubiquitous caching model, presented in Chapter 4, we carried out extensive experiments to contrast the performance of both paradigms. It was evident that ubiquitous in-network caching of multiple bitrates was mostly superior to transcoding models. Our detailed comparisons and experiments are detailed in Appendix A. A brief description of the performance comparison is provided below.

The operational difference between these two caching paradigms are depicted in Figure 2.4, where all cache resources are only allocated at edge nodes for transcoding and distributed evenly across all nodes for bitrate-aware caching. To highlight the potential gain of transcoding at the edge [27], in our experiments we assume zero processing delay of transcoding, to cater for an upper bound performance that the edge caching paradigms may achieve. This performance is compared against the best known bitrate-aware caching scheme. We discover that even under the assumption of zero transcoding delay, bitrate-aware caching can often match or even outperform the upper bound performance of transcoding across various bandwidth patterns, cache capacity, and popularity skewness measures. Based on our observations, bitrate-aware caching is more suitable to serve consumers with fixed and dedicated link capacity when cache resources are constrained. Online transcoding instead shows more potential to serve mobile consumers when there is a significant amount of caching space and computational power at the edge, but the ultimate performance must be verified in real applications where transcoding delay cannot be ignored.

Chapter 3

Bitrate-Selective Caching for Throughput Enhancement

3.1 Introduction

The inherent capacity of ICN to dynamically cache content enables novel models for provisioning different services, contents, and interest groups. Many research efforts have already investigated ICN caching [69], where typical schemes (e.g., [63, 53, 49]) target minimizing hop counts or cache-miss rates, in order to improve the network performance for adaptive streaming service.

Both hop distance and cache-miss rate are generic performance metrics. However, in adaptive streaming application, a consumer-centric QoE measure is more straightforward than generic metrics to reveal the essential satisfaction level of streaming service.

QoE includes many objective features, such as video quality, playback freezing and bitrate oscillation. While a Mean Opinion Score (MOS) is used to represent the overall QoE from a subjective perspective, to evaluate the performance of any streaming system, several researchers [14, 44] attempted to build a QoE model that

maps the combined objective QoE features to a subjective MOS. Although much progress has been made, there is still a lack of consensus on a standard QoE model. We adopt the approach of looking into each objective QoE metric in order to provide a comprehensive evaluation on the proposed caching system.

Among all QoE features, Duanmu et al. [14] discovered that video quality has the strongest correlation with overall satisfaction. Since video throughput has a direct impact on bitrate adaptation decisions, a higher video throughput would mostly trigger requests for better video quality that enhances consumers' QoE. Thus, we address the core caching problem for adaptive streaming by optimizing the video throughput. To this end, we

- develop *DaCPlace* as a benchmark caching scheme, generalized to scenarios where consumers demand adaptive bitrates. The objective of DaCPlace is to minimize access delay per bit of requested video, as a indicator of overall video throughput.

- perform a rigorous analytical model for adapting to heterogeneous requests in future Internet applications, and map them on ICN architecture to facilitate in-depth analysis of caching performance influenced by bitrate adaptation.

- design a heuristic scheme, *StreamCache*, where each router makes caching decisions using local statistics exchange to achieve low-overhead cache placement. Comprehensive simulations are conducted and demonstrate how, under various network settings (e.g., available cache storage and popularity distribution), StreamCache achieves near-optimal performance in contrast to DaCPlace.

The remainder of this chapter is organized as follows. Section 3.2 reviews related

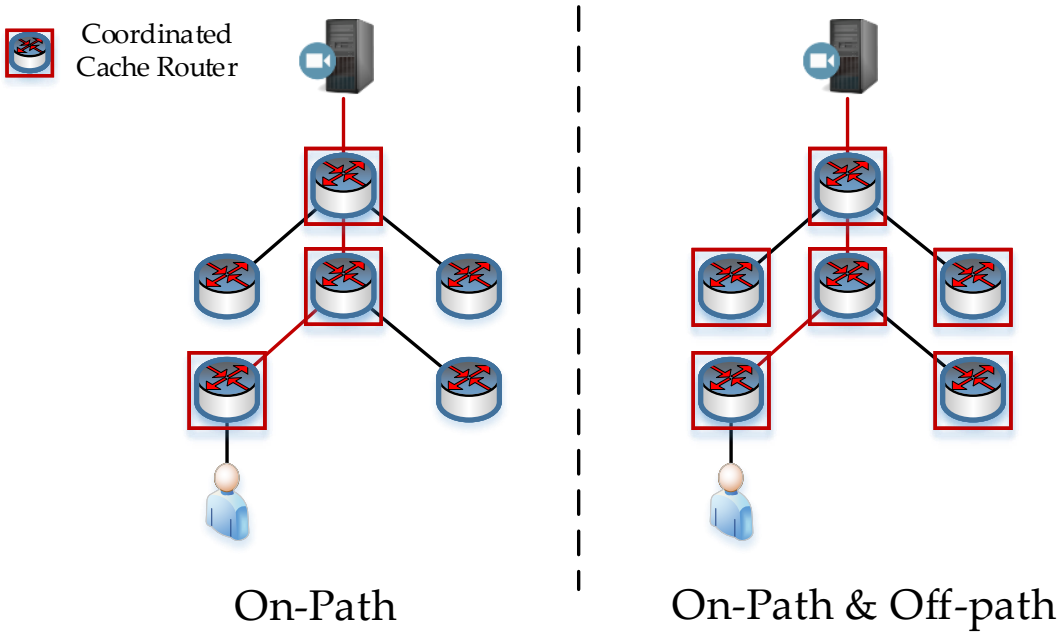


Figure 3.1: Different system settings between master thesis and this work. The red line indicates a forwarding path from consumers to a video producer. Routers in a rectangle are coordinated to make caching decisions.

work, where the contributions of my master thesis is explained, and the differences are highlighted to emphasize the novelty of this work. Section 3.3 describes the system upon which our designed caching algorithms are applied. We elaborate in Section 3.4 on the problem formulation of DaCPlace benchmark solution, and present the derivations for modeling the caching process, with specific emphasis on the queueing analysis. Section 3.5 describes the design of low-overhead scheme StreamCache, and we present experiment setup and performance evaluation results in Section 3.6 . Finally, we conclude in Section 3.7.

3.2 Related Work

The effect of ubiquitous caching for adaptive video delivery has been studied and reported in my master thesis [37], where cache placement problem is formulated and solved by optimizing the video throughput in a much constrained system. Specifically, I proposed an optimal on-path caching scheme *DASCache*, where cache decisions are coordinated among ICN routers along pre-determined routing paths to video producers. This chapter instead tackles a more sophisticated system, where both on-path and off-path caching are incorporated. As shown in Figure 3.1, the proposed caching algorithms optimize not only the caching placement on the path to producer, but also coordinate with resources in the neighborhood. A better caching decision relies on accurate modeling of adaptive video traffic over ICN. Hence we analyze the impact of *interest aggregation*, as a unique feature of ICN that reduces redundant traffic, to facilitate a more accurate estimate of video throughput.

3.3 System Description

We first detail our system, explaining the network architecture and routing protocol. Next, the video request pattern is elaborated to model the behaviors of consumers in adaptive streaming applications.

3.3.1 Network Architecture

Two different types of routers are considered: edge and intermediate routers. All consumers are served exclusively by edge routers.

Caching is not a standalone component of ICN, but highly coupled with *Interest* forwarding. It is hence essential for caching to work with routing and forwarding in a

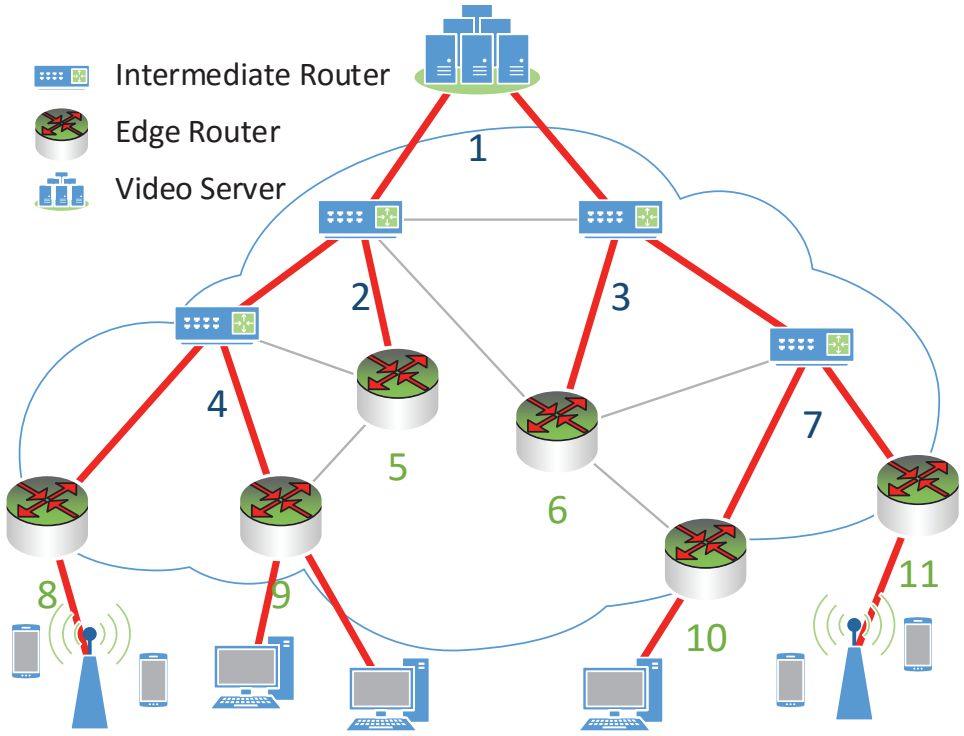


Figure 3.2: Network Topology. The bold lines indicate routing paths discovered by the OSPFN.

cooperative way. We adopt the Open Shortest Path First routing scheme for Named-data (OSPFN) [61], which is the most of both popular and commonly used routing protocol in ICN. OSPFN discovers the shortest routing path from each router to the video producer, as depicted in Figure 3.2, along which video requests can be satisfied by any cache node. However, we argue that the system would not reach the optimal performance when only the on-path caching resources are utilized. For instance, in Figure 3.2, if there is a request sent from consumers connecting to router 9 and the video content is not cached along the routing path to the producer but instead cached on router 8. on-path caching would ignore router 8 which may instead respond to the request faster than the video producer. This motivates our system to consider not

only on-path caching but also opportunities from the off-path caches.

When on-path and off-path caching are incorporated, each router has more than one option (interface) to forward a video request. Previous research by Rossini et al. [51] showed that the single-path forwarding, which relays requests through only one interface, results in the lowest network load, compared with existing multi-path forwarding schemes which send requests through multiple interfaces simultaneously. Thus, we adhere to ICN's single-path forwarding. Each router is assumed to apply the *best route* forwarding strategy, which chooses an interface with the minimal delay, to retrieve the video content.

3.3.2 Video Request Patterns

Video request patterns are described in terms of levels: file and chunk. Requests made on the file level represent the popularity of the video content. Once consumers decide to watch a video file, *Interest*s are tallied with the pattern on chunk level. These requests are generated sequentially, following the exact video playback.

Based on this pattern, gauging the video file request is an umbrella concept, which consists of a group of consecutive *Interest*s for content belonging to the same video file. Typically, video file requests are assumed to be made independently. However, since video chunk requests are generated sequentially, two consecutive *Interest*s from the same consumer yield notable correlation. In our system, each request for a file contains a batch of requests for chunks, and the number in each batch is variable. This feature corresponds to the typical viewing behavior: start playing from the beginning; keep watching for a period of time; terminate the media session when bored or out of time.

3.4 DaCPlace: Benchmark for Throughput Enhancement

DaCPlace is a benchmark solution to adaptive video caching, which is formulated as a Mixed Integer Linear Programming (MILP) problem. It optimizes video access time per bit, as an indicator of video throughput measured by consumers. When DaCPlace is executed, cached content is updated periodically on each router, over the period of a pre-determined round. In order to optimize caching decisions, statistics are collected to capture request patterns and cater to popular videos. At the beginning of each round, operational parameters are derived based on the statistics from the last round and are used as inputs to DaCPlace; the system then updates caches based on decisions made by DaCPlace and refreshes the statistics, preparing for the next round.

3.4.1 Cache Placement Problem Formulation

We model an ICN as a connected graph $G = (\mathbb{V}, \mathbb{E})$, where nodes in $\mathbb{V}$ are composed of video producers $\mathbb{P}$, edge routers $\mathbb{D}$ and intermediate routers $\mathbb{N}$. Each consumer is served exclusively by one edge router. Every node $v \in \mathbb{V}$ is equipped with content storage capacity C_v dedicated to adaptive video caching. The actual allocation of C_v has been investigated in related literature as a cache space allocation problem, which is well detailed and addressed in Section 2.3.3.

The number of video files in the system is represented by F. Our model reflects that content for adaptive streaming is fragmented into equal-duration segments, i.e., segments encoded at variable bitrates will be variable sizes. For ease of presentation, video files are fragmented into the the same number of fragments K. The number of bitrate encodings is B. Hence, video segments are identified by a (file, segment,

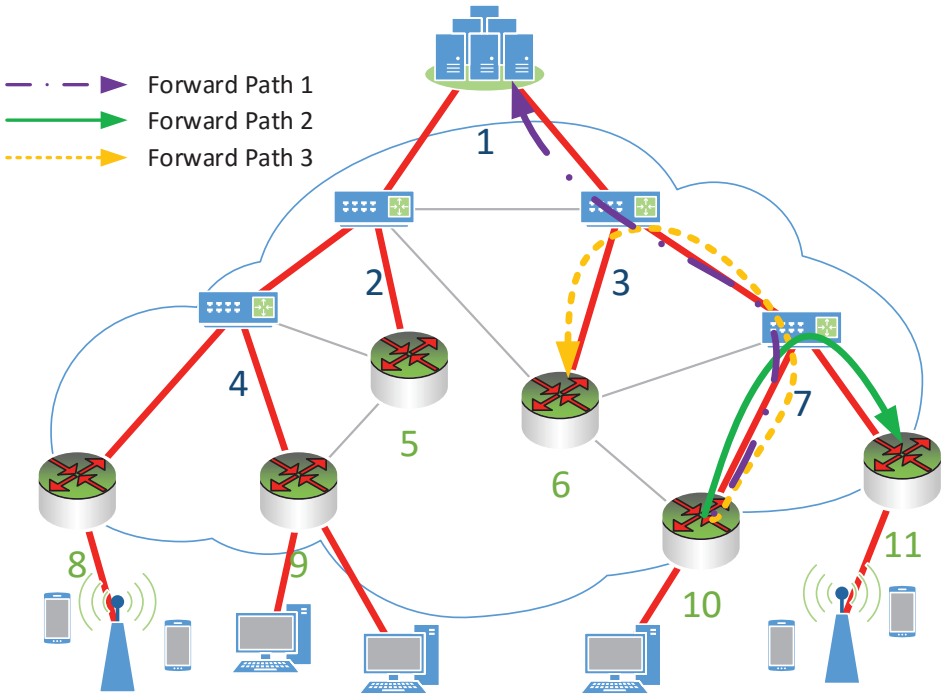


Figure 3.3: The possible content delivery paths

bitrate) triple, (f, k, b), where $1 \leq f \leq F, 1 \leq k \leq K$, and $1 \leq b \leq B$. Each video segment has size $S(f, k, b)$; we use $S(b)$ to simplify notation since equal duration video segments vary in size with bitrate encodings.

Let x_v denote the cache placement decision, where $v \in \mathbb{V}$ and $x_v(f, k, b) \in \{0, 1\}$. Thus a decision of $x_v(f, k, b) = 1$ indicates that video segment (f, k, b) is cached at node v. We use Γ_d to denote the set of possible endpoints of *Interest* forwarding paths starting from an edge router d. As current system utilizes on-path and off-path caching capacity, Γ_d includes not only video producers but also selective edge routers. These options in Γ_d define the boundary of cache coordination: each router only exchanges its caching status with other nodes along forwarding paths that can reach Γ_d. Let us take edge router 10 in Figure 3.3 as an example. In addition to the

video producer, edge router 6 and 11 are also included in Γ_{10} (i.e., $\Gamma_{10} = \{1, 6, 11\}$). However, it is important to note that other edge routers, such as routers 8, 9 and 5 are not included in Γ_{10} because *Interest* packets sent from edge router 10 would not reach them: the video producer cannot be an intermediate node in any forwarding path.

We define $[d, \Gamma_d(p)]$ as an array of nodes on a forwarding path, starting from an edge node d to an endpoint p in the set Γ_d. For simplicity, we use $[d, p]$ to denote $[d, \Gamma_d(p)]$. The length of this forwarding path $[d, p]$ is L. We further define the index of $[d, p]$ to start from 1, and the $(i+1)^{th}$ node denote the next-hop of i^{th} node on $[d, p]$. Thus $x^i_{[d,p]}$ represents the cache decision variable on the i^{th} router of $[d, p]$ (where f, k and b are implicit). Each $x^i_{[d,p]}$ also becomes an alias within x_v for example, $x^1_{[d,p]}$ is an alias of x_d, which provides a view of the edge router on the forwarding path to the end node p.

We further define binary variable $\delta^i_{[d,p]}$ as the caching status indicator, which reflects an 'aggregated' cache placement decision from d to i^{th} router of $[d, p]$. $\delta^i_{[d,p]} = 1$ only if any cache placement decision variable $x^j_{[d,p]} = 1$ for $1 \leq j \leq i$. In other words, $\delta^i_{[d,p]} = 1$ if content is already cached on a downstream router. Our scheme would evaluate all possible forwarding routes to make caching decisions, where the p is either a video producer or an edge router on the sibling branch. Finally, the number of requests on video segment (f, k, b) received by edge router d is denoted as $\xi_d(f, k, b)$, with ξ_d substituted for simplicity. Notation is additionally summarized in Table 3.1.

In adaptive streaming, video throughput is used by the adaptation algorithm to estimate the maximum supported bitrate. This is typically based on measuring the round trip time (RTT) delay of the most recently requested video chunk. A

Table 3.1: Summary of notations

Notation	Meaning
$\mathbb{V}$	Set of ICN nodes
$\mathbb{E}$	Set of links
$\mathbb{D}$	Set of edge routers
$\mathbb{N}$	Set of intermediate routers
$\mathbb{P}$	Set of video producers
S	Sizes of video segments
C	Cache capacity of ICN router
B	Number of supported bitrates
F	Number of adaptive video files
K	Number of video Segments in any file
q	Popularity distribution of video files
p	The probability of continuing watching the video
π	Stationary distribution on bitrate selection
x	Cache placement decision (0 or 1)
δ	Caching status indicator (0 or 1)
ξ	Number of video requests received by edge router
$[d, p]$	ICN routers on forwarding path from edge router d to node p
λ	Average request arrival rate before interest aggregation
$\tilde{\lambda}$	Average request arrival rate after interest aggregation
Γ	Endpoints of *Interest* forwarding paths
θ	link bandwidth
RT_{oh}	Round trip one-hop delay
RTT	Round trip time delay of a request from an edge router
$rRTT$	Residual RTT delay of a request from any router
μ	Average request miss / data arrival rate

consumer who wants to switch to a higher bitrate must achieve high throughput first. The problem is further compounded as consumers are highly sensitive to video delay. Therefore, we argue for focusing on minimizing the average access time per bit and optimize the ensuing cache placement. This objective is further specified via the following definitions.

Definition 3.1. $\tau_{[d,p]}(f,k,b)$ denotes the feasibility whether video requests for content (f,k,b) can be forwarded along path $[d,p]$. $\tau_{[d,p]}(f,k,b)$ is derived by

$$\tau_{[d,p]}(f,k,b) = \begin{cases} 0, & \text{if } p \text{ is the producer} \\ \infty \cdot (1 - \delta^{L}_{[d,p]}), & \text{otherwise} \end{cases} \tag{3.1}$$

We highlight that $\tau_{[d,p]}(f,k,b)$ would return **0** for a feasible forwarding path, i.e., if a request for video chunk (f,k,b) reaches the video producer or is satisfied by a cache along that path. Otherwise, $\tau_{[d,p]}(f,k,b)$ is assigned ∞. As τ is used in access delay definition which appears later in the optimization objective, an infinite value of $\tau_{[d,p]}(f,k,b)$, would imply that no cache along $[d,p]$ can satisfy such requests.

Definition 3.2. $\mathbb{E}[RTT_{[d,p]}(f,k,b)]$ is an accumulated round-trip delay for a video request (f,k,b) on a given forwarding path $[d,p]$. We denote $\mathbb{E}[RT^{i}_{oh[d,p]}(b)]$ as the single-hop delay on the i^{th} hop for video content encoded in bitrate b, where 'oh' denotes **O**ne-**H**op. $\mathbb{E}[RTT_{[d,p]}(f,k,b)]$ is derived by

$$\mathbb{E}[RTT_{[d,p]}(f,k,b)] = \tau_{[d,p]}(f,k,b) + \mathbb{E}[RT^{1}_{oh[d,p]}(b)] + \sum_{i=1}^{L-1}(1-\delta^{i}_{[d,p]}) \cdot \mathbb{E}[RT^{i+1}_{oh[d,p]}(b)] \tag{3.2}$$

Equation (3.2) sums round-trip delays on each hop over a given forwarding path. The appearance of $\tau_{[d,p]}(f,k,b)$ guarantees that an infeasible forwarding path would result in an infinite value of $\mathbb{E}[RTT_{[d,p]}(f,k,b)]$, which would not be considered by the optimization. $\mathbb{E}[RT^{1}_{oh[d,p]}(b)]$ is always included, as it denotes the delay on 'last-mile' link, which exists anyway no matter the caching decision. Whether the rest of single-hop delays on $[d,p]$ should be added into $\mathbb{E}[RTT_{[d,p]}(f,k,b)]$ depends on the caching status indicator $\delta^{i}_{[d,p]}$. Take the forwarding path from edge router 10 to video

producer 1 in Figure 3.3 as an example. If the requested video content can only satisfied from the producer, $\delta_{[10,1]}$ would always return $\mathbf{0}$ for any i where $1 \leq i < L$. Thus, $\mathbb{E}[RTT_{[d,p]}(f,k,b)]$ would include round-trip delays on every hop over that path.

Our system applies the best route forwarding strategy, whereby each router selects the next hop with the least data retrieval time. To implement this feature in the optimization formulation, we define $\mathbb{E}[RTT_d(f,k,b)]$ which ensures the smallest RTT to retrieve video content among all forwarding path candidates as follows.

Definition 3.3. $\mathbb{E}[RTT_d(f,k,b)]$ is the expected delay of video requests made by consumers served under edge router d, which is derived by

$$\mathbb{E}[RTT_d(f,k,b)] = \min_{p \in \Gamma_d} \Big\{ \mathbb{E}[RTT_{[d,p]}(f,k,b)] \Big\} \tag{3.3}$$

We choose to minimize the average access delay per bit, which reflects the performance of video downloading and relates to the throughput experienced by consumers.

$$\min \quad \sum_{d \in \mathbb{D}} \sum_{f=1}^{F} \sum_{k=1}^{K} \sum_{b=1}^{B} \xi_d q_f (1-p) p^{k-1} \pi_d(b) \frac{\mathbb{E}[RTT_d(f,k,b)]}{S(b)} \tag{3.4}$$

$$\text{s.t.} \quad x_v(f,k,b) \in \{0,1\}, \quad \forall v \in \mathbb{V} \tag{3.5}$$

$$\delta^i_{[d,p]} \in \{0,1\}, \quad \forall d \in \mathbb{D}, \forall p \in \Gamma_d, 1 \leq i \leq L \tag{3.6}$$

$$\sum_{f \in F} \sum_{k \in K} \sum_{b \in B} S(b) * x_v(f,k,b) \leq C_v, \forall v \in \mathbb{V} - \mathbb{P} \tag{3.7}$$

$$\delta^i_{[d,p]} \geq \delta^{i-1}_{[d,p]}, \tag{3.8}$$

$$\delta^i_{[d,p]} \geq x^i_{[d,p]}(f,k,b), \tag{3.9}$$

$$\delta^i_{[d,p]} \leq \delta^{i-1}_{[d,p]} + x^i_{[d,p]}(f,k,b), \tag{3.10}$$

$$\delta^0_{[d,p]} = 0, \tag{3.11}$$

$$x_p(f,k,b) = 1, \quad \forall p \in \mathbb{P} \tag{3.12}$$

$$\mathbb{E}[RTT_d(f,k,b)] \geq \mathbb{E}[RTT_{[d,p]}(f,k,b)] - \beta_{[d,p]}\mathbf{M}, \quad \forall d \in \mathbb{D}, \forall p \in \Gamma_d \tag{3.13}$$

$$\sum_{p \in \Gamma_d} \beta_{[d,p]} = |\Gamma_d| - 1 \tag{3.14}$$

$$\beta_{[d,p]} \in \{0,1\}, \quad \forall d \in \mathbb{D}, \forall p \in \Gamma_d. \tag{3.15}$$

Objective

The objective function minimizes the system-wide access delay per bit. This delay is represented by $\frac{\mathbb{E}[RTT_d(f,k,b)]}{S(b)}$. Our previous definition 3.3 on $\mathbb{E}[RTT_d(f,k,b)]$ is nonlinear. We transform it to a linear presentation by making $\mathbb{E}[RTT_d(f,k,b)]$ a continuous variable, and converting definition 3.3 into multiple linear expressions to play as constraints upon this variable $\mathbb{E}[RTT_d(f,k,b)]$.

The rest of components in the objective function are weights to the access delay per bit of different video segments. We use the number of requests ξ_d to distinguish the contributions from video consumers, and use q to denote video content popularity, where q_f represents the probability of interests in video file f. After a certain video file is determined, consumers start requesting a series of segments in that file. In Section 3.3.2, we described the most common pattern when people watch videos. Under this setting, the probability of requesting kth segment follows a geometric distribution, where the probability is $P(X = k) = (1-p)p^{k-1}, k \geq 1, k \in \mathbb{Z}$. p denotes the chance for a consumer to continue watching the next segment.

We denote $\pi_d(b)$ as the probability for consumer-side bitrate adaptation to request for content encoded with bitrate b. As the bitrate adaptation is influenced only by the throughput of its preceding request, the process of adaptive video requests inherently

satisfies the markov property and can be modelled as a discrete-time markov chain. Each state in this markov chain corresponds to one encoding bitrate. The process of bitrate adaptation, which switches from one video quality to another, is equivalent to the transition between two states. As a result, $\pi_d(b)$ is essentially the stationary distribution of a markov chain, which facilitates us to understand the dynamics of bitrate adaptation without tracking the real-time control.

Constraints

We build constraints which relate to the *Cache Capacity*, *Caching Status Indicator*, and *Best Route Selection*, as follows.

• The *Cache Capacity* defined in Constraint (3.7) ensures that the total size of cached video content is bound by available cache capacity over all cache routers except video producers.

• The relationship between *Caching Status Indicator* δ and cache placement decisions x is defined by Constraints (3.8)-(3.11). δ is an aggregation of cache placement decisions x. Constraints (3.8) and (3.9) give the lower bound of $\delta^i_{[d,p]}$, ensuring that its value should be greater than or equal to both its last hop indicator $\delta^{i-1}_{[d,p]}$ and the cache placement decision of the current router $x^i_{[d,p]}$. Constraint (3.10) gives the upper bound. When both $\delta^{i-1}_{[d,p]}$ and $x^i_{[d,p]}$ are 0, Constraint (3.10) will enforce $\delta^i_{[d,p]}$ to be assigned 0 since video content is not cached yet along $[d, p]$. Constraints (3.11) and (3.12) cope with the two special cases that are the consumer and the producer, respectively. As the index of $[d, p]$ starts from $i = 1$, we assign $\delta^0_{[d,p]} = 0$. Conversely, the caching decision on video producer x_p is equal to 1 for any content, since unavailable

content on in-network caches can always be found at the producer.

• Our interest forwarding applies *Best Route Selection* strategy, which is reflected via a non-linear *min* operator as appeared in definition 3.3. We transform this original definition using the '**big-M**' approach to linear expressions (3.13)-(3.14). In Constraint (3.13), **M** denotes a very large positive constant number. $\mathbb{E}[RTT_{[d,p]}(f,k,b)]$ is not a variable, but instead expanded via previous definitions 3.2 and 3.1. We also use an artificial binary variable $\beta_{[d,p]}$ (where (f,k,b) is implicit), to indicate the best route choice. When $\beta_{[d,p]} = 1$, the right hand side of Constraint (3.13) is a very small negative number. Since access delay $\mathbb{E}[RTT_d(f,k,b)]$ is always larger or at least equal to 0, $\beta_{[d,p]} = 1$ guarantees this constraint is always satisfied. When $\beta_{[d,p]} = 0$, Constraint (3.13) is essentially $\mathbb{E}[RTT_d(f,k,b)] \geq \mathbb{E}[RTT_{[d,p]}(f,k,b)]$. The continuous variable $\mathbb{E}[RTT_d(f,k,b)]$ now has a lower bound $\mathbb{E}[RTT_{[d,p]}(f,k,b)]$. As our optimization minimizes the access delay, $\mathbb{E}[RTT_d(f,k,b)]$ is forced to adopt this lower bound value, which is the round-trip delay along forwarding path $[d,p]$. Combined with Constraint (3.14), we have a total $|\Gamma_d| - 1$ number of β which could be assigned 1, with only one exception where $\beta_{[d,p]} = 0$. Our optimization implements the *Best Route Selection* by considering all possible p in Γ_d and assign the only $\beta_{[d,p^*]} = 0$, when video requests are forwarded along path $[d,p^*]$ that results in a minimal round-trip delay $\mathbb{E}[RTT_{[d,p^*]}(f,k,b)]$.

3.4.2 Expected Delay Derivation

To solve the cache optimization problem, we must derive the accumulated round trip delay $\mathbb{E}[RTT_{[d,p]}(f,k,b)]$. Since $\mathbb{E}[RTT_{[d,p]}(f,k,b)]$ is composed of multiple single-hop delays $\mathbb{E}[RT^i_{oh[d,p]}(b)]$ across all links over path $[d,p]$, we thus elaborate on our estimate

to this expected single-hop delay.

Four sources contribute to the video access delay: processing, propagation, transmission, queueing. We assume the video server and consumers are under the service of the same ISP, and thus the propagation delay is negligible. Processing delay is also omitted because of the *O(1)* hashing technique of table lookup on unsatisfied requests [25]. Thus, to compute $\mathbb{E}[RT^i_{oh[d,p]}(b)]$, we consider the transmission and queueing delays of video content. Let $\theta^i_{[d,p]}$ denote the bandwidth of link over which the video *Data* packet is delivered from the i^{th} router to the $(i-1)^{th}$ router (in the reverse order along the request forwarding path $[d,p]$). We use $\mathbb{E}[Q^i_{[d,p]}(b)]$ to represent the average queueing delay at i^{th} router when the video segments decoded in bitrate b are delivered to the $(i-1)^{th}$ node. The expected single-hop delay, $\mathbb{E}[RT^i_{oh[d,p]}(b)]$ is derived by

$$\mathbb{E}[RT^i_{oh[d,p]}(b)] = \frac{S(b)}{\theta^i_{[d,p]}} + \mathbb{E}[Q^i_{[d,p]}(b)]. \tag{3.16}$$

As both $S(b)$ and $\theta^i_{[d,p]}$ are constant input parameters, the problem of calculating $\mathbb{E}[RT^i_{oh[d,p]}(b)]$ becomes how to provide a good estimate of $\mathbb{E}[Q^i_{[d,p]}(b)]$. In the rest of this section, we first explain interest aggregation for a better queueing delay estimation in ICN, and then detail the derivation of $\mathbb{E}[Q^i_{[d,p]}(b)]$ using queueing theory.

Interest Aggregation

Quite often, large numbers of duplicate requests are witnessed in a short time frame for the same video. Under the current host-centric architecture, independent communication between a consumer and a producer must be maintained, which consumes a lot of resources (e.g., bandwidth) to repeatedly deliver the exact same content. ICN remedies this inefficiency via interest aggregation, where each router keeps track of

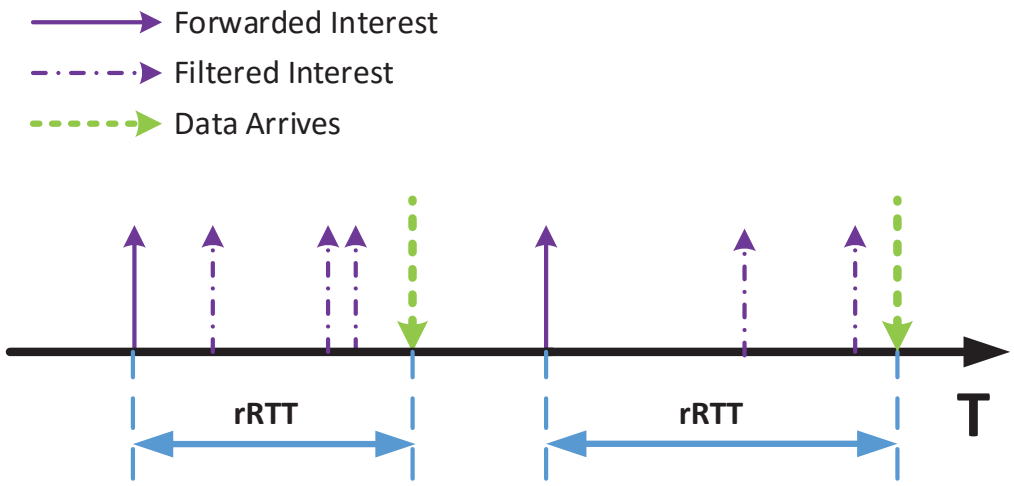


Figure 3.4: Filtering effect of interest aggregation. The length of rRTT changes according to real-time network condition.

unsatisfied requests and discards duplicate ones to prune unnecessary traffic. When a new *Interest* packet arrives at a certain router, not only would it be forwarded to the next hop but also the name of the content in that packet would be recorded. Next time, when an *Interest* packet for the same content arrives before the corresponding *Data* is sent back to the router, this duplicate request would be discarded.

