SVC-based Adaptive Video Streaming over Content-Centric Networking

Junghwan Lee[1], Jaehyun Hwang[2], Nakjung Choi[2], and Chuck Yoo[1]
[1] Department of Computer Science and Engineering, Korea University, Seoul, Korea
[e-mail: {jhlee, chuckyoo}@os.korea.ac.kr]
[2] Bell Labs, Alcatel-Lucent, Seoul, Korea.
[e-mail: {jh.hwang, nakjung.choi}@alcatel-lucent.com]
*Corresponding author: Chuck Yoo

Received June 21, 2013; revised August 31, 2013; accepted September 21, 2013; published October 29, 2013

Abstract

In recent years, HTTP adaptive streaming (HAS) has attracted considerable attention as the state-of-the-art technology for video transport. HAS dynamically adjusts the quality of video streaming according to the network bandwidth and device capability of users. Content-Centric Networking (CCN) has also emerged as a future Internet architecture, which is a novel communication paradigm that integrates content delivery as a native network primitive. These trends have led to the new research issue of harmonizing HAS with the in-network caching provided by CCN routers. Previous research has shown that the performance of HAS can be improved by using the H.264/SVC(scalable video codec) in the in-network caching environments. However, the previous study did not address the misbehavior that causes video freeze when overestimating the available network bandwidth, which is attributable to the high cache hit rate. Thus, we propose a new SVC-based adaptation algorithm that utilizes a drop timer. Our approach aims to stop the downloading of additional enhancement layers that are not cached in the local CCN routers in a timely manner, thereby preventing excessive consumption of the video buffer. We implemented our algorithm in the SVC-HAS client and deployed a testbed that could run Smooth-Streaming, which is one of the most popular HAS solutions, over CCNx, which is the reference implementation of CCN. Our experimental results showed that the proposed scheme (SLA) could avoid video freeze in an effective manner, but without reducing the high hit rate on the CCN routers or affecting the high video quality on the SVC-HAS client.

Keywords: bitrate selection, content-centric networking, HTTP adaptive video streaming, scalable video codec, video freeze time

This work was supported in part by a National Research Foundation of Korea (NRF) grant funded by the Korea government (MEST) (No.2010-0029180) and supported in part by the Seoul R&BD Program (WR080951).

http://dx.doi.org/10.3837/tiis.2013.10.006

1. Introduction

At present, HTTP adaptive streaming (HAS) is a new state-of-the-art video streaming technology, which is based on the success of HTTP. Major companies such as Microsoft, Apple, and Adobe have developed their own streaming technologies, i.e., HAS [1], Live Streaming [2], and Dynamic Streaming [3], respectively. In addition, 3GPP and ITU-T provide Dynamic Adaptive Streaming over HTTP (DASH) [4][5] as the standard streaming service. Although the names and the details of these algorithms are quite different, they have a common design goal, which is the provision of seamless video streaming by dynamically adjusting the quality of a video stream depending on the user's environment. More specifically, video content is divided into several video chunks and multiple quality levels are provided for each video chunk. Clients then select the most appropriate quality level by measuring the current network conditions on a moment-by-moment basis.

HAS is basically designed for current IP-based Internet environments, but we considered Content-Centric Networking (CCN) [6] as our target network environment in the present study. CCN is a novel networking paradigm that makes content delivery the main network primitive, which means that routers can route using the content names and serve content, if available. In particular, CCN supports an in-network caching functionality by equipping a cache (storage), known as a content-store, to each CCN router. Therefore, the total amount of network traffic can be reduced if large volumes of popular content are served from local CCN routers, rather than the original content provider.

