Quality of Experience in ICN: Keep Your Low-Bitrate Close and High-Bitrate Closer

Wenjie Li, Sharief M. A. Oteafy, *Senior Member, IEEE*,
Marwan Fayed, *Senior Member, IEEE, ACM*, and Hossam S. Hassanein, *Fellow, IEEE*

***Abstract*— Recent studies into streaming media delivery suggest that performance gains from ubiquitous caching in Information-Centric Networks (ICN) may be negated by Dynamic Adaptive Streaming (DAS), the de facto method for retrieving multimedia content. Bitrate adaptation mechanisms, that drive video streaming, clash with caching mechanisms in ways that affect users' Quality of Experience (QoE). Cache performance also diminishes as consumers dynamically select content encoded at different bitrates. In this article we use this evidence to draw a novel insight: in adaptive streaming over ICN, bitrates should be prioritized alongside popularity and hit rates. We build on this insight to propose *RippleCache* as a family of cache placement schemes that safeguard high-bitrate content at the edge and push low-bitrate content into the network core. Doing so reduces contention of cache resources, as well as congestion in the network. To validate *RippleCache* claims we construct two separate implementations. We design *RippleClassic* as a benchmark solution that optimizes content placement by maximizing a measure for ICNs shown to have high correlation with QoE. In addition, our lighter-weight *RippleFinder* is then re-designed with distributed execution for application in large-scale systems. *RippleCache* performance gains are reinforced by evaluations in NS-3 against state-of-the-art baseline approaches, using standard measures of QoE as defined by the DASH Industry Forum. Our results demonstrate that *RippleClassic* and *RippleFinder* deliver content that suffers less oscillation and rebuffering, all while achieving the highest levels of video quality; thus indicating overall improvements to QoE.**

***Index Terms*— Information centric network, named data networking, dynamic adaptive streaming, in-network caching, bitrate oscillation.**

I. Introduction

THE prevalence of video traffic on the Internet makes it a high-value and high-priority candidate for dedicated caching schemes in the development of Information-Centric Networks (ICNs) [1]. In the conventional IP-based Internet, streaming video traffic is known to defy the long-valued Internet tenets of stability, utilization, and fairness, in ways that are only beginning to be understood and addressed [2]–[4]. This suggests that video delivery services could be similarly problematic in ICN, in particular when the caching system is optimized for non-adaptive and non-video traffic. It is therefore instructive to understand caching behaviour and design for adaptive media within the context of ICNs.

Dynamic Adaptive Streaming over HTTP (DASH) is the application-layer standard that is used to deliver multimedia content over the network [5]. 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, adaptation 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 costs of delivery for content providers. However, as streaming video traffic approaches 80% of global Internet traffic [6], 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, which is the subject of this article, is known to be far from intuitive. Existing caching schemes (e.g., [7], [8]) fill this video-to-cache-placement gap by utilizing snapshots, or instantaneous inference, of adaptive video traffic in ICN. Despite some improvement, snapshots ignore the interplay between cache placement and consumer-side bitrate adaptation that can diminish cache performance [1].

The challenges stem from the interaction between caches and bitrate adaptation that cause "oscillation dynamics" [9]. Oscillation dynamics are intrinsically linked to inaccurate estimates caused by ever-changing network conditions that occur with intermittent cache hits and misses. Consider, for example, consumers that retrieve low-bitrate segments from edge caches and perceive good performance. A consumer-side bit-rate 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.

Manuscript received March 22, 2019; revised February 1, 2020; accepted November 14, 2020; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor E. Yeh. Date of publication December 29, 2020; date of current version April 16, 2021. This work was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) under Grant RGPIN-2014-06587 and Grant RGPIN-2017-06902. *(Corresponding author: Wenjie Li.)*

Wenjie Li and Hossam S. Hassanein are with the School of Computing, Queen's University, Kingston, ON K7L 2N8, Canada (e-mail: liwenjie@cs.queensu.ca; hossam@cs.queensu.ca).

Sharief M. A. Oteafy is with the School of Computing, DePaul University, Chicago, IL 60604 USA (e-mail: soteafy@depaul.edu).

Marwan Fayed is with Cloudflare, Inc., and also with the School of Computer Science, University of St Andrews, St Andrews KY16 9AJ, U.K. (e-mail: marwan@cloudflare.com).

Digital Object Identifier 10.1109/TNET.2020.3044995

1558-2566 © 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: Queen's University. Downloaded on April 28,2021 at 10:45:59 UTC from IEEE Xplore. Restrictions apply.

Oscillation dynamics are not inherent to ICNs only, and have previously been studied in the context of Content Delivery Networks (CDNs). For example, cache-aware bitrate adaptation [10] triggers independent threads of adaptation logic when cache hits occur. However, caching in CDNs differs significantly from that in ICNs. CDNs host all video content, and at fixed locations, so consumer estimates of system-wide performance are dominated by network effects. In contrast, caching in ICNs make it possible for video segments to reside 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 Quality of Experience (QoE).

In this article, 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 impact of network resource sharing on QoE for high-quality content requests is disproportional 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 [11], [12]. 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: rather than a single copy at upstream caches that sit on intersecting paths, QoE is best satisfied by placing copies of the largest segments at multiple edge caches.

This insight leads to, and is validated by, the main contribution of this article in *RippleCache*. We present *RippleCache* as a protocol that safeguards capacity at the edge routers for high-bitrate content, thereby pushing lower bitrate content along the forwarding path into 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 *RippleCache*-guided systems:

1) ***RippleClassic*** serves as a benchmark cache partitioning paradigm. Partitions are created by solving an optimization formulated as a binary integer programming problem. The objective of *RippleClassic* maximizes a metric designed specifically to measure ICN performance for adaptive streaming, that has been shown to have high correlation with consumers QoE [11]. The solutions that emerge place content in such a way that a *RippleCache* emerges.
2) ***RippleFinder*** is a distributed caching scheme that is built on our prior work [12] and executes in polynomial-time complexity. Execution begins at edge routers, from where cache partitions are created along the forwarding path to each video object origin. Placement decisions prioritize by utility, that we design as an indicator of the resource cost of a video segment (by size) and weighted by popularity.

Performance evaluations compare both *RippleCache* designs with ProbCache [13] as a baseline for probabilistic caching, as well as CE2 [14] as a baseline that commonly appears in literature. Metrics are selected and defined in accordance with DASH Industry Forum recommendations [15]. *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 article is organized as follows. In Section II we present related work, focusing on recent contributions to bitrate adaptation control and video caching in ICNs. 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 formulate a benchmark solution *RippleClassic* in Section IV, followed by a light-weight and practical embodiment *RippleFinder* in Section V. Section VI presents our experiment setup and performance evaluation. We conclude in Section VII and present our final remarks.

II. Related Work

Ubiquitous caching [14] 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 bound to a certain host, and can be retrieved from anywhere in the network. In-network caching schemes in ICN have been heavily investigated [16], [17]. A consensus is reached where caching performance can be enhanced by catering to content popularity [18], [19]. For example, request statistics may be processed to make caching decisions that reduce the hop distance between consumer and content [18]. The request frequency has also been utilized to annotate segments of popular content and resize caching windows [19].

