Bitrate Adaptation-aware Cache Partitioning for Video Streaming over Information-Centric Networks

Wenjie Li*, Sharief M.A. Oteafy†, Marwan Fayed‡ and, Hossam S. Hassanein*
*Queen's University, Canada
Email: {liwenjie, hossam}@cs.queensu.ca
†DePaul University, USA
Email: soteafy@depaul.edu
‡University of St Andrews, United Kingdom
Email: marwan.fayed@st-andrews.ac.uk

***Abstract*—Recent studies suggest that performance gains for content delivery over Information-centric Networks (ICNs) may be negated by Dynamic Adaptive Streaming (DAS), the de facto method for retrieval of multimedia content. The bitrate adaptation mechanism that drives video streaming appears to clash with generic ICN caching techniques in ways that affect users' Quality of Experience (QoE). Cache performance diminishes as video consumers dynamically select content encoded at different bitrates. Motivated by preliminary evidence suggesting the merits of bitrate-based cache partitioning, we introduce a scheme to dissect the cache capacity of routers along a forwarding path according to dedicated bitrates. To facilitate this partitioning, we propose a guiding principle *RippleCache*, which stabilizes bandwidth fluctuation while achieving high cache utilization by safeguarding high-bitrate content on the edge and pushing low-bitrate content into the network core. We further propose a cache placement scheme, *RippleFinder*, to realize this *RippleCache* principle and highlight its impact on users' QoE by cache partitioning. The performance gains are reinforced by evaluations in NS-3. Measurements show *RippleFinder* can significantly reduce bitrate oscillation, while ensuring high video quality, indicating overall improvement to QoE.**

***Index Terms*—Information Centric Networks; Named Data Networking; Dynamic Adaptive Streaming; In-network Caching.**

I. Introduction

The dominance of streaming video traffic on the Internet makes it a high-value, as well as high-priority, candidate for information-centric network (ICN) service [1]. Streaming video traffic is also known to defy the long-valued Internet tenets of stability, utilization, and fairness, that are only beginning to be understood and addressed [2], [3]. This suggests that video delivery services could be similarly problematic in an ICN, which is optimized to deliver non-adaptive and non-video traffic. It is therefore prudent to understanding and design for adaptive multimedia within the ICN context.

Dynamic Adaptive Streaming over HTTP (DASH) is the application-layer standard that is used to deliver multimedia content over the network [4]. A DASH implementation has three salient features. Content is first partitioned into equal duration segments. All segments are then encoded at multiple bitrates in order to accommodate a variety of network conditions. Finally, adaptive algorithms are used to retrieve the highest level of quality, subject to estimates of available network resources. These three attributes in combination have been central to maximizing consumer satisfaction, while minimizing content provider costs of delivery. However, as streaming video traffic approaches 80% [5], application-layer solutions are facing issues of scale.

In-network caching of video segments with variable bitrates is touted as being one solution. The placement of video segments with variable bitrates in ICN caches, which is the focus of this paper, is known to be far from intuitive. Existing caching schemes (e.g., [6], [7]) attempt to fill this video-to-cache-placement gap by utilizing a snapshot approach which is based on instantaneous picture of adaptive video traffic in ICN. Despite improvements on video throughput in a constrained network, this snapshot approach ignores the interplay between cache placement and consumer-side bitrate adaptation, which introduces dependencies that can diminish cache performance [1].

This dependency between caches and bitrate adaptation leads to a potential for "oscillation dynamics" [8]. For example, consumers that retrieve low-bitrate segments from nearby caches will perceive good performance. This observation will trigger a desire for higher-quality content that may be stored in the network core. As data from the network core has to be delivered via a longer path than the earlier cache, poor performance from the video source will cause the streaming application to reduce its video quality preference. Oscillation dynamics is intrinsically caused by estimates of unstable network conditions that occur with intermittent cache hits and misses. As a result, bitrate adaptation may recommend users to request for different bitrates, exceeding the expected behavior learned from the last snapshot of system; and cache placement that is derived from the last snapshot of system would become outdated immediately because of this shift on preferred bitrates.

The issue of oscillation dynamics has been studied within the conventional Internet. For example, a cache-aware bitrate adaptation [9] is proposed, where an independent thread of adaptation logic would be triggered when cache hits occur. However, ICN architecture significantly differs from conventional Internet where all content is hosted at known locations

978-1-5386-4413-3/18/$31.00 ©2018 IEEE

and consumer estimates of system performance are dominated by network effects. In an ICN where video segments with various bitrates may be stored in different caches, the consumer-side adaptation techniques have no means to distinguish poor performance in the network from poor performance at the cache. Thus, we argue that a "good" caching scheme that can stabilize the bandwidth fluctuation is the fundamental solution for ICN to relieve this oscillation.

In this paper, we design, implement, and evaluate such caching schemes that improve consumers Quality of Experience (QoE) by reducing bitrate oscillation. We first carry out exploratory experiments to understand the influence of cache placement on adaptive streaming. We find that bitrate oscillation pattern emerges with hop distance of requested content along the forwarding path. This insight then motivates us to safeguard cache capacity along each forwarding path for particular bitrates via cache partitioning. As a result, video content at a selected bitrate can be retrieved within an appropriate hop distance that stabilizes bandwidth fluctuation.

