Congestion-Aware Edge Caching for Adaptive Video Streaming in Information-Centric Networks

Yu-Ting Yu†, Francesco Bronzino§, Ruolin Fan†, Cedric Westphal‡, Mario Gerla†

†Computer Science Department, University of California, Los Angeles, CA, USA
§WINLAB, ECE Department, Rutgers University, New Brunswick, NJ, USA
‡Innovation Center, Huawei Technology, Santa Clara, CA, USA
‡Computer Engineering Deptartment, University of California, Santa Cruz, CA, USA
Email: †{yutingyu,ruolinfan,gerla}@cs.ucla.edu, §bronzino@winlab.rutgers.edu, ‡cwestphal@huawei.com

***Abstract*—This paper proposes a network-aware resource management scheme that improves the quality of experience (QoE) for adaptive video streaming in CDNs and Information-Centric Networks (ICN) in general, and Dynamic Adaptive Streaming over HTTP (DASH) in particular. By utilizing the DASH manifest, the network (by way of a logically centralized controller) computes the available link resources and schedules the chunk dissemination to edge caches ahead of the end-user's requests. Our approach is optimized for multi-rate DASH videos. We implemented our resource management scheme, and demonstrated that in the scenario when network conditions evolve quickly, our approach can maintain smooth high quality playback. We show on actual video server data and in our own simulation environment that a significant reduction in peak bandwidth of 20% can be achieved using our approach.**

I. Introduction

The consumption of video streams exhibits strong daily patterns, with a significant peak during "prime time" hours. From a network operator's perspective, not only video streaming will consume a lot of network resources, it will also require over-provisioning the network for a peak usage that is much higher than the average, resulting in a lot of unused capacity for most of the time.

Video streams using Dynamic Adaptive Streaming over HTTP (DASH) [30] or equivalent (Apple HLS, Adobe HDS, etc) exhibit some relatively specific properties. (i) Video streams are long lived, ranging from a few minutes for some YouTube clips to over an hour for some Netflix movies; (ii) Video streams are typically described in a manifest at the onset of the connection, therefore it is possible to know the semantics of the stream ahead of time; (iii) Video streams are predictable in the sense that the sequence of packets is predetermined by the video stream's description and the network conditions.

Many view a huge volume of traffic concentrated over a relatively short period of the day as a problem. We hope to demonstrate the properties of the video demands can be leveraged to our advantage. We claim that it is possible to time-shift a significant fraction of the traffic on the network during prime-time by pre-fetching the video streams which would be downloaded from the server during the peak hour to a server at the edge of the network as soon as the stream starts. The marginal cost for the network operator to pre-fetch traffic is close to zero if it is using empty capacity. If pre-fetching however adds traffic onto the network during period of congestion, it makes things worse. Therefore, we propose to monitor the network congestion, and to make the pre-fetching of traffic conditional on the amount of traffic already in the network.

For this approach to be feasible, the network layer needs to be aware of the user's consumption. Information-Centric Networks (ICN)[12][6][14] allow two significant changes for dynamic video streaming:

- It offers an opportunity to re-name content according to logical semantics, so that one can identify the proper rate and time segment from the naming structure. This allows to infer the names of future segments without necessarily having access to the manifest of the video stream.
- The file is explicitly requested using this name, letting the network between a client and a server easily identify that a series of packets belong to a video transfer.

These two changes lead to the following observation: in an ICN, it is easier for the edge network to infer what a user is streaming, and to derive what the user will request if he/she keeps watching the video stream. The network has also a view over the available resources, be it either available bandwidth towards the client and the server, or available storage/cache resource distributed within the network.

Combining the knowledge of the user's demands with that of the network resource, the network can therefore schedule the requests for video streams in order to maximize its incentive. One obvious first step for instance is for the network to decide whether or not to prefetch a video stream that is not currently cached within the network. The decision is based on the current network conditions, the expected variations of the traffic (based for instance on daily patterns), the available cache space, etc.

One important difference from the ICN architecture when compared with an overlay CDN, is that a network node has access to both the network congestion information (which an operator may be reluctant to share with an overlay) and to the clients' stream requests, using the ICN naming semantics. Therefore, an ICN node can populate its cache with the clients' demands using the network information. A video server in an overlay CDN knows the user's requests, but can only monitor the link quality between different CDN nodes through probing the paths (using a service like Conviva), and therefore can place the content in a different CDN cache in a way which impacts the network.

Another significant difference is the elasticity of the resources in an ICN. A typical ICN architecture assumes that many nodes in the network are able to cache content. Therefore, if the network is lightly congested, and assuming the video server can scale up with the demands, an ICN network can harness multiple caches to pre-fetch a video stream fast, and have a high speed up in the stream download, so that the video stream does not impact future conditions beyond a short horizon.

The contribution of this paper is to:

- study on data sets the potential gain of pre-fetching for video streaming during peak congestion;
- present an architecture for pre-fetching for video streams which combines network monitoring with ICN semantics to populate a cache ahead of the need of the video client;
- implement our architecture in a test-bed, with legacy video server and clients, demonstrating the feasibility of having the network take an active role in the video stream independently of the end points;
- and investigate the benefits of ICN cache video prefetching through experimental results.

We present our results in the context of ICN, but they could be translated into a CDN scenario provided that the CDN gateway has some interface to discover the network congestion in between CDN repositories.

The organization of this paper is as follows. In section II, we explain the benefits of prefetching using a motivating example and consider actual data sets. In section III, we briefly discuss the related work. In section IV, we introduce the resource allocation method for cache prefetching. The experiment results are carried out in section V. Finally, we conclude in section VI.