In the domain of adaptive video streaming, users' QoE can be improved by both client-side and server-side control [20], [21]. Rate adaptation on the client-side can be *Throughput*-based [22], [23], or *Buffer*-based [24], [25]. *Throughput*-based adaptation makes the best possible estimates on bandwidth by referring to received video throughput from previous segments, and adjusts bitrate selections to match bandwidth estimates. *Buffer*-based adaptation, is instead guided by indirect means of resource estimation such as buffer occupancy.

The relationship between in-network caching and bitrate adaptation has also attracted attention. For example, Jia *et al.* [26] designed a control layer for optimal Interest forwarding and adaptive video caching based on the virtual queue of each bit rate. Kreuzberge *et al.* [27] developed a cache-aware traffic-shaping policy in response to the unfair bandwidth sharing generated by rate-adaptive video streams. Liu *et al.* [28] studied caching behavior over ICN and demonstrated that clients could be served with bit rates even higher than their actual bandwidth, emphasizing the potential of ICN caching and the importance of designing caching schemes for adaptive video content. Other studies, such as our previous work on revealing the interplay between caching and bitrate adaptation [29], 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 [7], [30], [31]. However,

Authorized licensed use limited to: Queen's University. Downloaded on April 28,2021 at 10:45:59 UTC from IEEE Xplore. Restrictions apply.

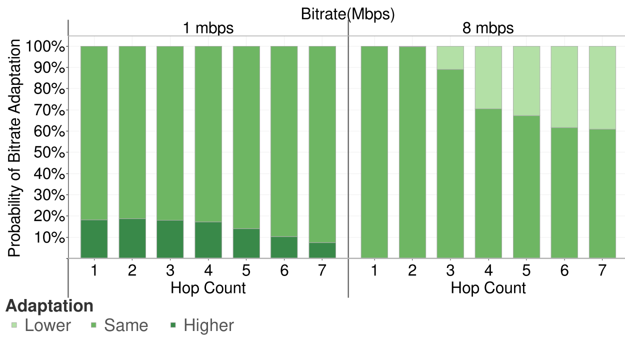


Fig. 1. Bitrate adaptations given cache distance: dark regions indicate switches to the higher bitrate; lighter regions indicate switches to the lower bitrate.

these works either drive caching mechanisms using the steady states that emerge from modelling bitrate adaptation as a Markovian process [7], or assume random behaviour from bitrate adaptation generated by Gaussian model [30]. While insightful, these studies are built on assumptions that overlook the real-world variations of client-side bitrate adaptations.

The observations and outcomes of this work are reinforced by orthogonal designs for datacenters [32]. We note that the achievable end-to-end throughput of a topology is limited, not just by the capacity of the network, but also the portion of capacity to deliver each byte, i.e.throughput is inversely proportional to the network capacity used to deliver data. This observations informs their Jellyfish architecture, that aims to reduce average path lengths, thereby increasing the number of high-throughput flows.

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 embodiments, are the first attempts to address the interplay between bitrate adaptation and cache placement to improve users' QoE.

III. Adaptive Streaming With Cache Partitioning

Our initial studies on the interplay between consumer-side adaptation and in-network cache placement provide us an opportunity to observe adaptation-level dynamics that vary by hop distance. These observations then demonstrate the need for safe-guarding cache capacity for a particular bitrate to facilitate fine-grained cache placement, which lead to our design of the *RippleCache* principle.

A. Adaptation Dynamics Characterized by Cache Placement

To study the impact of consumer-side bit-rate adaptation on cache placement, we carried out extensive experiments to elicit the intrinsic challenge of bitrate oscillation and high bit-rate placement. The following characterizations are drawn from evaluations of the benchmark Cache Everything Everywhere (CE2) with Least Frequently Used (LFU) cache-replacement policy [14]. The experiments were conducted on a 16-node topology where consumers send video requests, guided by the widely adopted FESTIVE adaptation control algorithm [23]. The details of our experiment settings are listed under the 'BIP-tractable' category in Table III under Section VI-A.

The salient results are summarized by Figure 1, depicting the likelihood of incurring a bitrate adaptation as a function of hop distance between the video consumer 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 large as possible.

Observations shown are limited to the highest (8 Mbps) and lowest (1 Mbps) bitrates, where bitrate adaptations occur most frequently relative to cache distances. We note that intermediate bitrates exhibit similar behaviors in a less pronounced fashion. As depicted in Figure 1, the left-most 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 proportion of medium shade increases in the last three hops.

In contrast, the rightmost bars in Figure 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 users' QoE.

B. Ripple-Like Cache Partitioning

Our early experiments underscore the need for cache partitioning. However, rather than conventional partitioning on individual caches, we propose the **RippleCache** partitioning principle that works on 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 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 given in Figure 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

Authorized licensed use limited to: Queen's University. Downloaded on April 28,2021 at 10:45:59 UTC from IEEE Xplore. Restrictions apply.

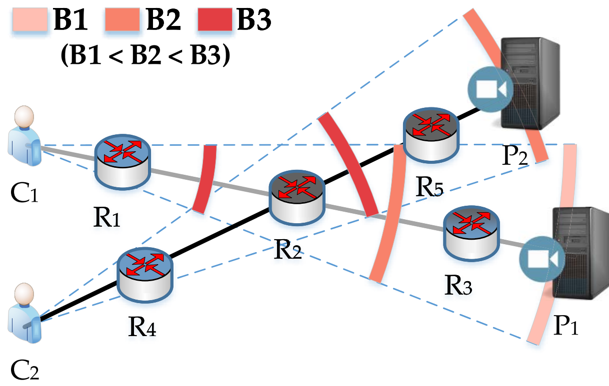


Fig. 2. Cache partitioning by encoding bitrates along each forwarding path.

of three available bitrates $B_3 > B_2 > B_1$, in decreasing bitrate from the consumers. The coloured arcs in Figure 2 mark the partition boundaries that delineate *Ripple*s. We note that *Ripple*s may contain 0 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 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.

IV. *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.

A. 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. Each user 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.

TABLE I

Summary of Notations Used in the Formulation

Notation	Meaning
$\mathbb{V}$	Set of ICN nodes
$\mathbb{E}$	Set of links
$\mathbb{D}$	Set of edge 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
x	Cache placement decision
$[d, p]$	ICN routers on forwarding path from edge d to producer p
δ	Caching status indicator (0 or 1)
θ	Number of video requests received by edge router
γ	Cache reward value
$RB^i_{[d,p]}$	Ripple Bitrate on i^{th} router along $[d, p]$

In this formulation single-path forwarding is assumed, where routers satisfy video requests by selecting the path with least delay to deliver content. Our formulation then optimizes the contributions of in-network caching along each forwarding path individually.