The video request pattern explained in Section 3.3.2 could be generalized as a batch renewal process [46]. We denote $rRTT^{i}_{[d,p]}(b)$ as the residual round trip time delay on i^{th} router along path $[d, p]$, which is the time interval between forwarding a video request and receiving the corresponding data from either a video producer or a cache. Caused by interest aggregation, there exists a filtering effect on requests received by any router, which is shown in Figure 3.4.

Our fundamental goal is to determine optimal video caching by estimating the round trip time delay, not to model video traffic over ICN. Due to the complexity of analyzing the superposition of two renewal processes, we argue that a good

approximation is critical to guiding the cache placement decision.

Carofiglio et al. [7] proposed an approximation, however, based on an assumption that once a video request is filtered, the following requests for the rest of video chunks in the same file will also be filtered. This assumption cannot be applied in a general scenario, since RTT depends on the cache placement and varies for video chunks of different bitrates. Instead, we analyze the effect of interest aggregation on a chunk level. The process of requests after filtering is approximated by a Poisson process.

The action of filtering is captured by the *thinning* of a Poisson process with probability $\tilde{p}^i_{[d,p]}(f,k,b)$, which is

$$\begin{aligned} \tilde{p}^i_{[d,p]}(f,k,b) &= P(\text{interval time} > \mathbb{E}[rRTT^i_{[d,p]}(b)]), \\ &= e^{-\lambda_v(f,k,b)\mathbb{E}[rRTT^i_{[d,p]}(b)]}. \end{aligned} \tag{3.17}$$

$\lambda_v(f,k,b)$ in Equation (3.17) is the sum of incoming request rates for a video segment (f,k,b) received by router v. v is also an alias with i^{th} router along path $[d,p]$. However, λ_v includes not only the input from edge node d but also rates from other paths through v. This approximation could be interpreted as: the video requests after filtering are independently picked from the original 'request flow' (as shown in solid arrow in Figure 3.4) with probability $\tilde{p}$. As a result, the time interval between two subsequent requests is statistically guaranteed to be larger than $rRTT$. The average request rate after filtering $\tilde{\lambda}$ is calculated by $\tilde{\lambda} = \tilde{p} \cdot \lambda$, which is an essential component when we derive the queueing delay.

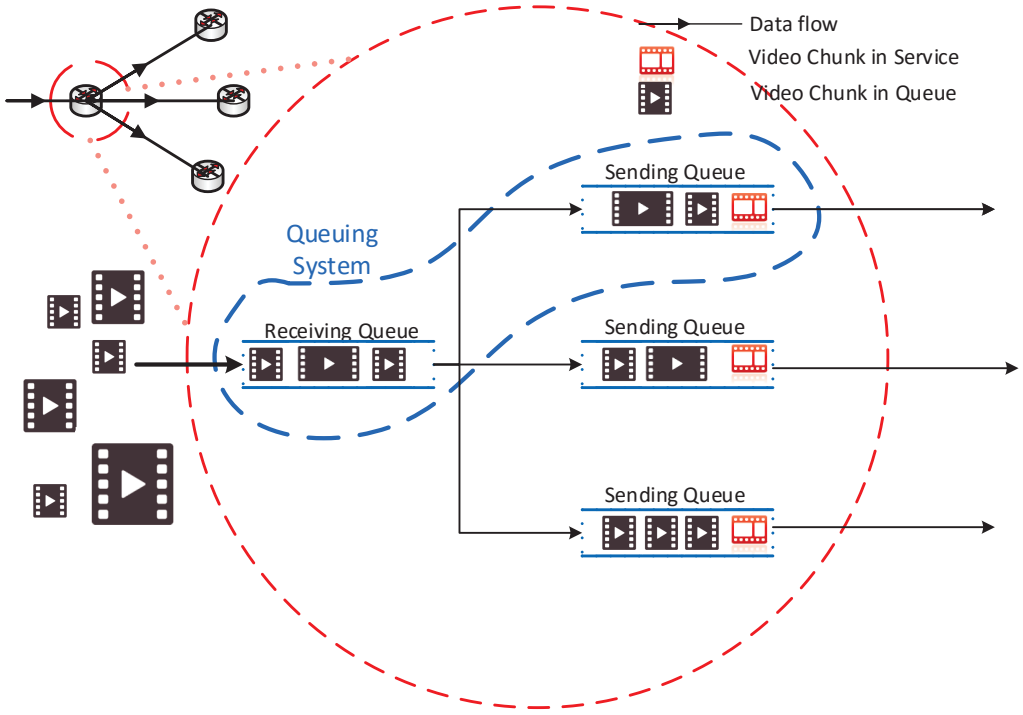


Figure 3.5: Queueing model for adaptive streaming system

Queueing Delay Analysis

For each video delivery path through a router, we model the queueing system to consist of the *Receiving Queue* and the corresponding *Sending Queue* on that path. As shown in Figure 3.5, both queues are assumed to be FIFO queues and are dedicated to serve streaming packets for ensuring QoE. The *Receiving Queue* dispatches the *Data* packet to a corresponding *Sending Queue* and that is where the queueing delay occurs.

As video requests is already modeled by an approximate Poisson process, *Data* packets are assumed to follow the same process as the requests, with the average input rate equal to the request arrival rate. The job service time of a video *Data* packet is determined by its corresponding size. As to adaptive streaming over ICN, the size

of a video segment (which is sent in one *Data* packet) only varies according to its encoding bitrate. *Data* packets with the same encoding bitrate belong to one class, sharing the same service time in the queue. As there are multiple available bitrates (classes), we adopt the multi-class M/G/1 model to analyze the queueing delay.

In a dynamic adaptive streaming system, we assume traffic load $\rho < 1$. This assumption is reasonable, since bitrate adaptation not only changes the requested video quality but also correspondingly adjust the network traffic load. $\rho \geq 1$ means that even requesting the lowest bitrate would cause significant video playback freezing and further lead to a total system failure, which is an extreme case and beyond the range of this work. Based on this assumption, the queueing delay is derived as follows.

Definition 3.4. Based on the multi-class M/G/1 model, the queueing delay $\mathbb{E}[Q^i_{[d,p]}]$ to deliver a *Data* packet from the i^{th} router to the $(i-1)^{th}$ router along the path $[d, p]$ is

$$\mathbb{E}[Q^i_{[d,p]}] = \frac{\sum_{b=1}^{B} \mu(b) S(b)^2 \theta^{-2}}{2 - 2\sum_{b=1}^{B} \mu(b) S(b) \theta^{-1}}, \tag{3.18}$$

where the superscript and subscript on $\mu(b)$ and θ are implicit. $\mu(b)$ denotes the arrival rate of video chunks encoded in bitrate b.

Proof. The arrival rate of *Data* packets $\mu(b)$ is calculated based on the superposition of independent Poisson processes [4] as follows.

$$\mu(b) = \sum_{f=1}^{F} \sum_{k=1}^{K} \tilde{\lambda}_v(f, k, b) \cdot (1 - x^i_{[d,p]}), \tag{3.19}$$

where $\tilde{\lambda_v}$ is the request arrival rate after interest aggregation.

The expected service time $\mathbb{E}[\varphi(b)]$ of given data packets encoded with bitrate b is

calculated by

$$\mathbb{E}[\varphi(b)] = \varphi(b) = \frac{S(b)}{\theta}. \tag{3.20}$$

The queueing system load ρ (where $[d, p]$ and i are implicit), can be derived by

$$\rho = \sum_{b=1}^{B} \rho(b) = \sum_{b=1}^{B} \mu(b)\mathbb{E}[\varphi(b)]. \tag{3.21}$$

Our assumption of $\rho < 1$ means that the input rate to the queue is less than the output rate, which guarantees the queueing model is not overloaded. The system load ρ is a key component in the queueing delay $\mathbb{E}[Q^i_{[d,p]}]$, as we derive by applying the Little's Theorem and *Pollaczek-Khinchin (P-K)* formula [4] as follows,

$$\mathbb{E}[Q^i_{[d,p]}] = \frac{\mathbb{E}[RS]}{1-\rho}, \tag{3.22}$$

where RS denotes the *Residual Service Time*, which is the remaining time seen by the new packet when it arrives in the queueing system until the current in-service packet is complete. Consider a time interval $[0, t]$, this *Residual Service Time* in a multi-class M/G/1 model is derived by

$$\mathbb{E}[RS] = \frac{1}{2t} \sum_{b=1}^{B} \sum_{m=1}^{M_b(t)} \varphi(b)^2, \tag{3.23}$$

where $M_b(t)$ denotes the number of packets, encoded with bitrate b, which complete

their services during $[0, t]$. As $t \to \infty$, we have

$$\begin{aligned}\mathbb{E}[RS] &= \frac{1}{2}\sum_{b=1}^{B}\lim_{t\to\infty}\frac{M_b(t)}{t}\cdot\lim_{t\to\infty}\frac{\sum_{m=1}^{M_b(t)}\varphi(b)^2}{M_b(t)} \\ &= \frac{1}{2}\sum_{b=1}^{B}\mu(b)\mathbb{E}[\varphi(b)^2]\end{aligned} \tag{3.24}$$

This is due to the service time of *Data* packets, $\varphi_{ij}(b)$, shown in Equation (3.20), is a constant value. Then, we have $\mathbb{E}[\varphi(b)^2] = \mathbb{E}[\varphi(b)]^2$. Therefore, synthesizing Equations (3.20)-(3.24), the expected queueing delay is

$$\mathbb{E}[Q^i_{[d,p]}] = \frac{\sum_{b=1}^{B}\mu(b)S(b)^2\theta^{-2}}{2-2\sum_{b=1}^{B}\mu(b)S(b)\theta^{-1}}.$$

□

3.4.3 DaCPlace Algorithm and Complexity

DaCPlace solves the optimal cache placement for throughput enhancement. The optimization is formulated as a MILP problem. Given the known complexity of MILP, solving this problem is NP-Complete.

The caching decisions by DaCPlace are updated periodically according to the most recent bitrate-specific statistics on video popularity. We refer to this popularity update as the ***outer iteration***. In addition, we also define a ***inner iteration*** to resolve a circular dependency that exists between $\mathbb{E}[RT^i_{oh[d,p]}]$ and caching placement decisions (x). $\mathbb{E}[RT^i_{oh[d,p]}]$ is a key component to derive the round-trip delay along a forwarding path $[d, p]$ as shown in definition 3.2. However, $\mathbb{E}[RT^i_{oh[d,p]}]$ depends on queueing delay $\mathbb{E}[Q^i_{[d,p]}]$ in Equation (3.16), which is further influenced by the

cache placement decisions (x) in Equation (3.18) and (3.19). That means the cache placement would alter the traffic load on each link and change the round trip delay. In remedy, we devise DaCPlace to iterate, by updating parameters and caching decisions reciprocally. Specifically, DaCPlace uses the cache placement decision of the last iteration to adjust the data input rate, which thereby yields a constant $\mathbb{E}[RT^{i}_{oh[d,p]}]$ value. This constant round-trip delay is then plugged into the current iteration to update the cache placement decision. Thus, in each iteration, the formulation is in a linear form that is solvable by MILP. The result of each inner iteration is non-increasing. Our simulations also show the execution of DaCPlace converges 100%.

Algorithm 1: DaCPlace

Input: Performance Difference Threshold (κ).
Output: Cache Placement Decision x.

Initialize $x \leftarrow 0$, $x' \leftarrow 0$
// Inner iteration starts
repeat
  $x \leftarrow x'$
  $OBJ(x) \leftarrow$ Last optimization objective value
  for all $d \in \mathbb{D}$ **do**
    for all $p \in \Gamma_d$ **do**
      for $i = 1$ **to** L of path $[d, \Gamma_d(p)]$ **do**
        Update round-trip delay $\mathbb{E}[RT^{i}_{oh[d,p]}]$
      end for
    end for
  end for
  $x' \leftarrow$ Solving a Mixed Integer Programming problem
  $OBJ(x') \leftarrow$ Current optimization objective value
until $|OBJ(x) - OBJ(x')| \leq \kappa$
return x

Algorithm 1 details the steps to compute the optimal cache placement. For readability, the index (f, k, b) of variables are omitted. This algorithm only describes the inner iteration which updates the cache placement variable x. The iteration stops once

predefined criterion (κ) is met, as shown in line 14, which measures the performance difference of two consecutive cache placements based on the optimization objective. This criterion should be set by the network provider considering the tradeoff between cache performance and time to achieve it. Each inner iteration is composed of two parts. Lines 5-11 calculate the expected delay and line 12 updates the cache placement result by solving a MILP problem. The complexity of DaCPlace is dominated by line 12 and is NP-Complete.

3.5 StreamCache: Low-Overhead Cache Placement

We propose StreamCache to narrow the gap between theoretical optimal solution and the practical implementation. StreamCache makes distributed caching decisions at each router with minimal coordination and aggregated statistical information. We show that, under various network settings, such as available cache storage and popularity distribution, StreamCache enables users to achieve close-to-the-optimal performance compared with our benchmark scheme DaCPlace and outperforms state-of-the-art caching placement policies in the literature.

The complexity of DaCPlace algorithm arises from the fact that inner iteration is required to update the cache placement, and in each iteration, a MILP formulated problem must be solved which is NP-Complete. We reduce the complexity by making each router keep track of *Interest* or *Data* packets in real time and measure the actual delay without building a queueing model. This can be easily done over ICN due to its unique feature in request forwarding. Next, we apply a greedy algorithm at each router to decide cache placement, avoiding the inner iteration and MILP formulation. Real-time measurement and heuristic design are two fundamental changes to

StreamCache, which effectively reduces the complexity and overhead.

Similar to DaCPlace, the objective of StreamCache is to maximize the average video throughput for adaptive streaming over ICN. To achieve this objective, StreamCache starts with video statistics collection at edge routers. The collected statistics are then used to derive cache utility, which reflects the contribution of each video segment towards throughput calculation. The cache placement is then made based on this utility value using greedy selection.

3.5.1 Cache Utility Derivation

The cache utility is calculated for all video segments at each router. For video chunk (f, k, b), the *Cache Utility* function is defined as

$$\mathbb{U}(f,k,b) = \frac{q(f)p^{k-1}\pi(b) \cdot rRTT(b)}{S(b)}, \tag{3.25}$$

where file popularity distribution (q), stationary bitrate adaptation (π) and probability of continuing watching next video chunk (p) are input parameters, following the same definitions as previously described in DaCPlace scheme.

Edge routers record video request statistics to derive these input parameters. Two statistics tables are involved in this derivation: **Video File Table** and **Bitrate Table** as shown in Figure 3.6. **Video File Table** tracks the request frequency on different video files to derive q. We also record the number of requests by encoding bitrates in **Bitrate Table** to derive π. We do not need a third table to track the popularity of different video segments. Instead, the maximum likelihood approach is applied to estimate the probability of continuing watching the next video chunk (p).

After edge routers have made caching decisions, **Video File Table** and **Bitrate**

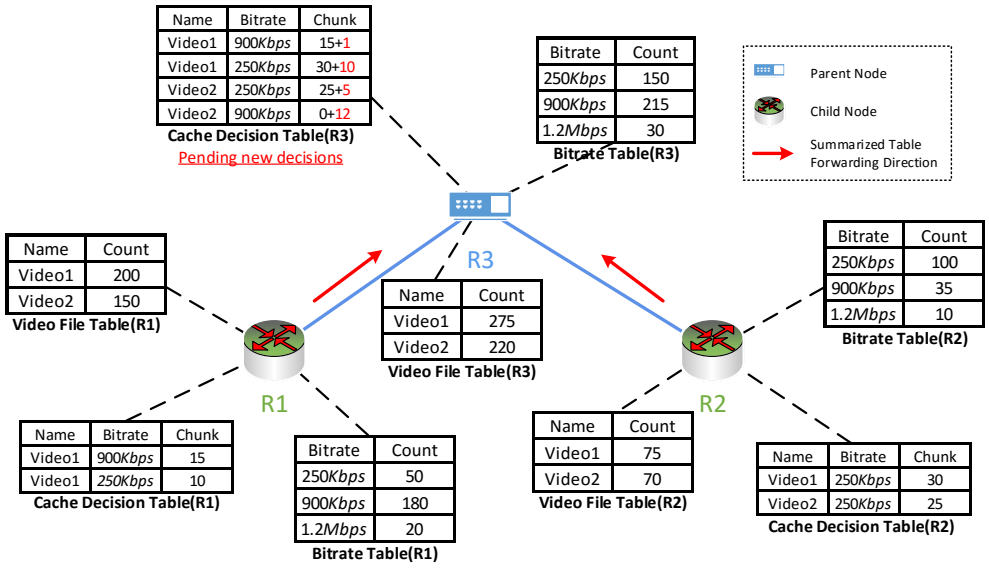


Figure 3.6: Summarized statistics and Cache Decision Table. Cache placement is made from the network edge towards the core. The local decisions are recorded by appending numbers in Cache Decision Table (in red color).

Table are delivered to upstream nodes along the forwarding path to video producers. Intermediate routers in the core network then intercept these tables, calculating cache utilities based on joined information from downstream nodes. After the cache placement is made, these two tables are forwarded to the next hop for further processing. As a result, StreamCache is executed distributedly, starting from the network edge towards the core.

3.5.2 Cache Decision with Greedy Selection

StreamCache sorts the cache utilities from high to low and caches video chunks with high utility. These decisions are local, but require coordination to utilize both on-path and off-path caching capability.

We design a **Cache Decision Table** for cache coordination based on an important

observation: as to any video file (f) encoded with bitrate (b), if the α^{th} video chunk is chosen to be cached, any video chunk k in the same file, where $1 \leq k < \alpha$, must have already been cached as well. It is because the caching utility for those video chunks is larger than the α^{th} chunk based on the Equation (3.25) and they should have been cached prior to the α^{th} chunk if there is enough cache capacity. Therefore, each router maintains a record of the maximum sequence number to indicate those video chunks which have already been cached. For instance, as shown in Figure 3.6, an entry of (*Video1*, *900kbps*, 15) in the table of router R1 means that the first 15 chunks of *Video1* encoded with *900kbps* have been cached.

Algorithm 2: StreamCache

Input: ICN Router v; Cache Capacity C_v; Video Segment Size $S(b)$; Video File Table; Bitrate Table; Cache Decision Table (CD_v).
Output: Cache Placement Decision x_v.

```
1: 𝕌 ← Calculate Utility
2: for all ∀i ∈ Downstream(v) do
3:    for all ∀e ∈ CD_i do
4:       if e ∉ CD_v then
5:          Insert e in CD_v
6:       else if CD_v(e).ChunkSeq < e.ChunkSeq then
7:          CD_v(e).ChunkSeq ← e.ChunkSeq
8:       end if
9:    end for
10: end for
11: 𝕌 ← Sort(𝕌)
12: for all ∀u ∈ 𝕌 do
13:    if C_v − S(u.b) ≥ 0 then
14:       if (u.f, u.b) ∉ CD_v or u.k > CD_v(u).ChunkSeq then
15:          x_v(u.f, u.k, u.b) to be cached
16:          C_v ← C_v − S(u.b)
17:       end if
18:    end if
19: end for
20: return x_v
```

In addition to **Video File Table** and **Bitrate Table**, **Cache Decision Table** is also passed from the network edge towards the core. Let W denote the set of all video chunks. Suppose a router v receives this **Cache Decision Table** $H_v(i)$ from a downstream router i. StreamCache on router v would only calculate cache utilities and apply greedy selection on video segments within $W - \bigcup_i H_v(i)$. That means video chunks already cached by sibling nodes are not considered by the parent router again, and requests for those chunks may be redirected, instead of being forwarding to the video producer, in order to utilize the off-path caching.

Algorithm 2 details StreamCache scheme. Lines 2-10 aggregate the caching decisions on router v from its downstream nodes by calculating the maximum sequence number in order to serve the common interests. Suppose the data structure like hash table is implemented which supports searching with $O(1)$ cost. The complexity of merging two decision tables is $O(FB)$ as the size of cache decision table is proportional to the number of bitrates and video files. Line 11 sorts the caching utility for all video chunks from the largest to smallest, with a complexity of $O(FB \log FB)$. Lines 12-19 utilize the greedy selection to fill the cache capacity of router v, which scans the sorted table with complexity $O(FB)$. Thus, the overall complexity of StreamCache is $O(FB \log FB)$.

3.6 Performance Results and Insights

We evaluate DaCPlace and StreamCache performance via simulation against known caching strategies on the NDN architecture. Caching schemes were implemented onto ndnSIM [2], an NS-3 based simulator. We claim without loss of generality that both DaCPlace and StreamCache designs and subsequent analyses can be applied to other

ICN architectures and cache hierarchies.

3.6.1 Simulation Setup

To evaluate DaCPlace, the simulation is composed of two phases. The first phase calculates the optimal cache decisions using Gurobi [22] to solve a MILP problem. The second phase simulates a NDN via ndnSIM, applying the caching decisions derived in the first phase. To evaluate StreamCache, we implement the distributed algorithm on each router directly in ndnSIM.

The settings of our experiments mimic the dynamic adaptive streaming application where requests are generated for different bitrates of video content. We consider four common bit rates: 250 Kbps, 400 Kbps, 600 Kbps and 900 Kbps. Without loss of generality, these four bit rates are typical viewing bit rates, which correspond to representative video quality levels [34]. We compare the performance of our proposed video caching schemes with three other cache placement schemes in the literature: CE2 [25], ProbCache [49] and the on-path optimal video caching scheme, DASCache [37]. We choose CE2 as it is a commonly used baseline [63, 8]. ProbCache is a popular approach in the literature (seen in [53, 63]) because of its effectiveness of reducing server hit ratio. DASCache is an optimal scheme which only utilizes the cache on the default forwarding path and we use it to compare with the performance of our proposed schemes in this more general system where both on-path and off-path caching are considered.

DaCPlace and StreamCache require an outer iteration to update bitrate-specific popularity. Our experiments only simulate one round in outer iteration since popularity and bitrate distributions remain as control parameters in the simulations. After

this round, the cached video content decided by DaCPlace or StreamCache would not be replaced, and then we start collecting statistics for performance evaluation. In order to make a fair comparison, simulation on CE2 and ProbCache mimics this procedure and we disable cache replacement (LRU) before the evaluation.

As our DaCPlace scheme targets optimizing QoE in terms of video throughput and quality, we choose the *Access Time Per Bit* of all consumers in the system as a performance metric. The delay is measured between the *Interest* packet sending and the corresponding *Data* packet arriving at the consumer's device. Another metric we choose is *Cache Hit*. This metric is commonly used to evaluate the performance of a caching system.

The routing scheme applied in our system is based on OSPFN. It generates a routing tree topology. At the same time, a tree is instructive because from the perspective of a video producer, the distribution topology is effectively a tree. Thus, in the simulations, we adopt a tree topology directly, with one layer of edge routers as leaf nodes and at least one layer of intermediate routers which connect to the producer. More routers between these two layers will only generate topologies with larger tree heights. We chose 20 nodes in our simulations to contrast performance results, as similar performance trends were observed for different network sizes.

3.6.2 Simulation Parameters

In the simulations, consumers generate video requests with certain parameters which are carefully chosen using the following rules:

- Any video request specifies the file index, chunk index and bitrate. The abstract requests for video files are determined by content popularity distribution (q).

Table 3.2: Simulation parameters for DaCPlace and StreamCache evaluation

Parameter	Value
Number of video files (F)	20
Number of video chunks per file (K)	15
Number of NDN routers ($\|\mathbb{V}\|$)	20
Number of edge routers ($\|\mathbb{D}\|$)	12
Number of video producers ($\|\mathbb{P}\|$)	1
Video segment playback time (sec)	2
Encoded bitrates (Kbps)	{250, 400, 600, 900}
Bandwidth (Mbps)	5
Topology tree height	4
Skewness factor (α)	0.8
Content store size percentage (ω)	15%
Cache allocation ratio (ϵ)	1
The probability of continuing watching (p)	0.9

We use the *Zipf*-like distribution [5] where the probability of requesting the f^{th} file is $q_f = \frac{\beta}{f^\alpha}$, where $\beta = \frac{1}{\sum_{f=1}^{F} f^\alpha}$. The parameter α controls the skewness of popularity distribution. A large α indicates that only few video files are frequently requested and a small α represents large number of video files have similar chance to be requested.

- The probability of continuing to watch the next chunk (p) is set manually, where we discuss the impact of this parameter on the performance in Section 3.6.3.

- The stationary distribution of bitrate selection (π) on each edge router, is generated randomly. Since video chunks encoded with different bit rates could incur different loads on the link, this distribution is determined before we choose the video request rate.

- The average video request rate (λ) is chosen randomly, with the only constraint

that the incurred load is less than the link capacity.

The complete list of simulation parameters are shown in Table 3.2. The cache capacity percentage (ω) indicates the total available system capacity for video streaming, calculated by $\omega FK \sum_{b=1}^{B} S(b)$. The cache capacity for each router C_v is influenced by the cache allocation ratio (ϵ), where $\epsilon = C_i/C_j, i \in \mathbb{D}, j \in \mathbb{N}$. All edge or intermediate routers are allocated with the same cache capacity.

3.6.3 Performance Evaluation

We study the effect of cache capacity, cache allocation patterns and content popularity on video access time per bit and cache hits. Simulation results are presented at a 90% confidence level.

The Impact of Cache Capacity Percentage

This experiment evaluates cache utilization and efficiency of caching schemes under uniform (equal allocation) cache storage settings. Figure 3.7 shows the average access delay per bit for different cache capacity (budgets). CE2 placement scheme yields relatively longer access delays across all test cases. The reason is that CE2 with LRU replacement keeps the most recent *Data* but cannot distinguish among video traffic which are mixed with popular and unpopular content. The performance of ProbCache outperforms CE2 because ProbCache caters to frequently requested content which significantly reduces delay. For example, at $\omega = 25\%$, ProbCache outperforms CE2 by 11.8%. However, DaCPlace still achieves lower access delay per bit by 26.7% over ProbCache. This is because DaCPlace optimizes video delivery, considering not only the delay but also storage utilization. Even though DaCPlace is a popularity-based

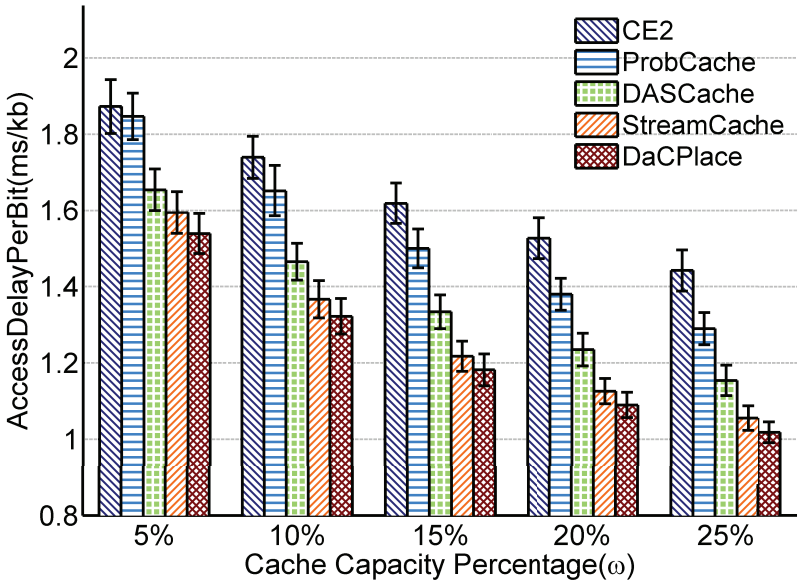
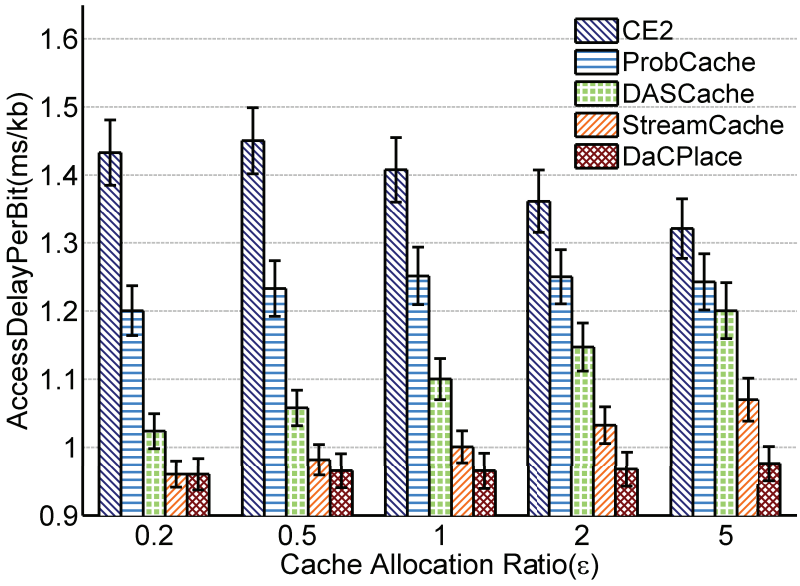


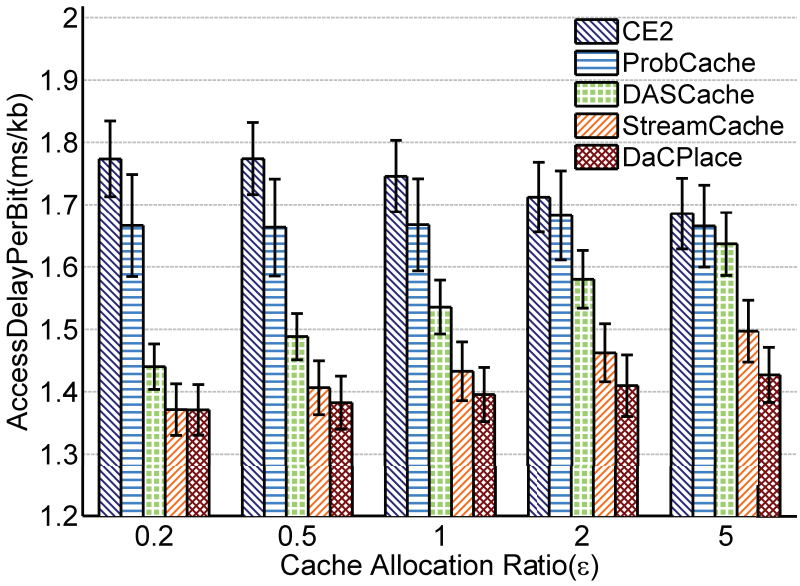
Figure 3.7: Access Delay Per Bit across different cache capacity

scheme, it may not keep the most popular content. Such available cache storage is used for multiple less popular content with lower bit rates (smaller size), which thereby achieves higher throughput.

Figure 3.7 also shows that StreamCache achieves close performance to DaCPlace (approximate 3% difference across all cases). This small difference represents our heuristic design considers important features of adaptive video caching system, resulting in a near-optimal performance for different total cache budgets. DaCPlace outperforms DASCache, especially for larger total cache capacity. For example, at $\omega = 5\%$, the access delay of DaCPlace is 7.5% lower than DASCache as opposed to 13.3% when $\omega = 25\%$. As mentioned earlier, DASCache scheme would not forward *Interest*s to routers which are not on the default routing path. Thus, once a request is missed by a cache, this request loses the opportunity to be satisfied on sibling branches. This loss increases with larger ω which explains the trend.



(a) $\alpha = 1.2$



(b) $\alpha = 0.4$

Figure 3.8: Access Delay Per Bit across cache allocation ratios

The Impact of Cache Allocation

We explore the distribution of cache allocation among edge routers and intermediate routers by using different allocation ratios, including homogeneous ($\epsilon = 1$) and heterogeneous ($\epsilon = 0.2, 0.5, 2, 5$) cases.

Cache storage on edge routers significantly impacts facilitating video streaming, since it is closest to users, thus could satisfy requests with minimal delay. Wang et al. [62] claim that when the content popularity distribution is highly skewed (i.e. where only a small portion of videos are frequently requested), edge routers should be allocated larger capacity. In contrast, when content has similar popularity score, more cache capacity should be allocated to intermediate routers to reduce cache redundancy. Figure 3.8a and 3.8b show the access delay under these two scenarios with varying Zipf popularity skewness ($\alpha = 1.2$ and 0.4 respectively).

The performance of all caching schemes has a similar trend across different allocation ratios in both scenarios. The access delay when $\alpha = 1.2$ is lower, compared to the case when $\alpha = 0.4$. This is attributed to cached content being more frequently requested with larger α, which generates more cache hits and results in less average delay. This is further analyzed in Section 3.6.3.