II. Motivation and Challenges

A. Prefetching Benefit

It is well known that video streaming has a lot of daily variability with traffic increasing dramatically during prime time hours. Prefetching allows to shift some of this traffic in time so as to reduce the peak bandwidth. Therefore it has an obvious applicability to the network operator.

Looking at data sets from YouTube observed at a European CDN server [7] and from aDailyMotion server [3], which we reproduce on Figure 1 and Figure 2 respectively, we see that the load increases dramatically during peak hour.

If the load were to reach a long plateau, there would be no benefit from prefetching. Shifting a video in time would only increase the load during part of the plateau, increasing the overall server peak. However, as seen from the figures, we see that the load exhibit a very spiky burstiness, which opens

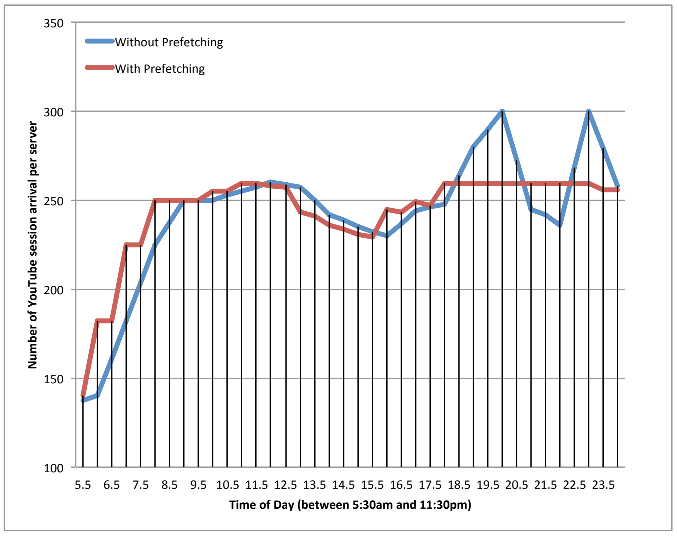


Figure 1: Daily variation of the traffic for the YouTube service at a European CDN server from [7].

up the door for smoothing the traffic and reducing the peak value of the load.

We can also compute the theoretic gain from prefetching, assuming a video stream length of 60 minutes (Netflix streams account for 30% of the Internet traffic during prime time and are long lived flows). This is plotted on both graph by the line which flattens up during the peak hour. With prefetching, the peak bandwidth usage saving at the busy hours (evening 3pm to 8pm) for both data sets is roughly in the 15% to 20% range over the different days. This means that even during a congested time, there is a significant gain from shifting traffic in time.

This means the network operator can possibly save the operating costs incurred by the extra usage without prefetching, or use the residual capacity to serve other types of traffic. From a user's point of view, the bandwidth can be used to improve their viewing experience, that is, higher quality video can be streamed with the available capacity.

We note that the popularity of the files is irrelevant to the study here: popular files will be served locally from the cache, and are not accounted in the load at the server. What we consider is: if a file is requested from the server, can the network shift the delivery of this file forward in time to a local cache, so that if the server load increases later on, this file will no longer impact the server.

B. Challenges

There are a few technical challenges to consider. First, there must be a way to identify the video contents with different rates so that the cache may prefetch higher rates in advance. This implies that the network must be DASH-aware, and the network, to some extent, must maintain the states of the current viewing. Second, a resource allocation mechanism is needed, to combine the capacity estimation and scheduling approaches,

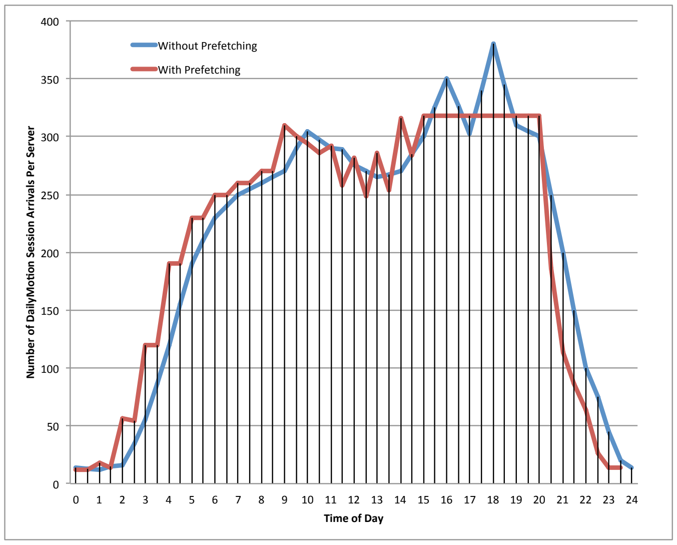


Figure 2: Daily variation of the traffic for the DailyMotion service from [3].

to decide when and which segments to prefetch at any given time. This links to the prediction of user behaviors.

Finally, the network's prefetching must not add to the congestion. There is little cost if un-used capacity is used to get the file locally from the network operator nor the server point of view. So there is little penalty for the network operator if the viewer ends up not watching the pre-fetched file. However, the network must never add to the traffic during the peak congestion.

III. Related Work

Information-Centric Network (ICN) architectures [12][6][14] have been proposed recently to allow the network to be aware of content semantics. We take full advantage of the ICN content abstractions in this paper.

Dynamic Adaptive Streaming over HTTP (DASH) [30] is a standard for video streaming that has been widely deployed by many major video streaming services such as Netflix, Hulu, and Youtube. In DASH, the video content is broken into a sequence of small segments, each containing a short playback time of the video. The DASH client first requests a manifest (the MPD) describing the video and available quality levels, and then adaptively requests the segment with proper video quality based on the current network conditions reflected during the streaming process.