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 have variable sizes. For ease of presentation, video files are fragmented into the same number of fragments K. The number of bitrate encodings is B. Hence, video segments are identified by a *(file, segment, 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 define $[d, p]$ as the sequence of routers on the forwarding path from edge router $d \in \mathbb{D}$ to the video producer $p \in \mathbb{P}$. The length of this forwarding path $[d, p]$ is L and the index of $[d, p]$ starts from 1. Thus $x^i_{[d,p]}$ represents the cache decision variable on the ith 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 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. Finally, the number of requests on video segment (f, k, b) received by edge router d is denoted as $\theta_d(f, k, b)$, with θ_d substituted for simplicity. Notation is additionally summarized in Table I.

Our formulation caters to diverse caching preferences by maximizing the sum of cache reward values. The cache reward of each request is denoted by $\gamma(RB^i_{[d,p]}, b)$, generated by a reward function that is described subsequently in Section IV-B. Briefly stated here, the reward function and value are derived from the *Ripple Bitrate*, $RB^i_{[d,p]}$, the highest sustainable bitrate

Authorized licensed use limited to: Queen's University. Downloaded on April 28,2021 at 10:45:59 UTC from IEEE Xplore. Restrictions apply.

at router i. The optimization is formulated as a BIP problem, as outlined below. The problem is solved in evaluations using Gurobi Optimizer [33]. Given the known complexity of BIP, solving this problem is NP-Complete.

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

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

$$\delta^{i}_{[d,p]}\in\{0,1\},\quad \forall d\in\mathbb{D},\forall p\in\mathbb{P},1\le i\le L \tag{2}$$

$$\sum_{f\in F}\sum_{k\in K}\sum_{b\in B}S(b)*x_v(f,k,b)\le C_v,\forall v\in\mathbb{V}-\mathbb{P} \tag{3}$$

$$\delta^{i}_{[d,p]}\ge\delta^{i-1}_{[d,p]}, \tag{4}$$

$$\delta^{i}_{[d,p]}\ge x^{i}_{[d,p]}(f,k,b), \tag{5}$$

$$\delta^{i}_{[d,p]}\le\delta^{i-1}_{[d,p]}+x^{i}_{[d,p]}(f,k,b), \tag{6}$$

$$\delta^{0}_{[d,p]}=0, \tag{7}$$

$$x_p(f,k,b)=1,\quad \forall p\in\mathbb{P} \tag{8}$$

$$\delta^{L-1}_{[d,p]}(f',k',b)-\delta^{i}_{[d,p]}(f',k',b)\le\mathbf{M}-\mathbf{M}*\delta^{i}_{[d,p]}(f,k,b), \tag{9}$$

1) Objective: The objective function maximizes the system-wide cache reward. A higher cache reward value correspond 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.

2) Constraints: Binary variables are defined in Constraints (1) and (2). The remaining constraints relate to the *Cache Capacity*, *Caching Status Indicator*, and *Popularity*, as follows.

• The *Cache Capacity* defined in Constraint (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)-(7). δ is an aggregation of cache placement decisions x. Constraints (4) and (5) 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 (6) gives the upper bound. When both $\delta^{i-1}_{[d,p]}$ and $x^{i}_{[d,p]}$ are 0, Constraint (6) will enforce $\delta^{i}_{[d,p]}$ to be assigned 0 since video content is not cached yet along $[d,p]$. Constraints (7) and (8) 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.

• *Popularity* contributes via Constraint (9). The catering to popularity is known to improve the performance of caching schemes [7], [14], [19]. Constraint (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-$\mathbf{M}$' approach [34] to ensure caching order, where $\mathbf{M}$ is any large positive constant number.

The popularity Constraint (9) benefits from additional remarks. A ranking table is assumed to exist for each forwarding path; in our own implementation (described in Section VI) 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 (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 (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 jth downstream router closer to consumers ($1\le j\le i$).

TABLE II

AN EXAMPLE OF AVERAGE CUMULATIVE DELAY OF 4-SECOND SEGMENTS BY HOP DISTANCE, FROM CONSUMER C TO CACHE ROUTER R_i ON FORWARDING PATH. GREYED CELLS DELINEATE IN-TIME DELIVERY FOR 4s SEGMENTS ENCODED AT BITRATE B_i

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

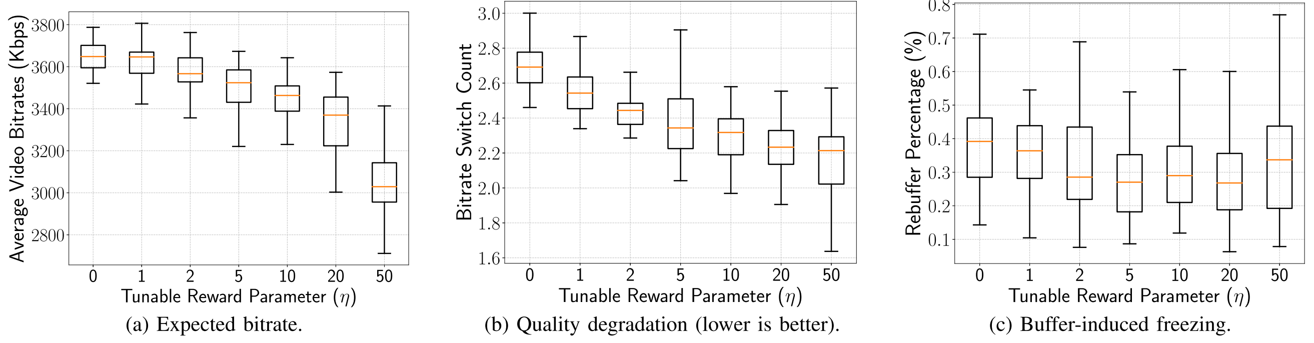
B. Cache Reward Function

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 γ. The design of γ 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 example in Table II, with entries captured from a simulation. Each cell is populated with the empirical average transfer delay for a 4-second video segment delivered to consumer C from any cache R_i 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 II, 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 II says that consumers would maintain or even increase their selected bitrate when content retrieved from R_2 or R_1. Conversely, that

Authorized licensed use limited to: Queen's University. Downloaded on April 28,2021 at 10:45:59 UTC from IEEE Xplore. Restrictions apply.



(a) Expected bitrate. (b) Quality degradation (lower is better). (c) Buffer-induced freezing.

Fig. 3. The impact of tunable cache reward η on users' QoE.

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 1.

The consumer-side adaptation and its interaction with in-network caches are then captured by reward function γ that we first introduced in [11], where the numerical reward values were shown to have a high correlation with traditional consumer-side 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 ith 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 II). We further use RB^i to denote $RB^i_{[d,p]}$, where forwarding path $[d, p]$ is implicit.

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

$$\gamma(RB^i, b) = \begin{cases} \mu(b), & \textbf{if } \; b = RB^i \\ \mu(b^{\uparrow}) * \beta(b) + \mu(b) * (1 - \beta(b)), & \textbf{if } \; b < RB^i \\ \mu(RB^i), & \textbf{if } \; (b > RB^i) \wedge (RB^i \geq b_1) \\ \mu(b_1), & \textbf{otherwise.} \end{cases} \tag{10}$$

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 γ 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. γ 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 $\gamma(RB^i, b) = \mu(b^{\uparrow})$ 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 users' QoE under a variable β, to highlight the trade-off under different design choices.