Our contributions in this work are threefold:

1) We carry out exploratory experiments on existing caching schemes to observe the cache distribution under adaptive streaming, and the observed interplay between caches along a forwarding path motivates the idea of a single aggregate of caches into a *cache path*.
2) We propose the novel concept of ***RippleCache***, as a guiding principle for adaptation-aware cache partitioning. *RippleCache* manages the entire cache capacity of ICN nodes along each forwarding path by safeguarding high-bitrate content on the edge and pushing low-bitrate content into the network core.
3) We present a cache placement scheme ***RippleFinder***, that realizes the *RippleCache* principle, to validate the concept of cache partitioning. Experimental results show *RippleFinder* can significantly reduce bitrate oscillation, increase video quality, and indicate overall improvement to QoE.

The remainder of this paper is organized as follows. In Section II we present related work, focusing on recent contributions to bitrate adaptation control and video caching in ICN. Section III pinpoints the challenges of adaptation-agnostic caching schemes on adaptive video streaming, and presents the *RippleCache* principle. To assess the potential gain of *RippleCache*, We elaborate on an embodiment scheme *RippleFinder* in Section IV. Section V presents our experiment setup and performance evaluation. We conclude in Section VI and present our final remarks.

II. Related Work

In the domain of adaptive video streaming, users' QoE can be improved by both client-side and server-side control [10]. Rate adaptation on the client-side can be *Stepwise* [11], [12], where changes in selected bitrate are constrained to contiguous levels of encoding. This has the effect of smoothing visual changes in resolution and quality. *Non-stepwise* adaptation has no such limitation on bitrate changes, and is instead guided by indirect means of resource estimation such as buffer occupancy [13], [14].

Ubiquitous caching [15] is a fundamental feature of ICN, and could effectively reduce redundant traffic generated by duplicate requests. Due to the decoupling of content and location in ICN naming mechanisms, information is not bonded with a certain host and can be retrieved from anywhere in the network. In-network caching schemes in ICN have been heavily investigated [16]. A consensus is reached where caching performance can be enhanced by catering to content popularity [17], [18]. For example, request statistics may be processed to make caching decisions that reduce the hop distance between consumer and content [17]. The request frequency has also been utilized to annotate segments of popular content and resize caching windows [18].

The relationship between in-network caching and bitrate adaptation has also attracted attention. For example, Kreuzberge et al. [19] develop a cache-aware traffic-shaping policy in response to the unfair bandwidth sharing generated by rate-adaptive video streams. Other studies, such as our previous work on revealing the interplay between caching and bitrate adaptation [20], motivates the need of bitrate-adaptation-aware caching. The focus on caching specifically for adaptive video streaming is comparatively recent. Examples include building cache models that accommodate multiple bitrates of the same content [6], [21], [22]. However, these work either drives caching mechanisms using the steady states that emerge from modelling bitrate adaptation as a Markovian process [6], or assumes random behaviour from bitrate adaptation generated by Gaussian model [21]. While insightful, these studies are built on assumptions that overlook the variations of realistic client-side bitrate adaptations.

In this work, we specifically address the interaction between ubiquitous caching and bitrate adaptation. To the best of our knowledge, our proposed *RippleCache* principle and its embodiment *RippleFinder* are the first attempts to harness the interplay between bitrate adaptation and cache placement to improve users' QoE.

III. Cache Partitioning for Adaptive Video Streaming

We begin by motivating our design choices with evidence from our exploratory studies to understand the interaction between adaptive video traffic and cache placement. Our results demonstrate that adaptation-level dynamics change with hop distance, which motivates us to safeguard portions of cache capacity for different bitrates, in order to improve cache utilization and reduce bitrate oscillation.

A. Impact of Cache Placement on Bitrate Adaptation

We evaluate the benchmark Cache Everything Everywhere (CE2) with Least Frequently Used (LFU) replacement [15], under experiment settings as described in Section V.

Figure 1 presents the likelihood of incurring a bitrate adaptation as a function of hop distance between the consumer and the cache. We present results for lowest (1 mbps) and highest

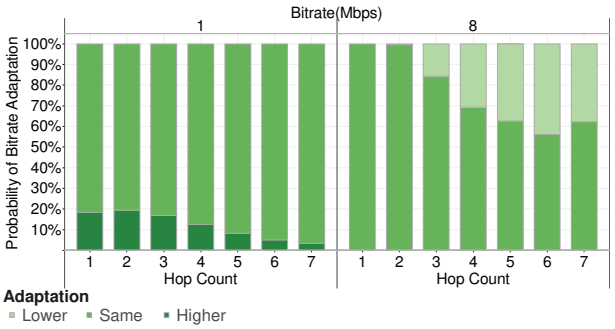


Fig. 1. Likelihood of bitrate adaptation at different hop distances.