DASH over ICN has been attracting some attention in the IRTF [16] and [9] examines the interaction of DASH with ICN. It observes potential issues, while we identify here synergies. [17] targets HTTP adaptive streaming (HAS) in Content-Centric Networking (CCN) [12] for Scalable Video Coding.

[23] has looked at how to predict the requests from the users by looking at their social interactions. Here, we use the natural predictability offered by video streaming.

[1] describes a web prefetching module running on the CDN main node (controller) which downloads web contents that will be requested in the future to the LAN CDN surrogates. Our work is different as the time domain is much smaller and the prefetching utilizes the bandwidth more dynamically. In addition, [1] does not specifically consider video contents.

[2] analyzes the potential benefits of CDN augmentation strategies can offer for Internet video workloads using a dataset of 30 million VOD and live sessions. It has also been observed in [27][8] that fractions of viewers typically watch only the first 10 minutes of video, around 4.5% of users are serial early quitters, and 16.6% of users consistently watch video to completion. This suggests that a user based prefetching policy should be a natural extension for our work.

[19] proposes a Network-Friendly DASH (NF-DASH) architecture for coordinating peer-assisted CDNs operated by ISPs and the traditional CDNs. [20] analyzes the waiting time and network utilization with service prioritization considering both on-demand fetching/caching and prefetching in a P2P-assisted video streaming system. [21] formulates the CDN assignment as an optimization problem targeting minimizing the cost based on the QoE constraints to CDN servers at different locations. [22] shows by measurement the shortcomings of today's video delivery infrastructure and proposes a control plane in the network for video distribution for global resource optimization.

Hybrid P2P-CDN video streaming enhancement has also been considered. That is, serving content from dedicated CDN servers using P2P technology.[11][10] and telco-CDN (CDNs operated by telecommunication companies, enabling users to reach CDN caches that are closer) federation are two emerging strategies. Telco-CDN federation can reduce the provisioning cost by 95%. Using P2P can lead up to 87.5% bandwidth savings for the CDN during peak access hours.

Proxy-assisted caching and prefetching has been widely studied in the literature. Some approaches consider the quality of the connections [13] and the usage of the client buffers [24]. Approaches for transcoding proxy caching are also presented in [18][28] in which the proxy caches different versions of the content to handle the heterogeneous user requirements. Prefix caching [29] caches only the frames at the beginning to minimize the average initial delay. Exponential segmentation [26] divides the video object such that the beginning of a video is cached as smaller segment. Lazy segmentation approach [5] determines the segment length according to the user access record at the late time.

Prefetching is common for video content to reduce the pause time during playback and service delay. [25] was the first to apply prefetching of web delivery to reduce latency. In this scheme, a server predicts the links that will be requested and prefetches accordingly. The use of proxy prefetching in VoD systems has been intensively studied.

[5] discusses a segment-based proxy pre-fetching for streaming delivery. [15] evaluates the 1-ahead, n-ahead, and priority-based segment prefetching. The results show that if the bottleneck link is between client and proxy, all prefetching

schemes achieve high cache hit rate after 2-3 client requesting a video. On the other hand, if the bottleneck link is between proxy and server, no prefetching helps. Our approach considers the link between the cache and the server and makes a prefetching decision accordingly.

IV. DASH-Aware Video Stream Prefetching

We propose a video prefetching approach that takes both video session context and network condition context into account.

In this paper, we use a (logically) centralized content controller, as in [4], to locate content, manage the cache, and monitor network congestion. The controller realizes the management and policy at the content level in ICN. As in [4], the URLs are used as data names. A controller is used to assign content to caches on the path from the client to server and to maintain the content state and location. In our implementation built on top of TCP/IP in SectionV, a proxy is built to maintain the TCP connection from the client so as to provide late binding of the connection with the content location and allow the client to dynamically switch between the cache and the server. However, this late binding is native to most ICN architectures. The proxy also interacts with the controller to perform content routing.

The controller monitors the network conditions, and makes a pre-fetching decision based upon the congestion in the network and the requests from the users.

The basic idea of our approach is as follows. The network controller collects statistics of current network capacity usage and ongoing DASH video sessions. Based on the collected information, the network controller assigns video segment prefetching task to the edge caches. One or several caches can be used to pre-fetch, depending on the network conditions and the speed with which one cache can get the data.

The task assignment is performed for a given period of time, which is defined as a *round*. Periodically, the controller uses its knowledge collected from the previous rounds such as the history of DASH video segment requests and the bandwidth usage to decide the segments to be prefetched by caches in the next round.

A. Protocol

The protocol message exchange is shown in Figure 3. Following [4], a proxy is used to dispatch HTTP requests. To dispatch the HTTP requests to cache or server, the proxy queries the controller to know whether the content is cached or prefetched. At this point, the controller, monitoring HTTP DASH requests, collects and maintains the video segments each client is requesting by simply parsing the query. The requests are then forwarded by the proxy to the server or cache as appropriate.

The controller monitors the current network bandwidth and data rates as the data flows back to the client. In order to retrieve video information, the controller may obtain MPD information by explicitly requesting the MPD from the server. Segment prefetching scheduling happens periodically. To schedule prefetching tasks, the controller computes the available capacity for prefetching (section IV-B) according to collected network context and projected future requests and compiles a list of segment identifiers to prefetch for caches (section IV-C). The lists are then sent to the caches. Upon receiving a segment identifier list, the cache initiates DASH HTTP requests for each specified segment to the server and fetch segments listed one by one. Whenever a segment is received by cache, a notification is sent to the controller so that the controller can maintain the sources of contents. Note that caches may support lower video rates by transcoding the retrieved segments to lower rate ones offline.