C. 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

$$\beta(b) = \frac{1}{\eta + rank(b)}, \tag{11}$$

where $rank(b)$ is the order of b in the sorted bitrate set (from smallest to largest). 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 γ rewards and consumer QoE implemented the inverse of $rank(b)$, alone [11]. 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 3. These plots show the impact of β on various measures of users'

Authorized licensed use limited to: Queen's University. Downloaded on April 28,2021 at 10:45:59 UTC from IEEE Xplore. Restrictions apply.

QoE for a range of η values. Performance metric definitions, as well as further experimental design details, are provided in Section VI. A smaller η value favours $\mu(b^{\uparrow})$, the reward that emphasizes higher quality. This can be seen in Figure 3a, where consumers receive the highest quality when $\eta = 0$ and diminishing quality as η values increase. Conversely, Figure 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 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 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 VI.

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.

V. *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.

A. 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, 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, 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.

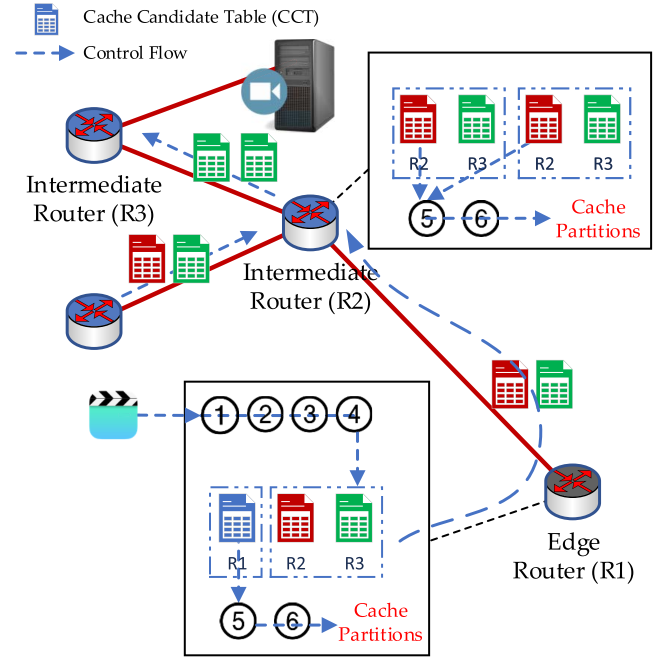


Fig. 4. Execution of *RippleFinder*. Edge router R_1 would create *Cache Candidate Tables* (CCTs) for R_1, R_2 and R_3. CCT for R_1 is processed immediately on R_1. CCTs for R_2 and R_3 are delivered upstream. The intermediate router R_2 would intercept all CCTs for R_2 (the icon in red color), and forward CCTs for R_3.

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.

B. *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 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,

$$\mathbb{U}_d(f, k, b) = \mu(b) * \theta_d(f, k, b). \tag{12}$$

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

Authorized licensed use limited to: Queen's University. Downloaded on April 28,2021 at 10:45:59 UTC from IEEE Xplore. Restrictions apply.

Video Request Statistics

(B3 > B2 > B1)

Name	Utility
/Video1/B3	100
/Video2/B3	70
/Video3/B3	40
• • •	
/Video10/B3	15
• • •	

Ranking Table (B3)

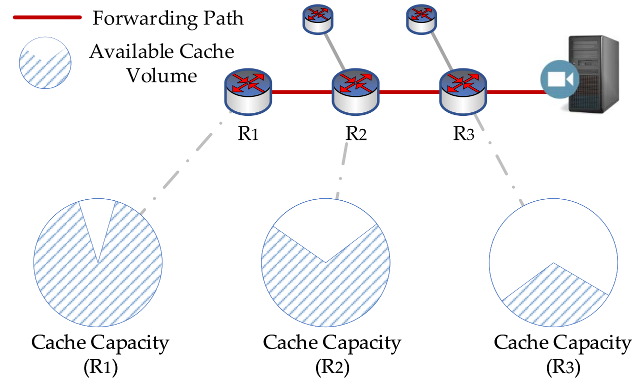
Name	Utility
/Video1/B2	80
/Video3/B2	60
• • •	
/Video24/B2	16
/Video25/B2	14
• • •	

Ranking Table (B2)

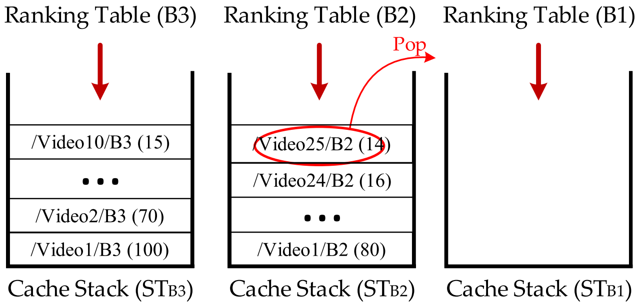
Name	Utility
/Video2/B1	40
/Video4/B1	35
/Video5/B1	30
• • •	
/Video20/B1	20
• • •	

Ranking Table (B1)

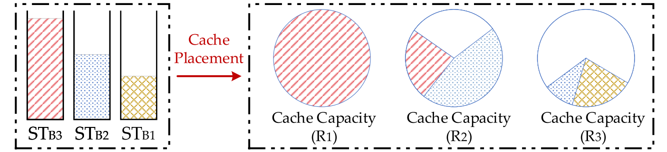
(1) **Ranking Table Construction**. Construct ranking tables for each bitrate (B_1, B_2 and B_3) from video statistics collected by edge router.



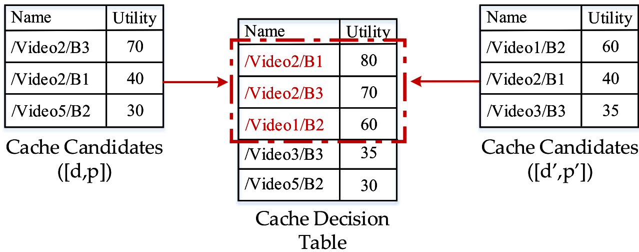
(2) **Cache Capacity Discovery**. The shaded volume of router R_1 , R_2 and R_3 would be used to cache video content delivered along forwarding path. These shaded volume is added together, with a size of $C_{[d,p]}$.



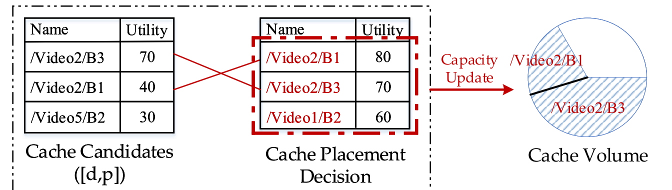
(3) **"Push" and "Pop"**. Video content is pushed into *Cache Stack* by ranking order. After content '*/Video25/B2*' is pushed into ST_{B_2}, the Equation 13 is violated, which triggers 'Pop' operation. Since utility of '*/Video10/B3*' on top of ST_{B_3} is higher than '*/Video25/B2*' on top of ST_{B_2}, video segment '*/Video25/B2*' is popped.