However, the problem is that CCN treats different quality level video chunks from the same file as different content, even when these chunks are the same content from a human's perspective. In this case, the HAS client might not benefit from the in-network caching of CCN if it requests a series of video chunks that are not cached in the local CCN routers[1]. To overcome this situation, the scalable video codec-based HAS (SVC-HAS) scheme [7][8][9] was proposed for content delivery network (CDN) environments where in-network caching is supported. In the H.264/SVC encoding scheme [10], each video chunk comprises one base layer and several enhancement layers. The base layer is essential for playing a video chunk, so it must always be downloaded after the request for each video chunk. Thus, the probability that the base layers persist in caches for a long period is quite high, which results in a high cache hit rate. However, we have observed that the SVC-HAS client frequently tries to select high quality level video chunks, for which the bitrate is larger than the bottleneck bandwidth, because it overestimates the current network bandwidth as the base layers are downloaded mostly from the cache of the local CCN routers. This misbehavior sometimes drains the client's buffer by unnecessarily downloading higher quality enhancement layers, which results in video freeze when the buffer fullness drops below zero.

In this study, we propose a new bitrate adaptation algorithm, referred to as SVC layer adaptation (SLA), for SVC-HAS in CCN environments. We develop a new drop timer to stop the downloading for the current video chunk in a timely manner and to move onto the next chunk when the timer expires. We implemented our algorithm in the SVC-HAS client and deployed it in a testbed that could run Smooth-Streaming, which is one of the most popular

[1] We note that this study focuses mainly on the in-network caching capacity of CCN routers, but the same problem would occur in CDN- or HTTP-caching environments. Of the various in-network caching solutions, we selected CCN as our target environment because it is one of the most popular emerging future Internet architectures.

HAS solutions, over CCNx [11], which is the reference implementation of CCN. Our experimental results show that the proposed scheme is very effective in the CCN environment because it takes advantage of in-network caching and prevents the client's buffer from being drained.

The remainder of this paper is organized as follows. Section 2 presents some background and related work on HAS, video encoding/decoding schemes, and CCN, which are important for a clear understanding of the paper. In Section 3, we compare AVC-HAS with SVC-HAS and explain our motivation. Section 4 describes our new bitrate adaptation algorithm in detail. Section 5 describes the experimental methodology we used to run the adaptive video streaming over CCN. Section 6 summarizes our major results and findings. Finally, Section 7 concludes the paper and discusses our future research plans.

2. Related Work

In this section, to facilitate a better understanding of this research, we provide an overview of previous studies related to adaptive video transmission, encoding/decoding frameworks, and CCN.

2.1 Adaptive Video Transmission

The main goal of HAS is to deliver video with a high quality of experience (QoE), even in dynamic network conditions. There are different implementations of HAS [1][2][3], but the basic idea is always the same. Before video streaming services, a single video is encoded at multiple bitrates and resolutions, typically 7–10 different rates, which range from 150 Kbps for mobile devices up to 6 Mbps for high definition. Each encoding is divided into video segments or chunks, which are typically 2–30 seconds in length. First, the client downloads a manifest file that contains information on the available audio and video streams, their encodings, and chunk durations.

The client requests each chunk of video using HTTP. For each chunk download, the client estimates the network bandwidth and runs a rate determination algorithm (RDA) to determine the bitrate used to request the next chunk. Each request gives the client the opportunity to change the bitrate. During the selection of the bitrate, the RDA must consider the available bandwidth, CPU processing power, screen size, and the fullness of its buffer. The RDA must balance the desire to request high quality video with the need to prevent its buffer from draining in order to deliver the highest sustainable quality without stops or stutters.

The HAS uses standard HTTP, so it easily traverses network address translations (NATs) and firewalls, and can utilize existing web infrastructures, such as caches, proxies, and CDNs. In addition, HTTP is stateless so no session information is required on the server side. Thus, the video bitrates and timing between requests are driven totally by the client. Recently, this type of approach has been discussed by standard bodies and MPEG-DASH (Dynamic Adaptive Streaming over HTTP) [5] is likely to be the first international standard for adaptive video streaming. We also note that the current HTTP (version 1.1) supports persistent connection, which allows multiple HTTP requests to use a single TCP connection. Therefore, it is assumed that there is only one TCP connection for each adaptive video streaming.