In short, the controller performs two tasks at the beginning of each round: 1. **Residual capacity estimation**: estimate capacity for the next round and decides how much spare capacity can be used by prefetching. 2. **Segment scheduling**: compile a list of video segments to prefetch based on the video information, request history and the existing contents resided in caches.

B. Residual capacity estimation

Suppose C_{total} is the current total capacity of the network, and B_{client} is the amount of capacity that is anticipated to be consumed by DASH clients, and W is the length of a round, the bandwidth available to prefetching at a cache, B_A, can be defined as follows.

$$B_A = max(C_{total}\delta - \frac{B_{client}}{W}, 0) \tag{1}$$

where δ is a pre-defined threshold preserved for traffic bursts.

To estimate the capacity consumed by DASH clients, consider B_{new} as the amount of bandwidth consumed by newly arrived video sessions in the next round, and B_{old} as the amount of bandwidth consumed by current video sessions in the next round, B_{client} can be represented as

$$B_{client} = B_{new} + B_{old} \tag{2}$$

Assuming a Poisson arrival process of video streams, the controller can formulate B_{new} as a function of video initial request arrival rate. However, the computation of B_{old} requires knowledge about the DASH video and the cached contents, as the controller needs to differentiate the segments that can be retrieved from caches and the segments only available on the original server.

The controller first needs to know which segments will be retrieved by the client in the next round. We can formulate this as a function of the last segment retrieved by a client and the number of segments to be requested in the next round. The information of the requested segments can be easily maintained by monitoring the DASH requests. However, predicting the identifier of future segments requires some knowledge of the available segments and the behavior of the streaming service client. In this paper, we focus on Netflix-like video service, which currently dominates the commercial video streaming industry. Commercial video streaming service allows the video player buffers only a fixed amount of data, in the unit of

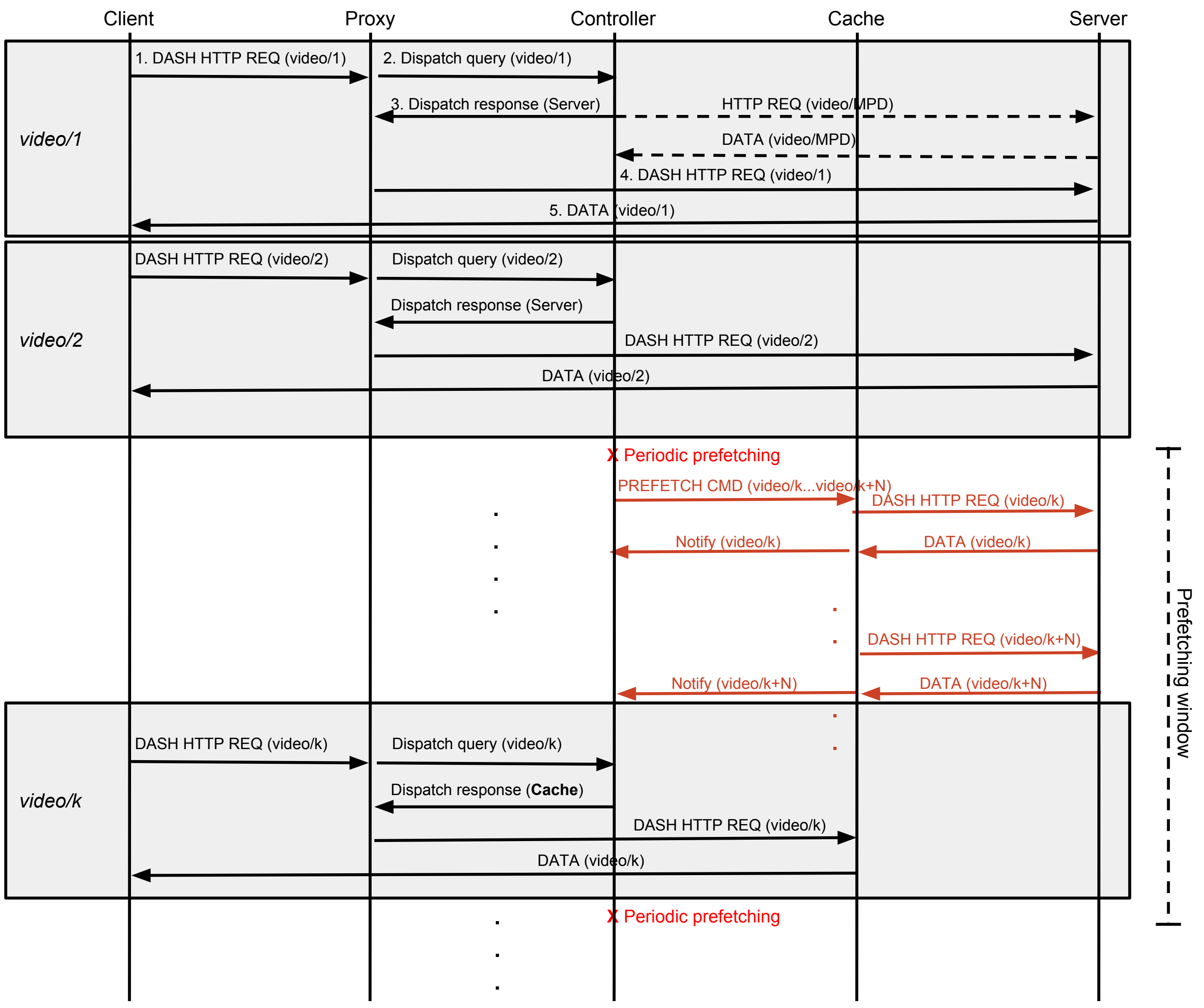


Figure 3: Message exchanges with prefetching

seconds or minutes, at any given time. The subsequent segments are only requested when equivalent playback has been consumed. In this case, we can reasonably expect a uniform DASH requests arrival for each client once its playback buffer becomes full. Before that, a client keeps requesting the next segments to fill up the buffer. More formally speaking, the number of segments to be requested by the client in the next round can be described by