It is straightforward to see that moving more cache storage to the edge would yield faster system response. However, this performance gain is coupled with the expense of less cache hits caused by worse cache utilization. In fact, this constitutes a core challenge in the design of any caching scheme, whereby striking the balance between efficient cache utilization and fast system response. For example, as shown in Figure 3.8, with increased ϵ, the performance of DASCache degrades and CE2 improves. These are two typical cases where cache redundancy and system response play as the dominating factor. However, as to DaCPlace, the performance difference

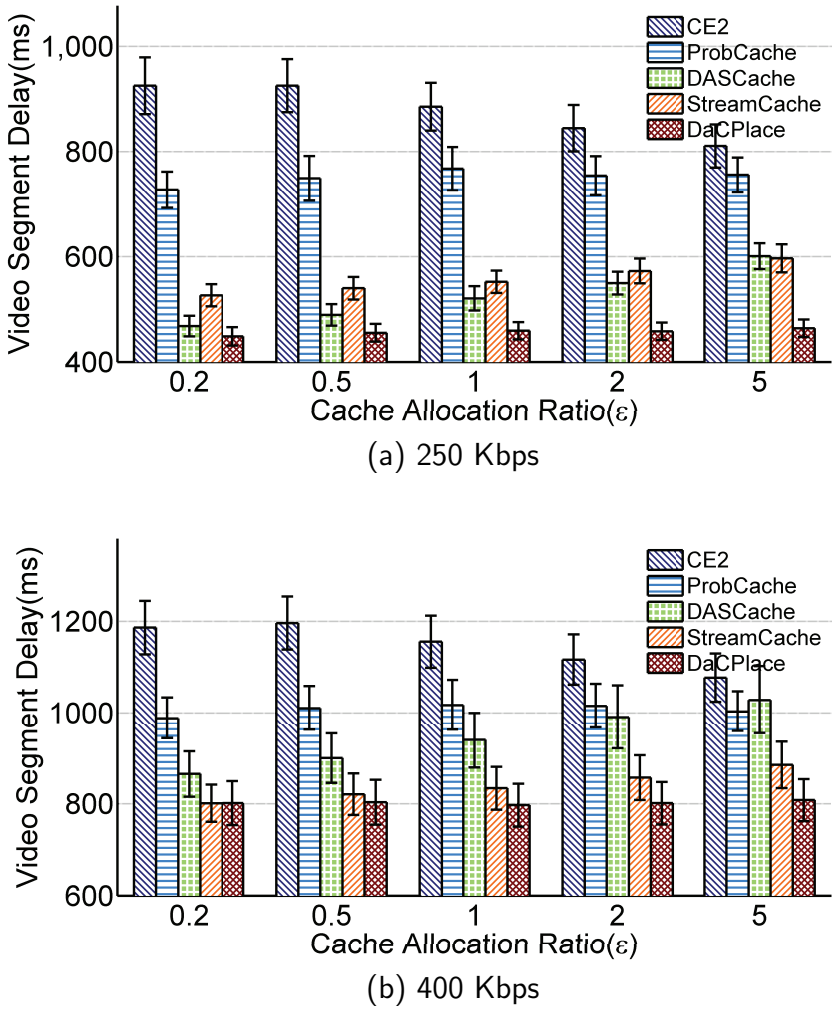
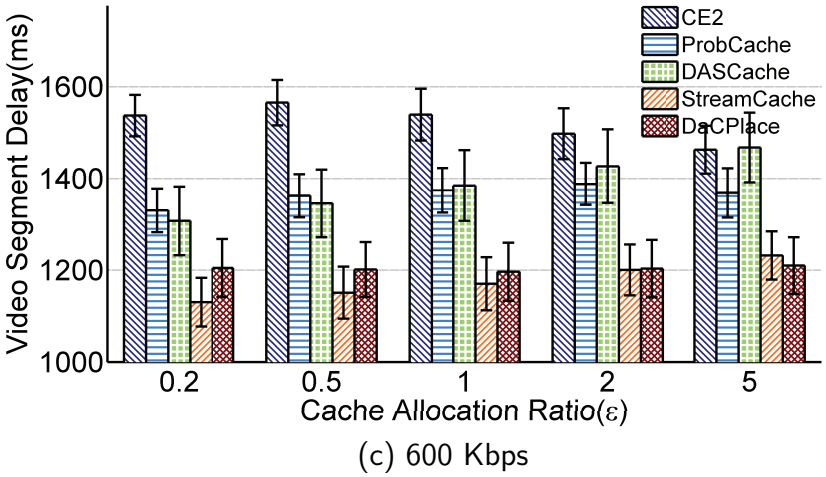


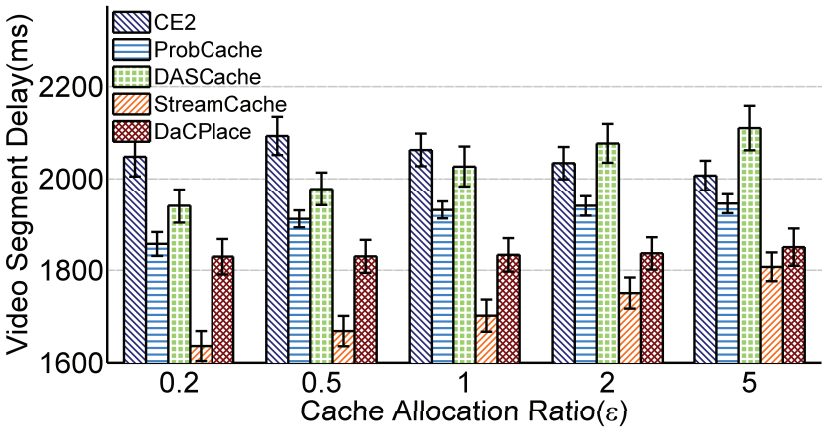
Figure 3.9: Average video segment delay ($\alpha = 1.2$)

is statistically insignificant. We then focus on the scenario when $\alpha = 1.2$ to present more features of our schemes.

Figure 3.9 presents the average delay grouped by bit rates. DaCPlace results in fastest download speed for video chunks encoded with 250 Kbps and 400 Kbps. For example, when $\epsilon = 1$, DaCPlace is 92.8% and 45.0% faster than CE2. It is important to note that the access delay of DaCPlace is longer than StreamCache



(c) 600 Kbps



(d) 900 Kbps

Figure 3.9: Average video segment delay ($\alpha = 1.2$) (cont.)

for higher bit rates (600 Kbps and 900 Kbps). Nevertheless, the overall access delay per bit (shown in Figure 3.8a) of DaCPlace is still lower than StreamCache. This reveals that DaCPlace caters to the requests for low bit rates, which is significant for adaptive video streaming: users who request low bit rates should be bound to benefit from caching the most. This is because the link bandwidth of users requesting low bit rates is relatively low, hence cached content would tremendously improve requested

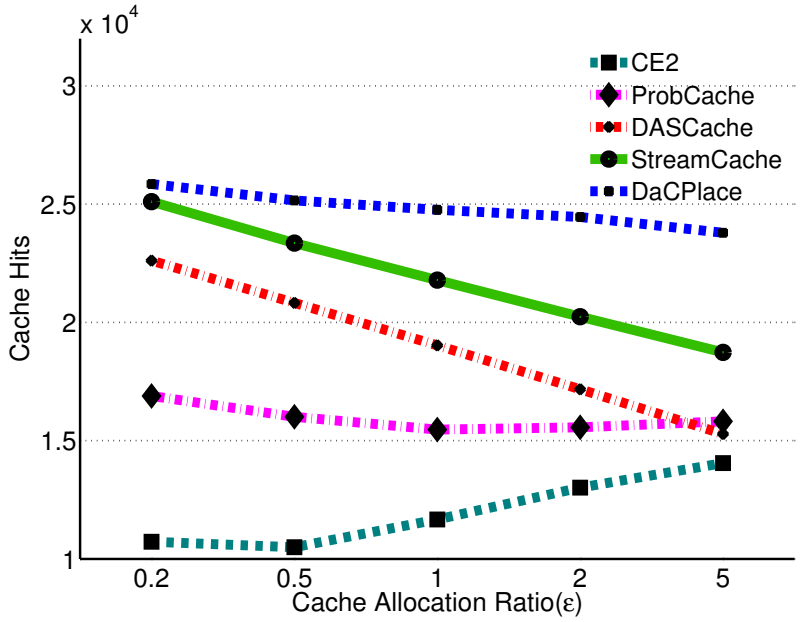
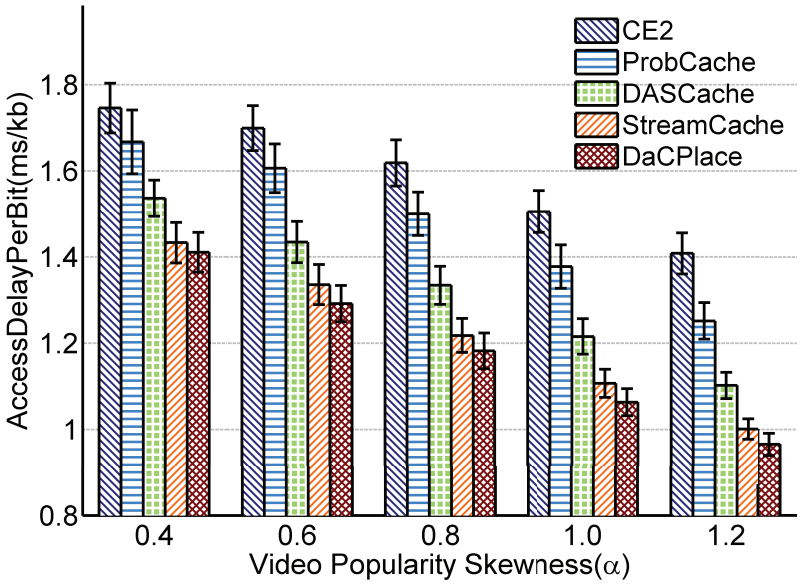


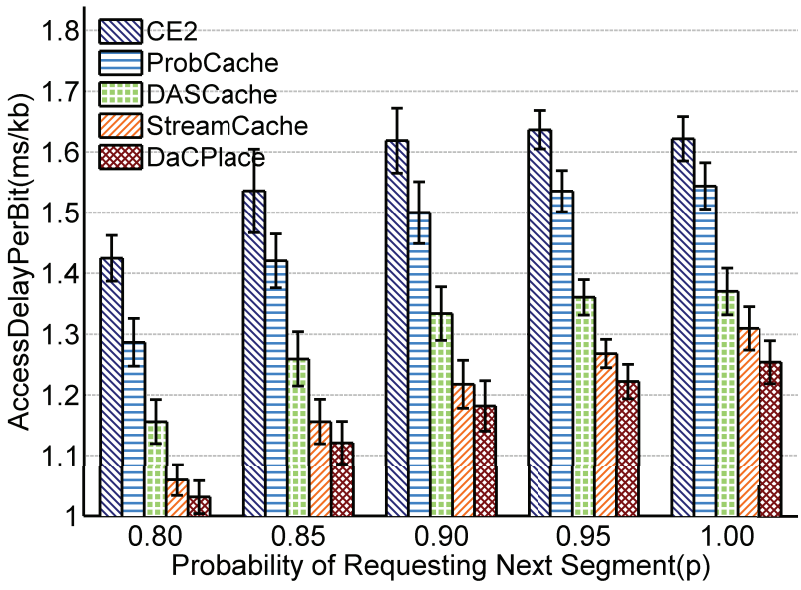
Figure 3.10: Cache hits across cache allocation ratios at $\alpha = 1.2$

video quality.

As shown in Figure 3.8a, the performance of DASCache degrades as the size of caching storage on edge routers increases. For example, as ϵ changes from 0.2 to 5, the access delay per bit of DASCache correspondingly increases by 17.3%. This impact is further investigated over the amount of cache hits. Figure 3.10 shows that cache his of DASCache decrease by 48.0% as ϵ changes from 0.2 to 5, but cache hits of DaCPlace are relatively unchanged across all tested cache allocation ratios. The superior performance is attributed to DaCPlace utilizing off-path and on-path caching in an optimized and cooperative manner. When $\epsilon = 5$, the amount of cache hits of DaCPlace increases by 55.7% over DASCache, which demonstrates the significant improvement caused by off-path caching.



(a) File Level Popularity



(b) Segment Level Popularity

Figure 3.11: Access Delay Per Bit across popularity skewness

The Impact of Content Popularity

The popularity of video content is modeled in two levels, as explained in Section 3.3.2. We control the skewness parameter α in the *Zipf* distribution, and p in our geometric distribution model, to vary content popularity on file and chunk levels; respectively.

Figure 3.11a shows the results across different skewness parameter values. For example, at $\alpha = 0.8$, DaCPlace improves by 37.0% and 26.9% over CE2 and ProbCache respectively. When α changes from 0.4 to 1.2, all tested caching schemes lead to less average access delay. This is because users' requests concentrate on a smaller set of popular content with larger α, which thereby increases the chance of cache hits. StreamCache captures popularity variance with different skewness values, and achieves equivalent performance (around 3.2% difference across all cases) compared with the optimal DaCPlace.

To capture the impact of the geometric distribution modeling of video requests, we present a comparative experiment in Figure 3.11b that depicts the access delay across different geometric parameter values. As p increases, more users are likely to finish viewing the entire video, which results in increased delay per bit for all caching schemes. The change in p would inherently influence the popularity per video chunk. Even though a certain video file is popular, the last few chunks could witness infrequent requests. Since the first several chunks of a popular video file are highly likely to be cached, as users continue to watch the video, chunks at the middle and (more towards) the end of that file may have to be retrieved directly from the server.

Therefore, the longer the time a user watches a video, the less likely a cache hit occurs. This explains why as p increases, the average access delay grows as well. As both our optimal DaCPlace and heuristic StreamCache schemes have considered such

request patterns, they outperform other caching schemes in capturing this variance over the extended duration of the video. Moreover, it is worthwhile to note that it is possible for users to experience bitrate switches while watching the video in the middle because the throughput of video chunks witnessing cache hit or miss are quite different. Our DaCPlace placement scheme is not designed for a particular user but the rate adaptation algorithm should be further considered for smooth playback.

3.7 Summary

We address the premise of dynamic adaptive streaming of video content, with the aim of minimizing the average access time per bit and improving QoE under varying network conditions. The future of video delivery is coupled with adaptive streaming, and caching schemes that address heterogeneous users cannot ignore tailored video delivery.

At the core of the work presented in this chapter, we argue for the importance of 1) capturing the characteristics of bitrate selection over varying user demands, which we modeled using a discrete-time markov chain, 2) establishing a solid queueing and service delay analysis to project link utilization and network variability, which we presented over a multi-class M/G/1 Queueing Model, 3) catering for interest aggregation in video demand, as it significantly reduces network overhead in handling equivalent content requests, which we analyzed and presented as thinning in a Poisson process, 4) presenting DaCPlace, as a benchmark solution for variable bitrate caching over ICNs, which incorporates content popularity as a core factor in optimizing caching performance, 5) designing a heuristic scheme, StreamCache, which

significantly decreases the computational complexity and achieves near-optimal performance to DaCPlace, and 6) enabling future improvements on our model and potential benchmarking by developing an NS-3 based ndnSIM simulation environment for StreamCache and DaCPlace.

We conclude that gauging popularity, on both the chunk and file levels, is critical to optimal cache placement. As adaptive video streaming yields requests for different bitrates, it is crucial to evaluate their respective effects on throughput to better optimize cache utilization. Moreover, utilizing ubiquitous in-network caching improves video delivery delay under popularity-based schemes, in comparison to edge-based caching. StreamCache and DaCPlace reduce cache redundancy by capitalizing on off-path caching, further building on ubiquitous in-network caching. Overall, StreamCache and DaCPlace outperform existing caching schemes under varying cache sizes and content popularity.

Chapter 4

Adaptive Streaming with Cache Partitioning

4.1 Introduction

Our previous research in Chapter 3 has shown the placement of video segments with variable bitrates in cache hierarchies is far from intuitive. To fill this video-to-cache-placement gap, DaCPlace and StreamCache are proposed to utilize snapshots, or instantaneous inference, of adaptive video traffic in ICN, and focus on video throughput as a direct factor that influences delivered video quality. Although evaluation results show significant improvement, our approaches did not look into other factors, such as bitrate oscillation and playback freezing, that impact QoE.

In this chapter, we tackle a different challenge that stems from the interplay between cache placement and consumer-side bitrate adaptation, named 'oscillation dynamics' [64]. To exemplify a common scenario, consumers that retrieve low-bitrate segments from edge caches will perceive good performance. A consumer-side bitrate adaptation protocol will thus invoke a request for higher-quality content that may be stored on a different (farther) cache in the network core. Data from the network core has to be delivered via a longer path than from the edge cache, and is more likely

to face contention or congestion. Poor performance from the higher-quality video source will cause the streaming application to reduce its video quality preference. Oscillation dynamics are intrinsically linked to inaccurate estimates caused by ever-changing network conditions that occur with intermittent cache hits and misses.

Oscillation dynamics are not inherent to ICNs only, and have previously been studied in the context of Content Delivery Network (CDN). For example, cache-aware bitrate adaptation [35] triggers independent threads of adaptation logic when cache hits occur. However, caching in CDN differs significantly from cache hierarchies in ICNs. CDN hosts all video content, and at fixed locations, so consumer estimates of system performance are dominated by network effects. In contrast, cache hierarchies in ICNs make it possible for video segments to appear at any cache router. As a result, consumer-side adaptation techniques have no means to distinguish between poor performance from network conditions and poor performance from cache conditions. This suggests that a "good" caching scheme may stabilize bandwidth fluctuations to reduce oscillation, and thereby improve consumer QoE.

We posit that one such family of caching schemes emerges when encoding bitrates are prioritized over - or alongside - conventional metrics associated with hit rates and popularity. In particular, we hypothesize that the QoE for high-quality content requests suffers disproportionately from resource sharing, relative to low-quality content. One implication would be that the highest bitrate content should be placed where there is least congestion. Our investigations into adaptation-based caching dynamics show that bitrate oscillation patterns emerge with hop distance [38, 39]. The pattern that emerges suggests that high-bitrate content is most stable when retrieved from edge caches. From a caching perspective this may be counter-intuitive: it entails

copies of the largest segments at multiple edge caches, rather than a single copy at upstream caches that sit on intersecting paths.

This insight leads to, and is validated by, the main contribution of this chapter. We present *RippleCache* as a cache guiding principle that safeguards capacity at the edge routers for high-bitrate content, thereby pushing lower bitrate content along the forwarding path towards the network core. This has the effect of *partitioning* cache capacity along a forwarding path, but raises questions with respect to partition boundaries and caches that sit on intersecting paths. In order to validate the main contribution, we construct two independent caching schemes, following RippleCache ideal:

- ***RippleClassic*** serves as a benchmark cache partitioning paradigm. Partitions are created by solving an optimization problem formulated as binary integer programming. The objective of RippleClassic maximizes a metric designed specifically to measure cache hierarchy performance for adaptive streaming, that has been shown to have high correlation with consumers QoE [38]. The solutions that emerge place content in such a way that a RippleCache emerges.

- ***RippleFinder*** is a distributed caching scheme that is built on our prior work [39] and executes in polynomial-time complexity. Execution begins at edge routers, from where cache partitions are created along the forwarding path to each video producer. Placement decisions prioritize utility, an indicator of the resource cost of a video segment (by size) and weighted by popularity.

Performance evaluations also significantly differ from Chapter 3. Instead of using a throughput-based metric, measures are selected and defined in accordance with

DASH Industry Forum recommendations [56]. The consumer-side adaptation behaviour is simulated via our own implementation of FESTIVE [26], a well-known mechanism that captures recent advancements in bitrate adaptation. Results show that RippleCache constructions consistently reduce oscillation and re-buffering, while meeting or exceeding the highest levels of competing video quality. The consistent performance, across varying levels of capacity and popularity-skew, lend weight to the argument that high-bitrate content should be kept close to consumers, and lower quality content pushed further away.

The remainder of this chapter is organized as follows. Section 4.2 pinpoints the challenges of adaptation-agnostic caching schemes on adaptive video streaming, followed by the RippleCache cache partitioning principle in section 4.3. To assess the potential gain of RippleCache, we formulate a benchmark solution RippleClassic in Section 4.4, followed by a light-weight and practical embodiment RippleFinder in Section 4.5. Section 4.6 presents our experiment setup and performance evaluation. We conclude in Section 4.7 and present our final remarks.

4.2 Why Do We Partition?

To study the impact of consumer-side bitrate adaptation on cache placement, we carried out extensive experiments to elicit the intrinsic challenge of bitrate oscillation and high bitrate placement. The following characterizations are drawn from evaluations of the benchmark CE2 with Least Frequently Used (LFU) [69]. The evaluation setup is described in Section 4.6.

The salient results are summarized by Figure 4.1, depicting the likelihood of incurring a bitrate adaptation as a function of hop distance between the video consumer

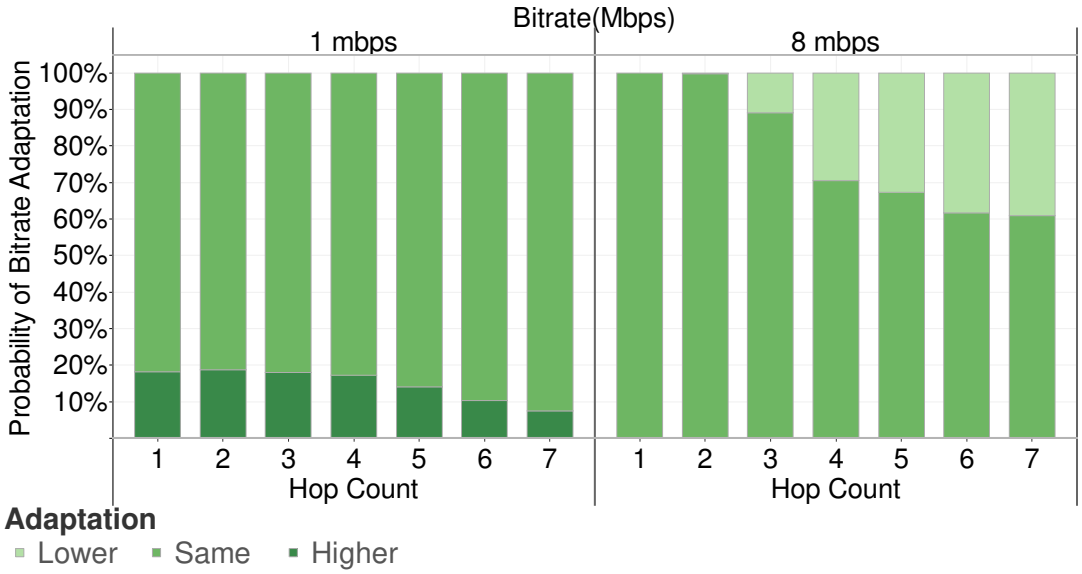


Figure 4.1: Bitrate adaptations given cache distance: dark regions indicate switches to the higher bitrate; lighter regions indicate switches to the lower bitrate.

and the cache. Each vertical bar is shaded according to the the direction of the adaptation: dark regions indicate switches to a higher bitrate; lighter regions indicate switches to lower bitrates; medium shade indicates no bitrate adaptation (same decision). We note that bitrate adaptations may be triggered in response to changes in either or both of network and caching conditions. Thus, the proportion of medium shade is an indication of stable or steady state between video requests with the network and caches that satisfy those requests. In order to reduce bitrate oscillation, this proportion of medium shade is expected to be as much as possible.

Bitrate adaptations occur most frequently relative to cache distances when users consume the lowest (1 mbps) and highest (8 mbps) bitrates under our experimental settings. As depicted in Figure 4.1, the leftmost bars show bitrate adaptations after successful requests for video content at 1 mbps. From among requests for low bitrates satisfied within the first four hops, measurements indicate no significant difference in

the likelihood of a bitrate increase. This suggests a degree of insensitivity to the location of low-bitrate content, with no obvious advantage to caching low-bitrate content closer to consumers at the edge. Instead, caching low-bitrate content in the core network provides an increasing adaptation stability, as the proportion of medium shade increases in the last three hops.

In contrast, the rightmost bars in Figure 4.1 show an opposing trend. Consumers that request high-bitrate content are increasingly likely to switch to lower quality as hop distance increases. Service degradation becomes increasingly unavoidable with hop distance for high-bitrate content. This happens because higher bitrate content consumes a disproportionately greater share of cache and network resources.

The combination of these two sets of observations suggest that lower-bitrate content should be moved into the core to make room for higher-bitrate content at the edges, which demonstrates the need for safe-guarding cache capacity for a particular bitrate. These observations then motivate our design of an adaptation-aware cache partitioning to reduce bitrate oscillation and improve consumers' QoE.

4.3 How Do We Partition?

Our early experiments underscore the need for cache partitioning. However, rather than conventional partitioning on individual cache, we propose **RippleCache** partitioning principle that works upon each *cache path*. A *cache path* is a concatenation of caches that sit on a forwarding path from consumers to a video producer. We say that a RippleCache principle safeguards content along the cache path by prioritizing bitrates in a monotonically decreasing fashion from edge routers.

The bitrate assignments in RippleCache effectively partition caches into concentric

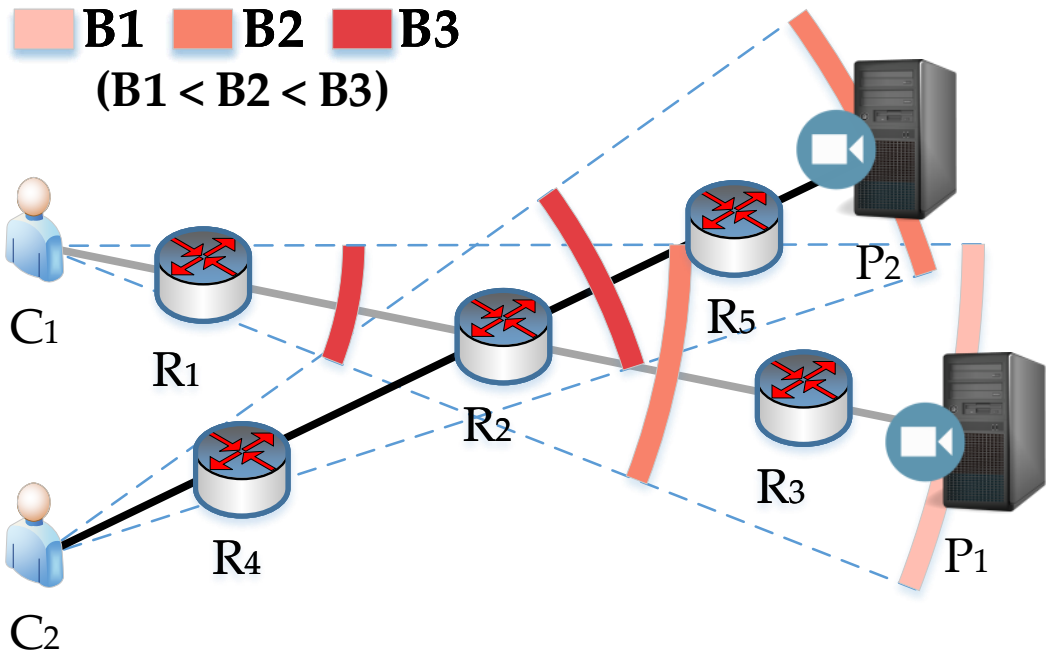


Figure 4.2: Cache partitioning by encoding bitrates along each forwarding path.

regions that we refer to as **Ripple**s. Ripple behaviours derive from its namesake: as much as ripples in liquid ebb and flow, partitions must be dynamic or re-definable in response to changing interest patterns.

A visual representation of a RippleCache is illustrated in Figure 4.2. It shows two independent forwarding paths from consumers C_1 and C_2 to their respective video producers P_1 and P_2. Caches, as guided by RippleCache, are assigned one of three available bitrates $B_1 < B_2 < B_3$, in decreasing bitrate from the consumers. The coloured arcs in Figure 4.2 mark the partition boundaries that delineate *Ripple*s. We note that *Ripple*s may contain zero or many routers. For example, the path from C_2 to P_2 assigns bitrate B_3 to the two routers closest to C_2; the same path omits the lowest bitrate from its partitions, leaving the video producer to satisfy lowest bitrate requests.

Same as ripples in liquid must coincide when they meet, caches that sit on multiple forwarding paths must share their capacity to resolve potential conflicts on cache partitions. For example, the forwarding paths in Figure 4.2 intersect at R_2, where cache space is reversed for different bitrates from these two paths: the same router R_2 is requested to cache both B_3 and B_2. As a result, a spontaneous solution is dividing the cache space at R_2 to ensure a fair share among these two *cache path*s, such that video content with B_2 and B_3 can coincide.

Our proposed RippleCache provides a manifestation of the 'ideal' cache partitioning. However, it is still a guiding principle and must be realized by a caching scheme in practice. A RippleCache implementation must 1. identify appropriate caching decision criteria so that placements may form partitions; and 2. implement a negotiation mechanism to ensure fair share allocations of cache capacity at nodes on intersecting paths. The following sections describe our implementations in RippleClassic as a benchmark and RippleFinder as a scalable and distributed heuristic.

4.4 RippleClassic Benchmark Optimization

Guided by the RippleCache principle described in the previous section, we hereby present the RippleClassic cache placement scheme. RippleClassic is an optimization formulated as a binary integer programming (BIP) problem. Its solutions are cache placements for adaptive video content under diverse network conditions and preferences. These placements serve as the benchmarks, against which we design and compare in later sections.

4.4.1 Cache Placement Problem Formulation

We apply the same settings and notations as detailed in Section 3.3.1, where differences and additional notations are described as follows. We model an ICN as a connected graph $G = (\mathbb{V}, \mathbb{E})$, where nodes in $\mathbb{V}$ are composed of video producers $\mathbb{P}$, edge routers $\mathbb{D}$ and intermediate routers. In this formulation, single-path forwarding and on-path caching are assumed.

Our formulation then caters to diverse caching preferences by maximizing the sum of 'cache reward' values. This cache reward of each request is denoted by $\phi(RB^i_{[d,p]}, b)$, which is an evaluation on the effect of caching on the i^{th} node along path $[d, p]$, $d \in \mathbb{D}, p \in \mathbb{P}$. Each video request is granted a cache reward when cache hit occurs, and the corresponding value is assigned by a reward function that is explained subsequently in Section 4.4.2.

The optimization is formulated as a BIP problem, as outlined below. Given the known complexity of BIP, solving this problem is NP-Complete.

$$\max \sum_{d\in\mathbb{D}} \sum_{p\in\mathbb{P}} \sum_{i=1}^{L} \sum_{f=1}^{F} \sum_{k=1}^{K} \sum_{b=1}^{B} \phi(RB^i_{[d,p]}, b)\xi_d[\delta^i_{[d,p]} - \delta^{i-1}_{[d,p]}]$$

$$\text{s.t.} \quad x_v(f,k,b) \in \{0,1\}, \quad \forall v \in \mathbb{V} \tag{4.1}$$

$$\delta^i_{[d,p]} \in \{0,1\}, \quad \forall d \in \mathbb{D}, \forall p \in \mathbb{P}, 1 \le i \le L \tag{4.2}$$

$$\sum_{f\in F} \sum_{k\in K} \sum_{b\in B} S(b) * x_v(f,k,b) \le C_v, \forall v \in \mathbb{V} - \mathbb{P} \tag{4.3}$$

$$\delta^i_{[d,p]} \ge \delta^{i-1}_{[d,p]}, \tag{4.4}$$

$$\delta^i_{[d,p]} \ge x^i_{[d,p]}(f,k,b), \tag{4.5}$$

$$\delta^i_{[d,p]} \le \delta^{i-1}_{[d,p]} + x^i_{[d,p]}(f,k,b), \tag{4.6}$$

$$\delta^{0}_{[d,p]} = 0, \tag{4.7}$$

$$x_p(f,k,b) = 1, \quad \forall p \in \mathbb{P}, \tag{4.8}$$

$$\delta^{L-1}_{[d,p]}(f',k',b) - \delta^{i}_{[d,p]}(f',k',b) \leq \mathbf{M} - \mathbf{M} * \delta^{i}_{[d,p]}(f,k,b). \tag{4.9}$$

Objective

The objective function maximizes the system-wide cache reward. A higher cache reward value corresponds to a better cache placement. The optimization traverses all forwarding paths starting from each edge router, and accumulates cache reward values on nodes where cache hits occur. Cache rewards are generated once per request where the cache hit occurs. The objective expression thus utilizes the difference between cache indicators δ to avoid infeasible reward values. In cases where a segment is cached multiple times along the forwarding path, $\delta^{i}_{[d,p]}$ and $\delta^{i-1}_{[d,p]}$ would be both equal to 1. Their difference $\delta^{i}_{[d,p]} - \delta^{i-1}_{[d,p]}$, being 0, ensures the correctness of reward calculation. Only where the segment first appears along the path can rewards be accumulated, i.e. where $\delta^{i}_{[d,p]} - \delta^{i-1}_{[d,p]}$ is non-zero.

Constraints

Binary variables are defined in Constraints (4.1) and (4.2). The remaining constraints relate to the *Cache Capacity*, *Caching Status Indicator*, and *Popularity*, as follows.

- The *Cache Capacity* defined in Constraint (4.3) ensures that the total size of cached video content is bound by available cache capacity over all cache routers except

video producer.

• The relationship between *Caching Status Indicator* δ and cache placement decisions x is defined by Constraints (4.4)-(4.7), as already described in Section 3.4.1.

• *Popularity* contributes via Constraint (4.9). The catering to popularity is known to improve the performance of caching schemes [9, 69]. Constraint (4.9) ensures that, whenever there is cache space, popular video content is selected for caching with a higher priority (close to consumers). We utilize the 'big-**M**' approach [21] to ensure caching order, where **M** is any large positive constant number.

The popularity Constraint (4.9) benefits from additional remarks. A ranking table is assumed to exist for each forwarding path; in our own implementation (described in Section 4.6) ranking tables are held and maintained at each edge routers d. Entries in the table are first categorized into bitrates, and then sorted by popularity for each category. We denote (f', k', b) and (f, k, b) as any two consecutive items in this table, where segment (f', k', b) is more popular than content (f, k, b). Constraint (4.9) guarantees that a less popular (f, k, b) cannot be cached closer to consumers than (f', k', b) on the forwarding path $[d, p]$. The result of left hand side is 1 if (f', k', b) is cached on any upstream router from i to penultimate node of the path $[d, p]$. To ensure Constraint (4.9) is not violated, $\delta^{i}_{[d,p]}(f, k, b)$ in the right hand side must then be assigned 0, which represents that (f, k, b) is never cached on a j^{th} downstream router closer to consumers $(1 \leq j \leq i)$.