2.2 Video Encoding/decoding

As an emerging standard, H.264/SVC [12] has the added ability to adapt bitrates, which is a single type of encoding technique that has been extended from the H.264/MPEG-4 standard.

In this framework, a single video stream comprises multiple layers, i.e., a base layer and more than one enhancement layer [13]. The base layer is a mandatory requirement for playing the video and decoding, while the other enhancement layers provide a better quality video stream. However, there is a strong dependency between layers and the higher layers are rendered useless if any of the lower layers are missed, which demands fairly complex scheduling disciplines [14]. At present, the H.264/SVC standard can provide scalability in terms of the spatial, temporal, and quality domains. Many recent studies have investigated the optimal number of layers required to provide better QoE, depending on the device capability, the available network bandwidth, and the error rate [15][16].

2.3 Content-Centric Networking

Over the last decade, the concept of Information-Centric Networking (ICN) has begun to attract much attention as a promising future Internet architecture, by adapting the network architecture to current network usage patterns, i.e., data dissemination. As a pioneering approach, TRIAD [17] explored the benefits of this novel networking paradigm. TRIAD uses URLs as names, i.e., user-friendly, structured, and location-independent identifiers, and an integrated directory is used to map from the DNS component of the URL to the closest available replica of the data. A number of other systems use distributed hash tables to route queries for content names, i.e., ROFL [18], i3 [19], and SEATTLE [20] are the most popular designs in this area. DONA [21] is another interesting design that replaces DNS using a combination of flat, self-certifying names, with a name-based anycast primitive on top of the IP layer.

CCN [6] was proposed recently, which may be characterized by routing-by-name and in-network caching. A user explicitly requests content by broadcasting their interest with the content name to the network and any content router that hears the interest and that has appropriate data can respond with a data packet. Data is retrieved only in the response to an interest so a single Interest packet corresponds to a single data packet in each link, thereby enabling inherently multicasting. In addition, in-network caching contributes to a low dissemination latency and network load reduction by storing popular content at the network edge close to users.

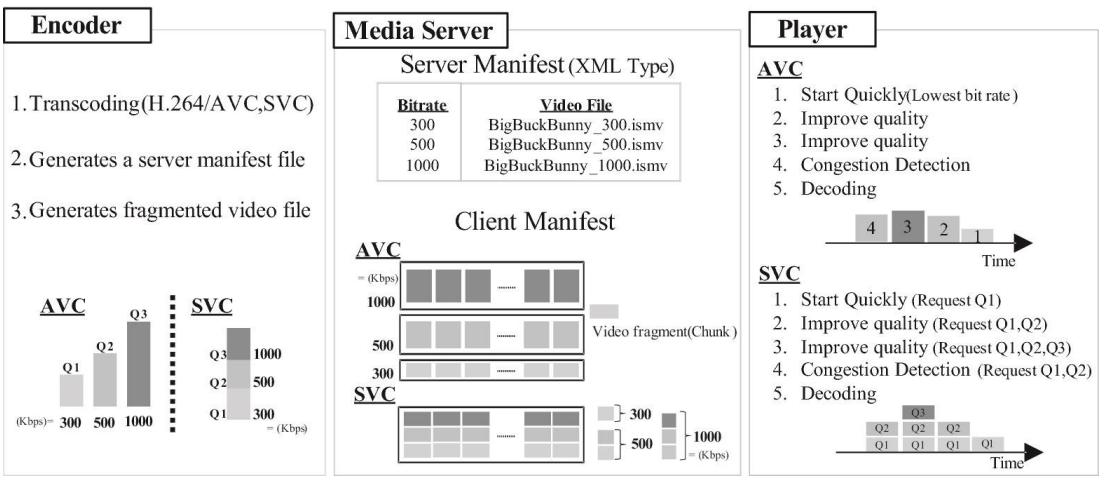
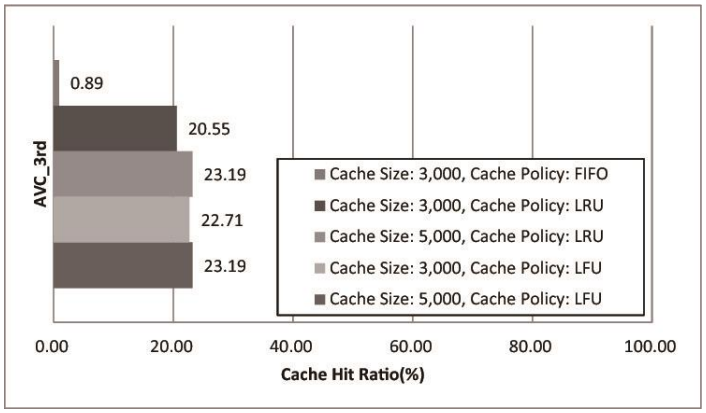