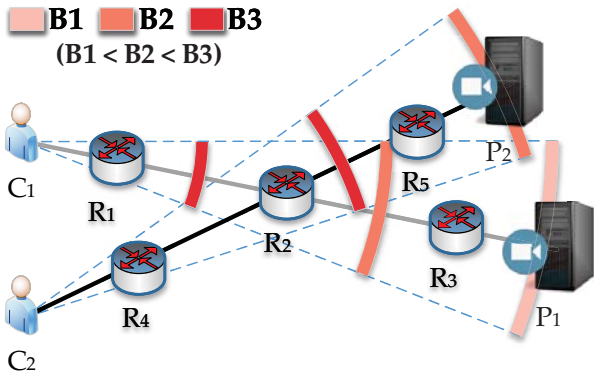


Fig. 2. cache partitioning by encoding bitrates along each forwarding path

(8 mbps) video quality, which are the most representative bitrates. We note that bitrate adaptations are triggered in response to changes in network and caching conditions. In order to reduce bitrate oscillation, the cache placement must be made with the least likelihood to trigger adaptation.

Figure 1 shows that, among 1 mbps streams, it is more likely to trigger an adaptation at low hop count while the stability increases at higher hop count. A contrasting trend emerges for high-bitrates. We observe from Figure 1 that the highest 8 mbps streams are more likely to remain the same quality within the first two hops. This implies high-bitrate content must be retrieved close to consumers. Otherwise, it becomes unavoidable to trigger bitrate adaptation when high-bitrate requests are forwarded into the core network. The combination of observations summarized by Figure 1 demonstrate the need for safe-guarding edge capacity for high-bitrate content by pushing low-bitrate content into the core. We expect that, by reserving cache capacity for a certain bitrate in an 'appropriate' range of hop distance from consumers, bandwidth fluctuation can be stabilized, which enables us to further reduce bitrate oscillation. This motivates our design of an adaptation-aware cache partitioning.

B. RippleCache Partitioning Principle

Our early experiments underscore the need for cache partitioning. However, deciding on which portions of the network should undergo cache partitioning, and the size of each partition, presents a significant challenge. This leads us to develop *RippleCache* as a guiding principle for partitioning, stated as follows:

Definition. ***RippleCache****: Considering a forwarding path from consumer(s) to a video producer, cache capacity along this path would be partitioned and safeguarded for video bitrates in a descending order.*

We further elaborate on this principle in Figure 2, where consumers can adapt video requests among three bitrates ($B_1 < B_2 < B_3$). The route from video consumer C_1 to provider P_1 follows routers R_1, R_2 and R_3. *RippleCache* would require any cache placement scheme that realize this principle to manage the cache capacity on R_1, R_2 and R_3 together, and safeguard caching space from R_1 to R_3 in the order of B_3, B_2 and B_1. Figure 2 depicts a representative partitioning example guided by *RippleCache*, where R_1 is arranged to cache video content particularly for B_3, R_2 for B_2 and R_3 for B_1. The boundary of partitions that separates the caching space for different bitrates is named as a *Ripple*.

RippleCache works on each forwarding path. As different consumers may experience varying bandwidth, complete different cache partitions may be constructed along each forwarding path. Thus, a certain router in the network may reside in multiple partitions simultaneously, facing a contradictory guidance for caching video content of different bitrates. That is, as shown in Figure 2, the cache placement on forwarding path from consumer C_2 to producer P_2 may request R_2 to cache bitrate of B_3, while another path from C_1 may drive the same router R_2 to cache B_2. The cache placement on R_2 must negotiate among these two routes, which dissects the available cache capacity for each route, in order to satisfy diverse caching preferences from different users.

RippleCache is intrinsically a guiding principle, and must be realized by a specific caching scheme. However, there still exist two issues to be solved before we can design any practical solution: 1) Which are appropriate caching decision criteria, so that the cache placement would form partitions? 2) When conflicting partitioning occurs, which negotiation mechanism can be applied to ensure a fair share on cache capacity among each path? We resolve these issues by proposing our *RippleFinder* scheme, as explained in Section IV.

IV. RippleFinder Cache Placement Scheme

RippleCache describes a high-level guiding principle of cache partitioning. In order to realize this principle, we manifest a cache placement scheme, named *RippleFinder*.

RippleFinder makes caching decisions along each forwarding path at one time, and iterates over all paths to complete the cache placement for all routers in an ICN. It executes upon a central controller, which can monitor the status of caches and send cache placement decisions. Video statistics are collected by edge routers and exchange with central controller to facilitate cache decisions. *RippleFinder* would be triggered once the request pattern derived from statistics significantly

changes, which would only be altered in hours as shown in previous research based on YouTube traces [23].

To facilitate explaining our proposal, an ICN is modelled as a connected graph $G = (\mathbb{V}, \mathbb{E})$. Every node $i \in \mathbb{V}$ is equipped content storage capacity C_i dedicated to adaptive video caching. We further assume single-path forwarding, where routers satisfy video requests by selecting only one interface with least delay to deliver content.

RippleFinder consists of the following five procedures as labeled from 1) to 5). The first four procedures are executed for each forwarding path while the last procedure is executed for each ICN router.