4.4.2 Cache Reward Function

RippleClassic decides content placement among caches, without explicit knowledge of the interplay between caches and consumers. This information is encoded in and

modelled by the reward function ϕ. The design of ϕ relies on the following intuition: A cache hit is valuable only if the transfer of video content from that cache to the consumer can be reliably sustained for the requested bitrate.

This intuition is demonstrated by an example in Table 4.1. Each cell in the table is the average transfer delay for a 4-second video segment delivered to consumer C from any cache on the forwarding path. The greyed cells delineate the routers from which content can be reliably retrieved within 4-second time for a given encoding. The duration of a video segment (4 seconds) is the deadline for video delivery: meeting this deadline means the requested bitrate is reliable, while missing this deadline may ultimately cause video playback freezing. For example, video requests for B_2 are only sustainable when satisfied on R_1 or R_2; video content retrieved from R_3 or R_4 will arrive 2.5s or 7s late on average.

Transfer delay also gives an indication on the value of a cache hit to the consumer. Referring again to Table 4.1, the delineation by greyed cells also corresponds with consumer adaptations. Consider a consumer that selects content encoded into 4-second segments at bitrate B_2. Table 4.1 says that consumers would maintain or even increase their selected bitrate when content retrieved from R_2 or R_1. Conversely, that same content retrieved from R_3 or R_4 will cause the consumer to avoid playback freezing by reducing its selected bitrate. This type of oscillation is the behaviour observed in Figure 4.1.

The consumer-side adaptation and its interaction with in-network caches are then captured by reward function ϕ that we first introduced in [38], where the numerical reward values were shown to have a high correlation with traditional consumer-side

Table 4.1: An example of average cumulative delay of 4-second segments by hop distance

	$(C..R_1)$	$(C..R_2)$	$(C..R_3)$	$(C..R_4)$
B_3	3s	6.5s	10.5s	16.5s
B_2	1s	3.5s	6.5s	11s
B_1	0.5s	1s	2s	3s

measures of QoE. The function takes two input parameters: (i) the consumer's requested bitrate b and, crucially, (ii) the router's assigned Ripple Bitrate, ($RB^i_{[d,p]}$). Given the i^{th} router on the forwarding path from edge node d to producer p, **Ripple Bitrate** ($RB^i_{[d,p]}$) denotes the *highest* sustainable bitrate that can be delivered to consumers (i.e., the top greyed cell of each column in Table 4.1). We further use RB^i to denote $RB^i_{[d,p]}$, where forwarding path $[d, p]$ is implicit.

The reward function $\phi(RB^i, b)$ is defined as,

$$\phi(RB^i, b) = \begin{cases} \mu(b), & \textbf{if } \ b = RB^i \\ \mu(b^{\uparrow}) * \beta(b) + \mu(b) * (1 - \beta(b)), & \textbf{if } \ b < RB^i \\ \mu(RB^i), & \textbf{if } \ (b > RB^i) \wedge (RB^i \geq b_1) \\ \mu(b_1), & \textbf{otherwise.} \end{cases} \tag{4.10}$$

We note that storage and transmission requirements for the encodings of any single video segment are non-uniform. In order to ensure that similar bias is reflected in the reward, μ is proportional to the base segment size. For the base bitrate at rank 1, $\mu(b_1) = 1$. Any other bitrate b is calculated as $\mu(b) = S_b/S_{b_1}$, where S_{b_1} as the size of the base bitrate segment. A bitrate $b^{\uparrow}$ denotes the next higher bitrate relative to b in the set of discrete bitrates used to encode the video.

Each entry in μ corresponds with a likely behaviour of the consumer relative to

the Ripple Bitrate. The first case triggers when the requested bitrate matches the target rate for the router $b = RB^i$. In this case there are sufficient resources to satisfy subsequent requests at the requested bitrate. The reward function returns $\mu(b)$.

The second case is left for discussion following third and fourth cases. The third case returns when $b > RB^i$, the requested bitrate is higher than Ripple Bitrate. Here a reward lower than $\mu(b)$ should be granted since the cache hit generates load or throughput that may cause consumers to reduce their video quality. As a result the lower reward discourages those cache partitions that can lead to video quality degradation.

The final case triggers when a cache is unable to maintain even the base rate video quality. In this case the ϕ function returns the lowest reward of $\mu(b_1)$, since requests satisfied under such circumstances are likely to lead to buffer-induced freezing, and should be avoided.

Returning to the second case $b < RB^i$, when the requested rate is lower than Ripple Bitrate for the cache. Recall that cache reward is only granted when there is a cache hit. Thus, this case represents a request that is satisfied by the cache, yet for content that should be pushed towards the network core. In this case the return value represents a trade-off. Strictly speaking, a cache hit encourages consumer to subsequently request a higher bitrate $b^{\uparrow}$. However, the additional load on the network could lead to bandwidth fluctuations that cause bitrate oscillation. Thus, care must be taken to avoid over-awards. ϕ returns a weighted sum of $\mu(b)$ and $\mu(b^{\uparrow})$, where the contribution of each component is controlled by parameter $\beta(b) \in [0, 1]$. $\beta(b) = 1$ returns $\phi(RB^i, b) = \mu(b^{\uparrow})$ and prioritizes video quality that consumers can achieve, while ignoring the risk of bitrate oscillation. Conversely, $\beta(b) = 0$ prioritizes

bitrate stability by returning $\mu(b)$. As RippleClassic optimizes cache placement from a system-wide perspective, $\beta(b) = 0$ encourages RippleClassic to relocate video content via matching Ripple Bitrate. As a result, lower quality would be eventually pushed towards the network core.

The question then emerges: What is an appropriate weight? A fixed β fails to capture the disproportional resource increases needed to satisfy requests for higher quality content. We then study consumers' QoE under a variable β, to highlight the trade-off under different design choices.

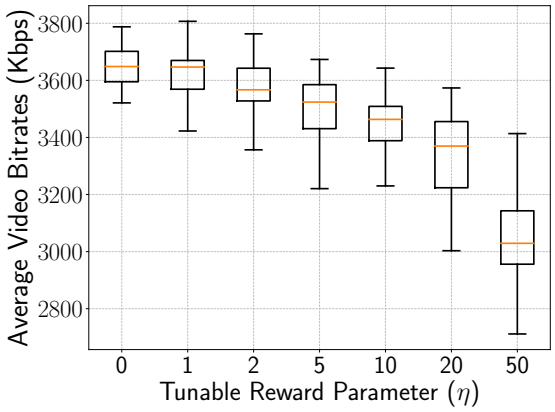
4.4.3 Tuning the Quality-Oscillation Tradeoff

We define $\beta(b)$ in a manner that is inversely proportional to the rank of the bitrate, $rank(b)$, such that

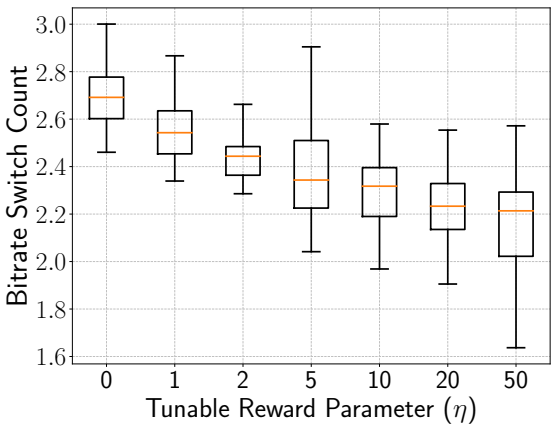
$$\beta(b) = \frac{1}{\eta + rank(b)}. \tag{4.11}$$

The inverse of $rank(b)$ echoes the increasingly conservative nature of rate adaptation controls at higher quality, corresponding with the disproportional increases in resources to support higher bitrates. The high correlation revealed in our previous study between ϕ rewards and consumer QoE implemented the inverse of $rank(b)$, alone [38]. Here we add a tunable parameter η to further explore the trade-off between quality and oscillation implied by β.

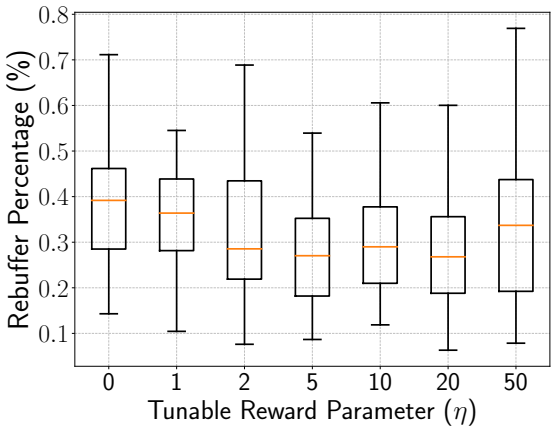
The competing demands between high quality and low oscillation are made evident by the box plots in Figure 4.3. These plots show the impact of β on various measures of consumers' QoE for a range of η values. Performance metric definitions, as well as further experimental design details, are provided in Section 4.6. A smaller η value favours $\mu(b^{\uparrow})$, the reward that emphasizes higher quality. This can be seen in



(a) Expected bitrate.



(b) Quality degradation (lower is better).



(c) Buffer-induced freezing.

Figure 4.3: The impact of tunable cache reward η on consumers' QoE.

Figure 4.3a, where consumers receive the highest quality when $\eta = 0$ and diminishing quality as η values increase. Conversely, Figure 4.3b shows that those larger values of η correspond with fewer adaptations that reduce quality. This happens because larger η values emphasize stability via $\mu(b)$. Finally, Figure 4.3c shows no significant difference in buffer-induced freezing. We take this as evidence that consumer-side adaptations are able to ensure the same degree of uninterrupted playback despite changes in network conditions.

Figure 4.3 points to $\eta = 1$ as striking a good balance between bitrate and oscillation. As $\eta = 0$ emphasizes video quality regardless of cache utilization and resulting bandwidth fluctuation, $\eta = 1$ **would reduce bitrate oscillation without sacrificing on received video quality**. We found this to be true throughout our wider evaluations in Section 4.6.

RippleClassic is designed to be a benchmark partitioning scheme that optimizes for high-bitrate content by pushing lower-bitrate content towards the core. The complexity of RippleClassic presents scalability challenges. In the next section, we design a distributed heuristic that can partition caches according to the RippleCache principles in polynomial time.

4.5 RippleFinder Cache Partitioning

The *NP-Complete* complexity class of RippleClassic is a barrier to deployment at scale. For larger networks, we design the distributed RippleFinder cache placement scheme. RippleFinder manages cache capacity per-forwarding path, rather than per-router. We begin with a high-level description, then follow with the details of each step, before showing that RippleFinder executes in polynomial time.

4.5.1 System Overview

A full execution consists of 6 procedures performed in sequence. RippleFinder begins at each edge router that (1) *ranks video segments* by their utility, and also (2) *discovers the total cache capacity* of the path. The edge router uses this information to (3) *push and pop* entries from the full ranking tables into new bitrate-specific stacks. Edge routers' final step is to (4) *nominate the caching candidates* for video content at each

router on the forwarding path.

A system-wide representation appears in Figure 4.4, where *Cache Candidate Tables (CCTs)* for routers R_1, R_2 and R_3 (in blue, red, and green, respectively) are generated following Steps (1) (2) (3) and (4). R_1 is processed immediately on R_1,

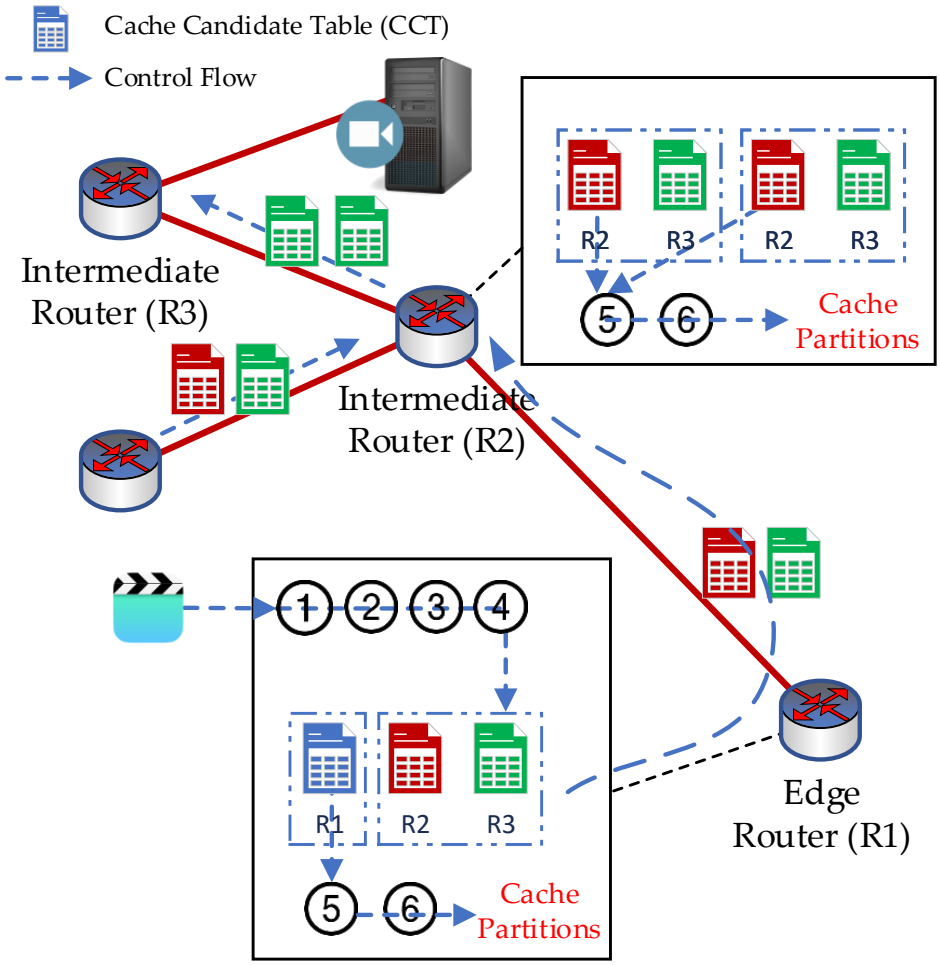


Figure 4.4: RippleFinder diagram. Edge router R_1 would create *Cache Candidate Tables* (CCTs) for R_1, R_2 and R_3. CCT for R_1 is processed immediately on R_1. CCTs for R_2 and R_3 are delivered upstream. The intermediate router R_2 would intercept all CCTs for R_2 (the icon in red color), and forward CCTs for R_3.

while the other two tables are delivered to upstream routers. Procedures (1)-(4) are repeated at each ICN edge router for each forwarding path.

Subsequent steps (5)-(6) are executed by all routers in the system. Routers that sit on intersecting paths must then (5) *negotiate their finite cache capacity* between competing paths once all candidate tables are received, since the total size of video segments in these candidate tables may exceed the cache capacity of this router. Note that the resulting cache capacity allocated to each path will differ from the initial values in Step (2). In the final Step (6), each router updates and returns this new values to the respective edge routers.

Steps (2) to (6) are repeated until cache capacity values at nomination phase (Step (4)) match the values after negotiation (Step (5)). This iteration is guaranteed to terminate, as is explained following the details of individual steps.

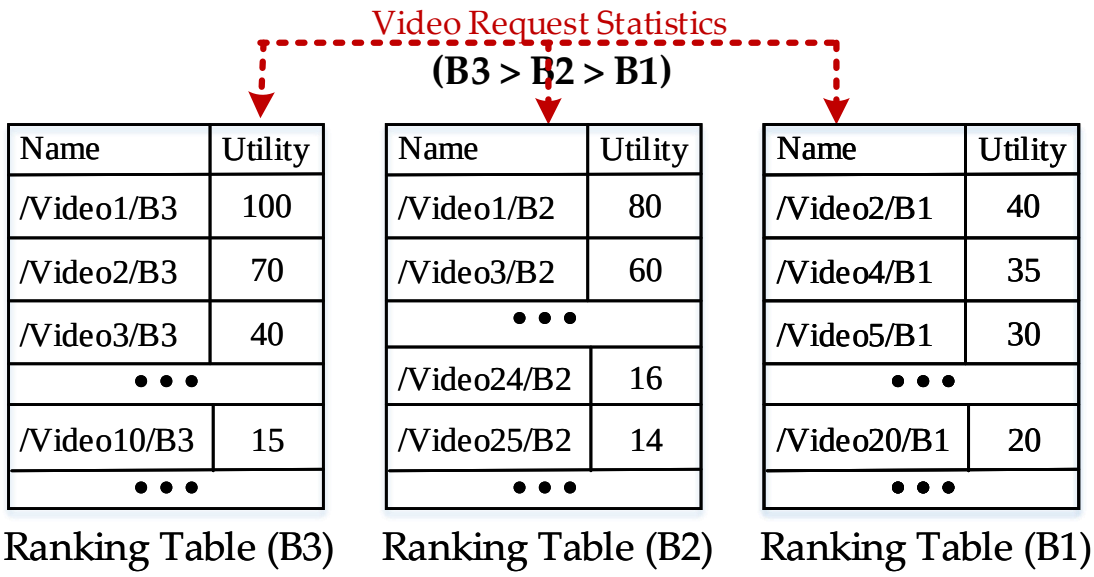
4.5.2 RippleFinder in Execution

Each individual step is described below with numbering that corresponds to the system overview.

(1) Ranking Table Construction. Video statistics are used to rank content by utility, for each bitrate, as shown in Figure 4.5-1. Every entry in a ranking table consists of the name of the content and the corresponding caching utility $\mathbb{U}$, sorted from high utility to low. The cache utility for video segment indexed by (f, k, b) is calculated as,

$$\mathbb{U}_d(f, k, b) = \mu(b) * \xi_d(f, k, b). \tag{4.12}$$

$\mu(b)$ and $\xi_d(f, k, b)$, both previously defined in Section 4.4, are a value proportional to the size of video segment and the number of requests, respectively. This notion

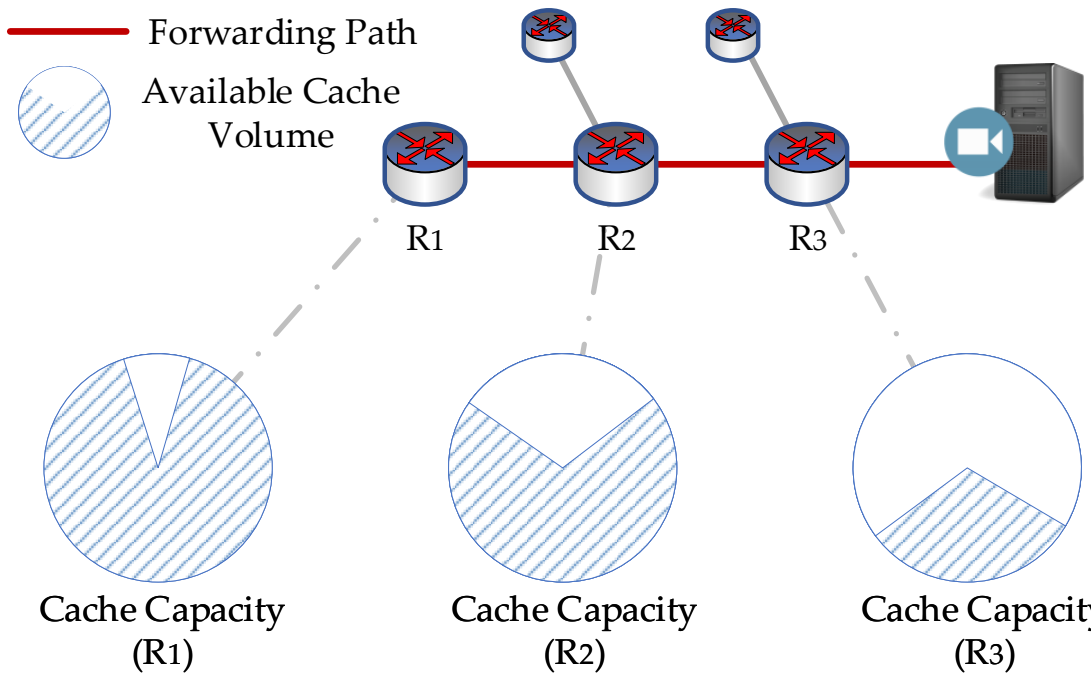


(1) **Ranking Table Construction**. Construct ranking tables for each bitrate (B_1, B_2 and B_3) from video statistics collected by edge router.

Figure 4.5: RippleFinder in execution [Step (1)].

of utility emphasizes video content that is both costly to deliver and highly popular. The caching decisions would then cater to video segments with high overall utility.

(2) Cache Capacity Discovery. The core of RippleFinder manages the entire cache capacity along each forwarding path. In this step, available cache volumes of routers dedicated to a forwarding path $[d, p]$ of length L are concatenated so that the total path capacity is $C_{[d,p]} = \sum_{j=1}^{L} C^j_{[d,p]}$. We note that $C^j_{[d,p]}$ differs from our earlier definition of C_v, where C_v represents the entire caching space on a certain router v. For any $j = v, C^j_{[d,p]} \leq C_v$ since the volume at a router dedicated to a path must be upper bounded by the router's cache capacity. In RippleFinder, the initial value for $C^j_{[d,p]} \leftarrow C_v$. However, as caching decisions are made along each forwarding path independently and routers in an ICN may be shared by multiple paths, one cannot guarantee that our initial assumption always remains valid. As shown in Figure 4.5-2,

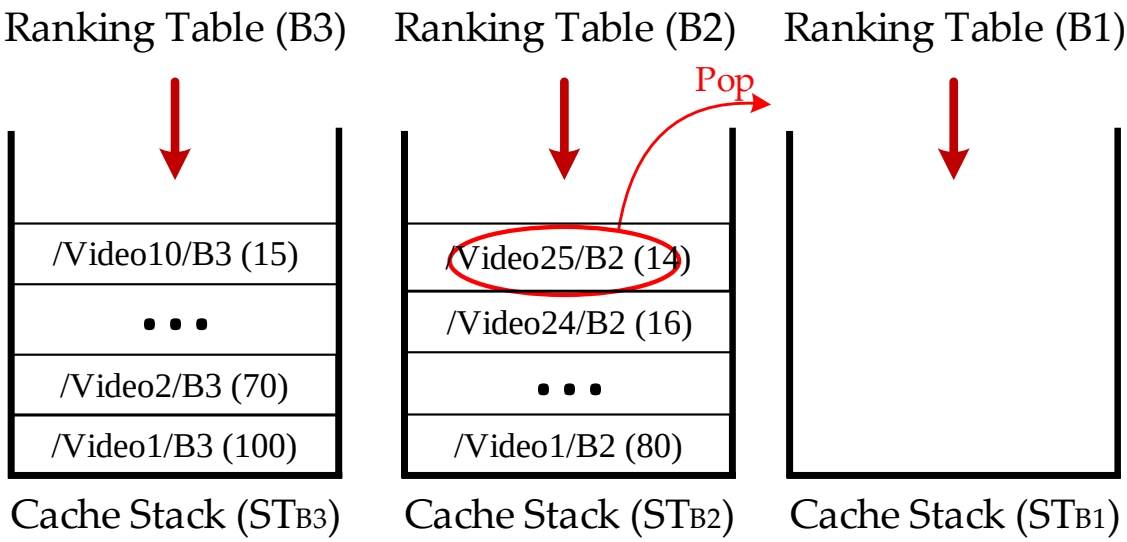


(2) **Cache Capacity Discovery**. The shaded volume of router R_1 , R_2 and R_3 would be used to cache video content delivered along forwarding path. These shaded volume is added together, with a size of $C_{[d,p]}$.

Figure 4.5: RippleFinder in execution [Step (2)].

only portions of the cache capacity at ICN nodes may be allocated to a forwarding path, so that some capacity may be reserved to content delivered through other paths. The volume of a cache on the path may be adjusted in later steps. Consequently, the cache capacity discovery marks the beginning of an iteration that ends with cache volume updates in Step (6).

(3) "Push" and "Pop". In this intermediate step, a cache stack ST_b is populated for each of the bitrate ranking tables in Step (1). Entries from ranking tables are pushed into the corresponding stack in descending order. The ordering can be seen in Figure 4.5-3, where higher bitrate stacks are filled before lower bitrate stacks, and within each stack the higher utility items sit deeper than lower utility items. Ranked



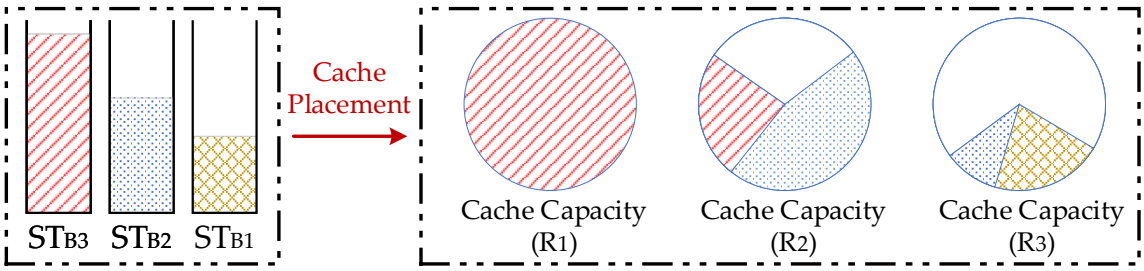
(3) **"Push" and "Pop"**. Video content is pushed into *Cache Stack* by ranking order. After content '*/Video25/B2*' is pushed into ST_{B_2}, the Equation 4.13 is violated, which triggers 'Pop' operation. Since utility of '*/Video10/B3*' on top of ST_{B_3} is higher than '*/Video25/B2*' on top of ST_{B_2}, video segment '*/Video25/B2*' is popped.

Figure 4.5: RippleFinder in execution [Step (3)].

entries are pushed into the stacks until the 'stacked' size of the video segments exceeds the total available cache capacity, i.e.

$$\sum_{b \in B} Size(ST_b) > C_{[d,p]}. \tag{4.13}$$

Once the cache size required by stack elements exceeds capacity $C_{[d,p]}$, RippleFinder pops and pushes entries as follows, and depicted by example in Figure 4.5-3. Until constraint $\sum_{b \in B} Size(ST_b) \leq C_{[d,p]}$ is restored, RippleFinder compares the top entries of each stack and pops the entry with lowest utility. Note that it is possible for a least-utility entry in a high-bitrate stack to have less utility than the least-utility entry in a lower-bitrate stacks. Once constraint $\sum_{b \in B} Size(ST_b) \leq C_{[d,p]}$ is restored, pushing resumes as normal until the known capacity is again exceeded. Stack operations

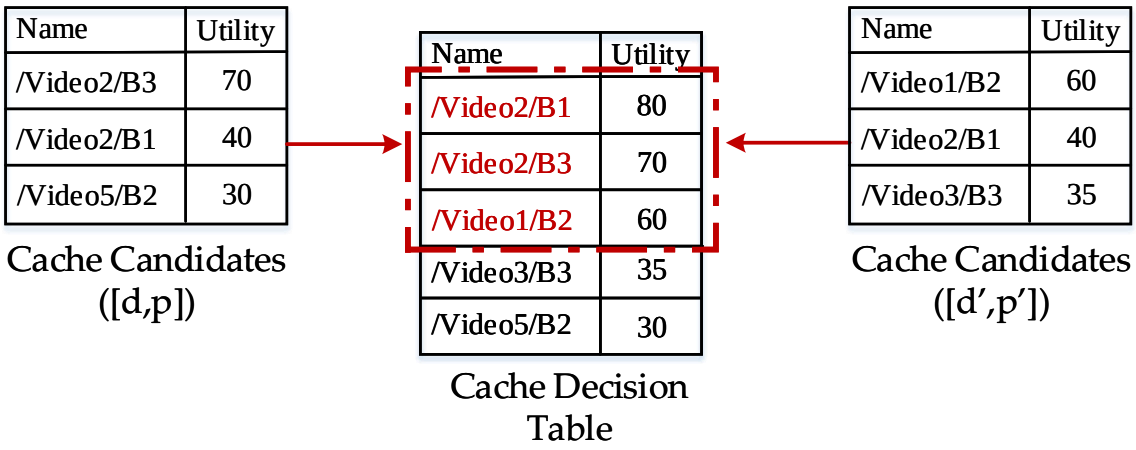


(4) **Cache Candidate Nomination**. Video segments in *Cache Stacks* are assigned to *Cache Candidate Tables* (CCTs). The assignment occurs first at CCT for R_1, followed by R_2 and R_3. Video content in ST_{B_3} is first arranged, followed by ST_{B_2} and ST_{B_1}.

Figure 4.5: RippleFinder in execution [Step (4)].

continue until the lowest bitrate stack is marked *complete*. A stack is marked complete when the popped video content is taken from the stack that is currently being filled since the overall cache utility can no longer be improved by continuing to push content into the current stack. The ordering of push operations ensures that higher bitrate stacks will always be marked complete before lower quality stacks. The content corresponding to entries that have been popped or that remain in the ranking tables would be excluded from cache placement.

(4) Cache Candidate Nomination. For each cache node along the forwarding path, the edge router constructs a Cache Candidate Table (CCT). Tables are populated with entries from the cache stacks, again in descending bitrate order, starting from the CCT for edge router itself. We note that content from any stack may span multiple tables. For example, the depiction in Figure 4.5-4 shows content from the stack for B_3 has filled the CCT for router R_1, and overflows into the CCT for R_2. The sum sizes of content assigned to candidate tables are capped by the capacities reported to the edge router during discovery in Step (2). Since the total space required by all items in all stacks is constrained by the path capacity $C_{[d,p]}$, every item



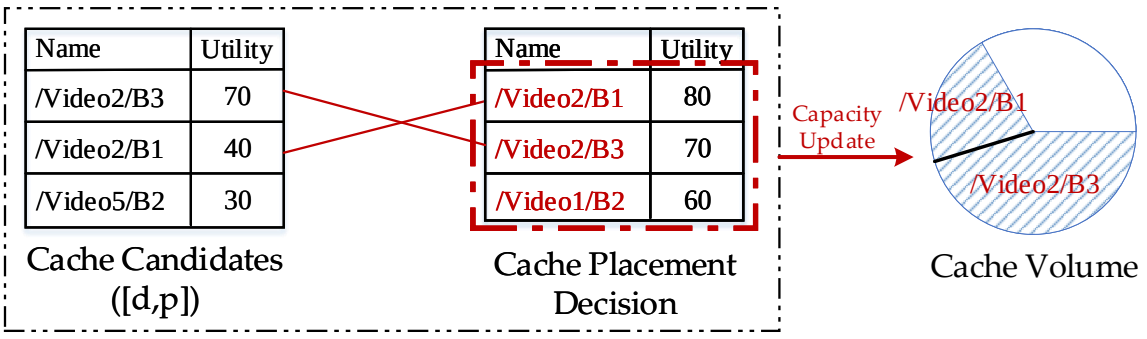
(5) **Cache Placement Negotiation**. The cache placement decision is made by (1) merging CCTs received from path $[d, p]$ and path $[d', p']$, and (2) choosing video segments with high utility as long as the cache capacity of this router C_v allows.

Figure 4.5: RippleFinder in execution [Step (5)].

in the stacks finds a place in a CCT. The result adheres to RippleCache ideals by assigning high-bitrate content in tables for ICN routers closer to consumers, leaving lower bitrate content for CCTs bound towards the core.

(5) Cache Placement Negotiation. Cache nodes receive a candidate table from each of its forwarding paths. The combined entries from all CCTs may exceed the cache's capacity and must be negotiated. Nodes rank video segments from all CCTs by the *sum* of the content's utility. In Figure 4.5-5 the individual utility values for '*/Video2/B1*' from left and right tables are summed to a utility of 80. Each router $v \in \mathbb{V}$ can cache up to its cache capacity C_v, according to the sorted utility from high to low.

(6) Cache Volume Update. Since sum utility is used to populate a cache, the portion of capacity dedicated to a path may be smaller than was previously reported in Step (2). In the example shown by Figure 4.5-6, a router would cache the three



(6) **Cache Volume Update**. The final caching decisions are compared against the items in each CCT. Segments '*/Video2/B1*' and '*/Video2/B3*' appear in both final cache placement and CCT. The updated cache volume of this router to path $[d, p]$ would be equal to the total size of these two segments.

Figure 4.5: RippleFinder in execution [Step (6)].

video segments enveloped in the red dot-dash line. The volume size dedicated to path $[d, p]$ is then equal to the size of both segments '*/Video2/B3*' and '*/Video2/B1*'. Updated volume sizes are returned to the respective edge routers along the reverse of forwarding paths, so that the available cache capacity $C_{[d,p]}$ for a entire path can remain current.

At this stage, the updated volume sizes are compared with previous values obtained in Step (2). A mismatch triggers another iteration of Steps (2)-(6). RippleFinder terminates when $C_{[d,p]}$ is unchanged for all forwarding paths between two consecutive iterations. RippleFinder is guaranteed to terminate. In any iteration, candidate tables are constructed with reported cache volume sizes. Since cache candidates nominated by edge routers may be omitted from the final placement at core routers, $C^j_{[d,p]}$ decreases monotonically. The worst possible case is that no cache capacity is allocated for a path, meaning that $C^j_{[d,p]}$ is capped by 0. Since no volume can be negative iteration must eventually end.

4.5.3 RippleFinder Algorithm and Complexity

RippleFinder is a distributed algorithm with polynomial time complexity of the number of paths in the system, $|\mathbb{D}| \cdot |\mathbb{P}|$.