Fig. 1. Comparison of H.264/AVC and H.264/SVC HTTP adaptive streaming architectures.

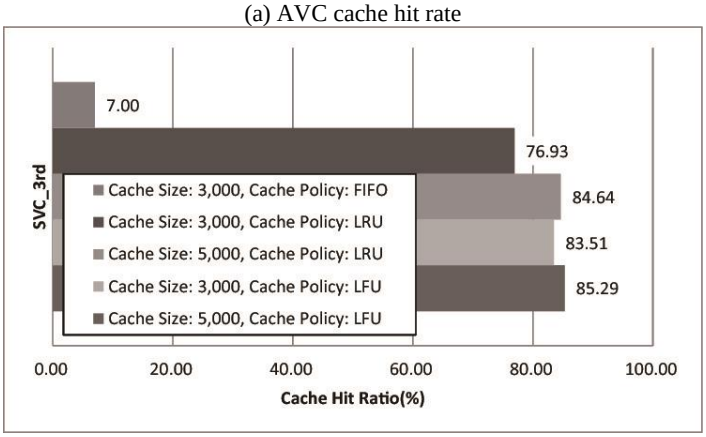
3. Motivation

3.1 Comparison of AVC-HAS and SVC-HAS

Despite having the same video in the user context, AVC-HAS generates multiple (independent) versions of the content from the server/network perspective. Therefore, there is competition for storage or caching space, even for a single item of video content at each content server or router. To alleviate this inefficiency, there have been recent efforts to exploit the H.264/SVC codec in the video HAS framework [7][8][9][22]. In H.264/SVC, a single item of video content comprises a base layer and more than one enhancement layer in a hierarchical form, depending on the number of quality levels supported [12], as shown in **Fig. 1**. In addition to the base layer, higher video quality can be achieved as more enhancement layers are assembled during the decoding process on the client side. Because of these H.264/SVC sub-hierarchies, the base layer is essential for every quality level, which means that all of the video requests share some common parts, i.e., at least the base layer. In these in-network caching environments, therefore, the probability that base layers are cached will be very high because of the high request rate. H.264/SVC can improve the in-network caching efficiency[2] by aggregating video requests, i.e., the cache hit rate.



(a) AVC cache hit rate



(b) SVC cache hit rate

Fig. 2. Comparison of the cache hit rates with AVC and SVC on the testbed.

[2] A similar trend was observed in [7].

Fig. 2 supports this suggestion, where SVC-encoded video content has a higher cache hit rate than AVC-encoded content with all cache size and replacement policy combinations. However, H.264/SVC requires more bits for each video quality level because of its hierarchical structure, i.e., there are encoding/decoding overheads of >10% between hierarchies [12]. To reduce these overheads, high efficiency video coding (HEVC) is the next generation codec, which is being developed by the ISO/IEC Moving Picture Experts Group (MPEG) and ITU-T Video Coding Experts Group (VCEG). SVC extensions are already in discussion, which will lead to lower encoding/decoding overheads compared with the existing H.264/SVC standard.