$$N_{req} = s_m - s_b + \frac{W}{s_t} \tag{3}$$

where s_m is the maximum buffer size in the unit of seconds, s_b is the current buffer size of the client, and s_t is the playback time of a segment in seconds. With the knowledge of last requested segment, the controller examines the next N_{req} segments and determines how many of them are already cached. If these segments are not cached yet at the time of scheduling starts, we assume the segment will be requested from the server and adds the amount of bandwidth needed for this segment to the anticipated network load N_i for a client i. B_{old} can be estimated as

$$B_{old} = \sum_i N_i s_{v,i} \tag{4}$$

where sv, i is the size of video segments in bytes. Note that we calculate B_{old} in a conservative way. That is, we assumes clients will request the highest quality segment and the segment size sv, i is approximated accordingly. This can be easily extended to a finer sv, i approximation on the controller side with knowledge of rate adaptation mechanism used by clients.

C. Segment prefetching scheduling

Once we have the residual capacity available for the cache, the controller needs to decide which segments to prefetch. We first explain the principle of our scheduling algorithm by an example with a single client. For simplicity, we first assume the cache-server path is able to sustain the highest video rate. Suppose each segment represents one second of the video and the client's last request was for segment 2, and our round length is 5 seconds. Therefore, we expect the client will retrieve segment 3 to segment 7 in the next round. Now let us first assume the cache does not have segment 3 to segment 7, and therefore the client will request the segments directly from the server. In this case, in order to prevent bandwidth waste (i.e. the cache prefetches the same segment as the client does), the cache must prefetch segments after segment 7, that is, starting from segment 8. In this simple case, the controller can calculate the expected number of segments to prefetch simply by

$$N_p = B_A W \tag{5}$$

That is, the cache will prefetch segment 8 to 8+N_p. If we assume segment 3 to 15 have been prefetched by the cache, in this case, the client will obtain these segments from the cache in the next round, and the cache should prefetch segment 16 to 16+N_p. Only the segments that have not been cached or played are scheduled.

Next, let us generalize the scenario to support multiple video rates and various network condition. Consider a case when the cache-server path cannot always sustain the highest video rate, say the highest video rate is 1Mbps but the residual bandwidth available for prefetching is only 700kbps. We have a question that whether we should still fetch the highest quality segment. In the case with a single client, suppose the earliest playback time among all segments awaiting prefetching is T, the data rate is R, the approximate segment size at rate r is s_r, the time the prefetching of segment i starts is t_i we can prefetch higher quality segments as long as the following holds:

$$T > \frac{R}{s_r} + t_i \tag{6}$$

In other words, the controller selects the video rate r by

$$\underset{r}{\text{argmax}} \frac{R}{s_r} + t_i - T \tag{7}$$

Note that in this case the controller may schedule less than N_p segment as it can only schedule the segments who satisfies $t_i < T$

Now consider the case where the cache prefetches for multiple clients watching different videos. By the same principle, the controller can decide for each client which segment to start from. The solution we are looking for is a way to prioritize the segments, and then we can take at most first N_p segment accordingly. For simplicity, we overlook the fairness between clients and simply order the segments by their estimated playback time, that is, the time the particular video segment will be needed. This time is estimated at the controller using the recorded requested time of the first segment request from a client, the duration of segments, and the order of the segment in the video. The duration and order of segments are available in the metadata carried by the mpd of DASH videos. If the residual capacity can sustain highest quality among all videos, the controller then selects the N_p segments with the earliest estimated playback time in the list to prefetch. In case when the residual capacity does not necessarily sustain all rates, equation 7 is used to decide which rate should be requested. Note that prefetching higher quality segment for one client does not affect the normal playback of other clients as we limit the end time of each prefetching to before T by equation 7. In addition, the capacity needed to sustain normal playback of all clients during the next round has been included in equation 4.

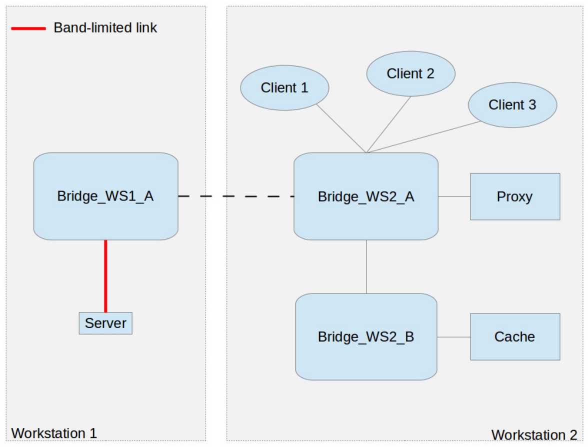


Figure 4: Testing scenario

V. EXPERIMENTS

A. Testbed and Settings

Our testing scenario consists of six nodes: a video server, a cache, a proxy, and three clients. The system is implemented using Open vSwitch and VirtualBox on top of two workstations as represented in Figure 4. Each node of the scenario runs on its own virtual machine. The algorithm logic is implemented extending the Floodlight Java based OpenFlow controller.