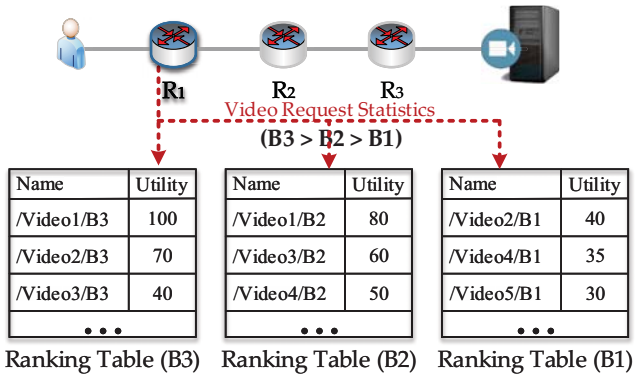
1) **Ranking Table Construction**. Video statistics are collected by the edge router of each forwarding path, and further utilized to construct ranking tables. As shown in Figure 3a, statistics on requested video content are categorized by bitrate, while a ranking table R_b is built for each bitrate b. Every entry in a ranking table consists of the *Name* of requested content and the corresponding caching utility $\mathbb{U}$, which is sorted from high to low in this table. Caching decisions would cater to video segments with high utility. The cache utility for video segment (identified by file ID f, chunk ID k, and encoding bitrate b) is calculated as follows:

$$\mathbb{U}(f, k, b) = \text{PopScore}(f, k, b) * \text{Cost}(b), \quad (1)$$

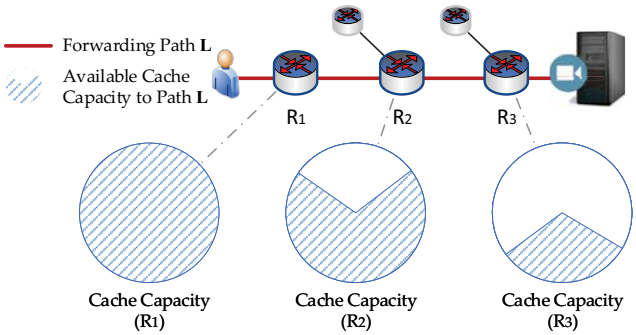
where $\text{PopScore}(f, k, b)$ is the number of requests appeared in statistics, and $\text{Cost}(b)$ is defined as the normalized size of video segment encoded with bitrate b. This utility reflects our design on the importance of caching: video contents that are 1) highly popular; 2) costly to deliver, are preferred caching.

2) **Discover Cache Capacity**. *RippleFinder* manages cache capacity of all ICN nodes along a forwarding path. Specifically, available caching storage along a forwarding path L is concatenated, where the total available size is defined as C_L. Initially, we start cache placement by assuming *RippleFinder* can manipulate the entire caching space of all routers along each path, i.e., $C_L = \sum_{i \in L} C_i$. However, since cache placement decisions are made along each forwarding path independently and routers in an ICN will be shared by multiple paths, one cannot guarantee that our previous assumption remains valid. In fact, as shown in Figure 3b, it is highly possible that only a portion of cache capacity of ICN nodes can be utilized by a forwarding path (e.g., R_2 and R_3 in Figure 3b), while the rest of capacity is reserved to cache content delivered through other paths. Thus, the available size C_L must be adjusted where the update rule is detailed in *Negotiation* procedure, as we explain later in this section.

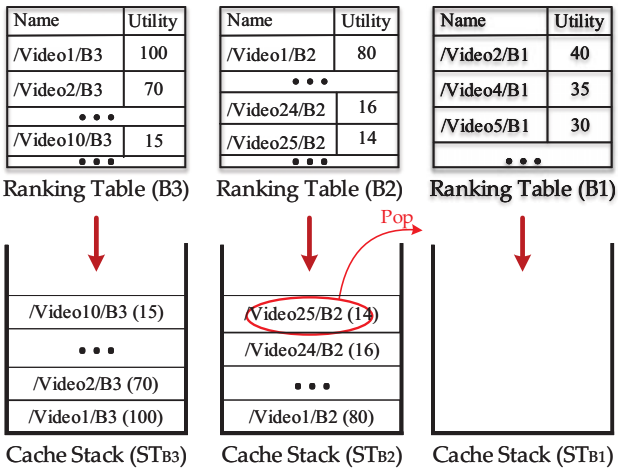
3) **"Push" and "Pop"**. We construct a *Cache Stack* (ST_b) for each bitrate b, as an intermediate data structure to hold video segments. ST_b is filled in a descending bitrate order. For example, ST_{B_3} is the first stack scheduled to be filled, followed by ST_{B_2} and ST_{B_1}. Video content that appears in table R_b will be "Push"ed into corresponding ST_b by ranking order until the size of video segments in all *Cache Stack*s exceeding the total available cache capacity C_L, i.e., "Push" operation will



(a) **Ranking Table Construction**. Construct ranking tables for each bitrate (B_1, B_2 and B_3) from video statistics collected by edge router R_1.



(b) **Discover Cache Capacity**. The shaded volume of router R_1 , R_2 and R_3 would be used to cache video content delivered along path L. These shaded volume is added together, with a size of C_L.



(c) **"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 2 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.