For ease of presentation, RippleFinder is written as a single-thread of execution in Algorithm 3. The analysis pertains to edge routers since only edge routers execute the full set of operations; intermediate routers are limited to negotiating placements and updating cache volumes. Line 4 constructs B ranking tables, each of up to size FK, with sorting complexity $\mathcal{O}(B \cdot FK \log(FK))$. Line 5 iterates over every router in each forwarding path to update cache capacity, with a complexity of $\mathcal{O}(|\mathbb{V}|)$. Both Lines 6 and 7 each scan over existing data structures in a time that is linear with the size of the structures, of $\mathcal{O}(BFK)$. Thus, the overall complexity for the full set of tasks (Line 3-8) is $\mathcal{O}(|\mathbb{P}|BFK \log(FK) + |\mathbb{P}||\mathbb{V}|)$, as the same operations have to repeat for totally $|\mathbb{P}|$ number of forwarding paths. The complexity incurred by Steps (5) and (6) at all routers is dominated by merging CCTs at Line 9. CCT is already a sorted table, and the length of each CCT is capped by cache capacity and also expected to be markedly less than the length of ranking tables (that contain all requested video content) at Line 4. As such the complexity at Line 9 is bound by that at Line 4. Therefore, the complexity of RippleFinder is $\mathcal{O}(|\mathbb{P}|BFK \log(FK) + |\mathbb{P}||\mathbb{V}|)$.

4.6 Performance Results and Insights

We evaluate RippleClassic and RippleFinder performance via simulation against known caching strategies on the NDN architecture. Results reinforce the broader merits of cache partitioning for adaptive streaming. We claim without loss of generality that the merits of RippleCache designs and subsequent analyses can be applied on other

Algorithm 3: RippleFinder

Input: Edge router d; set of producers $\mathbb{P}$; length of routing path L for each $(d, p), p \in \mathbb{P}$; dedicated cache volume $C^j_{[d,p]}$ at each hop.
Output: Adaptation-aware cache placement x_d on router d.

Initialize available cache volume $C_{[d,p]} \leftarrow \sum_{j=1}^{L} C^j_{[d,p]}$
repeat
  for all $p \in \mathbb{P}$ **do**
    Ranking Table Construction
    $C_{[d,p]} \leftarrow$ *Cache Capacity Discovery*
    "Push" and "Pop"
    $CCT \leftarrow$ *Cache Candidate Nomination*
  end for
  // $j = 1$ as RippleFinder is working on an edge node
  $C^1_{[d,p]}, x_d \leftarrow$ *Cache Placement Negotiation*
  for all $p \in \mathbb{P}$ **do**
    $C'_{[d,p]} \leftarrow$ *Cache Volume Update*
  end for
until $C_{[d,p]} = C'_{[d,p]}$
return x_d.

ICN architectures and cache hierarchies.

4.6.1 Simulation Setup and Parameters

The proposed cache partitioning schemes were implemented onto ndnSIM [2], an NS-3 based simulator. Each NDN router is allocated a Content Store (CS), where its size C_v is subject to a total available system capacity, controlled by ω, as

$$C_v = \frac{\sum \text{Size of Video}}{\#\text{ of NDN Routers}} * \omega, \forall v \in \mathbb{V}.$$

Consumer-side adaptation behaviour is simulated via our own implementation of FESTIVE [26], a throughput-based mechanism that captures recent advancements in bitrate adaptation. Users' interests in video content vary across different video files,

captured by a *Zipf*-like distribution (controlled via skewness parameter α). Videos are comprised of 4-second segments. Each video segment is prepared at 1, 2.5, 5, and 8 Mbps, which are recommended encoding bitrates by YouTube [67]. Consumers initiate a session first by requesting a video file and retrieving video-related metadata (i.e. the Media Presentation Description (MPD)) from the producer. Interests in videos are triggered following a Poisson process, with an average time interval between two consecutive interests being 300 seconds. Once interest for a video file is initiated, subsequent requests for the session are initiated by the bitrate adaptation algorithm.

Three additional caching schemes are evaluated alongside our proposed RippleFinder and RippleClassic for comparison. CE2 [69] with LRU, also with LFU, is a baseline that commonly appears in literature [69]. ProbCache [49] serves as a baseline for probabilistic caching [9]. Both RippleClassic and RippleFinder are cache placement schemes. As the interaction between in-network caches and consumer-side adaptation exists, the caching decisions from RippleClassic and RippleFinder are updated iteratively to keep up with the changes on consumers' preferred bitrates. The iteration on RippleClassic stops once two consecutive optimization produces similar cache rewards. RippleFinder stops after a fixed number of iterations, where we observe a stable performance on consumers' QoE.

Two separate networks are implemented for evaluation. A smaller 16-node ICN network with a maximum 7-hop distance from a video producer to consumers allows the binary integer programming from RippleClassic to find solutions in a reasonable time. Variations on hop distance are used to cause different video access delay by consumers. We choose network link capacity at 20 Mbps, and as a result, the highest

Table 4.2: Simulation parameters for RippleClassic and RippleFinder evaluation

NDN	BIP-tractable	Large-scale
Number of video files	25	500
Number of video segments per file	25	50
Number of NDN routers	16	42
Video segment playback time (sec)	4	4
Number of video producers	1	3
Number of video consumers	32	84
Encoded bitrates (Mbps)	{1, 2.5, 5, 8}	{1, 2.5, 5, 8}
Request interval on video file (sec)	300	300
Bandwidth (Mbps)	20	20
Cache reward parameter (η)	1	
Skewness factor (α)	1.2	1.2
Content store size percentage (ω)	0.2	0.05
FESTIVE		
Drop Threshold	0.8	0.8
Combine Weight	8	8

bitrate (8 Mbps) cannot be retrieved directly from the producer and must be provided by caches. We choose this relatively small link capacity to examine the performance that is enhanced by caching policies. Results for RippleClassic are shown for $\eta = 1$. Recall from Figure 4.3 and surrounding discussion that $\eta = 1$ appears to strike a good balance between prioritizing high bitrates and low oscillation.

A larger 42-node topology generated by BRITE [48] is used to evaluate cache partitioning in a realistic and large-scale system with multiple producers. The complete list of simulation parameters for both scenarios/topologies is listed in Table 4.2.

Results are evaluated using standard QoE metrics published by the DASH Industry Forum [56]. From the standard set, we adopt three metrics, *Average Video Quality*, *Bitrate Switch Count* and *Rebuffer Percentage*, as described in their relevant sections. Each set of evaluations is repeated across a range of content store size ratio

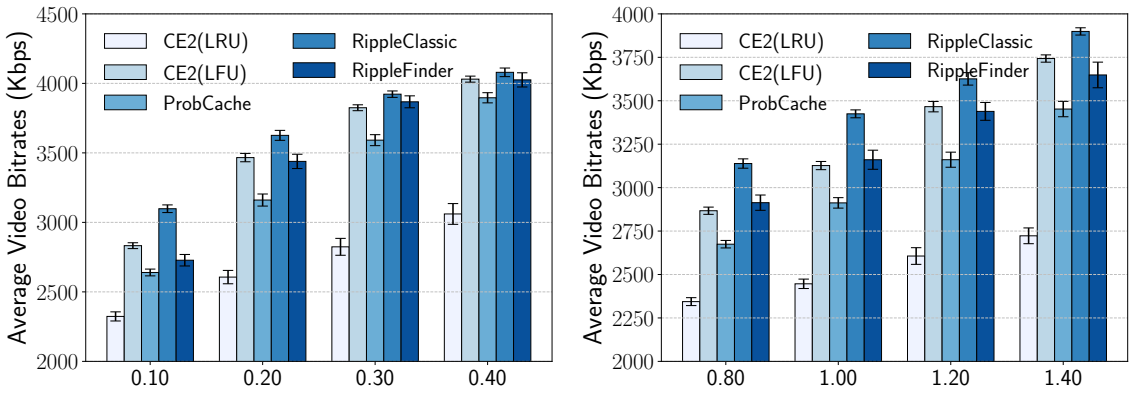
ω and popularity-skewness parameter α. All results are presented at a 95% confidence level.

4.6.2 Average Video Quality

Figures 4.6a and 4.6b show the Average Video Quality, defined as the average video bitrate that consumers request among all video sessions [56]. Measurements indicate that RippleClassic and RippleFinder performance meet or exceed CE2 with LFU and ProbCache. We observe that the gap in performance against the benchmark RippleClassic grows proportionally larger as cache resources diminish. For example, Figure 4.6a shows that when the cache capacity ratio is $\omega = 0.2$, RippleClassic delivers higher video quality than CE2 with LFU by 4.6%. When the total cache capacity drops to 0.1, RippleClassic delivers an average bitrate 9.4% better than CE2 with LFU. RippleFinder results in a similar performance as CE2 with LFU across all tested cache capacity, with one exception. At $\omega = 0.1$, RippleFinder delivers an average bitrate 3.9% lower than CE2 with LFU. Later observations show that RippleFinder magnifies such small differences by substantially reducing bitrate oscillation irrespective of caching resources.

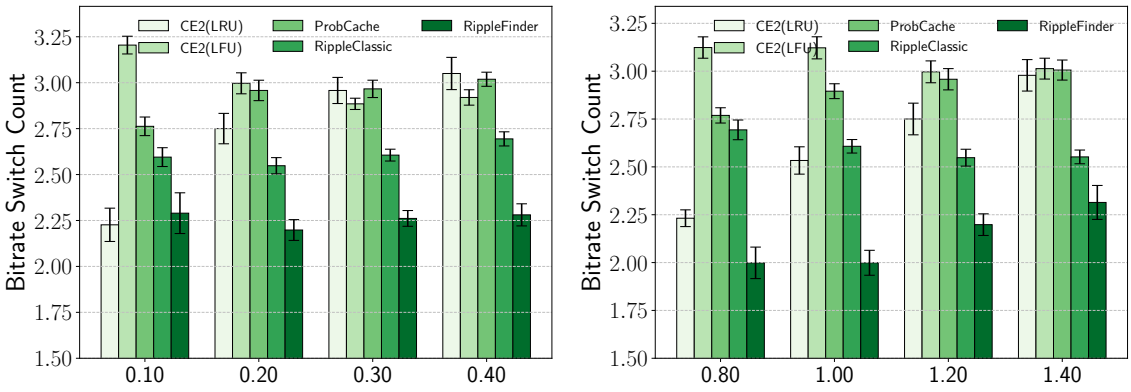
The trend is similar to popularity skewness, shown in Figure 4.6b. As expected, the average video bitrates increase among all caching schemes as the skewness parameter α grows from 0.8 to 1.4, since a greater number of requests target fewer video content. Here, too, RippleClassic and RippleFinder will distinguish themselves via improvements in reducing oscillation.

When comparing both RippleCache-guided schemes to each other, we observe measurable differences when cache capacity and skew diminish. This is explained by



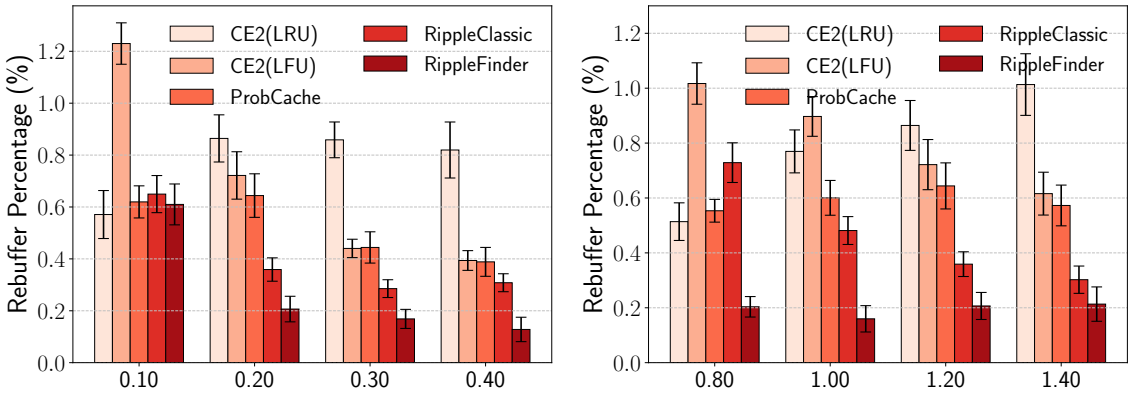
(a) Content Store Size Percentage (ω) (b) Popularity Skewness (α)

Figure 4.6: Average Video Bitrate under 'BIP-tractable' settings



(a) Content Store Size Percentage (ω) (b) Popularity Skewness (α)

Figure 4.7: Bitrate Switch Count under 'BIP-tractable' settings



(a) Content Store Size Percentage (ω) (b) Popularity Skewness (α)

Figure 4.8: Rebuffer Percentage under 'BIP-tractable' settings

the design of RippleClassic to optimize the use of available resources against request patterns. In contrast the advantages of optimization over popularity-based schemes, including RippleFinder, diminish as capacity resources grow or request patterns become predictable.

4.6.3 Bitrate Switch Count

The bitrate switch count measures oscillation by recording the frequency of bitrate switches (including both upgrade and downgrade) in a video session [56]. Results are shown in Figures 4.7a and 4.7b, as cache capacity and popularity distribution are made to vary, respectively. In all evaluations, both RippleClassic and RippleFinder reduce bitrate oscillation when compared with popularity-based CE2 with LFU and ProbCache. When cache capacity is lowest, or popularity least skewed, CE2 with LRU appears to meet or exceed RippleFinder or RippleClassic scheme. The corresponding video bitrate observations for LRU show that this comes at the cost of video quality. The lower video quality for LRU also explains the low degrees of oscillation. Coupled with Figures 4.6a and 4.6b, we see that even when $\omega = 0.1$ (or $\alpha = 0.8$), our RippleCache-guided schemes are seen to reduce oscillation while sustaining the highest levels of video quality.

Observations in Figures 4.7a and 4.7b also indicate that RippleFinder outperforms RippleClassic, despite both adhering to RippleCache ideals. The performance gap may be explained by their difference in caching decision criteria. Recall that RippleClassic implements a reward system that approximates adaptation behaviour. This has the effect of optimizing for the consumer's criteria, namely maximal sustainable bitrate. Conversely, the use of utility in RippleFinder embeds conventional

notions of hit ratio, albeit on a per-path basis. This has the effect of stabilizing video throughput across a single logical cache despite being distributed over multiple volumes. The differences in performance between the two RippleCache schemes are reflective of their different emphases on consumer vs. cache performance.

4.6.4 Rebuffer Percentage

Short-term variations in network and system conditions can adversely affect playback before bitrate adaptations are triggered. One such indication is buffer-induced pausing during playback that manifests on-screen as 'freezing'. We measure the impact of 'freezing' in terms of rebuffer percentage, which is the average time spent in a video freezing state over the active time of a video session [56]. Results are shown in Figures 4.8a and 4.8b.

Since video playback freezing relates to the access delay of media segments, caching schemes that achieve high hit ratios must be able to deliver segments before they are needed for playback, otherwise the playback will freeze. This can be seen in Figures 4.8a and 4.8b, where both RippleCache-guided caching schemes outperform the others. As large amount of video segments with high quality would significantly increase the network delay and choke video traffic, RippleFinder and RippleClassic reduce system-wide traffic load by satisfying high-bitrate requests as early as possible. Only when the request distribution is least skewed ($\alpha = 0.8$) or there exists limited cache capacity ($\omega = 0.1$), does RippleFinder or RippleClassic performance diminish to a degree matched by popularity-based schemes. It is also noticeable that the freezing time for all tested caching schemes is relatively short. This is because the simulation settings assume a small video file to measure the video freezing. A small video

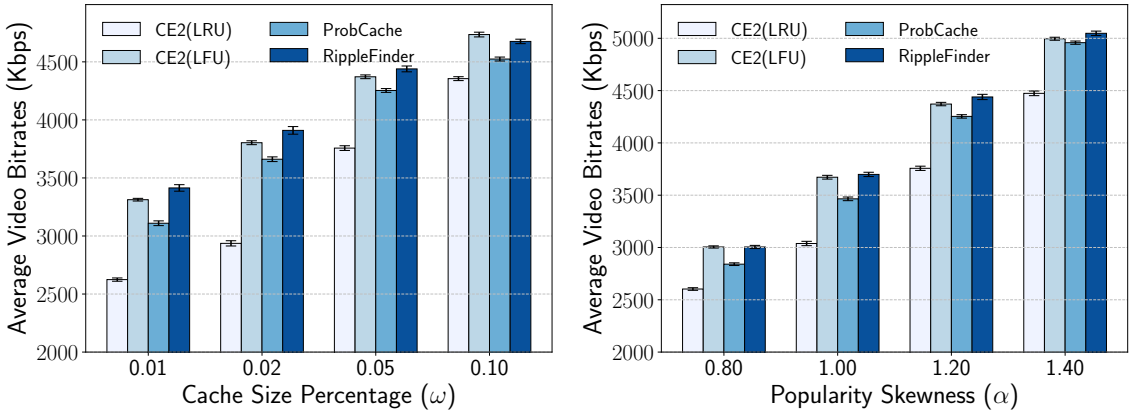
file is already representative to differentiate the performance between our proposed RippleCache schemes and other benchmark solutions. It can be expected that for a larger video file, the freezing time would increase correspondingly.

Intuitively, average video bitrate and rebuffer percentage are conflicting measures, i.e, a higher video bitrate probably leads to a worse playback freezing. However, simulation results from Figures 4.6 and 4.8 imply that the relationship between these two metrics is more subtle. In support of intuition, for example, at cache capacity $\omega = 0.1$ LFU delivers higher video quality than LRU, but causes almost twice playback freezing. The perceived relationship between metrics is broken when comparing RippleFinder at the same $\omega = 0.1$. Here, RippleFinder delivers the higher video quality matching LFU but maintains the same rebuffer ratio as LRU. Collectively these observations reinforce that, in distributed multimedia caching systems, cache hits have value only if their occurrence is useful to the consumer.

4.6.5 Evaluation On A Realistic Topology

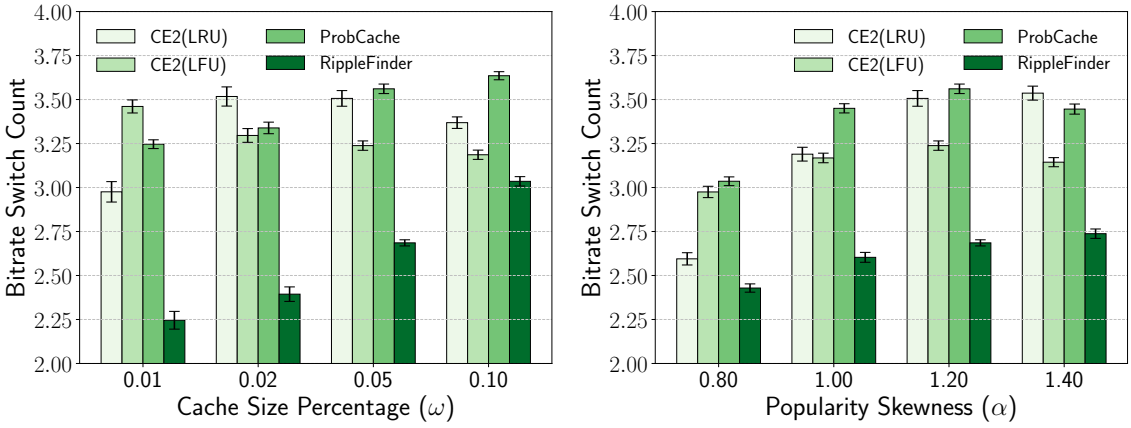
We evaluate over a large 42-node autonomous system (AS) topology generated using BRITE [48]. The Barabási-Albert (BA) model is first selected to build an autonomous system(AS)-level structure. Connections between ICN routers within each AS are established randomly. A total of 84 video consumers are connected to this network, and request for video content from three producers. Each producer provides 500 video files, each consisting of 50 segments. Remaining simulation settings for this large-scale scenario are listed in Table 4.2.

Results in Figures 4.9-4.11 show that RippleFinder performance trends are similar to previous observations. In particular, RippleFinder meets or exceeds competing



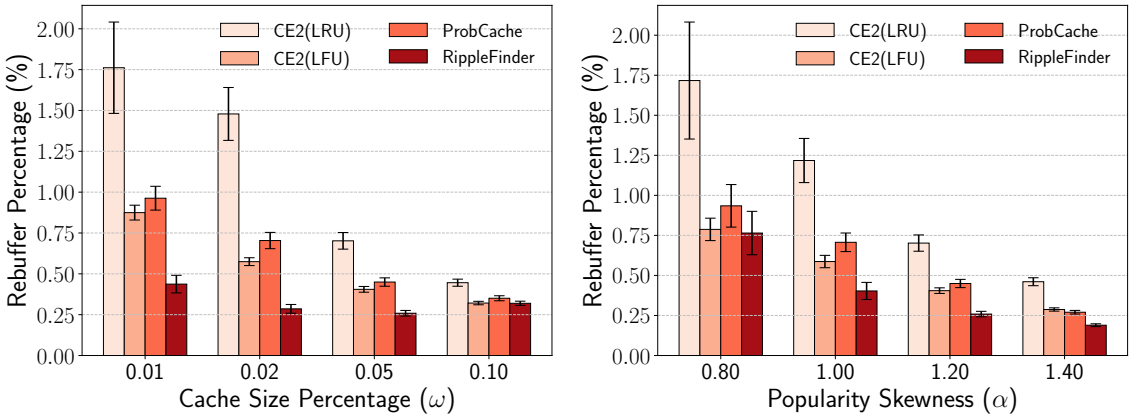
(a) Content Store Size Percentage (ω) (b) Popularity Skewness (α)

Figure 4.9: Average Video Bitrate under 'Large-scale' settings



(a) Content Store Size Percentage (ω) (b) Popularity Skewness (α)

Figure 4.10: Bitrate Switch Count under 'Large-scale' settings



(a) Content Store Size Percentage (ω) (b) Popularity Skewness (α)

Figure 4.11: Rebuffer Percentage under 'Large-scale' settings

levels of video quality and rebuffering, while significantly reducing bitrate oscillation.

This consistent performance of RippleCache-guided design across topologies is also noteworthy. When compared to trends of the smaller network captured in Figures 4.6 and 4.8, the performance of competing schemes appears to be affected by size and topology. For example, Figures 4.8a and 4.8b show CE2 with higher rates of rebuffering for LFU than LRU at small ω or small α, respectively. However, in the representative topology LFU and LRU performance is inverted, as can be seen in Figures 4.11a and 4.11b. These differences further demonstrate the poorly understood interactions between caching and adaptation controls. The consistent QoE performance delivered by RippleFinder across different traffic patterns and topologies is important for real-world deployments.

4.6.6 Discussion of Results

Throughout our evaluations we notice the ability of CE2 with LFU in terms of delivered video quality and playback freezing. Looking ahead, the robustness of LFU suggests that performance gains promised by ICNs specifically, and caching hierarchies generally, may be dependent on their ability to exploit content characteristics. Otherwise caching mechanisms may be mooted by simple popularity, alone, and the corresponding simplicity of LFU.

The general hypothesis that cache placement should be informed by content characteristics is reinforced by RippleFinder/RippleClassic observations. By designing a cache placement scheme for adaptive streaming content, we draw insights that run counter to convention. Lower quality content that is pushed into the core, for example, can improve end-user QoE. Edge caches are left with additional capacity for

higher-quality content. Consequently, content quality at all bitrates becomes network-limited rather than cache-limited.

4.7 Summary

In this chapter, we have argued that ICN cache placement should be tailored for adaptive streaming, as bitrate adaptation mechanisms appear to clash with generic ICN caching techniques. We highlight the issue of oscillation dynamics which is caused by the interplay between in-network caching and bitrate adaptation control, present a primer in a novel approach to caching, and establish the premise of safe guarding cache partitions for higher bitrates, allowing for more ideal cache placement strategies for adaptive video content.

Our proposed safe-guarding mechanism enforces bitrate-based partitioning of cache capacities, named RippleCache, in order to stabilize bandwidth fluctuation. In RippleCache, a network of caches is viewed along each forwarding path from consumers, where the essence is safeguarding high-bitrate content on the edge and pushing low-bitrate content into the network core. To validate the concept and demonstrate the potential gain of RippleCache, we implement two cache placement schemes, RippleClassic and RippleFinder, where our experiment results contrast to leading caching schemes, and demonstrate how cache partitioning would improve consumers' QoE, in terms of high video quality and significant reduction on bitrate oscillation.

More importantly, our explorations yield the following three conclusions. 1. The operational mandate of bitrate adaptation algorithms significantly impacts in-network caching schemes, thus caching must seamlessly cooperate with adaptation. Existing schemes that apply a snapshot approach cannot be applied directly for adaptive

streaming application, as they ignore the need of cooperation, which results in severe bitrate oscillation. 2. The problem of bitrate oscillation can be tackled by concatenating caches along a forwarding path into a *cache path*. Although cache hits on a standalone router would result in similar throughput, adaptation-level dynamics vary across encoding bitrates such that even exact same throughput will not bring the same adaptation decision for each bitrate. By zooming out our view from one cache to the range of a forwarding path, we can arrange video content of different bitrates at a hop distance from consumers that maximizes the chance of maintaining the same adaptation decision, which is the key to avoid bitrate oscillation. 3. It is possible for a caching scheme to deliver video consumers high-quality content while ensuring near-zero playback freezing and minimal bitrate oscillation. Our experiments demonstrate that there is significant room of improvement for future caching policies to enhance QoE by practicing cache partitioning and inheriting from RippleCache principle. This study paves the way for caching schemes that can interact with bitrate selection algorithms, and handle the dependency between adaptation control and caching via network prediction for future request patterns.

Chapter 5

Predictive Cache Partitioning

5.1 Introduction

Bitrate oscillation is one of the enduring challenges in cache management, caused by the interplay of cache placement and throttling actions by adaptation control [64]. This challenge was addressed in Chapter 4, where it was shown that cache partitioning can effectively reduce bitrate oscillation while maintaining high streaming quality. However, similar to most popularity-based caching schemes, both *RippleClassic* and *RippleFinder* derive caching decisions from recent request statistics. This operation typically involves iteratively updating the cached bitrates, with the assumption that these selected bitrates would remain popular in the near future.

However, since cached video content would significantly change the throughput experienced by users, bitrate adaptation schemes could (based on recent statistics) attempt to request bitrates which exceed the expected network behavior. That is, as we improve the user's experience due to better cache management, the user's bitrate adaptation scheme may in fact attempt to surpass the available throughput, based on its conceived 'improvement' in network conditions. Thus, if caching decisions are

made without *inferring* future video requests, the cached bitrates would typically lag behind real-time changes. Therefore, we argue that a better video caching strategy must cater to incoming traffic by considering not only the (currently) most requested bitrates, but also *predicting* the bitrate adaptation to occur in the future.

This insight motivated the work presented in this chapter. Specifically, we devise the *PredictiveRipple* that attempts to predict future video traffic, to remedy the dependency between in-network caches and bitrate adaptation. PredictiveRipple is guided by the RippleCache principle, and optimizes the same cache reward indicator that is central to RippleClassic. However, PredictiveRipple presents significant improvements in contrast to previous schemes, specifically in the following aspects:

- PredictiveRipple is a cache partitioning scheme. Compared with our previous proposals in Chapter 4, where cache partitions are created via cache placement, PredictiveRipple determines the cache size for each bitrate, rather than deciding on which specific video segments to cache, and leaves content placement/replacement to an independent caching strategy. The decoupling of cache partitioning from content placement reduces the complexity of PredictiveRipple, and yields flexibility to handle video content with time-varying popularity.

- PredictiveRipple captures the dynamics of cache-partitioning via a Multi-Agent Reinforcement Learning (MARL) model, aiming to optimize the user's QoE. At its core, the MARL-based model guides each router to independently monitor the consumers' input, learning from its reaction to bitrate adaptation, and cooperating with other routers to discover the best (joint) cache partitioning actions.

- PredictiveRipple is an adaptive system, which evolves to improve cache partitioning under various video traffic patterns.

The results of our evaluation show that although the QoE improvements by RippleFinder are significant, PredictiveRipple optimizes the long-term QoE performance of caching system and resolves an overfitting issue which is rooted in popularity-based caching schemes.

The remainder of this chapter is organized as follows. We present the overview of PredictiveRipple caching system in Section 5.2, and elaborate on the formulation of its MARL framework in Section 5.3. Section 5.4 details the training process of MARL, elaborating on the heuristics employed to facilitate efficient cache partition exploration. Section 5.5 presents the experiment setup and performance evaluation. We summarize our findings in Section 5.6.

5.2 Cache Partitioning with PredictiveRipple

We design ***PredictiveRipple***, as an adaptive scheme to perform cache partitioning. PredictiveRipple is guided by the same RippleCache principle as in Chapter 4. Each router coordinates with other nodes - along forwarding paths - to adjust the cache capacity dedicated to each bitrate.

We model an ICN as a connected graph $G = (\mathbb{V}, \mathbb{E})$, where nodes in $\mathbb{V}$ are composed of video producers $\mathbb{P}$, edge routers $\mathbb{D}$ and intermediate routers. All users are served exclusively by edge routers. Every node v, where $v \in \mathbb{V}$, is equipped with content storage with capacity C_v which represents the capacity dedicated to adaptive video caching. In PredictiveRipple, we assume each router relays video requests following single-path forwarding, and applies the *best route* forwarding strategy.

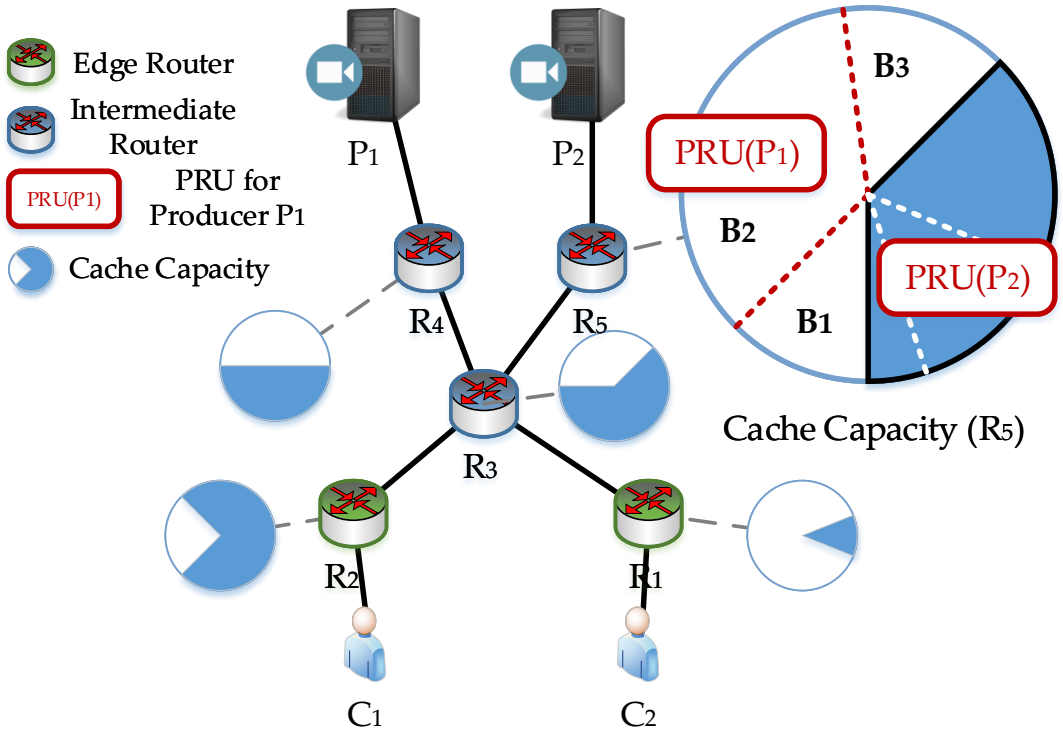


Figure 5.1: PredictiveRipple system architecture. The cache capacity of every ICN router is divided into two portions, where each portion serves the content from producer P_1 and P_2 respectively. Each cache portion is managed by a *PredictiveRipple Unit (PRU)* module, and is partitioned into three sub-partitions for content encoded with bitrate B_1, B_2 and B_3.

The available cache capacity on any router $v \in \mathbb{V}$ is further divided into portions to serve video content from different producers. We use $C_v(p)$ to denote cache capacity that is dedicated to caching video content from producer p, where $C_v = \sum_{p \in \mathbb{P}} C_v(p)$. As shown in Figure 5.1, multiple *PredictiveRipple Unit*s (PRUs) are installed on ICN routers. Each PRU is responsible for managing cache capacity $C_v(p)$ for a certain video producer p, and coordinating with other PRUs which serve the same producer, to adjust its cache partitioning. In Section 5.3, we elaborate on this PRU, which is trained by reinforcement learning to cater to future bitrate adaptation. Each PRU

determines the cache size for each bitrate, not which specific video segments should be cached. As catering to video content popularity is known to improve the performance of caching schemes [69], we implement and evaluate CE2 with LFU as a representative video placement/replacement scheme for PredictiveRipple.

5.3 Reinforcement Learning Framework

While PRU adjusts cache partitions based on bitrate adaptation prediction, understanding the interaction between cache placement and bitrate adaptation is a fundamental prerequisite to build PRU. Our first attempt involved modelling this interaction, but the challenge lies in the diversity of heuristics employed in adaptation algorithms [23, 58, 66]. These heuristics are designed for various scenarios, rendering it almost impossible to build a generic model which can represent the specifics of these algorithms. Thus, we set to devise a model-agnostic approach, and built the proposed PRU in PredictiveRipple using a Multi-Agent Reinforcement Learning (MARL) framework [55, 6]. As a result, PRU is not limited by any specific type of bitrate-adaptation, and can be applied to general network settings.