In our scenario, the server contains three different sample videos of the same length, and with the same video rates available: 200Kbps, 250Kbps, 300Kbps, and 400Kbps. The testing video is trimmed from the DASH data set and is 200 seconds long. Each segment's duration is 2 seconds. Finally the video is played at the client by a modified VLC player. In order to observe the behavior of Netflix-like video services, we modified the latest VLC player's DASH plug-in with two additional features:

1) **Limited download buffer**: the client downloads and keeps only the video segments for the following t seconds of playback at any given time. We set $t = 20$ in our experiment.

2) **Video quality adaptation**: the client adapts video quality by a moving average of the data rate estimates of the previous k segments. This allows the video quality adaptation to be more responsive, but also avoids unnecessary quality fluctuation. In the experiment, we use the value $k = 5$.

To emulate the behavior of resource competition between multiple clients, the server link rate has been reduced to 900 kbps, which provides enough bandwidth to serve only two flows at the maximum bitrate available.

B. Results

We experiment using the previously described settings and compare the results with and without our prefetching approach. In particular we analyze two scenarios: in the first one we look at how a quick transition from a lightly loaded network to a congested network influences the perceived video quality for the user; in our setup, this corresponds to starting a first client at time 0 and the other two clients half way through the reproduction of the video for the initial client (after 100 seconds). In the second scenario, we look at how the network reacts to an increasing load over time by starting the three clients progressively at times 0 seconds, 70 seconds and finally 140 seconds. To evaluate the perceived quality we look at the decisions taken by VLC's DASH plugin. The adaptation logic used inside the DASH module selects the bitrate to fetch depending on two factors: estimated throughput and video reproduction needs. Hence, the bitrate selected reflects the perceived quality by the end user.

Figures 5a and 5d show the time traces of the first scenario with and without prefetching. From this trace we can see how the cache exploits the unused capacity available during the first part of the experiment to prefetch the entire video that the first client is retrieving. By doing so, the server link never reaches a congested status and this allows the second and third clients to have enough bandwidth available to provide the best video quality to the users. When prefetching is not enabled, the arrival of the two new clients corresponds to a drastic reduction in the bitrates of the segments fetched. We notice that this is further influenced by the behavior of DASH clients that retrieve only a time limited buffer window. This window is not big enough to compensate for the fluctuation in available bandwidth and causes the player of the first client to reduce the video quality displayed. In a similar manner, 5b and 5e show the same analysis for the second scenario. While the contention period, when all three clients are reproducing the content, is shorter, it is still evident how prefetching benefits maintaining a higher quality reproduction. Moreover, in this scenario, both the first two clients can leverage the presence of the cache in the network.

To get a closer look at the distribution of the choices taken by the adaptive logic, we present the CDFs of the segment bitrates in Figures 5c and 5f. As we can see, more than 20% of the chunks for the first scenario and more than 10% for the second scenario are fetched at a reduced quality if the prefetching algorithm is not enabled in contrast with the close

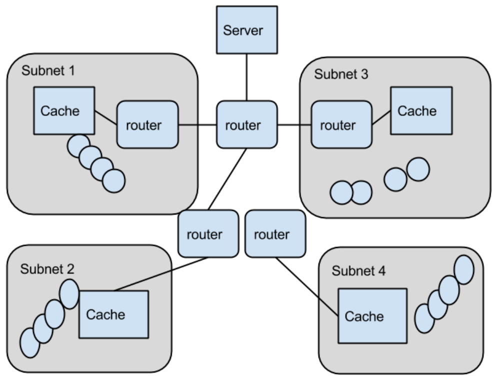


Figure 6: Simulation Scenario

to 0% results obtained with prefetching. This results are even more encouraging considering the fact that they include also the portions of time where only one or two clients are actively fetching data and then the provided bandwidth is enough to retrieve the best video quality possible.

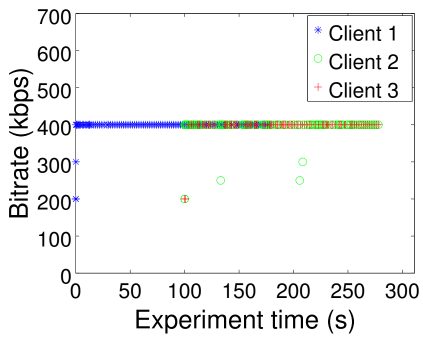
C. Simulation

We also conducted a simulation to investigate the performance of the proposed scheme in larger network using the *ns3* simulator. The simulation scenario consists of 1 video server and 16 clients, each downloads a different video, from 4 subdomains, as shown in Figure 6. There are 4 clients in each subdomain. Each subdomain has a cache that may prefetch video segments for its 4 clients. All links have data rate 5Mbps. To simulate the prime time traffic, we divide all clients to two groups: prim-time clients and non-prime-time clients. We define 400-800 seconds as prime-time, at 400 seconds, 3 clients from each subdomain starts their video downloading within 10 seconds. The non-prime-time clients start before prime-time period with inter-arrival interval 10 seconds. The entire simulation lasts for 1000 seconds. All other settings such as application and video rate configurations are the same as in the testbed experiments. The Netflix player-like client behavior including video rate adaptation based on exponential average of session data rate and HTTP protocol are implemented.