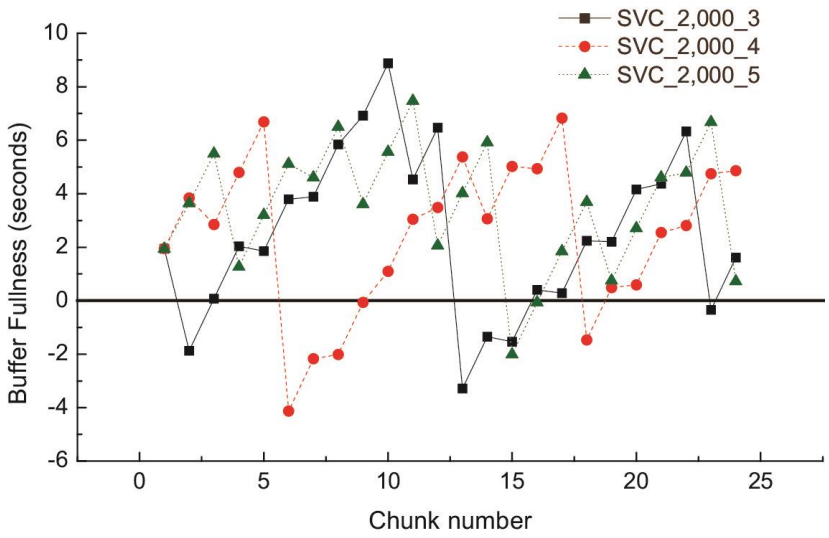


Fig. 3. Buffer fullness with SVC-HAS RDA (cache size = 2000, bottleneck = 300 Kbps).

3.2 Novel RDA for SVC-HAS

In general, most RDAs used in widely deployed HAS clients assume a client-to-server model [22]. In this case, each client estimates the available bandwidth on the end-to-end path between the client and server to adjust the quality levels of the video chunks. In in-network caching environments, part of the video chunks can be cached at multiple points so the client may download the video chunk from any of multiple sources or in parallel [23]. This behavior makes it difficult to estimate and predict the network bandwidth. In the SVC-HAS framework, in particular, each video chunk comprises more than one segment (or layers), which makes the estimation and prediction of the network bandwidth even more difficult. For example, if the base layers of the requested video segments are cached in multiple content routers near the client, most of the video requests are satisfied from the caches, so the existing RDAs may decide to increase quality level. However, if no video segments related to higher quality levels of the video chunks in the caches are present near the client, their retrieval from the server takes a long time and the client's buffer is consumed continuously. **Fig. 3** shows the buffer fullness[3] when the same content is requested repeatedly three, four, and five times, where the cache size of the intermediate content routers is 2,000 (in terms of CCN data packets) and the bottleneck link is 300 Kbps. In all cases, the buffer fullness always begins with 2 seconds because the client requests the same quality level for the first video chunk, e.g., 221 Kbps, and

[3] The buffer fullness is defined as the playback buffer size at the player, which is measured in seconds [29].

the RDA starts to work after the first chunk has been downloaded (the length of each video chunk is 2 seconds). Next, the buffer fullness exhibits different patterns depending on the cache hit rate and the quality level used. In particular, the buffer fullness drops below zero (which is referred to as a play-out freeze) if the downloading time of the enhancement layers is longer than the current buffer fullness. Thus, a novel RDA needs to be designed to prevent video freeze in an effective manner while maintaining a high quality level, even in highly dynamic environments.

Recently, a new RDA was proposed that uses a sloping-based SVC heuristic [30][31]. In this scheme, a client performs either: (i) prefetching (downloading the base layer for future segments) or (ii) backfilling (downloading the enhancement layer for the current segments). A sloping-based heuristic was defined for this purpose where more backfilling is performed if the slope is steeper, whereas more base layers of new segments are downloaded if the slope is flatter. However, this method assumes that the rate (bandwidth) distribution is provided in advance and there is no dynamic slope adjustment scheme. Furthermore, to develop a dynamic adjustment policy for the sloping-based approach, an accurate bandwidth measurement is required, which is impossible in CCN environments due to the cache effect. Thus, the failure of slope adjustment will also result in video freeze in the same way as other RDAs.