Fig. 3. *RippleFinder*'s procedures oriented around cache paths.

be paused once

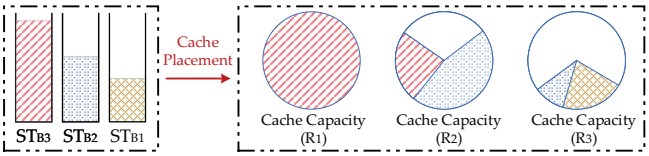
$$\sum_{b \in B} Size(ST_b) \leq C_L \quad (2)$$

is violated. Thereafter, a "Pop" operation is triggered to adjust the content in *Cache Stack*s, in order to restore Equation 2 and resume "Push" operation. Figure 3c depicts a snapshot of this procedure when ST_{B_2} is being filled and Equation 2 is just violated after segment '$\backslash Video25 \backslash B_2$' is pushed. The "Pop" operation would compare the utility of video segments on the top of each *Cache Stack*, and pop up content with the least utility until Equation 2 is restored valid again. As a result, "Push" and "Pop" will occur intermittently, which improves the average utility of video segments in all stacks. When the popped video content belongs to the *Cache Stack* that is currently being filled (i.e., ST_{B_2} in the example as shown in Figure 3c), this stack would be marked as 'complete' since overall cache utility can no longer be improved by continuing pushing content into current *Cache Stack*. The popped content and entries left in the ranking table would be excluded from cache placement. Next, we move on to the following *Cache Stack* (i.e., ST_{B_1} in Figure 3c) and repeat the same operations.

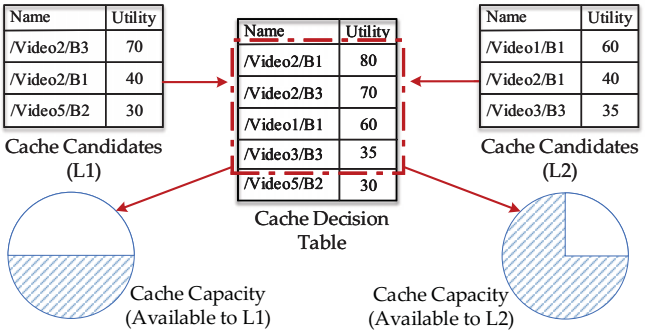
4) **Nominate Cache Candidates**. Video segments are kept in *Cache Stack*s after *"Push" and "Pop"*, while this procedure takes segments out from stacks, and assigns them over ICN nodes along a forwarding path. In order to take advantage of cache partitioning and realize the *RippleCache* principle, video segments in *Cache Stack*s are placed in a descending bitrate order, starting from edge router along path L. As shown in Figure 4a, video content in ST_{B_3} is first to be placed, followed by ST_{B_2} and ST_{B_1}. The number of segments assigned to each router is determined according to the contributed volume of each node to C_L. We nominate these assigned video segments as cache candidates. This procedure is also a precursor for *Negotiation*, where the final cache placement would choose from those candidates.

5) **Negotiation**. This procedure completes two tasks: a) it evaluates nominated cache candidates by all forwarding paths, and determines final cache placement from these candidates. b) it updates C_L based on cache placement result. To complete task a), a cache decision table is built on each router, as shown in Figure 4b, where every entry consists of the *Name* of a cache candidate and the merged cache utility. Each router caches as many content as its cache capacity allows, according to the sorted cache utility from this table. To complete task b), we first define $C_i(L)$ as the available capacity for cache placement on router i along path L. $C_i(L)$ is derived by total size of nominated video segments from path L that appear in the final cache placement. As a result, after all $C_i(L)$ on router i along path L is calculated, C_L can be updated by $C_L = \sum_{i \in L} C_i(L)$.

The coordination among these five procedures is detailed in Algorithm 1. *RippleFinder* executes by iteration: the first four procedures make cache placement based on assumed path capacity C_L, while the last *Negotiation* procedure updates C_L and triggers the cache placement for another iteration. Iterations would stop once C_L is not changed for any forwarding path between two consecutive rounds. Since C_L decreases monotonically during iterations, and is capped by 0 (i.e., *RippleFinder* cannot utilize any size on the path to make cache placement), iterations are guaranteed to converge in the end.



(a) **Nominate Cache Candidates**. Video segments in *Cache Stack*s are placed back to ICN routers. The placement occurs first at R_1, followed by R_2 and R_3 along path L. Video Content in ST_{B_3} is first arranged to place, followed by ST_{B_2} and ST_{B_1}.



(b) **Negotiation**. The cache placement on path L_1 and L_2 nominates cache candidates. Videos that reside inside the dashed rectangle represent the final cache placement decisions. As two cache candidates from L_1 (three from L_2) appear in the final cache placement, $C_i(L_1)$ is then equal to the size of segment '*/Video2/B3*' plus the size of '*/Video2/B1*'. The same rule applies to $C_i(L_2)$, which sums up the size of three video segments.

Fig. 4. Negotiating partitions among individual caches shared along intersecting paths.

RippleFinder has a polynomial time complexity. Let us first denote the number of video files in the framework as F, number of segments in each file as K, number of encoding bitrates as B. The overall complexity of this algorithm is $O(|\mathbb{V}|BFK \cdot log(BFK) + |\mathbb{V}|^3)$, which is dominated by Line 8 and Line 13. *Discover Cache Capacity* iterates over each forwarding path and every router to update cache capacity, with a complexity of $O(|\mathbb{V}|^2)$. *Negotiation* sorts the utility values from a mix of cache candidates of different paths, with a complexity of $O(BFK \cdot log(BFK))$.