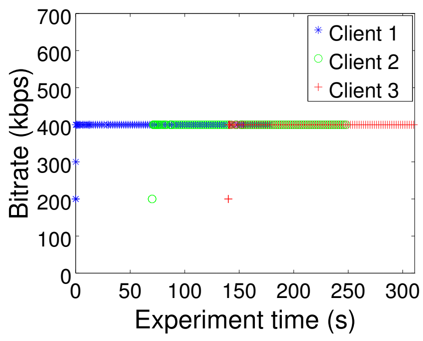
We focus on the time-shifting server load in this experiment. The result is shown in Figure 7. The server load is defined as the traffic sent by the server. With prefetching, the server load is shifted ahead of time and is more "flat" at prime time. The peak traffic is lighter when rush time users come in. The peak server load is 3300kbps without prefetching, and is reduced to 2650kbps with prefetching. This reduction in peak demand of 20% is similar to the values depicted on Figure 1 and 2. Note that in this experiment the bottleneck link between the server and the core router is not congested, and thus all users are able to stream highest quality video.

VI. CONCLUSIONS

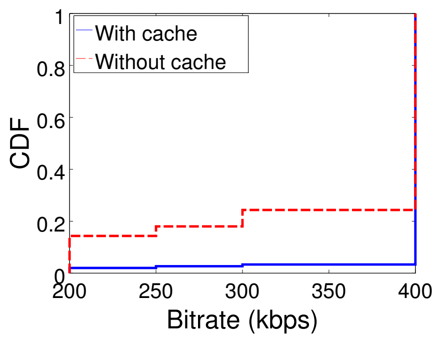
We propose a DASH-aware scheduling algorithm for edge cache prefetching in ICN. Our algorithm utilizes the knowl-



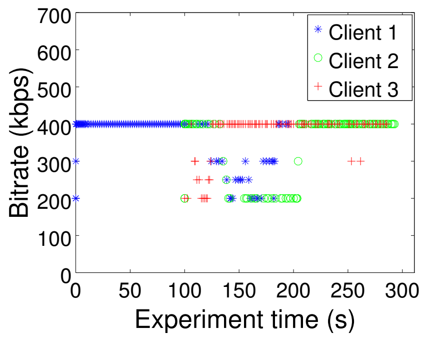
(a) Scenario 1 with pre-fetching



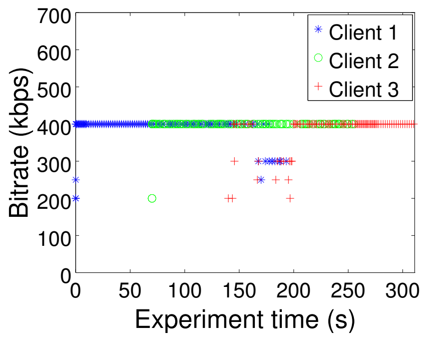
(b) Scenario 2 with pre-fetching



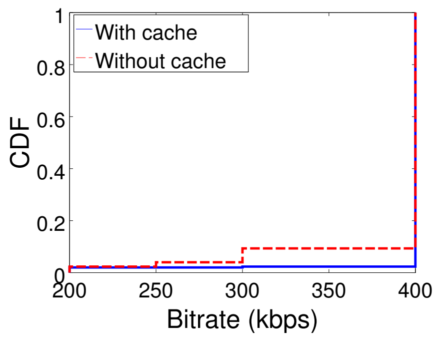
(c) Scenario 1 bitrates CDF comparison



(d) Scenario 1 without pre-fetching



(e) Scenario 2 without pre-fetching



(f) Scenario 2 bitrates CDF comparison

Figure 5: Experimental results

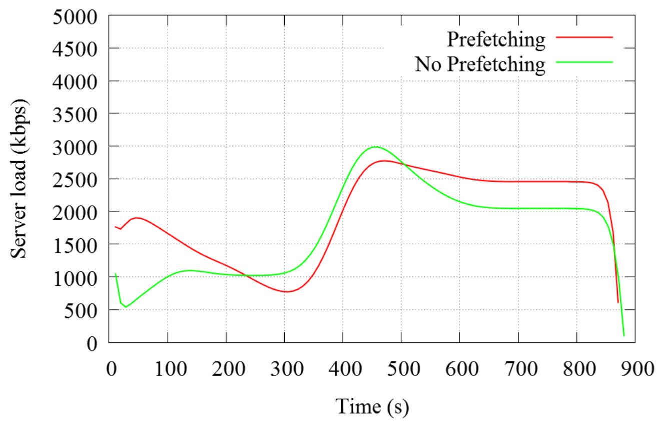


Figure 7: Simulation results: server load

edge of the video segments obtained by DASH MPDs, and schedules the segment prefetching according to the current network condition and the context of video playback including the video rates and client request history. The results show that this method improves the quality of experience on client side by utilizing the residual bandwidth and requesting video segments in advance.

References

[1] L. Ariyasinghe, C. Wickramasinghe, P. Samarakoon, U. Perera, R. P. Buddhika, and M. Wijesundara. Distributed local area content delivery approach with heuristic based web prefetching. In *Computer Science & Education (ICCSE), 2013 8th International Conference on*, pages 377–382. IEEE, 2013.

[2] A. Balachandran, V. Sekar, A. Akella, and S. Seshan. Analyzing the potential benefits of cdn augmentation strategies for internet video workloads. In *Proceedings of the 2013 conference on Internet measurement conference*, pages 43–56. ACM, 2013.

[3] L. Carlinet, T. Huynh, B. Kauffmann, F. Mathieu, L. Noirie, and S. Tixeuil. Four months in daily motion: Dissecting user video requests. In *Wireless Communications and Mobile Computing Conference (IWCMC), 2012 8th International*, pages 613–618, Aug 2012.

[4] A. Chanda and C. Westphal. Contentflow: Mapping content to flows in software defined networks. *in Proc. IEEE Globecom'13, Atlanta, GA*, November 2013.