(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}.



(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.



(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.

Fig. 5. *RippleFinder* in execution.

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 5-2, 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 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 entries are pushed into the stacks until the 'stacked' size of the video segments exceeds the total available cache capacity, i.e.

$$\sum_{b \in B} Size(ST_b) > C_{[d,p]}. \tag{13}$$

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 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

Authorized licensed use limited to: Queen's University. Downloaded on April 28,2021 at 10:45:59 UTC from IEEE Xplore. Restrictions apply.

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 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 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 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 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 5-6, a router would cache the three 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.

Upon termination *RippleFinder* identifies the target bitrates for each router. In our implementation, caches are populated opportunistically as content arrives, with content replaced appropriately to prioritize the target bitrate.

C. 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 1. 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 $\mathcal{O}(|\mathbb{P}|BFK \log(FK) + |\mathbb{P}||\mathbb{V}|)$.

Algorithm 1 *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.

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

VI. Performance Results and Insights

We evaluate *RippleClassic* and *RippleFinder* performance via simulation against known caching strategies on the Named Data Networking (NDN) architecture. Results reinforce the broader merits of cache partitioning for adaptive streaming.

Authorized licensed use limited to: Queen's University. Downloaded on April 28,2021 at 10:45:59 UTC from IEEE Xplore. Restrictions apply.

We claim without loss of generality that the merits of *RippleCache* designs and subsequent analyses can be applied on other ICN architectures and cache hierarchies.

A. Simulation Setup and Parameters

The proposed cache partitioning schemes were implemented onto ndnSIM [35], 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

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

Consumer-side adaptation behaviour is simulated via our own implementation of FESTIVE as it is described in [23]. FESTIVE is an adaptation 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 [36]. Consumers initiate a session first by requesting a video file and retrieving video-related meta-data (i.e. the Media Presentation Description (MPD)) from the producer. We assume that all cache nodes have apriori knowledge of MPD files, so that media descriptions are known to the caching algorithm. Each consumer starts a streaming session following a Poisson process, with an average time interval as 300 seconds. Each streaming session consists of requests for video segments in a certain video file, where the requested bitrate is guided by the bitrate adaptation algorithm.

Three additional caching schemes are evaluated alongside our proposed *RippleFinder* and *RippleClassic* for comparison. *Cache Everything Everywhere (CE2)* [14] with Least Recently Used (LRU) is a baseline that commonly appears in literature [14]. *ProbCache* [13] serves as a baseline for probabilistic caching [19]. In addition, a modified version of *ProbCache* guided by *RippleCache* principle is also evaluated, in order to demonstrate that *RippleCache* can facilitate existing caching policies achieving a better QoE. This modified approach is denoted as *ProbCache(p)*, where p is a filtering probability and would be applied, in addition to normal probability from *ProbCache*, when request bitrate is different from *Ripple Bitrate*. As a result, *ProbCache* is also denoted as *ProbCache(1.0)* in this section for consistency. Many filtering probabilities are evaluated and a performance trend appears as we decrease this value. As the performance difference is insignificant with smaller filtering probability, we decide to present the modified *ProbCache* with $p = 0.2$.

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 users' preferred bitrates. The iteration on *RippleClassic* stops once the difference between two consecutive optimization objective values fall below a threshold. *RippleFinder* stops after a pre-defined constant number of iterations, where this number is determined as we observe a relative stable performance on users' QoE.

Two separate networks are implemented for evaluation. A smaller 16-node topology with an extra video producer is adopted that forms a binary tree with a maximum 7-hop distance from the producer to consumers. This allows the binary integer programming from *RippleClassic* to find solutions in 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 our measurement indicates that no consumer has enough bandwidth to retrieve the highest bitrate (8 Mbps) content directly from the producer when all consumers are streaming simultaneously. 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 3 and surrounding discussion that $\eta = 1$ appears to strike a good balance between prioritizing high bitrates and low oscillation.

TABLE III
SIMULATION PARAMETERS

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

A larger 42-node topology randomly generated by BRITE [37] is used to evaluate cache partitioning in a realistic and large-scale system with three producers. The complete list of simulation parameters for both scenarios/topologies are listed in Table III.

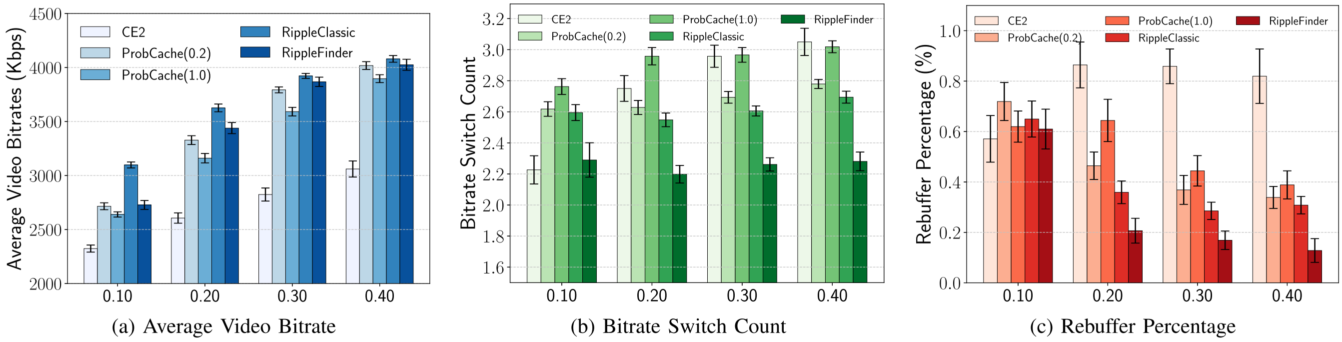
Results are evaluated using standard QoE metrics published by the DASH Industry Forum [15]. From the standard set, we adopt three metrics, *Average Video Quality*, *Bitrate Switch Count* and *Rebuffer Percentage*, as described in their relevant sections. Each set of evaluations is repeated across a range of content store size ratio ω and popularity-skewness parameter α. Each experiment is executed with 30 runs and results are presented at a 95% confidence level.

B. Average Video Quality

Figures 6a and 7a show the *Average Video Quality*, defined as the average video bitrate that consumers request among all video sessions [15]. Measurements indicate that *RippleClassic* and *RippleFinder* performance meet or exceed the best among *CE2* and *ProbCache*. Another Observation is *ProbCache(0.2)* outperforms normal *ProbCache(1.0)* across different cache capacity. This result indicates by restricting the caching areas and only allowing popular content to be cached closer to consumers, *RippleCache* can enhance cache utilization of existing approach.