V. Performance Results And Insights

We validate *RippleFinder* performance via simulation against known caching strategies. The results and observations inform and reinforce the broader merits of cache partitioning for adaptive streaming. Our evaluations are conducted on the Named Data Networking (NDN) architecture. We maintain without loss of generality that *RippleFinder* design and subsequent analyses can be applied on other ICN infrastructures.

A. Simulation Setup and Parameters

We implement *RippleFinder* over ndnSIM [24], an NS-3 based simulator. Each NDN router is allocated a Content Store

Algorithm 1 *RippleFinder*

Input: Set of forwarding paths $\mathbb{L}$; Set of ICN Routers $\mathbb{V}$; Cache Capacity C_i of Router $i, i \in \mathbb{V}$.
Output: Adaptation-aware cache placement x_i on router i.

for all $L \in \mathbb{L}$ **do**
  $C_L \leftarrow \sum_{i \in L} C_i$
end for
while $C_L <> C'_L$ **do**
  for all $L \in \mathbb{L}$ **do**
    $C'_L = C_L$
    Do **Ranking Table Construction**
    Do **Discover Cache Capacity**
    Do **"Push" and "Pop"**
    Do **Nominate Cache Candidates**
  end for
  for all $i \in V$ **do**
    $C_L, x_i \leftarrow$ Do **Negotiation**
  end for
end while
return x_i.

(CS), where its size C_i is subject to a total available system capacity, controlled by ω, as

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

Client-side adaptation behaviour is simulated via our own implementation of FESTIVE [12], a buffer occupancy-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 200 seconds in duration, 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. Requests on video files are triggered following a Poisson process, with an average time interval between two consecutive requests as 400 seconds. After the request for a video file is triggered, each segment within that file is requested under the control by FESTIVE. In our simulations, we build a 16-node ICN network with a maximum hop distance from a video producer to consumers capped at 7 levels. In this topology, edge routers connect to an equal number of consumers. Link capacity is 20Mbps, which represents dedicated bandwidth for adaptive streaming. The simulation parameters are listed in Table I.

Three additional caching schemes are evaluated alongside *RippleFinder* for comparison. *Cache Everything Everywhere (CE2)* [15] with LRU, also with LFU, is a baseline that commonly appears in literature [15]. *ProbCache* [25] serves as a baseline for probabilistic caching [6], [18]. As our proposed *RippleFinder* is a placement scheme, a snapshot on cache status of other caching schemes is taken at steady state, in order to make a fair comparison.

We present results of three performance metrics: *Expected Bitrate* ($\mathcal{E}_{br}$), *Video Freezing Duration* ($\mathcal{P}_{vf}$) and *Average Times of Bitrate Switches* ($\mathcal{A}_{bo}$), which are inherited from our definitions in previous work [20]. These metrics are used to (i) assess the average video quality a consumer can expect; (ii) identify the duration of a video pausing to buffer; and (iii) track bitrate oscillation at the consumer. Each set of evaluations is repeated to account for the total size of the content store ω and popularity-related bias α. All results are presented at a 95% confidence level.

TABLE I
SIMULATION PARAMETERS

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)	0.8
Content store size percentage (ω)	0.1
FESTIVE	
Drop Threshold	0.8
Combine Weight	8

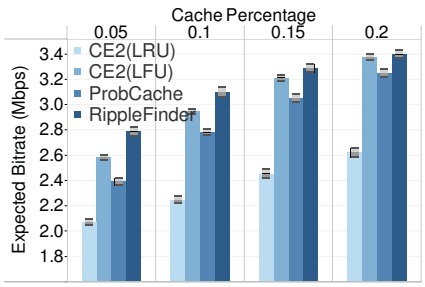
B. The Impact on Expected Bitrate

Expected Bitrate ($\mathcal{E}_{br}$), plotted across Figure 5, is an aggregate measure of the video quality that can be supported by network and cache resources. $\mathcal{E}_{br}$ is calculated by averaging the requested bitrates among consumers during entire simulation period. *RippleFinder* performs as well as other popularity-based schemes (e.g., LFU or *ProbCache*) across different cache capacity and popularity distribution, and achieves advantages especially at low cache capacity and high popularity skewness. For example, Figure 5a shows that at ω = 0.05, *RippleFinder* achieves higher $\mathcal{E}_{br}$ than *CE2* with LFU by 8.1%, and is 34.6% better than *CE2* with LRU. The improvement of *RippleFinder* is significant over LRU but is minor over LFU and *ProbCache*. This is because *RippleFinder* is also a popularity-based scheme, which achieves high cache utilization by capturing request pattern and catering to popular content. In addition, *RippleFinder* considers the factor of delivery cost in its utility function (as shown in Eq 1), which further reduces bandwidth consumption and gains advantages over LFU and *ProbCache* especially with small cache capacity.

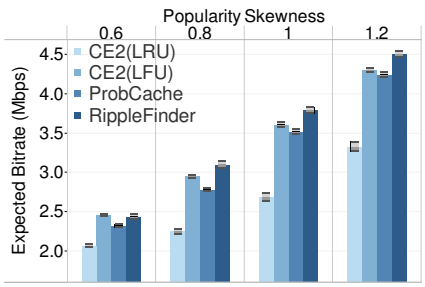
Figure 5b shows $\mathcal{E}_{br}$ under different popularity distributions, controlled by skewness parameter α. When α changes from 0.6 to 1.2, expected video quality gets improved by all tested caching schemes. This is because users' requests concentrate on a smaller set of popular content with larger α, which thereby increases the overall cache hit ratio. *RippleFinder* delivers a consistently higher expected bitrate and outperforms other schemes with large α values.