[5] S. Chen, H. Wang, X. Zhang, B. Shen, and S. Wee. Segment-based proxy caching for internet streaming media delivery. *MultiMedia, IEEE*, 12(3):59–67, 2005.

[6] C. Dannewitz. Netinf: An information-centric design for the future internet. In *Proc. 3rd GI/ITG KuVS Workshop on The Future Internet*, 2009.

[7] P. Fiadino, A. D'Alconzo, A. Br, and P. Casas. On the detection of network traffic anomalies in content delivery network services. In *Proceedings of IFIP/IEEE Networking conference*, 2014.

[8] A. Finamore, M. Mellia, M. M. Munafò, R. Torres, and S. G. Rao. Youtube everywhere: Impact of device and infrastructure synergies on user experience. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, pages 345–360. ACM, 2011.

[9] R. Grandl, K. Su, and C. Westphal. On the interaction of adaptive video streaming with content-centric networking. *in Proc. Packet Video Workshop, San Jose, CA*, December 2013.

[10] C. Huang, J. Li, and K. W. Ross. Can internet video-on-demand be profitable? In *ACM SIGCOMM Computer Communication Review*, volume 37, pages 133–144. ACM, 2007.

[11] C. Huang, A. Wang, J. Li, and K. W. Ross. Understanding hybrid cdn-p2p: why limelight needs its own red swoosh. In *Proceedings of the 18th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, pages 75–80. ACM, 2008.

[12] V. Jacobson, M. Mosko, D. Smetters, and J. Garcia-Luna-Aceves.

Content-centric networking. *Whitepaper, Palo Alto Research Center*, pages 2–4, 2007.

[13] S. Jin, A. Bestavros, and A. Iyengar. Network-aware partial caching for internet streaming media. *Multimedia systems*, 9(4):386–396, 2003.

[14] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica. A data-oriented (and beyond) network architecture. In *ACM SIGCOMM Computer Communication Review*, volume 37, pages 181–192. ACM, 2007.

[15] V. Krishnamoorthi, N. Carlsson, D. Eager, A. Mahanti, and N. Shahmehri. Helping hand or hidden hurdle: Proxy-assisted http-based adaptive streaming performance. In *Proc. IEEE MASCOTS*, 2013.

[16] S. Lederer, C. Westphal (Editor), C. Mueller, A. Detti, D. Corujo, A. Azgin, C. Timmerer, and D. Posch. Adaptive streaming over icn. *IETF draft draft-irtf-icnrg-videostreaming-01.txt*, July 2014.

[17] J. Lee, J. Hwang, N. Choi, and C. Yoo. Svc-based adaptive video streaming over content-centric networking. *KSII Transactions on Internet and Information Systems (TIIS)*, 7(10):2430–2447, 2013.

[18] K. Li, K. Tajima, and H. Shen. Cache replacement for transcoding proxy caching. In *Web Intelligence, 2005. Proceedings. The 2005 IEEE/WIC/ACM International Conference on*, pages 500–507. IEEE, 2005.

[19] Z. Li, M. K. Sbai, Y. Hadjadj-Aoul, A. Gravey, D. Alliez, J. Garnier, G. Madec, G. Simon, and K. Singh. Network friendly video distribution. In *International Conference on the Network of the Future*, volume 1, 2012.

[20] F. Liu, B. Li, and H. Jin. Peer-assisted on-demand streaming: characterizing demands and optimizing supplies. *Computers, IEEE Transactions on*, 62(2):351–361, Feb 2013.

[21] H. H. Liu, Y. Wang, Y. R. Yang, H. Wang, and C. Tian. Optimizing cost and performance for content multihoming. *ACM SIGCOMM Computer Communication Review*, 42(4):371–382, 2012.

[22] X. Liu, F. Dobrian, H. Milner, J. Jiang, V. Sekar, I. Stoica, and H. Zhang. A case for a coordinated internet video control plane. In *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*, pages 359–370. ACM, 2012.

[23] F. Malandrino, M. Kurant, A. Markopoulou, C. Westphal, and U. C. Kozat. Proactive seeding for information cascades in cellular networks. In *INFOCOM, 2012 Proceedings IEEE*, pages 1719–1727. IEEE, 2012.

[24] Z. Miao and A. Ortega. Proxy caching for efficient video services over the internet. In *in 9th International Packet Video Workshop (PVW'99*. Citeseer, 1999.

[25] V. N. Padmanabhan and J. C. Mogul. Using predictive prefetching to improve world wide web latency. *ACM SIGCOMM Computer Communication Review*, 26(3):22–36, 1996.

[26] S.-H. Park, E.-J. Lim, and K.-D. Chung. Popularity-based partial caching for vod systems using a proxy server. In *Proceedings of the 15th International Parallel & Distributed Processing Symposium*, page 115. IEEE Computer Society, 2001.

[27] L. Plissonneau and E. Biersack. A longitudinal view of HTTP video streaming performance. In *Proceedings of the 3rd Multimedia Systems Conference*, pages 203–214. ACM, 2012.

[28] W. Qu, K. Li, H. Shen, Y. Jin, and T. Nanya. The cache replacement problem for multimedia object caching. In *Semantics, Knowledge and Grid, 2005. SKG'05. First International Conference on*, pages 26–26. IEEE, 2005.

[29] S. Sen, J. Rexford, and D. Towsley. Proxy prefix caching for multimedia streams. In *INFOCOM'99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1310–1319. IEEE, 1999.

[30] I. Sodagar. The MPEG-DASH standard for multimedia streaming over the internet. *MultiMedia, IEEE*, 18(4):62–67, 2011.