In the context of this work, PRU is intrinsically a learning agent in the MARL framework. Our proposed PredictiveRipple has multiple PRUs installed on ICN routers for different video producers. Those PRUs, which serve video content from the same producer, logically form a MARL instance. Thus, PredictiveRipple encompasses multiple MARL instances, where each instance configures the in-network caching for a particular producer. As the formulation of each MARL instance is intrinsically the same, in this section, we focus on explaining the MARL framework of a single producer.

MARL predicts future bitrate adaptation by referring to cache reward in the long term. The evaluation of this reward reflects diverse caching preferences from different video consumers. We apply the same reward function design as in Section 4.4.2, which quantifies the 'quality' of cached video content based on the RippleCache principle. Our cache reward design is inherently distributed, since reward is evaluated independently at each router, and granted per cache hit. Thus, each PRU implements the same reward function, measuring and recording the received reward in real-time. Each MARL instance evaluates its own performance by capturing the sum of rewards contributed by all corresponding PRUs. The evolution of MARL instances in the long term relies on PRUs for exchanging and maximizing their local cache rewards, which is elaborated on later in this section.

The rest of this section is organized as follows. First we elaborate on how each learning agent (PRU) is executed under the MARL framework. Second, we detail the problem formulation, defining the state and action space for PRUs. Third, we elaborate on the need for space aggregation as a practical implication, and finally highlight the distributed coordination among agents to reach the global optimum.

Overview

The timeline for a learning agent (PRU) is shown in Figure 5.2. Before each agent starts learning, an initialization period is triggered between $[0, t_1]$. During this period, a popularity-based caching scheme (such as CE2 with LFU) would run on each router. Video statistics are collected and video content is ranked by popularity. Each PRU allocates cache space for popular video content for as much as its cache capacity allows, which eventually forms the initial cache partitions for each bitrate.

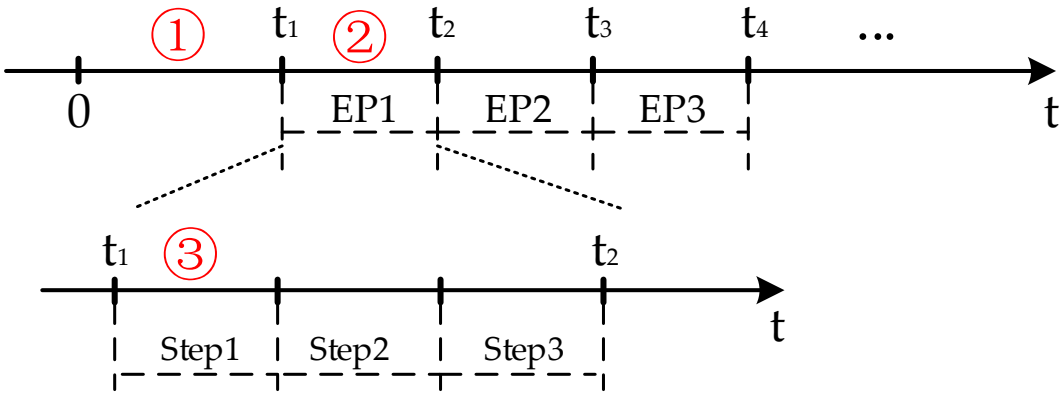


Figure 5.2: Timeline for a learning agent. ① Capture initial cache partition. ② A learning episode (EP1) between $[t_1, t_2]$. ③ A learning step where an action to adjust cache partitions occurs.

Each PRU then continues from t_1 and cycles through multiple *episode*s. For instance, the time interval between $(t_1, t_2]$ forms episode 1 (EP1), during which multiple actions are taken by this PRU. At the end of each episode, regardless of the current cache partitions, the initial partitioning (derived at time t_1) would be restored in order to prepare for the next training episode.

Each episode consists of several *steps*, and each step starts with an action that adjusts the size of cache partitions. The cached content in each partition is managed by a popularity-based caching scheme, where we adopt CE2 with LFU without loss of generality. Cache replacement can only occur between video content encoded with the same bitrate, in order to refresh the cached content while maintaining the size of partitions. During the time interval of a *step*, the cache reward is also computed by each agent to indicate the performance of recent action on cache partitions.

5.3.1 MARL Formulation

We formulate the behavior of agents in MARL as a Markov Decision Process (MDP), where both the *state* and *action* of an agent define its status .

We consider a MARL instance with m learning agents, indexed by $\{1, ..., m\}$, which are installed on different ICN routers. The local *state* of any agent i ($1 \leq i \leq m$) is represented by $s_i = \langle \Lambda_P^i, \Lambda_R^i \rangle$, which includes the status of cache partitions (Λ_P^i) and incoming requests (Λ_R^i) as an input to MARL instance. We include these features in an agent's state as they will eventually influence the cache reward that each agent can receive.

Given that there are B available bitrates for users to request, the status of cache partitions is defined by $\Lambda_P^i = \langle \theta_{b_1}^i, \theta_{b_2}^i, \ldots, \theta_{b_B}^i \rangle$, where $\theta_{b_j}^i$ ($1 \leq j \leq B$) is the (discretized) percentage of cache capacity allocated for encoded bitrate b_j. The incoming requests Λ_R^i are similarly represented by $\langle \upsilon_{b_1}^i, \upsilon_{b_2}^i, \ldots, \upsilon_{b_B}^i \rangle$, where $\upsilon_{b_j}^i$ denotes the percentage of requests for bitrate b_j received from video consumers. If an agent i is located on an edge router (the topological boundary of PredictiveRipple), $\upsilon_{b_j}^i$ can be derived directly from video statistics, which form the input to the MARL instance. Otherwise, $\upsilon_{b_j}^i$ is assigned value 0 to indicate that agent i is installed on an intermediate router.

Each agent in a certain *state* would select an action, which triggers a transition to adjust cache partitions. The local *action* of agent i is represented by

$a_i = \langle \tau^i_{b_1}, \tau^i_{b_2}, \ldots, \tau^i_{b_B} \rangle$. $\tau^i_{b_j}$ and is defined by

$$\tau^i_{b_j} = \begin{cases} +1, & \textit{increase size by } \zeta\textit{(\%)} \\ 0, & \textit{no action} \\ -1, & \textit{decrease size by } \zeta\textit{(\%)}, \end{cases} \tag{5.1}$$

where ζ represents the action granularity. For example, if $\zeta = 10(\%)$, $\tau^i_{b_j} = +1$ means increasing the cache size for bitrate b_j by 10% within the total cache capacity managed by this agent (PRU). The values of $\tau^i_{b_j}$ for any local action a_i must satisfy the following constraints:

$$\sum_{j=1}^{B} \tau^i_{b_j} = 0 \tag{5.2}$$

$$\sum_{j=1}^{B} |\tau^i_{b_j}| \leq 2, \tag{5.3}$$

where increasing and decreasing the cache capacity for bitrates must occur pairwise. For example, if $B = 4$, a legitimate action a_i can be $\langle +1, 0, -1, 0 \rangle$, which means increasing the cache size for b_1 and decreasing for b_3. Another example is $\langle 0, 0, 0, 0 \rangle$ which means the current cache partition remains unchanged.

In a MARL instance, all agents are adjusting cache partitions simultaneously and each one needs to be aware of both its local status and the status of other agents; to reach a global optimum. We thus define the agent-dependent state $\boldsymbol{s_i}$ and action $\boldsymbol{a_i}$, where we use a **bold** typeface to distinguish each from their local versions. Formally, $\boldsymbol{s_i} = \boldsymbol{s}_{\mathcal{H}(i)}$, where $\mathcal{H}(i)$ is the dependency set of agent i. $\mathcal{H}(i)$ contains agents, where their hosts (routers) relay video requests which impact the

cache reward received by agent i. Figure 5.3 shows an example of this dependency set $\mathcal{H}(i)$. $\mathcal{H}(i)$ includes all agents along the request forwarding paths, since cache hits on those nodes could change video throughput and influence the adaptation decision. Suppose $|\mathcal{H}(i)| = k$. $k \geq 1$ is guaranteed since each $\mathcal{H}(i)$ must include i itself. If agents $i_1, \ldots, i_k$ are contained in $\mathcal{H}(i)$, $\boldsymbol{s_i}$ is then represented by $\boldsymbol{s_i} = \langle s_{i_1}, s_{i_2}, \ldots, s_{i_k} \rangle$. The agent-dependent action, $\boldsymbol{a_i}$, is similarly defined, where $\boldsymbol{a_i} = \langle a_{i_1}, a_{i_2}, \ldots, a_{i_k} \rangle$, which contains local actions applied by agents in $\mathcal{H}(i)$. The size of $\mathcal{H}(i)$ depends on the size of routing topology and could be very large when MARL is applied in a complex ICN. We propose space aggregation later in Section 5.3.2 for practical considerations.

After agent i applies action a_i, the cache reward is recorded and then MARL instance transits to a different state. The local reward of agent i is denoted by $R_i(\boldsymbol{s_i}, \boldsymbol{a_i})$. We apply the same cache reward function ϕ as in Section 4.4.2. Suppose we denote Ψ_i as a set of video requests satisfied by caches that are managed by agent i. The local reward, $R_i(\boldsymbol{s_i}, \boldsymbol{a_i})$, is then defined by $R_i(\boldsymbol{s_i}, \boldsymbol{a_i}) = \sum_{\Psi_i} \phi(RB^i, b)$.

The objective of the MARL framework is to maximize the global cache reward; computer as the sum of local rewards from all agents. To accomplish this goal, we apply $\mathcal{Q}$-Learning [55] as the underlying technique to track the variations in cache reward through multiple transitions. $\mathcal{Q}$-Learning, where $\mathcal{Q}$ stands for *Quality*, is a model-free approach, and has been extensively studied in the literature. $\mathcal{Q}$ is a weighted sum of reward in an episode and is updated by each transition. The update rule of $\mathcal{Q}$ for each agent i, derived in [28], is shown as

$$\mathcal{Q}_i(\boldsymbol{s_i}, \boldsymbol{a_i}) = \mathcal{Q}_i(\boldsymbol{s_i}, \boldsymbol{a_i}) + \rho[R_i(\boldsymbol{s_i}, \boldsymbol{a_i}) + \gamma \mathcal{Q}_i(\boldsymbol{s_i'}, \boldsymbol{a_i'}^*) - \mathcal{Q}_i(\boldsymbol{s_i}, \boldsymbol{a_i})], \tag{5.4}$$

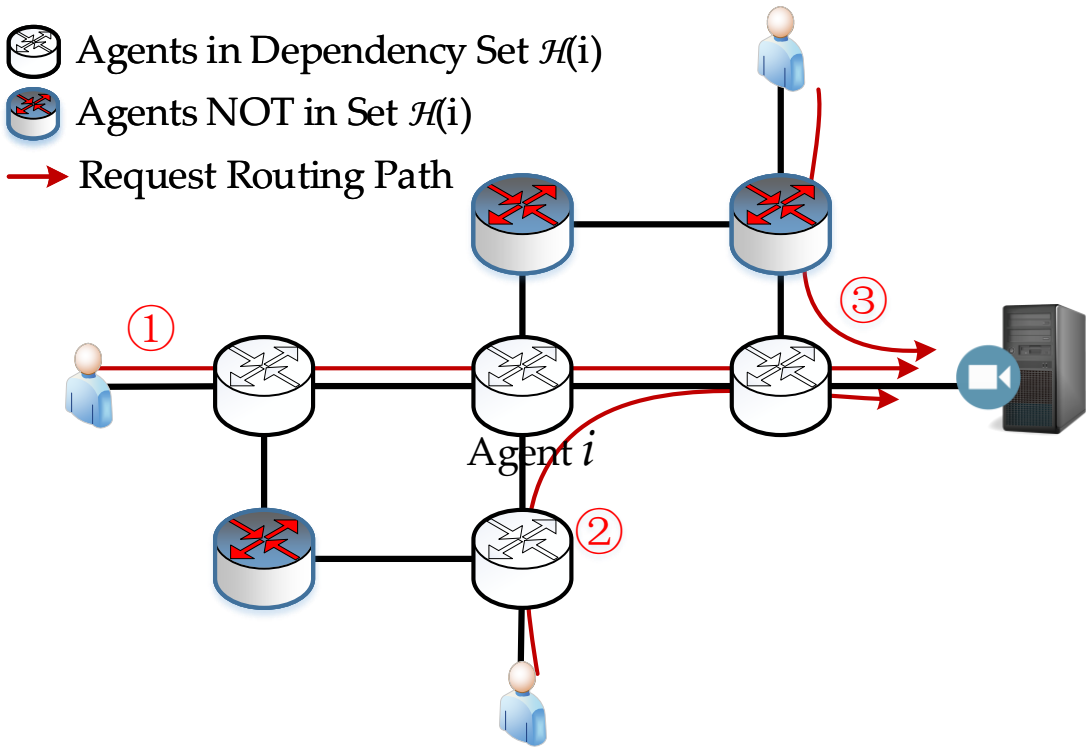


Figure 5.3: An example of a dependency set. Agent i is installed on a router where routing paths ①② go through. All agents that are installed on these two paths will be added into $\mathcal{H}(i)$. Other routers (agents), for example the edge node along path ③, are not affected.

where ρ denotes the learning rate and γ represents the discount factor, which are common notations in reinforcement learning. As presented in Equation (5.4), updating $\mathcal{Q}$ relies on the instant cache reward $R_i(\boldsymbol{s_i}, \boldsymbol{a_i})$ and the best possible future reward $\mathcal{Q}_i(\boldsymbol{s_i'}, \boldsymbol{a_i'}^*)$ on the next state $\boldsymbol{s_i'}$. The best action is derived by $\boldsymbol{a_i'}^* = \arg\max_{\boldsymbol{a'}} \sum_{j \in \mathcal{H}(i)} \mathcal{Q}_j(\boldsymbol{s_j'}, \boldsymbol{a_j'})$. Apparently, each agent i needs to reach a consensus with other agents in the dependency set $\mathcal{H}$ to achieve this $\boldsymbol{a_i'}^*$, and we elaborate in Section 5.3.3 a max-sum algorithm [50] to coordinate behaviors among agents.

5.3.2 Space Aggregation

The dependency set $\mathcal{H}(i)$ contains agents along routing paths which impact the local cache reward at agent i. The agent-dependent state $\boldsymbol{s_i}$ and action $\boldsymbol{a_i}$ are built based on $\mathcal{H}(i)$, and thus the dimension of $\boldsymbol{s_i}$ and $\boldsymbol{a_i}$ increases with the size of $\mathcal{H}$. In the worst case, when agent i runs on a router where most routing paths are merged (e.g., the last hop before reaching the video producer), $\mathcal{H}$ may possibly include all routers in the network which causes space explosion. In this section, we propose aggregation approaches on $\boldsymbol{s_i}$ and $\boldsymbol{a_i}$ respectively, based on our insights of cache partitioning problem. The objective of our approach is to reduce the dimension of $\boldsymbol{s_i}$ and $\boldsymbol{a_i}$ while maintaining the necessary features that reveal the essence of transitions in MARL framework. We show in this section that the dimension of aggregate state/action is bounded by the length of routing path.

State Space Aggregation

All agents - which serve the same video producer - have to coordinate with each other on cache partitioning. As every MARL instance serves one video producer, its routing topology intrinsically has a tree structure. Our proposed aggregation on state space is thus performed based on this tree topology. Specifically, as shown in Figure 5.4, agents in $\mathcal{H}(i)$ are grouped into three topology classes 'downstream', 'local' and 'upstream', based on their relative locations to agent i in the tree. Our state space aggregation then attempts to merge the states in 'downstream' class into 'local' class, while keeping the same states in 'upstream' class.

Formally, we denote the aggregated state as $\tilde{\boldsymbol{s}}_{\boldsymbol{i}}$. The dependency set $\mathcal{H}(i)$ can be represented as $\mathcal{H}(i) = \mathcal{H}_{up}(i) \cup \{i\} \cup \mathcal{H}_{down}(i)$, where $\mathcal{H}_{up}(i)$ denotes the agent

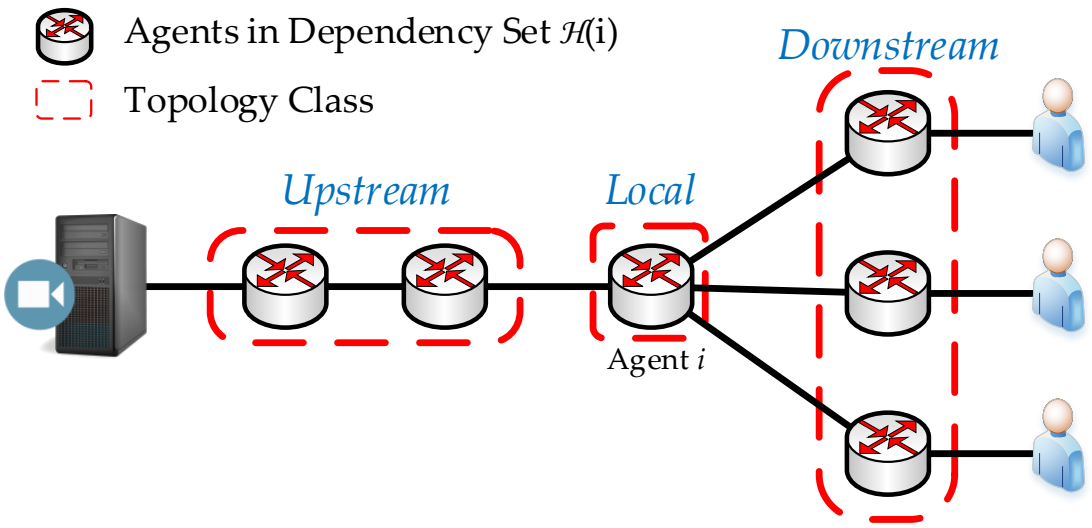


Figure 5.4: The dependency $\mathcal{H}(i)$ of agent i is divided into three topology classes.

set in the 'upstream' class and $\mathcal{H}_{down}(i)$ denotes the 'downstream' class. Suppose $|\mathcal{H}_{up}(i)| = k$ and agents $i_1, \ldots, i_k$ are contained in $\mathcal{H}_{up}(i)$. The aggregated state $\tilde{\boldsymbol{s}}_{\boldsymbol{i}}$ is constructed as $\tilde{\boldsymbol{s}}_{\boldsymbol{i}} = \langle s_{i_1}, s_{i_2}, \ldots, s_{i_k}, \tilde{s}_i \rangle$, where $\tilde{s}_i$ is the modified local state after the aggregation with downstream states. As a result, the dimension of $\tilde{\boldsymbol{s}}_{\boldsymbol{i}}$ equals to $|\mathcal{H}_{up}(i)| + 1$ and is bounded by the length of routing path. When the agent i is installed on an edge router, there are no other agents in its 'downstream' class, and thus state space aggregation is not performed ($\tilde{s}_i = s_i$).

Similarly, when agent i runs on an intermediate router, we should construct $\tilde{s}_i$, where the local state of agent i is represented by $s_i = \langle \Lambda_P^i, \Lambda_R^i \rangle$, which includes both cache partitioning status and video request pattern. Initially, Λ_R^i is assigned to 0 if the agent is not directly connected to video consumers. This is due to the fact that such agents are inside a PredictiveRipple and only agents on the boundary (i.e., installed at the edge router) can receive input to the system.

The state space aggregation process then executes in the following manner: any

agent i at an intermediate router would monitor the incoming request patterns independently, assigning a non-zero value to $\tilde{\Lambda}_R^i$, and discarding all states from other agents in its 'downstream' class. As a result, the modified local state $\tilde{s}_i$ is then defined as $\tilde{s}_i = \langle \Lambda_P^i, \tilde{\Lambda}_R^i \rangle$. Since cache partitions and consumers' input have a combined impact on video traffic, we argue that the incoming request pattern $\tilde{\Lambda}_R^i$ at an intermediate router is a good indicator of the aggregated cache status from downstream agents. Our state space aggregation then looks as though we merge all states in 'downstream' class into the 'local' class.

Action Space Aggregation

Action space aggregation is a challenging problem, since optimal global behavior of the MARL framework relies on the action coordination among all agents in $\mathcal{H}(i)$. As agents whose actions are aggregated would not be involved in the coordination, action space aggregation may cause an quasi-optimal result. The aggregated action $\tilde{\boldsymbol{a}}_{\boldsymbol{i}}$ is determined by evaluating the degree of dependency (relationship) in $\mathcal{H}$ and actions are aggregated from agents with minimal degrees. As a result, the aggregation would make the least impact on the MARL performance.

Designing an aggregation approach pivots on quantify the dependency relationship. Rogers et al. [50] used $\mathcal{Q}(\boldsymbol{s_i}, \boldsymbol{a_i})$ to calculate the dependency degree, and further reduced the complexity of deriving the approximate optimal action. As $\mathcal{Q}(\boldsymbol{s_i}, \boldsymbol{a_i})$ changes when a MARL instance is being trained, the dependency degree thus changes correspondingly, so is the representation of aggregated action $\tilde{\boldsymbol{a}}_{\boldsymbol{i}}$. More importantly, since $\mathcal{Q}(\boldsymbol{s_i}, \boldsymbol{a_i})$ has to be recorded before action space aggregation, $\mathcal{Q}(\boldsymbol{s_i}, \boldsymbol{a_i})$ is indexed by $\boldsymbol{a_i}$ such that a complete action space of agent i must be tracked. In fact,

although Rogers's approach is insightful, it forms a paradox: we need $\mathcal{Q}(\boldsymbol{s_i}, \boldsymbol{a_i})$ to reduce the action space, but to acquire $\mathcal{Q}(\boldsymbol{s_i}, \boldsymbol{a_i})$, we cannot reduce the action space. As brought forward by [50], we argue that to resolve such contradiction, the action space aggregation approach must have a static rule ('static' referring to the approach not being affected by the runtime status) to remove dependency relationship in $\mathcal{H}(i)$ during the learning procedure.

The aggregation approach is guided by our experiments on ideal cache partitioning. We note that the dependency relationship in $\mathcal{H}$ is symmetric, meaning if we find the agent $j \in \mathcal{H}(i)$, we can also find the agent $i \in \mathcal{H}(j)$. Our action aggregation approach then removes one dependency from this symmetric dependency pair which is less necessary when deriving a good cache partitioning. Recall from our previous description on RippleCache principle, an ideal cache partitioning would push low quality video content towards to the core network while safeguarding cache capacity for high quality content on the edge. Once dependencies with upstream agents are removed, cache router has no idea whether upstream caches have enough space to relocate low quality content. When this critical information is missing, safeguarding cache capacity for high quality content becomes useless since the users' basic needs on low quality content must be satisfied before bitrate adaptation can upgrade video quality. On the contrary, the dependencies from downstream agents are less necessary. Since whatever actions are taken by downstream agents, these actions would alter the incoming video request pattern ($\tilde{\Lambda}_R^i$) and local agent can still derive the best possible cache partitioning under current traffic. Therefore, we remove the original dependencies between agents in 'downstream' class and others in the rest of $\mathcal{H}$. Formally, The aggregated action $\tilde{\boldsymbol{a}}_{\boldsymbol{i}}$. is constructed as $\tilde{\boldsymbol{a}}_{\boldsymbol{i}} = \langle a_{i_1}, a_{i_2}, \ldots, a_{i_k}, a_i \rangle$, where

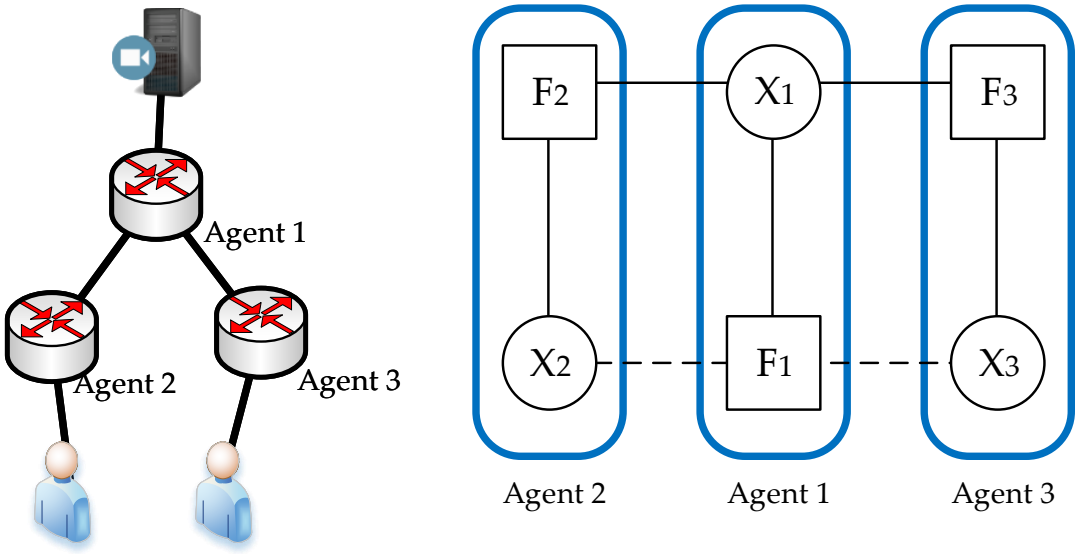


Figure 5.5: Since $\mathcal{H}(2) = \{1, 2\}$, the factor graph constructs a link between function node of agent 2 (F_2) and variable node of agent 1 (X_1), and another link between F_2 and X_2. The same rule applies to $\mathcal{H}(3)$. The action space aggregation trims $\mathcal{H}(1)$ by ignoring dependencies from 'downstream' agents, which are represented by dashed lines.

$i_k \in \mathcal{H}_{up}(i)$. The dimension of $\tilde{\boldsymbol{a}}_i$ again equals to $|\mathcal{H}_{up}(i)| + 1$ and is bounded by the length of the forwarding path.

5.3.3 Distributed Coordination

Both state and action space aggregation mandate that coordination shall be limited within the same pruned dependency set $\mathcal{G}$, where $\mathcal{G}(i) = \mathcal{H}_{up}(i) \cup i$. We adopt a distributed max-sum algorithm [50] to manage the coordination among agents in $\mathcal{G}$ and derive $\boldsymbol{a}_i^*$ as an estimate of optimal action at each transition.

The max-sum algorithm works based on a bipartite factor graph. In this graph, every agent is represented by a variable component and a function component. Each dependency relationship after state/action space aggregation creates a link between

a variable and a function component. Figure 5.5 shows an example of this factor graph. Max-sum algorithm then operates by specifying coordination messages that are exchanged between variable and function component (along solid lines as shown in Figure 5.5). The coordination messages are then denoted as

- **From variable i to function j:**

$$q_{i\to j}(a_i) = c_{ij} + \sum_{h\in\overline{\mathcal{G}}(i)\setminus j} r_{h\to i}(a_i) \tag{5.5}$$

  where c_{ij} is a normalization factor which prevents the messages from increasing endlessly. $\overline{\mathcal{G}}(i)$ is defined as the 'opposite' of $\mathcal{G}(i)$. That means if $j \in \mathcal{G}(i)$, we then have $i \in \overline{\mathcal{G}}(j)$.

- **From function j to variable i:**

$$r_{j\to i}(a_i) = \max_{\boldsymbol{a_j}\setminus a_i}[\mathcal{Q}_j(\boldsymbol{s_j}, \boldsymbol{a_j}) + \sum_{h\in\mathcal{G}(j)\setminus i} q_{h\to j}(a_h)] \tag{5.6}$$

As $q_{i\to j}$ depends on r from the *function component*, and $r_{j\to i}$ depends on q from the *value component*, message exchanges carried out by the max-sum algorithm between variable and function components are iterated, and terminated when both q and r converge. Each agent i can find its approximate optimal action a_i^* by locally calculating $a_i^* = \arg\max_{a_i} z_i(a_i)$, where $z_i(a_i) = \sum_{h\in\overline{\mathcal{G}}(i)} r_{h\to i}(a_i)$. We observe that the max-sum algorithm typically converges to good solutions. However, to safe guard PredictiveRipple, in case the max-sum algorithm does not converge, we establish a secondary measure to terminate the max-sum algorithm when the number of iterations reaches a pre-defined threshold.

5.4 MARL in Execution

We hereby elaborate on how agents in the MARL framework are trained over ICN to predict bitrate adaptation. The behavior of any agent in an episode is specified as shown in Algorithm 4.

Algorithm 4: MARL Training - in an episode

Input: Learning policy ψ; time interval between two transitions ν; Total number of transitions in an episode Ω.

1: Discover initial cache partitions $\boldsymbol{s_i}$
2: Current transition $m \leftarrow 0$
3: **repeat**
4: $\quad \boldsymbol{a_i} \leftarrow \emptyset$
5: $\quad$ **if** $\psi =$ 'Exploitation' **then**
6: $\quad\quad \boldsymbol{a_i} \leftarrow \boldsymbol{a_i'^*}$ at last transition $(m-1)$
7: $\quad$ **else if** $\psi =$ 'Exploration' **then**
8: $\quad\quad$ **for** $\forall i_k \in \mathcal{G}(i) \setminus \{i\}$ **do**
9: $\quad\quad\quad$ Receive local action a_{i_k} from agent i_k
10: $\quad\quad\quad$ Insert a_{i_k} into $\boldsymbol{a_i}$
11: $\quad\quad$ **end for**
12: $\quad\quad$ Choose local action a_i for agent i randomly
13: $\quad\quad$ Insert a_i into $\boldsymbol{a_i}$;
14: $\quad$ **end if**
15: $\quad$ Carry out action $\boldsymbol{a_i}$ to adjust cache partitions
16: $\quad$ Start measuring cache reward $R_i(\boldsymbol{s_i}, \boldsymbol{a_i})$ for current transition
17: $\quad$ **Wait** ν **do**
18: $\quad\quad$ Discover state $\boldsymbol{s_i'}$ in the next transition
19: $\quad\quad$ Derive approximate optimal action $\boldsymbol{a_i'^*}$ using Max-Sum coordination
20: $\quad\quad$ Update $Q_i(\boldsymbol{s_i}, \boldsymbol{a_i})$ using Equation (5.4)
$\quad\quad$ // Prepare for the next transition
21: $\quad\quad$ $m \leftarrow m + 1$
22: $\quad\quad$ $\boldsymbol{s_i} \leftarrow \boldsymbol{s_i'}$
23: $\quad$ **end Wait**
24: **until** $m \geq \Omega$

Each agent starts with an initialization on cache partitions (Lines 1-2) at the beginning of an episode, and then performs cache partition adjustments at multiple

transitions. Lines 4-15 detail the action selection procedure, where all agents face a problem of '*Exploitation*' vs. '*Exploration*' (Lines 5 and 7). This problem is essentially determining whether agents shall apply the known optimal action or explore the performance of a new cache partitioning. The choice ψ between these two options are made by the video producer and is broadcast inside the network to ensure each agent would apply the same strategy. Specifically, we apply ϵ-greedy [55], which generates a probability ϵ in the range of $[0, 1]$, to help this decision. When 'Exploitation' mode is selected (Line 6), each agent utilizes the known optimal action derived from the last transition; when 'Exploration' mode is selected (Lines 8-13), all agents then explore the impact of a random action.

Our preliminary studies show that although only a random action for 'Exploration' is required, agents rarely improve the cache reward in the actual experiments. We discover that constraints on action selection must be placed for efficient exploration. In practice, the video producer not only decides ψ but also chooses a pair of bitrates (b_x, b_y) for exploration. Thereafter, the explorations by all agents are limited within possible combinations of bitrates from this pair. For example, if we consider a total of 4 bitrates in the system and the bitrate pair is (b_1, b_4), actions by agents can only choose from $\langle +1, 0, 0, -1\rangle$, $\langle -1, 0, 0, +1\rangle$ and $\langle 0, 0, 0, 0\rangle$.

After a certain action is taken, agents wait for ν time to collect cache reward. Lines 18-20 completes a transition. Each agent detects the video request pattern based on the recent video statistics and transits from the original state $\boldsymbol{s_i}$ to the next $\boldsymbol{s_i'}$. Meanwhile, the approximate optimal action is derived by max-sum algorithm and $\mathcal{Q}$ value is updated to record the experience learned from the recent transition. As MARL cycles through multiple episodes, learned experience is accumulated until it

is trained to deal with common request patterns.

5.5 Performance Results and Insights

We evaluate PredictiveRipple performance via simulations on the NDN architecture to highlight the improvement made by online learning and prediction. Compared with the performance of RippleFinder presented in Section 4.6, PredictiveRipple driven by MARL framework achieves better or at least equal performance across multiple QoE metrics, which reinforces the need of predictive cache partitioning schemes for adaptive video streaming.

5.5.1 Simulation Setup and Parameters

We apply a similar simulation setup as Section 4.6 to examine the performance of PredictiveRipple, and the differences in our setup are described as follows.

PredictiveRipple is a predictive online cache partitioning scheme driven by MARL framework. Essentially, a MARL instance improves its performance by optimizing the $\mathcal{Q}$ value, as defined in Equation (5.4). A $\mathcal{Q}$ value is composed of cache reward in an entire episode across many transitions. As a result, the performance measurement on PredictiveRipple starts from the initial cache partitions until the end of an episode. The same methodology is applied to RippleFinder. This approach differs from what we previously used in Section 4.6, where RippleFinder iteratively updates its cache partitioning, hoping that it reaches an equilibrium. An equilibrium for RippleFinder means the recent cache partitioning would not alter the consumer-side bitrate adaptation decision. The results of RippleFinder in Section 4.6 are reported at an equilibrium

status. We argue that our current measurement is more appropriate to reveal the performance of a streaming service, because (1) the equilibrium status is not always easy to reach in real applications, and (2) the improvement of a predictive scheme can be highlighted when we focus on the system performance in a given period.