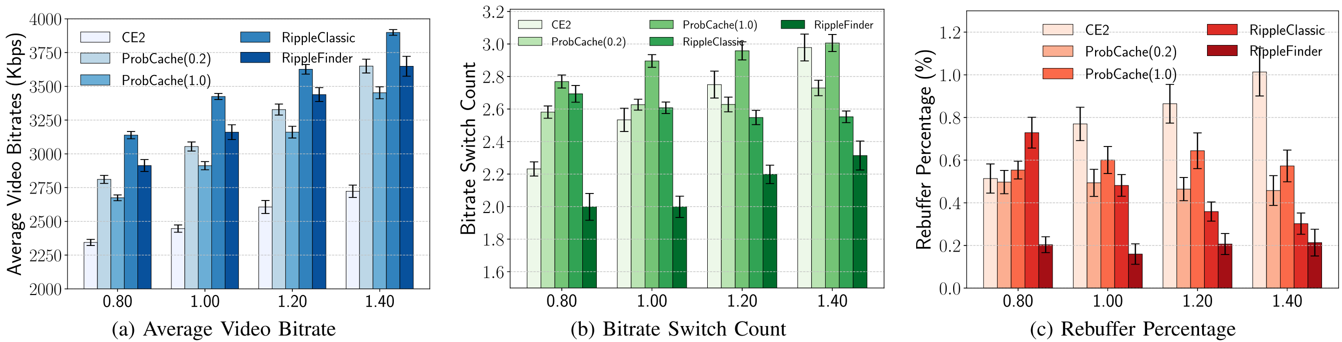
We observe that the gap in performance against the benchmark *RippleClassic* grows proportionally larger as cache

Authorized licensed use limited to: Queen's University. Downloaded on April 28,2021 at 10:45:59 UTC from IEEE Xplore. Restrictions apply.



(a) Average Video Bitrate (b) Bitrate Switch Count (c) Rebuffer Percentage

Fig. 6. Content Store Size Percentage (ω) for 'BIP-tractable' settings.



(a) Average Video Bitrate (b) Bitrate Switch Count (c) Rebuffer Percentage

Fig. 7. Popularity Skewness (α) for 'BIP-tractable' settings.

resources diminish. For example, Figure 6a shows that when the cache capacity ratio is $\omega = 0.4$, *RippleClassic* delivers higher video quality than *ProbCache(0.2)* by 1.5%. When the total cache capacity drops to 0.1, *RippleClassic* delivers an average bitrate 14.1% better than *ProbCache(0.2)*. *RippleFinder* results in a similar performance as *ProbCache(0.2)* across all tested cache capacity. Later observations show that *RippleFinder* magnifies such small differences by substantially reducing bitrate oscillation irrespective of caching resources.

The trend is similar as popularity skewness, shown in Figure 7a. As expected, 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. *RippleClassic* and *RippleFinder* will also 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 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.

C. Bitrate Switch Count

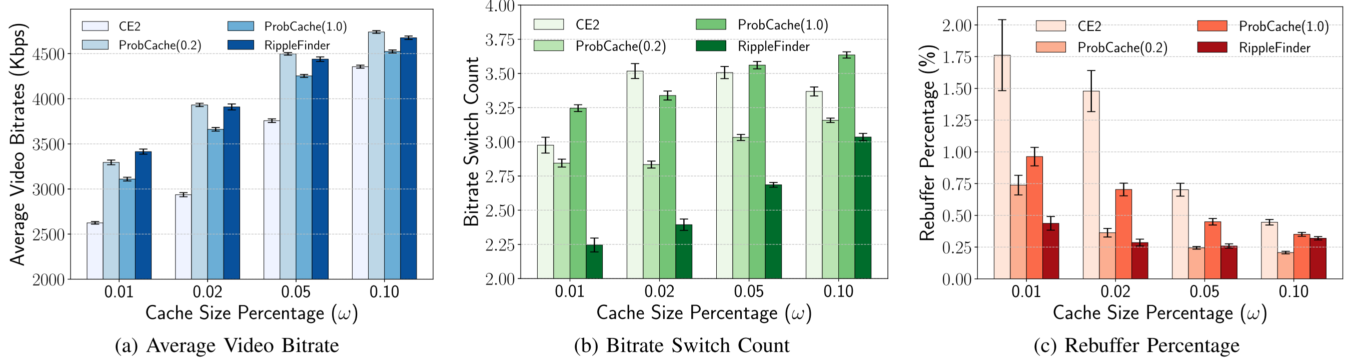
The *Bitrate Switch Count* measures oscillation by recording the number of times that bitrate is switched up and down in a video session when consumers request for a certain file [15]. Results are shown in Figures 6b and 7b, as cache capacity and popularity distribution are made to vary, respectively. In all evaluations, both *RippleClassic* and *RippleFinder* reduce bitrate oscillation when compared with popularity-based *ProbCache(1.0)*. It is also noticeable that *ProbCache(0.2)* effectively reduces the bitrate oscillation from *ProbCache(1.0)* and achieves a similar performance as *RippleClassic*, thanks to the adoption of *RippleCache* principle. 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 *CE2* show that this comes at the cost of video quality. The lower video quality for *CE2* also explains the low degrees of oscillation. Coupled with Figures 6a and 7a, we see that our *RippleCache*-guided schemes are seen to reduce oscillation while sustaining the highest levels of video quality.

Observations in Figures 6b and 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.

D. Rebuffer Percentage

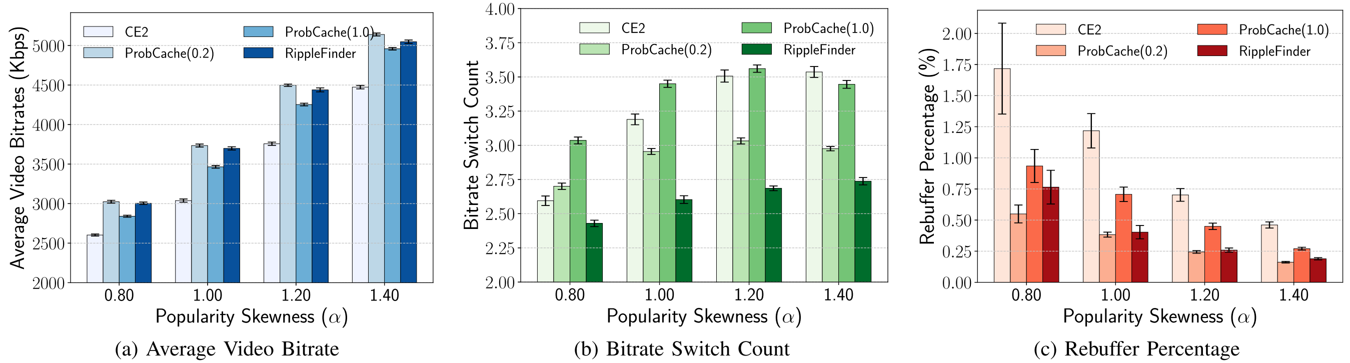
Short-term variations in network and system conditions can adversely affect playback before bitrate adaptations are

Authorized licensed use limited to: Queen's University. Downloaded on April 28,2021 at 10:45:59 UTC from IEEE Xplore. Restrictions apply.



(a) Average Video Bitrate (b) Bitrate Switch Count (c) Rebuffer Percentage

Fig. 8. Content Store Size Percentage (ω) for 'Large-scale' settings.