4. Proposed Method

Our bitrate adaptation algorithm has two basic components: (i) bitrate selection for the next video chunk and (ii) SVC layer adaptation (SLA) based on the downloading status. We explain these functions in detail in the following subsections.

4.1 Bitrate Selection

Let $Q = \{q_1, q_2, \cdots, q_n\}$ be a set of available video quality levels extracted from a manifest file, assuming that q_1 indicates the lowest bitrate and q_n is the highest. To select the bitrate, we need to measure the downloading bandwidth for the ith chunk, DBW_i, as follows:

$$DBW_i = ChunkSize_i / t_i \tag{1}$$

where $ChunkSize_i$ is the chunk size and t_i is the download time for the ith chunk. Using this information, the bitrate for the ith chunk, x_i is obtained as follows:

$$x_i = \begin{cases} q_1, & i = 1 \\ \max\left\{ y \middle| \forall y \in Q : y \le DBW_{i-1} \right\}, & i > 1 \end{cases} \tag{2}$$

Thus, our algorithm always starts with q_1 as the first video chunk. The next bitrate is determined by the downloading bandwidth of the previous chunk. For example, if Q = {*50Kbps, 100Kbps, 150Kbps*} and DBW_{i-1} is 120 Kbps, the next selected bitrate, x_i, would be 100 Kbps. We note that our algorithm is based on H.264/SVC, assuming that each chunk contains several layers. Therefore, the chunk size should be the summed data size for all the downloaded layers in a chunk. Similarly, the download time is determined by the layer with the longest download time. Moreover, some layers may be dropped during the download with our layer adaption algorithm, which we will explain in the next subsection. Thus, only completely downloaded layers will be considered for the chunk size and the download time.

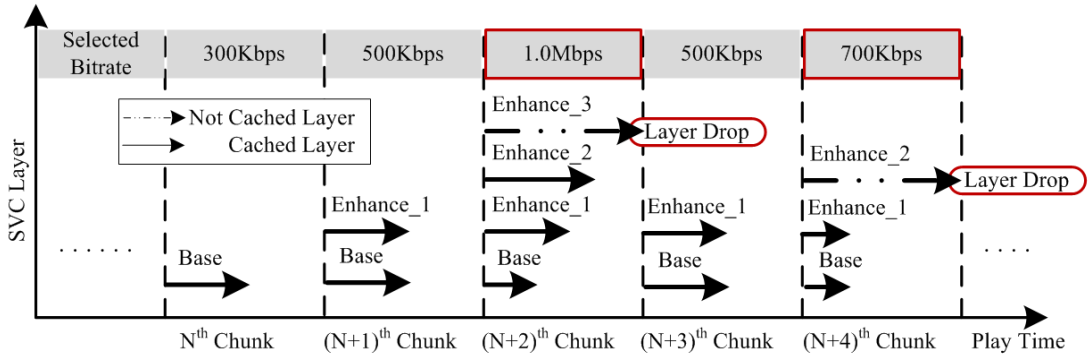


Fig. 4. Example of SVC layer adaptation.

4.2 SVC Layer Adaptation (SLA)

The main design aim of our algorithm is to avoid video freeze in an effective manner, as discussed in Section 3. **Fig. 4** outlines our basic strategy, i.e., dropping layers where the download time is too long. This figure shows that the selected bitrate increases to 1.0 Mbps based on the cache effect, by requesting the base layer and three enhancement layers in the $(N+2)^{th}$ chunk. The third enhancement layer is not cached by the cache of the local CCN nodes in this example, which means that it will eventually drain the video buffer when the client tries to download it from the original content provider. This may lead to video freeze. The easiest way of preventing this phenomenon is to simply stop the download for the layer. It is difficult to predict whether a specific layer is cached in the local CCN nodes. Thus, our approach uses a drop timer, which drops the layers that are not completely downloaded before the timer expires[4].