C. The Impact on Bitrate Oscillation

Bitrate oscillation manifests on-screen by images that appear to improve and degrade in quality. In our measurements,

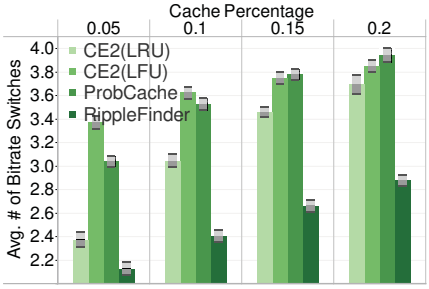


(a) $\mathcal{E}_{br}$ across cache capacity (ω)

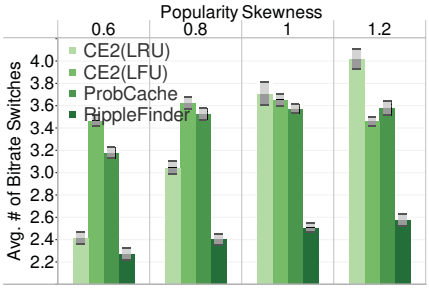


(b) $\mathcal{E}_{br}$ across popularity skewness (α)

Fig. 5. Expected bitrate across tunable ω, α

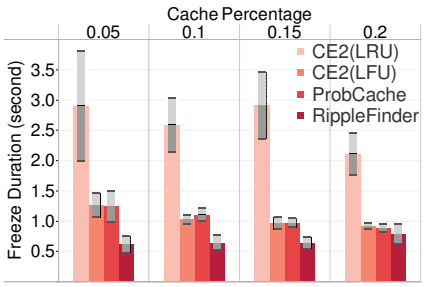


(a) $\mathcal{A}_{bo}$ across cache capacity (ω)

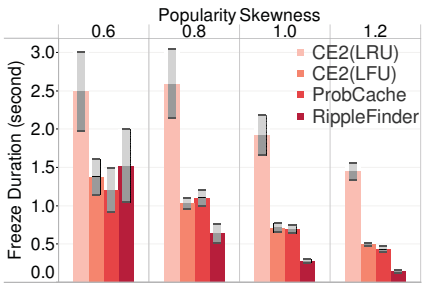


(b) $\mathcal{A}_{bo}$ across popularity skewness (α)

Fig. 6. Bitrate oscillation across tunable ω, α



(a) $\mathcal{P}_{vf}$ across cache capacity (ω)



(b) $\mathcal{P}_{vf}$ across popularity skewness (α)

Fig. 7. Video freezing across tunable ω, α

we average the count of bitrate increases and decreases during watching an entire video file. A representative set of measurements appears in Figure 6, where *RippleFinder* achieves the least bitrate oscillation compared with other caching schemes. In combination with Figure 5, the performance of *RippleFinder* is remarkable since it can not only reduce bitrate oscillation but meanwhile deliver the best video quality.

Figure 6 shows that bitrate switches increase with cache capacity and popularity skewness. This is because either large cache size or skewed popularity distribution would improve cache hit ratio, which triggers bitrate adaptation to upgrade requested video quality. However, as cache resource of each ICN node is limited, a downgrade on video quality would unavoidably occur thereafter. Popularity-based schemes (e.g., *ProbCache*) achieve a higher cache hit ratio than LRU, but this high cache hit ratio also comes with a higher probability of performance downgrade. As a result, popularity-based schemes deliver a better video quality on average with a cost of frequent bitrate adaptation. A good caching scheme, such as *RippleFinder*, would maintain consumers' requests on high-quality content as long as possible, which thereby improves the overall QoE.

D. The Impact on Video Freezing

Short-term variations in network and system conditions can adversely affect playback before bitrate adaptation occurs. One such indication is buffer-induced pausing during playback that manifests on-screen as 'freezing'. The *Video Freezing Duration* ($\mathcal{P}_{vf}$) is shown in Figure 7, calculated as the average playback time spent in a 'frozen' state.

The probability of video playback freezing relates to the video access delay of each segment. Caching schemes that achieve high cache hit ratio can reduce average access delay, which causes less playback freezing. In contrast to its rivals, *RippleFinder* outperforms other schemes since it not only caters to popular content (which improves cache hit ratio), but also realizes the *RippleCache* principle where high-bitrate content is placed closer to consumers. As large amount of video segments with high quality would significantly increase the network delay and choke video traffic, *RippleFinder* is effective on relieving traffic load by satisfying high-bitrate requests as early as possible. Only when the request distribution is least skewed, does *RippleFinder* performance diminish to a degree matched by popularity-based caching.

E. Discussion of Results

Throughout our evaluations we were surprised by the ability of *CE2* with LFU to meet or exceed all but *RippleFinder* in terms of delivered video quality and playback freezing. Looking ahead, the robustness of LFU suggests that performance gains promised by ICNs may be dependent on 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 further reinforced by *RippleFinder* 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. Content quality at all bitrates consequently becomes network- rather than cache-limited.