(a) Average Video Bitrate (b) Bitrate Switch Count (c) Rebuffer Percentage

Fig. 9. Popularity Skewness (α) for 'Large-scale' settings.

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 [15]. Results are shown in Figures 6c and 7c.

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 Figures 6c and 7c, 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 *CE2* or *ProbCache(1.0)*.

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 6 and 7 imply that the relationship between these two metrics is more subtle. In support of intuition, for example, at cache capacity $\alpha = 0.8$ *RippleClassic* delivers the highest video quality, but causes the most playback freezing. The perceived relationship between metrics is broken when when comparing *RippleFinder* at the same $\alpha = 0.8$. Here, *RippleFinder* delivers the higher video quality matching *ProbCache(0.2)* but maintains the least rebuffer ratio.

Collectively these observations reinforce that, in distributed multimedia caching systems, cache hits have value only if their occurrence is useful to the consumer.

E. Evaluation on a Realistic Topology

We evaluate over a large 42-node autonomous system (AS) topology generated using BRITE [37]. 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 III.

Results in Figures 8 and 9 show that *RippleFinder* performance trends are similar to previous observations. In particular, although another *RippleCache*-guided approach *ProbCache(0.2)* meets or exceeds *RippleFinder* in terms of video quality and rebuffering, *RippleFinder* compensates the overall QoE with significantly reduced bitrate oscillation, especially at low ω or low α.

This consistent performance of *RippleCache*-guided design across topologies is also noteworthy. When compared to trends of the smaller network captured in Figures 6 and 7, the performance of competing schemes appears to be affected by size and topology. For example, Figures 6c and 7c show *CE2* with higher rates of rebuffering at large ω or large α.

Authorized licensed use limited to: Queen's University. Downloaded on April 28,2021 at 10:45:59 UTC from IEEE Xplore. Restrictions apply.

However, in the representative topology *CE2* performance is inverted, where rebuffering rate decrease with cache capacity and popularity skewness, as can be seen in Figures 8c and 9c. 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.

F. Discussion of Results

Throughout our evaluations we notice the ability of popularity-based caching schemes (such as *ProbCache*) in terms of delivered video quality and playback freezing. Looking ahead, the robustness of popularity-based approaches suggests that performance gains promised by ICNs specifically, and caching hierarchies generally, may be dependent on their ability to exploit content characteristics.

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- rather than cache-limited.

VII. Conclusion

In this article, 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 two cache placement schemes, *RippleClassic* and *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 explorations yield the following conclusions: 1) The operational mandate of bitrate adaptation algorithms significantly impacts in-network caching schemes, thus caching must seamlessly cooperate with adaptation. Schemes that execute on snapshots are ill-suited for direct application to adaptive streaming applications because the absence of cooperation leads to bitrate oscillation. 2) Bitrate oscillations can be reduced by concatenating caches along a forwarding path into a *cache path*. By aggregating isolated caches along the forwarding path video content can be placed according to the distance from consumers that reduces the likelihood of adaptation, thereby avoiding oscillation. 3) Such a broad-view of caching entities maintains high-quality content delivery while diminishing playback freezing. 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 *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 *Proc. Workshop Internet QoE*, 2016, pp. 55–60.

[3] A. Mansy, M. Fayed, and M. Ammar, "Network-layer fairness for adaptive video streams," in *Proc. IFIP Netw. Conf. (IFIP Netw.)*, May 2015, pp. 1–9.

[4] T. Flach *et al.*, "An Internet-wide analysis of traffic policing," in *Proc. Conf. ACM SIGCOMM Conf. (SIGCOMM)*, 2016, pp. 468–482.

[5] MPEG. *DASH*. Accessed: Dec. 16, 2020. [Online]. Available: https://dashif.org/guidelines/

[6] *Cisco Visual Networking Index: Forecast and Methodology, 2016-2021*, Cisco, San Jose, CA, USA, 2017.

[7] W. Li, S. M. A. Oteafy, and H. S. Hassanein, "Rate-selective caching for adaptive streaming over information-centric networks," *IEEE Trans. Comput.*, vol. 66, no. 9, pp. 1613–1628, Sep. 2017.

[8] Z. Ye, F. De Pellegrini, R. El-Azouzi, L. Maggi, and T. Jimenez, "Quality-aware DASH video caching schemes at mobile edge," in *Proc. 29th Int. Teletraffic Congr. (ITC)*, Sep. 2017, pp. 205–213.

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

[10] D. H. Lee, C. Dovrolis, and A. C. Begen, "Caching in HTTP adaptive streaming: Friend or foe?" in *Proc. Netw. Operating Syst. Support Digit. Audio Video Workshop (NOSSDAV)*, 2013, pp. 31–36.

[11] W. Li, M. Fayed, S. M. A. Oteafy, and H. S. Hassanein, "A cache-level quality of experience metric to characterize ICNs for adaptive streaming," *IEEE Commun. Lett.*, vol. 23, no. 2, pp. 262–265, Feb. 2019.

[12] W. Li, S. M. A. Oteafy, M. Fayed, and H. S. Hassanein, "Bitrate adaptation-aware cache partitioning for video streaming over information-centric networks," in *Proc. IEEE 43rd Conf. Local Comput. Netw. (LCN)*, Oct. 2018, pp. 401–408.

[13] I. Psaras, W. K. Chai, and G. Pavlou, "Probabilistic in-network caching for information-centric networks," in *Proc. 2nd Ed. ICN Workshop Inf.-Centric Netw. (ICN)*, 2012, pp. 55–60.

[14] M. Zhang, H. Luo, and H. Zhang, "A survey of caching mechanisms in information-centric networking," *IEEE Commun. Surveys Tuts.*, vol. 17, no. 3, pp. 1473–1499, 3rd Quart., 2015.

[15] *Dash Industry Forum. Dash-If Position Paper: Proposed QOE Media Metrics Standardization for Segmented Media Playback*. Accessed: Oct. 25, 2018. [Online]. Available: https://dashif.org/docs/ProposedMediaMetricsforSegmentedMediaDelivery-r12.pdf

[16] F. Khandaker, S. Oteafy, H. S. Hassanein, and H. Farahat, "A functional taxonomy of caching schemes: Towards guided designs in information-centric networks," *Comput. Netw.*, vol. 165, Dec. 2019, Art. no. 106937.

[17] A. Ioannou and S. Weber, "A survey of caching policies and forwarding mechanisms in information-centric networking," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 4, pp. 2847–2886, 4th Quart., 2016.

[18] J. Li *et al.*, "Popularity-driven coordinated caching in named data networking," in *Proc. 8th ACM/IEEE Symp. Archit. Netw. Commun. Syst. (ANCS)*, 2012, pp. 15–26.

[19] 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 *Proc. IEEE INFOCOM Workshops*, Mar. 2012, pp. 316–321.

[20] J. Kua, G. Armitage, and P. Branch, "A survey of rate adaptation techniques for dynamic adaptive streaming over HTTP," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1842–1866, 3rd Quart., 2017.