For this purpose, we define two initial states in terms of the buffer fullness: low-buffer state and steady state. We also define a threshold for state transition, α, where the state begins with the low-buffer state when $BF < \alpha$ and is transited to the steady state when $BF \geq \alpha$, where BF is the current buffer fullness. In the low-buffer state, we need to fill up the buffer because the buffer fullness is relatively low. Therefore, the drop timeout is set to the length of a video chunk so the buffer fullness is at least not reduced during this state. If all of the requested layers are downloaded earlier than the timeout, the client immediately requests the next chunk, which increases the buffer fullness. In the steady state, it is permissible to increase the drop timeout to a larger number than the chunk length because the buffer fullness is sufficient to download more enhancement layers that are probably not stored in the local caches. Next, we define another threshold, β, which prevents video freeze. We set the drop timeout so the buffer fullness is not less than β during the download. In summary, the drop timeout (DTO) is reset as follows whenever a new chunk is requested:

$$DTO = \begin{cases} ChunkLength, & BF < \alpha \\ ChunkLength + (BF - \beta), & BF \geq \alpha \end{cases} \tag{3}$$

[4] After the timer expires, we stop the ongoing HTTP download by closing its TCP connection. For the next video segments, a new TCP connection will be re-established because each video streaming maintains only one TCP connection based on the HTTP 1.1 specifications.

where *ChunkLength* is the length of a video chunk. If a timeout occurs, downloading is stopped for the video chunk and the next chunk is requested with a new selected bitrate. We note that the base layer is not dropped by the timer in all cases because the base layer is essential for playing the video chunk whereas the other enhancement layers are optional. Thus, although the download time of the base layer is larger than DTO, the next chunk request will be delayed until downloading of the base layer is completed.

5. Experimental Environment

This section describes the experimental environment we used to evaluate our new bitrate adaptation algorithm over CCNx [11], which is the reference implementation of CCN. **Fig. 5** outlines our testbed and its three main elements: the Smooth-Streaming server, CCN, and HAS client.

We ran a well-known web server, Apache2, with the Smooth-Streaming extension [24] on a Linux (openSUSE 11.4) machine as the Smooth-Streaming server. We enabled adaptive video streaming for a test video clip, "Big Buck Bunny," which is provided by Microsoft [25]. To enable SVC-based HAS, we encoded an original video clip using JSVM (version 9.19.14) [26] and the SVC reference codec, where the video clip was available at the following video encoding bitrates: 221, 429, 683, 1050, and 1503 Kbps. Note that each video chunk comprised one base layer and four enhancement layers. The base layer corresponded to the lowest quality level (i.e., 221 Kbps) and the encoding bitrate could increase up to 1,503 Kbps because the enhancement layers were sequentially stacked on top of the base layer. This information was transferred to the HAS client with a traditional Smooth Streaming manifest file. The length of each video chunk was 2 seconds.

For the CCN, we ran the CCN reference code (ccnx-0.6.2) [11] on three Linux (openSUSE 11.4) machines (CCN1 to CCN3) to enable CCN. In the following discussion, we refer to these machines as CCN nodes. We used HTTP Proxy and NetFetch, which are provided in the ccnx-0.6.2 release, to translate (IP) HTTP request/data packets into (CCN) interest/data packets and vice-versa. We used DummyNet [27] to control the network bandwidth and one-way delay, as shown in **Fig. 5**. In addition, we implemented LFU (in-cache LFU [28]) caching strategies in the CCN reference code.