VI. Conclusion

In this paper, 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, and present a primer in a novel approach to caching, and establishes the premise of safe guarding cache partitions for higher bit-rates, allowing for more ideal cache placement strategies for adaptive video content.

Our proposed safe-guarding mechanism enforces bitrate-based partitioning of cache capacities, named as *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 an embodiment caching scheme, *RippleFinder*, where our experiment results contrast to leading caching schemes, and demonstrate how cache partitioning would improve users' QoE, in terms of high video quality and significant reduction on bitrate oscillation.

More importantly, our experimentation with *RippleFinder* yields the following 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.

References

[1] C. Westphal, S. Lederer, D. Posch, C. Timmerer *et al.*, "Adaptive video streaming over information-centric networking (icn)," RFC 7933, IRTF, 2016, [Online]. Available:http://www.rfc-editor.org/rfc/rfc7933.txt.

[2] J. Chen, M. Ammar, M. Fayed, and R. Fonseca, "Client-driven network-level qoe fairness for encrypted 'dash-s'," in *SIGCOMM Internet-QoE Workshop*, 2016.

[3] A. Mansy, M. Fayed, and M. H. Ammar, "Network-layer fairness for adaptive video streams," in *Proc. IFIP Networking*, 2015.

[4] MPEG, "DASH," http://dashif.org/mpeg-dash.

[5] Cisco, "Cisco visual networking index: Forecast and methodology, 2015–2020," 2016.

[6] W. Li, S. Oteafy, and H. Hassanein, "Rate-selective caching for adaptive streaming over information-centric networks," *IEEE Transactions on Computers*, vol. 66, no. 9, pp. 1613–1628, 2017.

[7] Z. Ye, F. De Pellegrini, R. El-Azouzi, L. Maggi, and T. Jimenez, "Quality-aware dash video caching schemes at mobile edge," in *IEEE International Teletraffic Congress (ITC 29)*, 2017, pp. 205–213.

[8] R. Grandl, K. Su, and C. Westphal, "On the interaction of adaptive video streaming with content-centric networking," in *Packet Video Workshop (PV), 2013 20th International*. IEEE, 2013, pp. 1–8.

[9] D. Lee, C. Dovrolis, and A. Begen, "Caching in http adaptive streaming: Friend or foe?" in *ACM Network and Operating System Support on Digital Audio and Video Workshop*, 2014, pp. 31–36.

[10] J. Kua, G. Armitage, and P. Branch, "A survey of rate adaptation techniques for dynamic adaptive streaming over http," *IEEE Communications Surveys & Tutorials*, 2017.

[11] Z. Li, X. Zhu, J. Gahm, R. Pan, H. Hu, A. Begen, and D. Oran, "Probe and adapt: Rate adaptation for http video streaming at scale," *IEEE Journal on Selected Areas in Communications*, vol. 32, no. 4, pp. 719–733, 2014.

[12] J. Jiang, V. Sekar, and H. Zhang, "Improving fairness, efficiency, and stability in http-based adaptive video streaming with festive," in *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies (CoNEXT)*. ACM, 2012, pp. 97–108.

[13] 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*, vol. 44, no. 4, pp. 187–198, 2015.

[14] G. Tian and Y. Liu, "Towards agile and smooth video adaptation in dynamic http streaming," in *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies (CoNEXT)*. ACM, 2012, pp. 109–120.

[15] M. Zhang, H. Luo, and H. Zhang, "A survey of caching mechanisms in information-centric networking," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 3, pp. 1473–1499, 2015.

[16] A. Ioannou and S. Weber, "A survey of caching policies and forwarding mechanisms in information-centric networking," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 4, pp. 2847–2886, 2016.

[17] J. Li, H. Wu, B. Liu, J. Lu, Y. Wang, X. Wang, Y. Zhang, and L. Dong, "Popularity-driven coordinated caching in named data networking," in *ACM/IEEE symposium on Architectures for Networking and Communications Systems (ANCS)*, 2012, pp. 15–26.

[18] 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)*, 2012, pp. 316–321.

[19] 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)*, 2015.

[20] 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)*, 2017.

[21] Y. Jin, Y. Wen, and C. Westphal, "Towards joint resource allocation and routing to optimize video distribution over future internet," in *IFIP Networking Conference*, 2015.

[22] A. Araldo, F. Martignon, and D. Rossi, "Representation selection problem: optimizing video delivery through caching," in *IFIP Networking Conference*, 2016, pp. 323–331.

[23] P. Gill, M. Arlitt, Z. Li, and A. Mahanti, "Youtube traffic characterization: a view from the edge," in *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, 2007, pp. 15–28.

[24] A. Alexander, I. Moiseenko, and L. Zhang, "ndnsim: Ndn simulator for ns-3," *Technical Report NDN-0005*, 2012.

[25] I. Psaras, W. K. Chai, and G. Pavlou, "Probabilistic in-network caching for information-centric networks," in *Proc. of the 2nd ICN Workshop on Information-centric Networking*. ACM, 2012, pp. 55–60.