Authorized licensed use limited to: Queen's University. Downloaded on April 28,2021 at 10:45:59 UTC from IEEE Xplore. Restrictions apply.

[21] P. Georgopoulos, Y. Elkhatib, M. Broadbent, M. Mu, and N. Race, "Towards network-wide QoE fairness using openflow-assisted adaptive video streaming," in *Proc. ACM SIGCOMM Workshop Future Hum.-Centric Multimedia Netw. (FhMN)*, 2013, pp. 15–20.

[22] Z. Li *et al.*, "Probe and adapt: Rate adaptation for HTTP video streaming at scale," *IEEE J. Sel. Areas Commun.*, vol. 32, no. 4, pp. 719–733, Apr. 2014.

[23] J. Jiang, V. Sekar, and H. Zhang, "Improving fairness, efficiency, and stability in HTTP-based adaptive video streaming with FESTIVE," in *Proc. 8th Int. Conf. Emerg. Netw. Exp. Technol. (CoNEXT)*, 2012, pp. 97–108.

[24] 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 Comput. Commun. Rev.*, vol. 44, no. 4, pp. 187–198, 2015.

[25] G. Tian and Y. Liu, "Towards agile and smooth video adaptation in dynamic HTTP streaming," in *Proc. 8th Int. Conf. Emerg. Netw. Exp. Technol. (CoNEXT)*, 2012, pp. 109–120.

[26] R. Jia, Z. Liu, X. Wang, X. Gan, X. Wang, and J. J. Xu, "Modeling dynamic adaptive streaming over information-centric networking," *IEEE Access*, vol. 4, pp. 8362–8374, 2016.

[27] C. Kreuzberger, B. Rainer, and H. Hellwagner, "Modelling the impact of caching and popularity on concurrent adaptive multimedia streams in information-centric networks," in *Proc. IEEE Int. Conf. Multimedia Expo Workshops (ICMEW)*, Jun. 2015, pp. 1–6.

[28] Y. Liu *et al.*, "Dynamic adaptive streaming over CCN: A caching and overhead analysis," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2013, pp. 3629–3633.

[29] W. Li, S. M. A. Oteafy, and H. S. Hassanein, "On the performance of adaptive video caching over information-centric networks," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2017, pp. 1–6.

[30] Y. Jin, Y. Wen, and C. Westphal, "Towards joint resource allocation and routing to optimize video distribution over future Internet," in *Proc. IFIP Netw. Conf. (IFIP Netw.)*, May 2015, pp. 1–9.

[31] A. Araldo, F. Martignon, and D. Rossi, "Representation selection problem: Optimizing video delivery through caching," in *Proc. IFIP Netw. Conf. (IFIP Netw.) Workshops*, May 2016, pp. 323–331.

[32] A. Singla, C.-Y. Hong, L. Popa, and P. B. Godfrey, "Jellyfish: Networking data centers randomly," in *Proc. USENIX Symp. Netw. Syst. Design Implement. (NSDI)*. San Jose, CA, USA: USENIX, 2012, pp. 225–238.

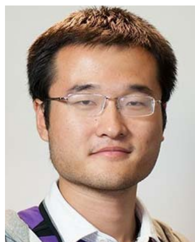
[33] *Gurobi Optimizer*, L. Gurobi Optimization. Accessed: Dec. 16, 2020. [Online]. Available: https://www.gurobi.com/

[34] I. Griva, S. G. Nash, and A. Sofer, *Linear and Nonlinear Optimization*, vol. 108. Philadelphia, PA, USA: SIAM, 2009.

[35] A. Alexander, I. Moiseenko, and L. Zhang, "ndnSIM: NDN simulator for NS-3," Univ. California, Los Angeles, CA, USA, Tech. Rep. NDN-0005, 2012.

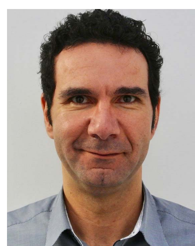
[36] *YouTube Help*. Accessed: Sep. 20, 2018. [Online]. Available: https://support.google.com/youtube/answer/1722171?hl=en

[37] A. Medina, A. Lakhina, I. Matta, and J. Byers, "BRITE: An approach to universal topology generation," in *Proc. 9th Int. Symp. Modeling, Anal. Simulation Comput. Telecommun. Syst. (MASCOTS)*, 2001, pp. 346–353.



Wenjie Li received the B.S. degree in computer science and engineering from Southeast University, Nanjing, Jiangsu, China, in 2013, and the M.Sc. and Ph.D. degrees from the School of Computing, Queen's University, in 2015 and 2019, respectively. His research interests include future network architecture, ubiquitous caching in named data networking, and efficient video streaming over hierarchical cache networks.

Sharief M. A. Oteafy (Senior Member, IEEE) received the Ph.D. degree from Queen's University, Canada, with a focus on adaptive resource management in the IoT. He is currently an Assistant Professor with the School of Computing, DePaul University, USA. He also co-leads the team working on the Tactile Internet Architecture, under the development of the IEEE P1918.1 Standard Working Group. He has coauthored a book on dynamic wireless sensor networks, presented more than 60 publications, and delivered multiple IEEE tutorials on the IoT and BSD. His current research interests include developing the tactile Internet, caching in information centric networks, and managing the proliferation of big sensed data (BSD). He co-chaired a number of IEEE symposia and workshops in conjunction with IEEE ICC and IEEE LCN. He is currently an Associate Technical Editor of *IEEE Communications Magazine* and IEEE ACCESS and on the editorial board of *Internet Technology Letters* (Wiley).



Marwan Fayed (Senior Member, IEEE) received the M.A. degree from Boston University and the Ph.D. degree from the University of Ottawa. He is currently the Research Lead of Cloudflare, Inc., and an Associate Member of faculty with the University of St Andrews, U.K. He is also the Co-Founder and the former Director of rural Internet backhaul provider, HUBS cic., a recipient of the EU Commission's Broadband Award for Future-Proof and Quality of Service in 2016. His current research interests include network, transport, and measurement in next-generation edge systems. He has coauthored multiple publications with IEEE, ACM, and IFIP. He is a Senior Member of the ACM. He was a recipient of the IEEE Best Paper Award. He was appointed as the Theme Leader for networking research in Scotland, from 2014 to 2016.

Hossam S. Hassanein (Fellow, IEEE) received the Ph.D. degree in computing science from the University of Alberta, Canada, in 1990. He is currently a leading authority in the areas of broadband, wireless and mobile networks architecture, protocols, control, and performance evaluation. His record spans more than 500 publications in journals, conferences, and book chapters, in addition to numerous keynotes and plenary talks at flagship venues. He is also the Founder and the Director of the Telecommunications Research Laboratory, School of Computing, Queen's University, Kingston, ON, Canada, with extensive international academic and industrial collaborations. He is an IEEE Communications Society Distinguished Speaker and the Past Chair of the IEEE Communication Society Technical Committee on AHSN. He has received several recognitions and best papers awards.

Authorized licensed use limited to: Queen's University. Downloaded on April 28,2021 at 10:45:59 UTC from IEEE Xplore. Restrictions apply.