We evaluate our findings on a 16-node NDN network with a single video producer. The topology is randomly generated by BRITE [48]. We do not consider multiple producers in the experiments since the performance of PredictiveRipple depends on multiple identical MARL instances, while each MARL instance manages video content from the same producer. Thus, a network with a single producer is already representative to test the MARL design and reflect the overall PredictiveRipple performance. In addition, we present results for a smaller 16-node topology to reveal the dynamics of a MARL instance since with our state/action space aggregation, the difficulty of training a MARL is already bounded.

The parameters we apply to train and evaluate PredictiveRipple are listed in Table 5.1. We choose the granularity $\zeta = 5$ (%) since the MARL instance produces the best results among other trials in our preliminary experiments. The learning rate ρ, as appeared in Equation (5.4), is dynamically adjusted during the MARL training. Specifically, $\rho = \frac{\kappa}{\#\text{ of updates} + \kappa}$, where κ is a constant input. The learning rate depends on the number of times the same $(\boldsymbol{s_i}, \boldsymbol{a_i})$ pair in $\mathcal{Q}$ function that has been updated. As a result, ρ is close to 1 when there are few updates, which makes the value of $\mathcal{Q}$ change fast towards the recent cache reward; the learning rate drops to near 0 after many updates to help $\mathcal{Q}$ value converge and reach a consensus on the optimal action.

Table 5.1: Simulation parameters for PredictiveRipple evaluation

NDN	
Number of video files	200
Number of video segments per file	50
Number of NDN routers	16
Video segment playback time	4 sec
Number of video consumers	32
Encoded bitrates	{1, 2.5, 5, 8} mbps
Average time interval on video file requests	400 sec
Bandwidth	20 mbps
Skewness factor (α)	1.2
Content store size percentage (ω)	0.1
Cache reward parameter (η)	1
MARL	
Learning discount (γ)	0.99
Learning rate coefficient (κ)	20
State/Action granularity (ζ) (%)	5
Number of episodes for training	25000
Number of episodes for evaluation	100

5.5.2 The Impact of Online Prediction

We evaluate PredictiveRipple using the same performance metrics as defined in Section 4.6, which are *Bitrate Switch Count*, *Rebuffer Percentage* and *Average Video Quality*. Compared to the bar charts we used in Chapter 3 and 4, box plots were chosen here as they effectively filter abnormal data, to focus on the performance that appears most frequently.

Figure 5.6 shows that PredictiveRipple achieves the least bitrate oscillation matter regardless of our choice of the cache size or popularity skewness. Compared against CE2 with LFU, PredictiveRipple is more efficient in reducing the *Bitrate Switch Count* at a low cache size ω or low popularity skewness α setting, and the performance gap

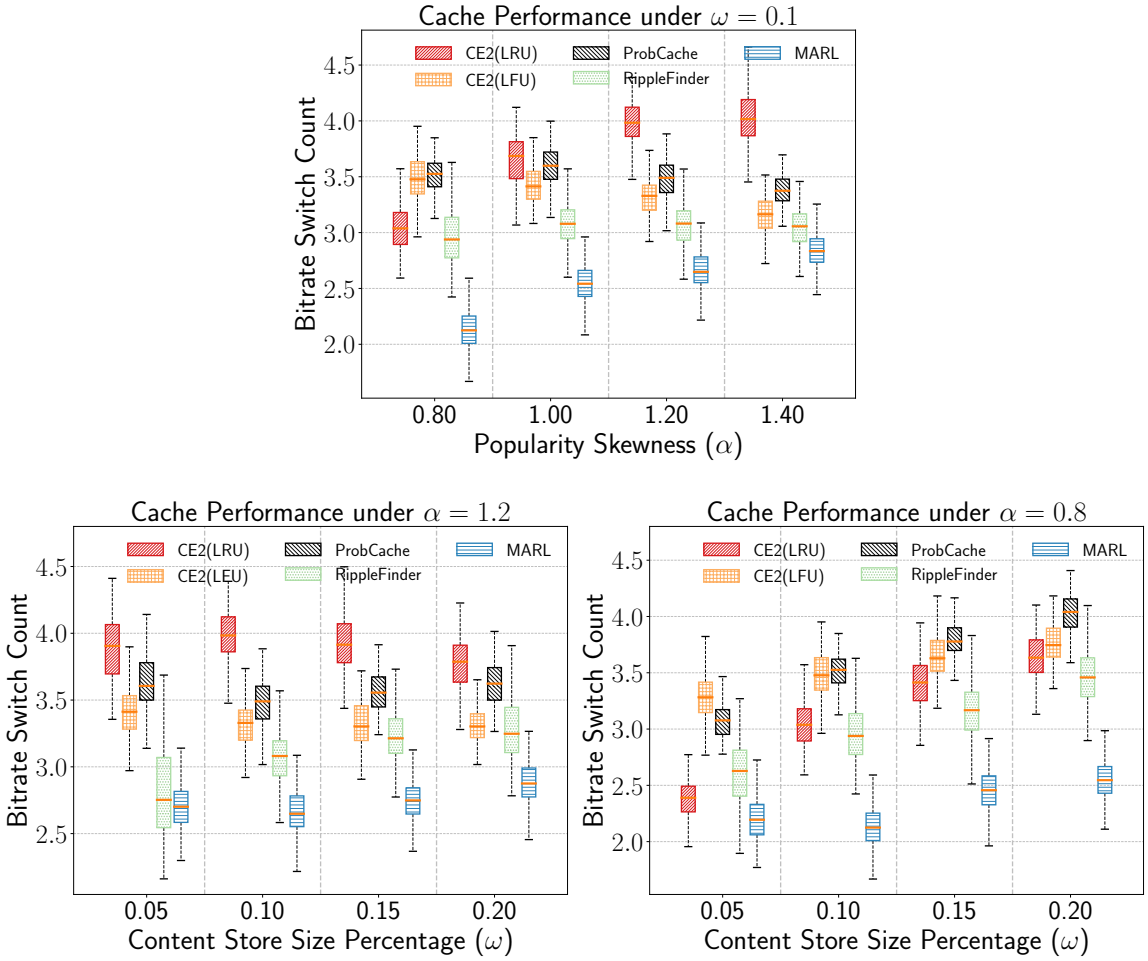


Figure 5.6: Bitrate Switch Count under MARL evaluation settings

between PredictiveRipple and CE2 with LFU shrinks as we increase cache size ω or popularity skewness α. This is because when there is sufficient cache storage or video requests are highly concentrated on a few options, popularity-based caching schemes already have enough resources to satisfy a large number of requests or capture the request pattern. Thus, *Bitrate Switch Count* as shown in Figure 5.6 is contributed mostly by bitrate switch up at a higher skewness α or a larger cache size ω, which matches the higher video quality as presented later in Figure 5.8. The impact of cache

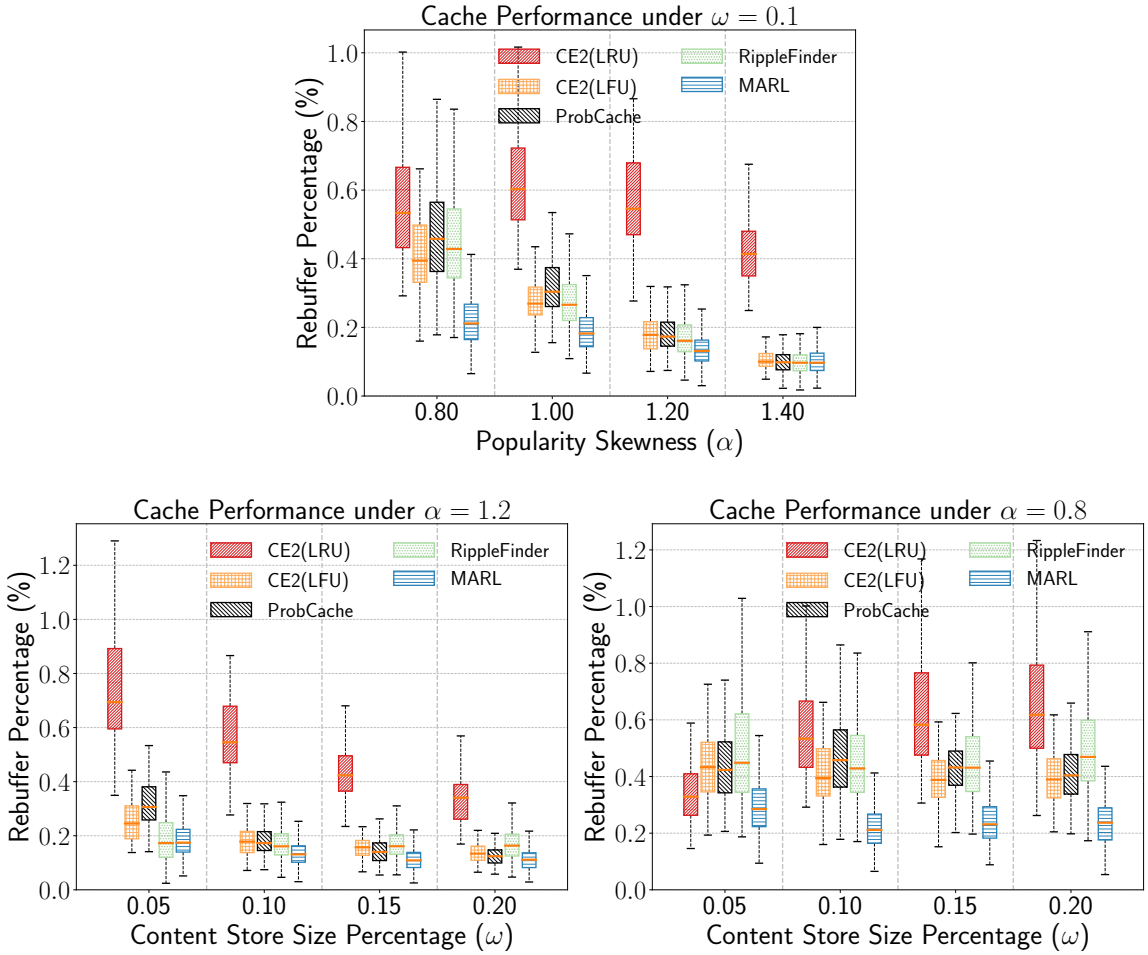


Figure 5.7: Rebuffer Percentage under MARL evaluation settings

partitioning is limited under this setting (since the popularity-based caching scheme already performs well). On the contrary, at a smaller skewness α or cache size ω, general caching schemes struggle with unstable video throughput and result in both high bitrate switch up and down. The effect of cache partitioning, as represented by RippleFinder and PredictiveRipple, is highlighted due to their capability of reducing the number of bitrate switch down.

Although both PredictiveRipple and RippleFinder are guided by RippleCache principle, PredictiveRipple outperforms RippleFinder in a given episode for its online

prediction capability. RippleFinder derives the cache partitioning based on only the current video traffic pattern. The lack of foreseeing the future bitrate adaptation would harm the performance of RippleFinder as there exists an 'overfitting' problem: RippleFinder may work perfectly under the current traffic but once bitrate adaptation triggers a different request pattern, the performance of RippleFinder may degrade significantly.

Figure 5.7 shows that PredictiveRipple causes the least playback freezing compared against other caching schemes. Special attention must be paid when the popularity skewness $\alpha = 0.8$ (bottom right figure), where the rebuffer percentage of RippleFinder increases to the same (or even higher) level as CE2 with LFU but PredictiveRipple is still capable of keeping a low level of playback freezing. The reason is similar to our previous analysis for *Bitrate Switch Count*; the overfitting of RippleFinder worsens the performance, especially in a heavy load system caused by lower popularity skewness α. In addition, PredictiveRipple only controls the size of each cache partition and allows LFU to refresh the cached content in real time, while RippleFinder is essentially a cache placement scheme and must be refreshed manually. Such difference makes PredictiveRipple more suitable in a highly dynamic environment.

Figure 5.8 shows CE2 with LFU delivers the highest video quality to consumers across all scenarios, which is consistent with our previous observations in Section 4.6. Our proposed PredictiveRipple achieves a similar performance as CE2 with LFU across different cache sizes (ω) at a higher popularity skewness $\alpha = 1.2$. However, at $\alpha = 0.8$, PredictiveRipple performs worse than RippleFinder and other popularity-based caching schemes. This performance gap increase at a larger cache size (as

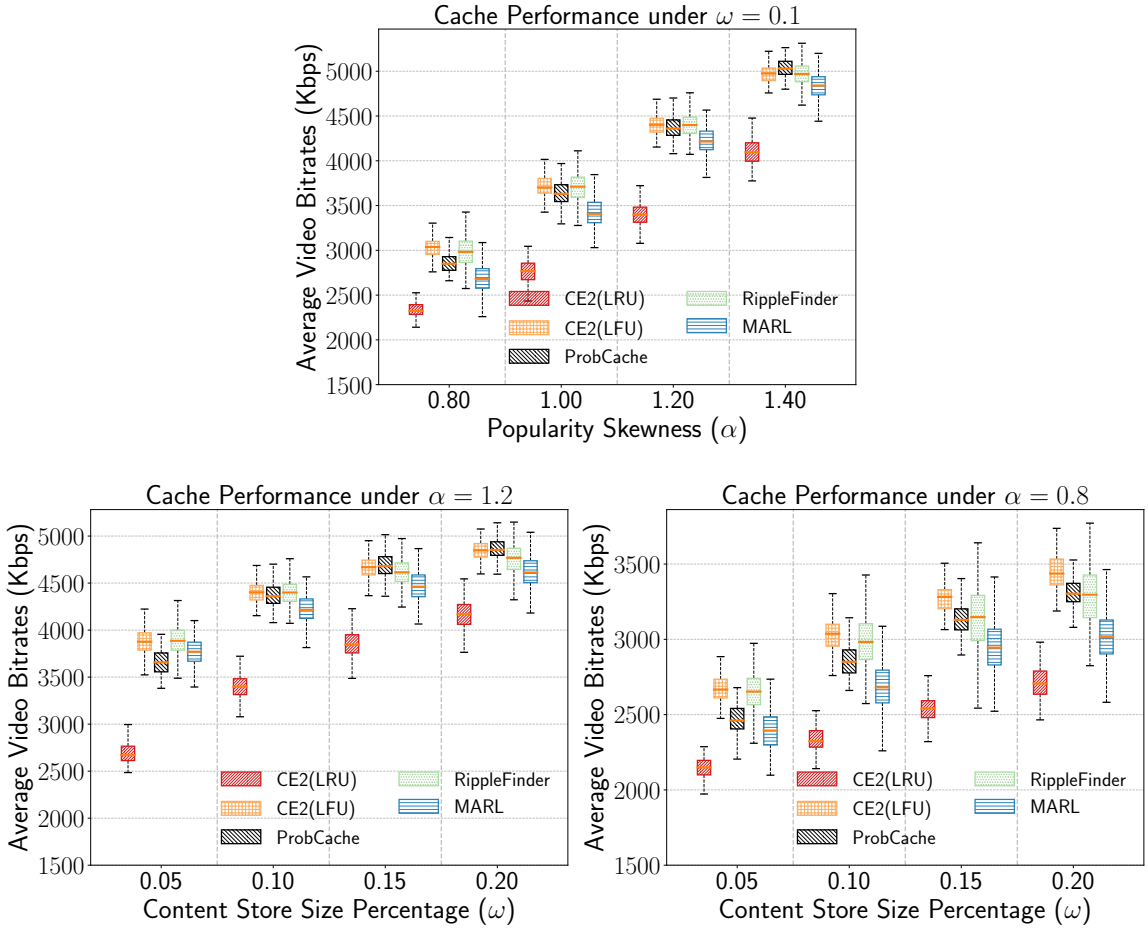


Figure 5.8: Average Video Bitrate under MARL evaluation settings

illustrated in the bottom right figure). Combined with the results for *Bitrate Switch Count*, we argue that a relatively worse video quality does not indicate the overall performance of PredictiveRipple. A caching scheme that leads to frequent bitrate oscillation would always deliver a higher video quality on average, since the adaptation control attempts to increase the requested bitrate without considering whether such video quality can be sustained or not. Our proposed PredictiveRipple makes the necessary compromise on the delivered video quality to achieve a balance between multiple performance metrics and finally ensures an overall improvement to QoE.

5.6 Summary

In this chapter, we have shown that the dependency relationship between in-network caches and bitrate adaptation may harm the performance of caching schemes that rely on recent request statistics to make caching decisions. In remedy, bitrate adaptation *prediction* can resolve such dependency by considering not only what the current most requested bitrates are, but also predict popular choices in the future. As our existing designs of cache partitioning schemes were shown to improve overall QoE, we proposed the novel caching system *PredictiveRipple*, building upon this cache partitioning concept with prediction capability; catering to dynamic video content in the long term.

PredictiveRipple manages cache partitions via a MARL framework. In the MARL, we utilized a $\mathcal{Q}$-learning approach to perform online prediction. $\mathcal{Q}$-learning optimizes the impact of a cache partition adjustment by evaluating both immediate and future cache reward. Our experiments on PredictiveRipple against the representative caching partitioning approach RippleFinder reinforces the need for adaptation prediction. We explained how the performance of popularity-based caching is hindered by an overfitting challenge, rooted in its inherent design where caching decisions depend on instantaneous video statistics. Although popularity-based caching schemes, like RippleFinder, may efficiently cache currently popular video content, future bitrate adaptation degrades their performance. PredictiveRipple achieves significant improvement over classic cache partitioning schemes by further reducing bitrate oscillation and re-buffering levels. Moreover, it is only when cache resources are constrained or popularity distribution is less skewed, that PredictiveRipple makes a necessary compromise on video quality to ensure smooth video playback.

Chapter 6

Conclusion and Future Directions

6.1 Summary

In this thesis, we demonstrated the QoE performance when cache hierarchies in ICN can capture, understand and react to bitrate adaptation. This was motivated by the fact that video streaming is dominating the current Internet, and the lack of prior research on caching schemes that are optimized for adaptive streaming applications. We proposed solutions which emphasized various QoE metrics and achieved comprehensive improvements to QoE performance.

Chapter 1 gave an overview of the research problem and our contributions in each chapter. Chapter 2 detailed the background and related work, where we explained the concepts of bitrate adaptation in DASH and related research which attempted to improve QoE from the consumer side. We emphasized and compared two different caching paradigms for adaptive streaming by conducting preliminary experiments. Our results showed that a representative bitrate-aware caching approach can meet or even exceed the ideal video transcoding at the edge (where we assume zero transcoding delay), which reinforced the advantage of utilizing cache hierarchies to handle

video requests for variable bitrates. While our intuition was to pursue the latter, we were further reassured via experimental studies that transcoding would not outperform ubiquitous network-wide caching. Our experiments are further detailed in Appendix A.

Chapter 3 presented our designs to improve video quality, regarded as an important metric that affects consumers' satisfaction score. As video throughput is used as the primary indicator that guides the requested video quality selection, we proposed caching schemes that reduced the video access delay under varying encoded bitrates which eventually led to a higher video quality. The first part of this chapter presented our adaptive video traffic modelling, which later facilitated the queueing delay analysis and video throughput derivation. A benchmark caching scheme DaCPlace then utilized the delay estimates to decide the cache placement when both on-path and off-path cache resources were jointly optimized. The second part of this chapter aimed to reduce the complexity in DaCPlace and presented StreamCache as a greedy heuristic to handle adaptive streaming.

In Chapter 4, we shifted our focus to bitrate oscillation, as another critical QoE metric. In order to reduce the impact of intermittent cache hits and misses which ultimately caused bitrate oscillation, a novel notion of cache partitioning was introduced. We first displayed an ideal cache partitioning, as summarized by a RippleCache principle. Two separate designs RippleClassic and RippleFinder were developed to cater to RippleCache ideal. The performance gains were reinforced by evaluations against state-of-the-art baseline approaches, using standard measures of QoE as defined by the DASH Industry Forum. Measurements showed that RippleClassic and RippleFinder delivered content that suffered less oscillation and rebuffering, as well as the highest

levels of video quality, indicating overall improvements to QoE.

Chapter 5 extended the foundation laid in Chapter 4 to predictive cache partitioning. We proposed a MARL framework, driven by $\mathcal{Q}$-learning to estimate the cache performance in the long term. MARL was presented as a distributed framework, where cache routers work cooperatively to adjust cache partitions based on both current and future popular bitrates. Experiment results indicated the overfitting problem, rooted in popularity-based caching schemes, was resolved by bitrate adaptation prediction, and the proposed predictive caching framework further enhanced QoE.

6.2 Limitations

Our priority in system modelling is to mimic realistic adaptive streaming scenarios. Although several efforts have been made, including implementing an existing bitrate adaptation control algorithm and adopting recommended encoding bitrates from YouTube, the proposed schemes are evaluated under a simulation environment which may not be able to replicate the exact same settings from the real world. Recently, an NDN testbed has been released [43], which gives ICN researchers an opportunity to examine their ideas and proposals over a real network. However, for adaptive streaming studies, a large number of real consumers must also be involved in the experiments to provide feedback online under different caching decisions. Existing video trace data over the current Internet is not useful as it cannot reflect the caching status, as a key metric in my work. Video traces particularly under ICN must be generated in order to produce hard evidence on the performance of the proposed solutions.

6.3 Future Directions

This thesis has presented a novel direction in bitrate adaptation-aware caching over ICN. Our proposals achieve comprehensive improvements on various consumer-side QoE metrics. However, there are still open challenges and potential implementation opportunities that can be explored in order to fully leverage the ICN caching capability. Some of these avenues are listed as follows.

6.3.1 Bitrate Adaptation Prediction with Deep Learning

Recent advancements in Deep Learning enable future work on training MARL frameworks more efficiently. This could be accomplished by applying deep neural networks to estimate unexplored $\mathcal{Q}$ value before the optimal action derivation. In order to accelerate the training procedure, existing Deep Learning platforms, such as Tensorflow and Keras, shall be incorporated in the experiments. Our current experiment setup requires MARL to interact with a network simulator for real-time measurement on consumers input. As a result, the performance of this network simulator would highly influence the training speed of the entire system. Our choice of ndnSIM [2], in spite of the best simulator to mimic the NDN architecture, has limited support of multi-threading, which would be the bottleneck of experiments. Future work must develop an experimental system that matches the speed of Deep Learning and network simulation. We also want to highlight this issue prevails in all practical systems which depend on the real-time input for further processing by Deep Learning.

6.3.2 Cache-Friendly Bitrate Adaptation

The currently adopted DASH protocol cannot perform as desired in a network with cache hierarchies, because bitrate adaptation is unable to differentiate whether video throughput variations are caused by network jiter or intermittent cache hits and misses. Thus, it is important for bitrate adaptation to identify the source of variations. In remedy, it would be fruitful to investigate cache hit locations for video requests, and explore schemes that could capitalize on such information. As a result, consumers can evaluate the impact of cache hits and then determine if a given bandwidth fluctuation is more likely caused by in-network caches or not. Thereafter, bitrate adaptation schemes can be developed to handle different sources of bandwidth fluctuation. We envision this type of adaptation control as *cache-friendly* rate adaptation, as it would aim to enhance its performance by coordinating with caching schemes in cache hierarchies.

Bibliography

[1] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman. A Survey of Information-Centric Networking. *IEEE Communications Magazine*, 50(7):26–36, 2012.

[2] A. Alexander, I. Moiseenko, and L. Zhang. ndnSIM: NDN simulator for NS-3. *Technical Report NDN-0005*, 2012.

[3] A. Araldo, F. Martignon, and D. Rossi. Representation Selection Problem: Optimizing Video Delivery Through Caching. In *IFIP Networking Conference*, pages 323–331. IEEE, 2016.

[4] D. P. Bertsekas, R. G. Gallager, and P. Humblet. *Data Networks*, volume 2. Prentice-Hall Englewood Cliffs, NJ, 1987.

[5] L. Breslau, P. Cao, L. Fan, G. Phillips, S. Shenker, et al. Web Caching and Zipf-like Distributions: Evidence and Implications. In *IEEE INFOCOM*, pages 126–134. IEEE, 1999.

[6] L. Bu, R. Babu, B. De Schutter, et al. A Comprehensive Survey of Multiagent Reinforcement Learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38(2):156–172, 2008.

[7] G. Carofiglio, M. Gallo, L. Muscariello, and D. Perino. Modeling Data Transfer in Content-Centric Networking. In *IEEE International Teletraffic Congress (ITC)*, pages 111–118. IEEE, 2011.

[8] W. K. Chai, D. He, I. Psaras, and G. Pavlou. Cache "Less for More" in Information-centric Networks (Extended Version). *Computer Communications*, 36(7):758–770, 2013.

[9] K. Cho, M. Lee, K. Park, T. T. Kwon, Y. Choi, and S. Pack. Wave: Popularity-Based and Collaborative In-Network Caching for Content-Oriented Networks. In *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 316–321. IEEE, 2012.

[10] Cisco. Cisco Visual Networking Index: Forecast and Methodology, 2016–2021, 2017.

[11] A. Dabirmoghaddam, M. M. Barijough, and J. Garcia-Luna-Aceves. Understanding Optimal Caching and Opportunistic Caching at the Edge of Information-Centric Networks. In *ACM conference on Information-Centric Networking*, pages 47–56. ACM, 2014.

[12] C. Dannewitz, D. Kutscher, B. Ohlman, S. Farrell, B. Ahlgren, and H. Karl. Network of Information (NetInf) - An Information-Centric Networking Architecture. *Computer Communications*, 36(7):721–735, 2013.

[13] L. De Cicco, V. Caldaralo, V. Palmisano, and S. Mascolo. Elastic: a Client-side Controller for Dynamic Adaptive Streaming over HTTP (DASH). In *IEEE International Packet Video Workshop (PV)*. IEEE, 2013.

[14] Z. Duanmu, A. Rehman, and Z. Wang. A Quality-of-Experience Database for Adaptive Video Streaming. *IEEE Transactions on Broadcasting*, 64(2):474–487, 2018.

[15] S. Eum, K. Nakauchi, Y. Shoji, N. Nishinaga, and M. Murata. CATT: Cache Aware Target Identification for ICN. *IEEE Communications Magazine*, 50(12):60–67, 2012.

[16] S. K. Fayazbakhsh, Y. Lin, A. Tootoonchian, A. Ghodsi, T. Koponen, B. Maggs, K. Ng, V. Sekar, and S. Shenker. Less Pain, Most of the Gain: Incrementally Deployable ICN. *ACM SIGCOMM Computer Communication Review*, 43(4):147–158, 2013.

[17] M. Fiedler, T. Hossfeld, and P. Tran-Gia. A Generic Quantitative Relationship Between Quality of Experience and Quality of Service. *IEEE Network*, 24(2):36–41, 2010.

[18] T. Flach, P. Papageorge, A. Terzis, L. Pedrosa, Y. Cheng, T. Karim, E. Katz-Bassett, and R. Govindan. An Internet-Wide Analysis of Traffic Policing. In *ACM SIGCOMM Conference*, pages 468–482. ACM, 2016.

[19] N. Fotiou, P. Nikander, D. Trossen, and G. C. Polyzos. Developing Information Networking Further: from PSIRP to PURSUIT. In *International Conference on Broadband Communications, Networks and Systems*. Springer, 2010.

[20] R. Grandl, K. Su, and C. Westphal. On the Interaction of Adaptive Video Streaming with Content-Centric Networking. In *IEEE Packet Video Workshop (PV)*. IEEE, 2013.

[21] I. Griva, S. G. Nash, and A. Sofer. *Linear and Nonlinear Optimization*, volume 108. Siam, 2009.

[22] Gurobi. Gurobi Optimizer Reference Manual. http://www.gurobi.com/documentation/. Accessed on 2019-03-18.

[23] T.-Y. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson. A Buffer-based Approach to Rate Adaptation: Evidence From a Large Video Streaming Service. *ACM SIGCOMM Computer Communication Review*, 44(4):187–198, 2015.

[24] A. Ioannou and S. Weber. A Survey of Caching Policies and Forwarding Mechanisms in Information-Centric Networking. *IEEE Communications Surveys & Tutorials*, 18(4):2847–2886, 2016.

[25] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard. Networking Named Content. In *International Conference on Emerging Networking Experiments and Technologies*. ACM, 2009.

[26] J. Jiang, V. Sekar, and H. Zhang. Improving Fairness, Efficiency, and Stability in HTTP-Based Adaptive Video Streaming with FESTIVE. In *ACM International Conference on Emerging Networking Experiments and Technologies (CoNEXT)*, pages 97–108. ACM, 2012.

[27] Y. Jin, Y. Wen, and C. Westphal. Optimal Transcoding and Caching for Adaptive Streaming in Media Cloud: an Analytical Approach. *IEEE Transactions on Circuits and Systems for Video Technology*, 25(12):1914–1925, 2015.

[28] J. R. Kok and N. Vlassis. Collaborative Multiagent Reinforcement Learning by Payoff Propagation. *Journal of Machine Learning Research*, 7(Sep):1789–1828, 2006.

[29] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica. A Data-oriented (and Beyond) Network Architecture. *ACM SIGCOMM Computer Communication Review*, 37(4):181–192, 2007.

[30] C. Kreuzberger, B. Rainer, and H. Hellwagner. Modelling the Impact of Caching and Popularity on Concurrent Adaptive Multimedia Streams in Information-Centric Networks. In *IEEE International Conference on Multimedia & Expo Workshops (ICMEW)*. IEEE, 2015.

[31] J. Kua, G. Armitage, and P. Branch. A Survey of Rate Adaptation Techniques for Dynamic Adaptive Streaming over HTTP. *IEEE Communications Surveys & Tutorials*, 19(3):1842–1866, 2017.

[32] N. Laoutaris, H. Che, and I. Stavrakakis. The LCD Interconnection of LRU Caches and Its Analysis. *Performance Evaluation*, 63(7):609–634, 2006.

[33] S. Lederer, C. Mueller, C. Timmerer, and H. Hellwagner. Adaptive Multimedia Streaming in Information-Centric Networks. *IEEE Network*, 28(6):91–96, 2014.

[34] S. Lederer, C. Müller, and C. Timmerer. Dynamic Adaptive Streaming over HTTP Dataset. In *ACM Multimedia Systems Conference*, pages 89–94. ACM, 2012.

[35] D. Lee, C. Dovrolis, and A. Begen. Caching in HTTP Adaptive Streaming: friend or Foe? In *Network and Operating System Support on Digital Audio and Video Workshop (NOSSDAV)*, pages 31–36. ACM, 2014.

[36] J. Lee, K. Lim, and C. Yoo. Cache Replacement Strategies for Scalable Video Streaming in CCN. In *IEEE Asia-Pacific Conference on Communications (APCC)*, pages 184–189. IEEE, 2013.

[37] W. Li. Popularity-driven Caching Strategy for Dynamic Adaptive Streaming over Information-Centric Networks. Master's thesis, Queen's University, Kingston, 2015. Available Online: http://hdl.handle.net/1974/13414.

[38] W. Li, M. Fayed, S. M. Oteafy, and H. S. Hassanein. A Cache-Level Quality of Experience Metric to Characterize ICNs for Adaptive Streaming. *IEEE Communications Letters*, 23(2):262–265, 2019.

[39] W. Li, S. Oteafy, M. Fayed, and H. S. Hassanein. Bitrate Adaptation-Aware Cache Partitioning for Video Streaming over Information-Centric Networks. In *IEEE Conference on Local Computer Networks (LCN)*. IEEE, 2018.

[40] W. Li, S. M. Oteafy, and H. S. Hassanein. On the Performance of Adaptive Video Caching over Information-Centric Networks. In *IEEE International Conference on Communications (ICC)*. IEEE, 2017.

[41] Z. Li and G. Simon. Time-Shifted TV in Content Centric Networks: the Case for Cooperative In-Network Caching. In *IEEE International conference on communications (ICC)*. IEEE, 2011.

[42] Z. Li, X. Zhu, J. Gahm, R. Pan, H. Hu, A. C. Begen, and D. Oran. Probe and Adapt: Rate Adaptation for HTTP Video Streaming at Scale. *IEEE Journal on Selected Areas in Communications*, 32(4):719–733, 2014.

[43] H. Lim, A. Ni, D. Kim, Y.-B. Ko, S. Shannigrahi, and C. Papadopoulos. NDN Construction for Big Science: Lessons Learned from Establishing a Testbed. *IEEE Network*, 32(6):124–136, 2018.

[44] Y. Liu, S. Dey, F. Ulupinar, M. Luby, and Y. Mao. Deriving and Validating User Experience Model for DASH Video Streaming. *IEEE Transactions on Broadcasting*, 61(4):651–665, 2015.

[45] Y. Liu, J. Geurts, J.-C. Point, S. Lederer, B. Rainer, C. Muller, C. Timmerer, and H. Hellwagner. Dynamic Adaptive Streaming over CCN: a Caching and Overhead Analysis. In *IEEE International Conference on Communications (ICC)*, pages 3629–3633. IEEE, 2013.

[46] D. M. Lucantoni. New Results on the Single Server Queue with a Batch Markovian Arrival Process. *Communications in Statistics. Stochastic Models*, 7(1):1–46, 1991.

[47] A. Mansy, M. Fayed, and M. Ammar. Network-Layer Fairness for Adaptive Video Streams. In *IFIP Networking Conference*. IEEE, 2015.

[48] A. Medina, A. Lakhina, I. Matta, and J. Byers. BRITE: an Approach to Universal Topology Generation. In *International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pages 346–353. IEEE, 2001.

[49] I. Psaras, W. K. Chai, and G. Pavlou. Probabilistic In-Network Caching for Information-Centric Networks. In *ACM ICN Workshop on Information-Centric Networking*, pages 55–60. ACM, 2012.

[50] A. Rogers, A. Farinelli, R. Stranders, and N. R. Jennings. Bounded Approximate Decentralised Coordination via the Max-Sum Algorithm. *Artificial Intelligence*, 175(2):730–759, 2011.

[51] G. Rossini and D. Rossi. Evaluating CCN Multi-Path Interest Forwarding Strategies. *Computer Communications*, 36(7):771–778, 2013.

[52] A. Sackl, P. Casas, R. Schatz, L. Janowski, and R. Irmer. Quantifying the Impact of Network Bandwidth Fluctuations and Outages on Web QoE. In *IEEE International Workshop on Quality of Multimedia Experience (QoMEX)*. IEEE, 2015.

[53] L. Saino, I. Psaras, and G. Pavlou. Hash-Routing Schemes for Information Centric Networking. In *ACM SIGCOMM workshop on Information-Centric networking*, pages 27–32. ACM, 2013.

[54] M. Seufert, S. Egger, M. Slanina, T. Zinner, T. Hoßfeld, and P. Tran-Gia. A Survey on Quality of Experience of HTTP Adaptive Streaming. *IEEE Communications Surveys & Tutorials*, 17(1):469–492, 2015.

[55] R. S. Sutton, A. G. Barto, et al. *Introduction to Reinforcement Learning*, volume 135. MIT press Cambridge, 1998.

[56] MPEG. DASH Industry Forum. DASH-IF Position Paper: Proposed QoE Media Metrics Standardization for Segmented Media Playback. (Date last accessed 25-October-2018).

[57] MPEG. DASH. http://dashif.org/mpeg-dash.

[58] G. Tian and Y. Liu. Towards Agile and Smooth Video Adaptation in Dynamic HTTP Streaming. In *ACM International Conference on Emerging Networking Experiments and Technologies (CoNEXT)*, pages 109–120. ACM, 2012.

[59] A. V. Ventrella, G. Piro, and L. A. Grieco. Publish-Subscribe in Mobile Information Centric Networks: Modeling and Performance Evaluation. *Computer Networks*, 127:317–339, 2017.

[60] J. M. Wang, J. Zhang, and B. Bensaou. Intra-AS Cooperative Caching for Content-Centric Networks. In *ACM SIGCOMM Workshop on Information-Centric Networking*, pages 61–66. ACM, 2013.

[61] L. Wang, A. Hoque, C. Yi, A. Alyyan, and B. Zhang. OSPFN: An OSPF Based Routing Protocol for Named Data Networking. *Technical Report NDN Technical Report NDN-2012-13*, 2012.

[62] Y. Wang, Z. Li, G. Tyson, S. Uhlig, and G. Xie. Design and Evaluation of the Optimal Cache Allocation for Content-Centric Networking. *IEEE Transactions on Computers*, 65(1):95–107, 2016.

[63] Y. Wang, M. Xu, and Z. Feng. Hop-based Probabilistic Caching for Information-Centric Networks. In *IEEE Global Communications Conference (GLOBECOM)*, pages 2102–2107. IEEE, 2013.

[64] C. Westphal, S. Lederer, D. Posch, C. Timmerer, et al. Adaptive Video Streaming over Information-Centric Networking (ICN), 2016. [Online]. Available: http://www.rfc-editor.org/rfc/rfc7933.txt.

[65] G. Xylomenos, C. N. Ververidis, V. Siris, N. Fotiou, C. Tsilopoulos, X. Vasilakos, K. V. Katsaros, G. C. Polyzos, et al. A Survey of Information-Centric Networking Research. *IEEE Communications Surveys & Tutorials*, 16(2):1024–1049, 2014.

[66] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli. A Control-Theoretic Approach for Dynamic Adaptive Video Streaming over HTTP. *ACM SIGCOMM Computer Communication Review*, 45(4):325–338, 2015.

[67] YouTube Help. https://support.google.com/youtube/answer/1722171?hl=en. [Online; accessed 20-September-2018].

[68] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, P. Crowley, C. Papadopoulos, L. Wang, B. Zhang, et al. Named Data Networking. *ACM SIGCOMM Computer Communication Review*, 44(3):66–73, 2014.

[69] M. Zhang, H. Luo, and H. Zhang. A Survey of Caching Mechanisms in Information-Centric Networking. *IEEE Communications Surveys & Tutorials*, 17(3):1473–1499, 2015.

[70] H. Zhao, Q. Zheng, W. Zhang, B. Du, and H. Li. A Segment-Based Storage and Transcoding Trade-off Strategy for Multi-version VoD Systems in the Cloud. *IEEE Transactions on Multimedia*, 19(1):149–159, 2017.

Appendix A

Performance Comparison of Transcoding and Bitrate-Aware Caching

We compare the performance of caching under the ubiquitous and edge caching paradigms, via simulation. One of the important observations in our simulations is that conventional cache metrics, such as cache hit ratio, are not ideal for measuring streaming performance. Our evaluations are conducted on the NDN [68] architecture, as a favored representative that implements ICN primitives.

A.1 Simulation Setup

We build an NDN environment using the NS-3 based simulator ndnSIM [2]. Caching space is either distributed evenly across all NDN nodes or only at edge nodes to represent both caching paradigms. We ensure equal cache capacity between edge and ubiquitous caching. This total capacity is allocated proportional to the size of all video content provided by the video producer, formally as $\sum_{n=1}^{N}\sum_{b=1}^{B} S_n^b * \omega$ where S_n^b is the size of a video segment with index n encoded at bitrate b. N is the number of video segments, and B is the number of video encodings, and ω is a control parameter

Table A.1: Simulation parameters for Transcoding and Bitrate-aware Caching performance

NDN	
Number of video files	250
Number of video segments per file	40
Number of NDN nodes	16
Video segment playback time	4 sec
Number of video consumers	32
Request interval on video file/session (sec)	400
Skewness factor (α)	1.2
Content store size percentage (ω)	0.02
FESTIVE	
Drop Threshold	0.8
Combine Weight	8

that enables us to examine the caching performance under different cache sizes.

Consumer-side adaptation is simulated via our implementation of FESTIVE [26]. FESTIVE is a highly-cited approach, and a representative of throughput-based adaptation control algorithm. A video session would be triggered following a Poisson process, where consumers' interests on video content are captured by a *Zipf* distribution (controlled by skewness parameter α). Once a video session is initialized, video segments within the requested video file are retrieved under control by FESTIVE. Video files are 160 seconds in duration and are divided into 4-second segments. Each video segment is prepared at 1, 2.5, 5, and 8 Mbps, which are recommended encoding bitrates by YouTube [67]. The simulation parameters are listed in Table A.1.

We evaluate RippleFinder and the transcoding model in [27], which we refer by name as *Transcode*. RippleFinder is a cache placement scheme, where its caching decisions are updated until a steady state is reached. We assume zero transcoding cost/delay for Transcode to highlight the upper bound performance of transcoding

at the network edge. In addition, we evaluate CE2 with LFU replacement. Although CE2 is not designed specifically for adaptive streaming, it is a widely used benchmark for ubiquitous caching. LFU caters to content popularity and outperforms LRU. We also test a variation of CE2 where only edge nodes are allocated cache capacity, along with LFU for content replacement. We name this approach as *EdgeOnly* to represent generic edge caching. All results are presented at a 95% confidence level.

A random topology is generated by BRITE [48] to mimic a realistic streaming scenario [59]. In this topology, we chose a video producer such that the hop distance between any consumer and the producer ranges from three to six hops. This variation in hop distance would cause different video access delay for consumers. We choose in-network link capacity at 20 Mbps, and the 'last-mile' link bandwidth varies by fluctuation patterns as we detail in A.2. As a result, the highest bitrate (8 Mbps) cannot be retrieved directly from the producer and must be provided by caches. We choose this relatively small link capacity to examine the performance that is enhanced by caching policies.

A.2 Bandwidth Fluctuation Pattern

We investigate different bandwidth variation patterns on the 'last-mile' link between each consumer and their edge node, to mimic wired and wireless networks. Three patterns were discovered in studies on real measurements for mobile users [52], in addition to a stationary pattern for benchmarking. Thus, we adopted four variation patterns are evaluated as shown in Figure A.1. Throughout our experiments, we discovered that the variations in performance of caching schemes under Patterns $\mathcal{C}$ and $\mathcal{D}$ were not statistically significant. We thus opted to present results under

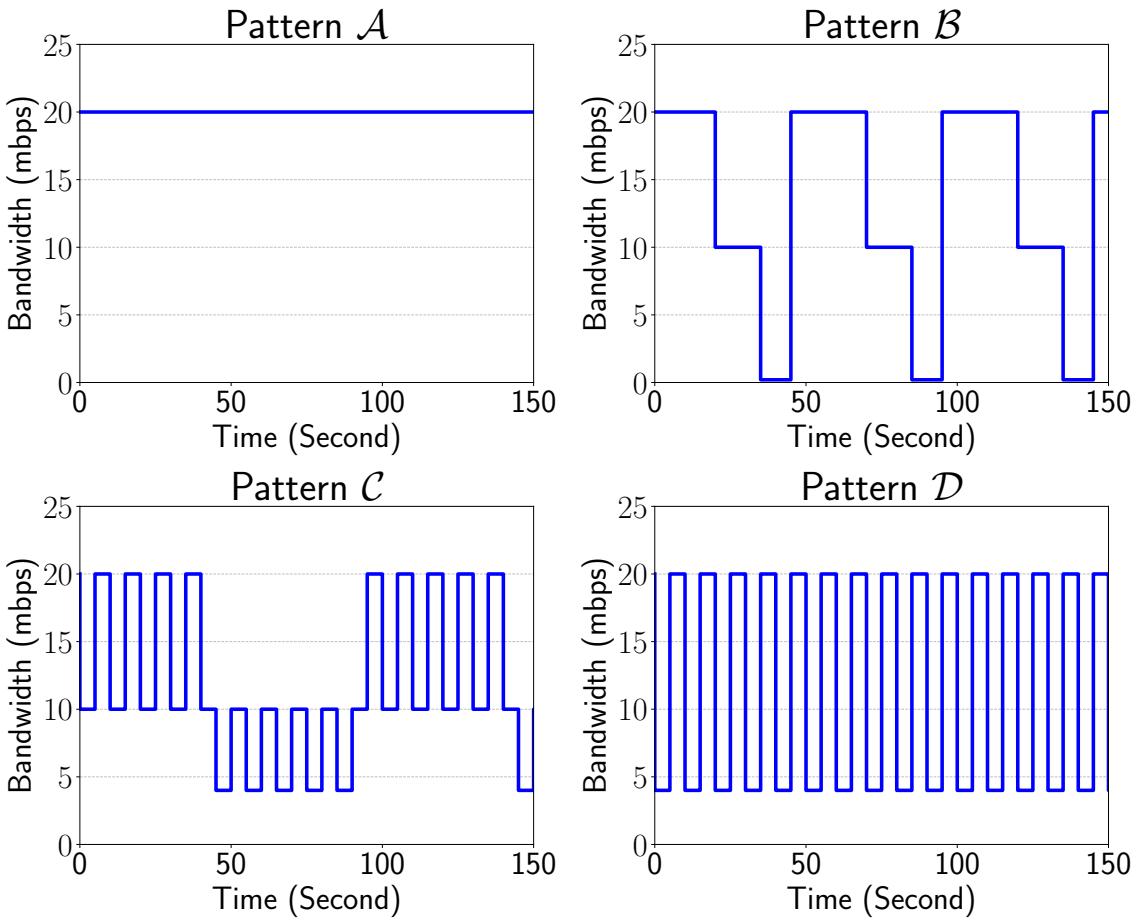


Figure A.1: 'Last-mile' bandwidth fluctuation pattern

Pattern $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$ only.

A.3 Cache Hit Ratio

Cache hit ratio is a classic metric to evaluate the performance of caching schemes. As shown in Figure A.2, we observe even the baseline CE2 with LFU outperforms Transcode, because of high cache redundancy caused by edge caching. This relationship remains the same no matter the bandwidth fluctuation pattern, cache capacity

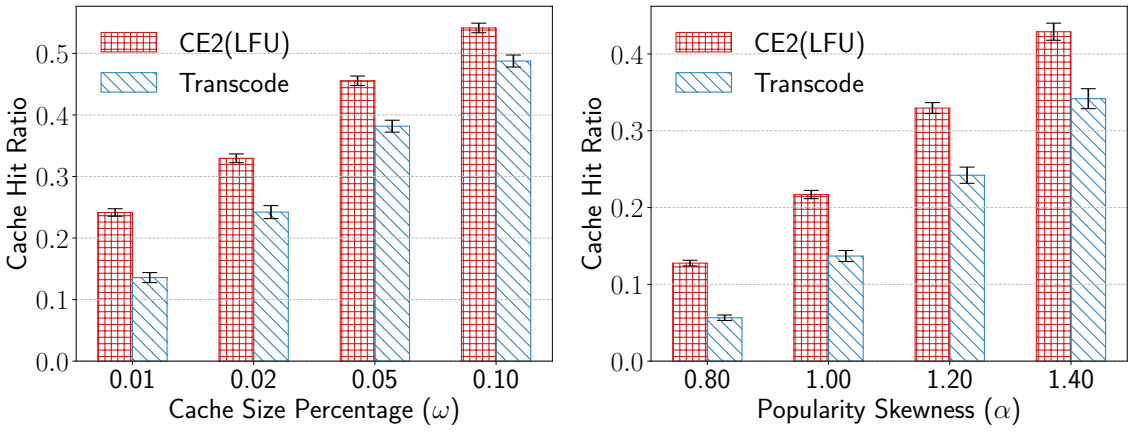
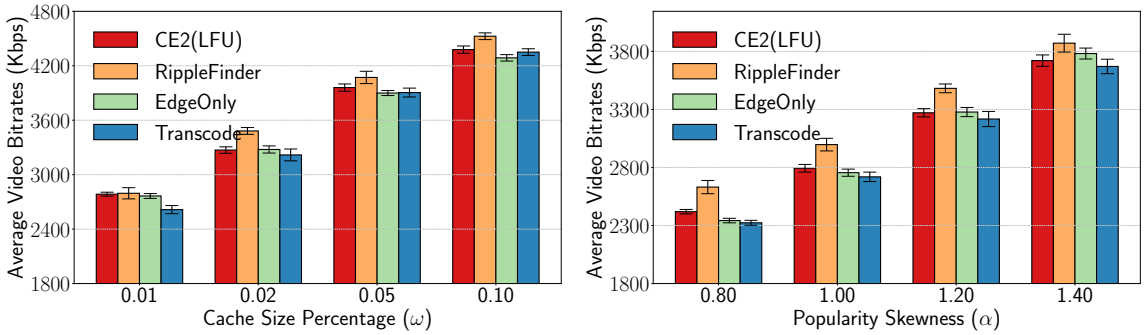


Figure A.2: Cache hit ratio under bandwidth pattern $\mathcal{A}$

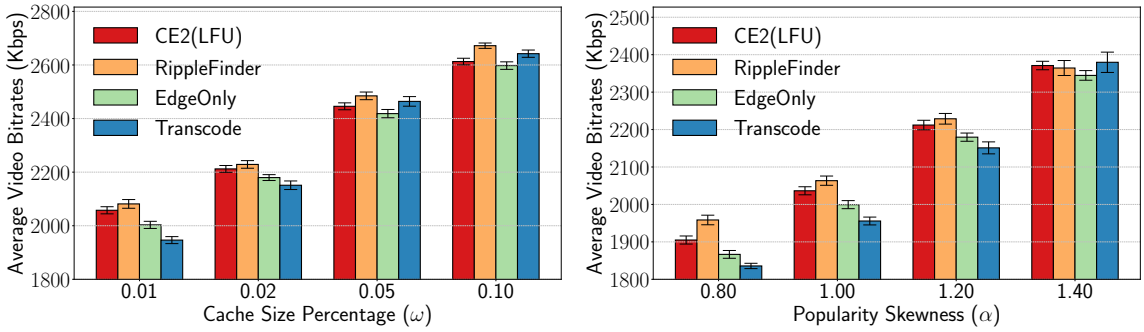
or content popularity skewness. However, our following observations on QoE contradict the current result, where Transcode outperforms CE2 across almost every QoE metric that we examined. As the effectiveness of caching for video streaming must be verified by consumers' QoE, QoE metrics are thus a more direct indicators of system performance. Cache hit ratio itself, as a conventional cache metric, has critical flaws when measuring schemes particularly for video streaming. As cache hit ratio cannot distinguish 'where' this hit occurs, cache hits at the edge or within the network can cause significantly different video throughput that alters the behaviour of consumer-side bitrate adaptation, impacting users' QoE.

A.4 QoE Metrics

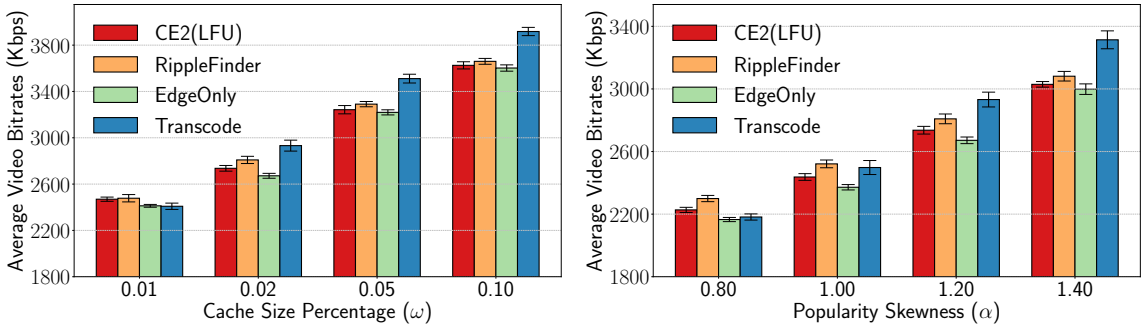
DASH industry forum has published a standard set of QoE metrics [56]. In our experiments, we selectively adopt three metrics from the standard set, *Average Video Bitrate*, *Rebuffer Percentage* and *Bitrate Switch Count*. Other metrics, such as *Rebuffer Count* or *Bitrate Switch Rate* are also evaluated but not reported, since they either



(a) Pattern $\mathcal{A}$



(b) Pattern $\mathcal{B}$



(c) Pattern $\mathcal{C}$

Figure A.3: Average Video Bitrate across cache size and popularity skewness.

share a similar trend with presented metric or the performance difference (caused by caching) is insignificant.

A.4.1 Average Video Bitrate

This metric represents the average video quality that consumers request among all video sessions. Results are grouped by bandwidth fluctuation pattern, and in each group we present the performance across cache capacity and content popularity skewness. As shown in Figure A.3a, transcoding at the network edge has no advantage over ubiquitous caching with constant bandwidth at the 'last-mile' link (pattern $\mathcal{A}$). The video quality difference between CE2 and Transcode was statistically insignificant. Instead, RippleFinder delivers the highest quality to consumers, which reinforces bitrate-aware caching as the superior paradigm under pattern $\mathcal{A}$. This is because Transcode only maximizes its utilization of cached content when it assumes that ' requests to all bitrates are equally likely'. However, a constant link capacity fails to provide enough bandwidth fluctuation, and video throughput variations are mainly caused by in-network traffic congestion. Thus, this assumption is not always satisfied across all video segments, which diminishes the performance of Transcode.

Figure A.3b presents delivered video quality when there is intermittent connection failure. The performance trend is similar to pattern $\mathcal{A}$. As requests for lower quality content are dominating under this pattern $\mathcal{B}$, not all bitrates are frequently requested for each segment which impacts the performance of Transcode. Besides, it is noticeable that at low capacity or low popularity skewness, EdgeOnly delivers even higher video quality than Transcode. This means online transcoding, even with zero processing delay, brings no benefits to system performance. This is because Transcode requires caching only the highest quality no matter what are the frequently requested bitrates. When requests for low quality content are dominating, Transcode forces edge caches to store the highest quality segments, which not only consumes more

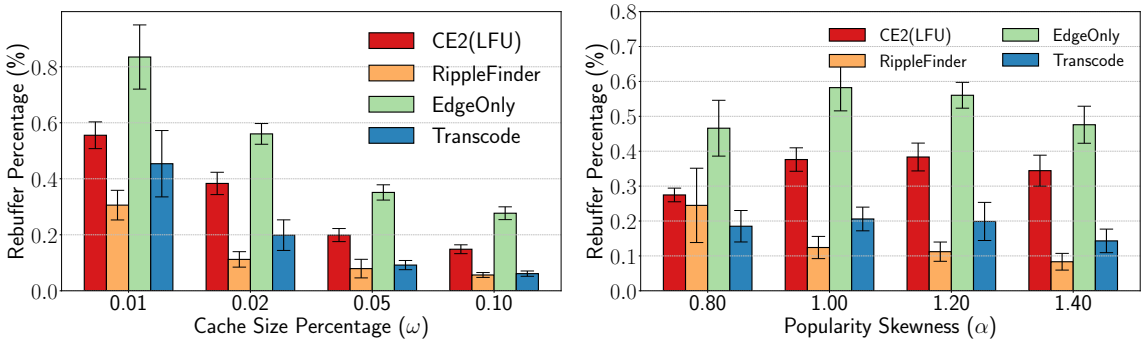
caching space than needed (for lower quality content) but also reduces the amount of video content that can be served by the cache.

Transcode presents superior performance mainly under pattern $\mathcal{C}$ as shown in Figure A.3c. This is due to bandwidth fluctuation between 4 Mbps and 20 Mbps creating more chances for all encoding bitrates to be frequently requested, which boosts the performance of Transcode. At large cache volume, this enhancement by Transcode is significant since large caching space gains more advantage from the efficient cache utilization of Transcode that allows to cache only the highest quality video segment.

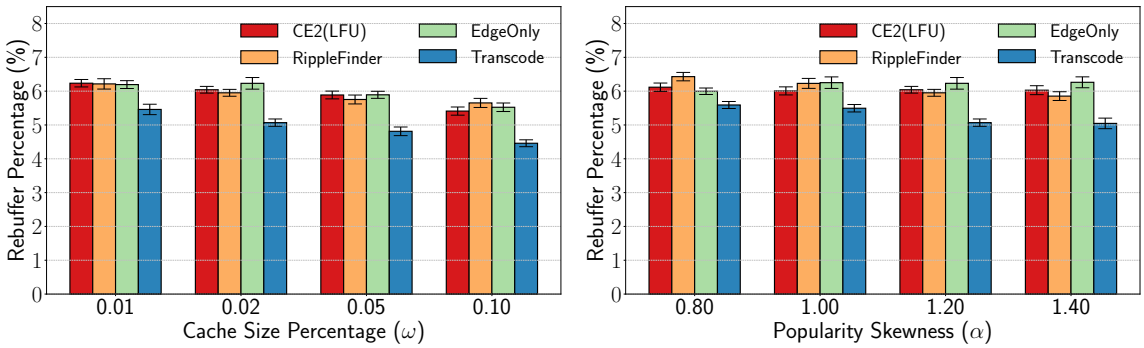
A.4.2 Rebuffer Percentage

This metric is defined as the time spent in a video freezing state over the active time of a video session. It is noticeable that EdgeOnly causes a higher chance of video freezing than ubiquitous caching scheme CE2. Intuitively, as a representative of edge caching, EdgeOnly would satisfy requests closer to the consumer, which should lead to less video access delay than CE2. This counter-intuitive result is affected by consumer-side bitrate adaptation. Cache hits on edge caches have a higher chance than in-network caches to trigger a video quality upgrade. However, this upgrade is harmful once high quality content is not sustainable. The follow-up cache misses would require consumers to retrieve content directly from the producer, which results in a even longer access delay and a higher chance of video stalling.

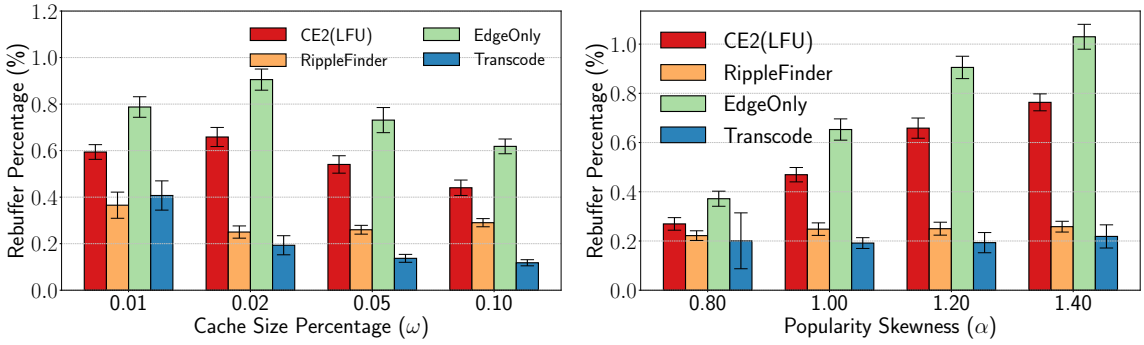
In contrast, Transcode performs better than EdgeOnly, since it can satisfy video requests for any version of the content, with a constant cost of caching space (by storing only the highest version). RippleFinder achieves less video freezing than



(a) Pattern $\mathcal{A}$



(b) Pattern $\mathcal{B}$



(c) Pattern $\mathcal{C}$

Figure A.4: Rebuffer Percentage across cache size and popularity skewness.

Transcode under constant link capacity as shown in Figure A.4a. Transcode re-gains its advantage under fluctuated link capacity in Figure A.4c. Under an intermittent network connection, all tested schemes are prone to significant video stream stalls

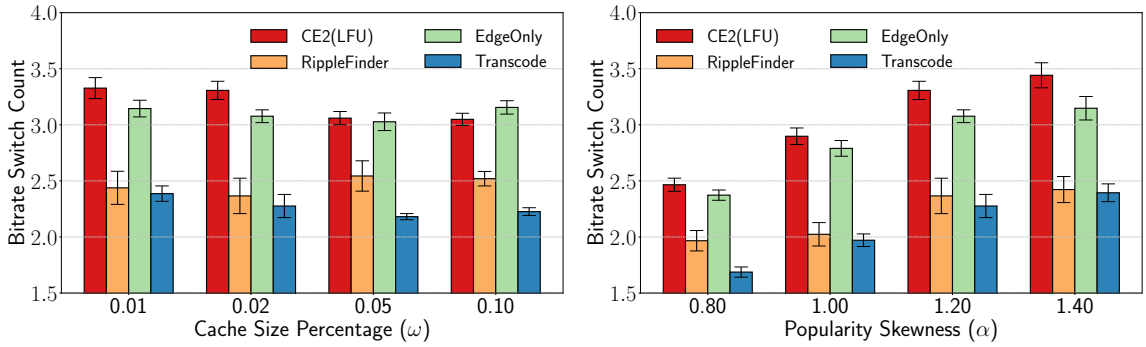
(although Transcode may perform slightly better). The impact of caches on video stalling is thus negligible under pattern $\mathcal{B}$.

A.4.3 Bitrate Switch Count

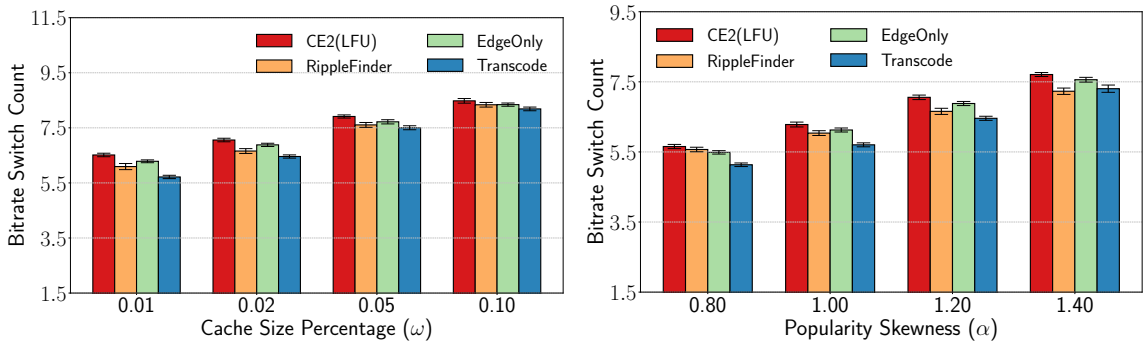
This metric is defined as the number of times the requested video bitrate changed during a video session. It is evident that EdgeOnly causes more bitrate oscillations than Transcode, which indicates that the edge caching paradigm alone is not the key contributor to smooth video playback. We suspect our assumption of zero transcoding delay is the main reason for such performance, as the same video throughput is guaranteed across all versions of popular video content. Thus, a higher degree of bitrate oscillation is expected when Transcode is applied under a realistic setting that factors in the inevitable processing delay, which varies as the highest quality version is transcoded to different bitrates. In addition, RippleFinder can still achieve a similar bitrate switch count as Transcode under bandwidth pattern $\mathcal{A}$ and $\mathcal{C}$. As shown in Figure A.5a, RippleFinder even matches the upper bound performance of Transcode at high popularity skewness (e.g., at $\alpha = 1.2$ or 1.4). This result highlights the potential of bitrate-aware ubiquitous caching schemes in controlling bitrate oscillations.

A.5 Insights on Ubiquitous Caching vs. Edge-Transcoding

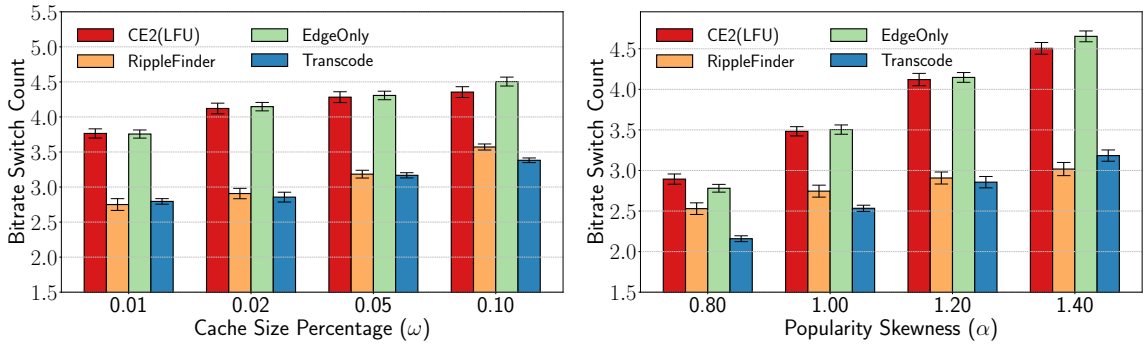
Throughout our experiments, we discover that neither transcoding nor bitrate-aware caching present blanket solutions across all bandwidth fluctuation patterns. When



(a) Pattern $\mathcal{A}$



(b) Pattern $\mathcal{B}$



(c) Pattern $\mathcal{C}$

Figure A.5: Bitrate Switch Count across cache size and popularity skewness.

consumers are dedicated to a fixed bandwidth or connect via a wired link, bitrate-aware caching is better suited to facilitate streaming services. RippleFinder outperforms even the upper bound performance of transcoding with regards to video quality

and playback freezing, while almost matching Transcode in bitrate oscillation.

However, when consumers are connecting via mobile devices, transcoding has a great potential in overall QoE improvement. In reality, as the processing delay of transcoding exists and varies by case, it is necessary to evaluate against best-known bitrate-aware caching schemes to validate this advantage.

We also noticed the performance of Transcode diminishes at low cache capacity or low popularity skewness. Edge caching would cause a higher degree of cache redundancy and the online-transcoding assumption may not hold in many scenarios. Both of these factors undermine cache utilization, which impacts performance when the available cache capacity is limited. We thus emphasize the edge transcoding approaches when high cache capacities are guaranteed, coupled with minimal-contention on caching resources. Same reason applies to high popularity skewness, where few video content is popular and competes for cache.

In designing caching models, it is important to take into consideration the impact of transcoding on computing and networking resources. That is, simply assuming that edge nodes are computationally more equipped than core routers, does not warrant an assumption of superior performance that could handle both computationally-intensive transcoding along with edge computing requirements.

As we previously mentioned, transcoding also has significant storage and communication cost, which should be factored into the design. In addition, even if edge nodes have significant caching capacity, the notion that they can equate to the caching capacity found in ubiquitous caching models is not always true. Thus, relying on edge caching alone inherently sacrifices potential space that could benefit more content.

A.6 Conclusions

In this work, we addressed the seldom investigated comparison between the impact of transcoding and bitrate-aware caching on consumers' QoE. We conducted extensive experiments to examine their performance across various bandwidth patterns, cache capacity, and popularity skewness measures. Our experiments demonstrated that conventional metrics, such as cache hit ratio, are not ideal indicators of video-related system performance, as they often contradict QoE performance benchmarks. We thus adopted industry-leading benchmarks in quantifying QoE, and accordingly contrasted the performance of both paradigms.

Even under the assumption of zero transcoding delay, we discovered that bitrate-aware caching can often match or outperform the upper bound performance of transcoding. Based on our observations, bitrate-aware caching is more suitable to serve consumers with fixed and dedicated link capacity when cache resources are constrained. Online transcoding is more suitable to serve mobile consumers when there is a significant amount of caching space and computational power at the edge, in excess to the operational needs of the omnipresent edge computing architecture.

One of the important future directions to evaluate different caching models, is investigating user-centric video request patterns in edge networks. We are in need of more representative models for video behavior in mobile environments that capture video viewing activity. Evidently, the co-existence of transcoding functionalities with other edge tasks is a problem that requires further investigation. This is especially important when edge computing architectures are tasked with significant offloading and migration requirements, which may hinder their responsiveness to time-sensitive video traffic management.