For the HAS client, we used a Windows XP machine with the SVC-HAS client. This client had two different adaptive streaming schemes: the traditional (original) scheme and the proposed (new) scheme. Similar to the existing HAS, the SVC-HAS client decided and requested the quality level of the video chunks based on an estimation of the bandwidth. It could then play each video chunk after receiving all of the requested base and enhancement layers of the chunk. The RDA of the SVC-HAS was the same as that used by the existing HAS. We refer to this scheme as the "traditional scheme" in the next section. The proposed scheme included our new layer adaptation algorithm (SLA) on top of the traditional scheme to avoid video freeze.

Next, we provide an example to illustrate how the testbed operated (**Fig. 5**). The HAS client sent an HTTP request for a video chunk to the Smooth Streaming server. This HTTP request reached the HTTP Proxy (CCN1) where it was converted into several interest packets, each of which was forwarded to the next CCN node. If the content was not cached at any CCN node, the interest packets propagated toward NetFetch (CCN3). NetFetch converted the interest packets into a HTTP request and retrieved the desired content from the Smooth Streaming server. NetFetch split the HTTP response into several data packets, which were

then delivered as regular CCN data. These data were then forwarded across the CCN using information in the Pending Interest Table (PIT) and were cached at each node. The data packets were aggregated into a single video chunk at the HTTP Proxy, which then delivered the chunk to the HAS client as a response to the original HTTP request. Note that this strategy permits the running of a legacy HAS solution over CCN without any modifications to the HAS client.

We considered a single item of video content (Big Buck Bunny) with a playback time of 124 seconds. The length of each video chunk is 2 seconds in HAS, which means that there were 62 video chunks. In the experiments, we fixed the network delay at 10 ms while varying the network bandwidth among 300, 500, and 1000 Kbps. It has been reported that the delay does not have any significant effects on SVC-HAS solutions whereas the bandwidth significantly affects the results [31]. We also varied the cache size among 2000, 2500, and 3000 content-objects[5], and LFU was used as the cache replacement policy. For each set of network parameters, we instructed the client to download the same video content 10 consecutive times to investigate the effect of in-network caching on the performance of SVC-HAS. The maximum buffer fullness was 21 seconds.

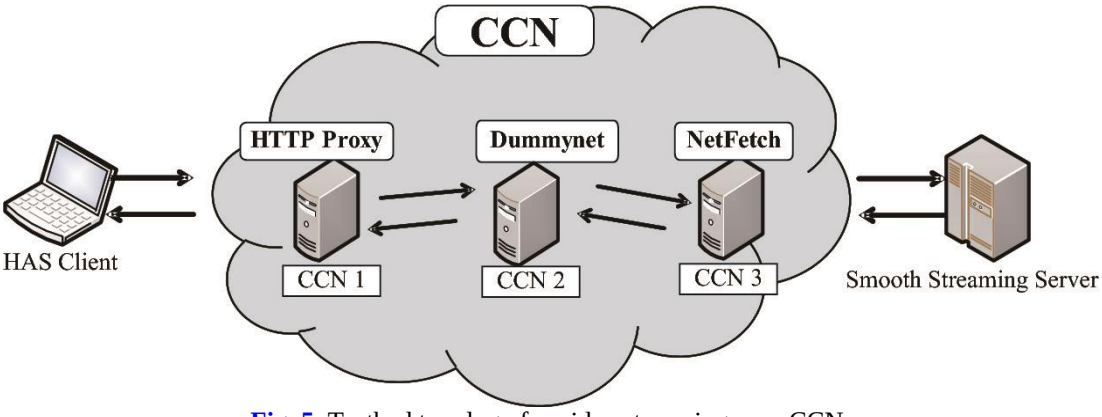


Fig. 5. Testbed topology for video streaming over CCN.

6. Performance Analysis

In this section, we present our evaluation of the performance of the proposed scheme (SLA) compared with the traditional scheme (i.e., existing RDA within the SVC-HAS framework). We performed several experiments to select the SLA parameters, α and β. The following performance metrics were measured: (i) video freeze time, (ii) buffer fullness, (iii) video quality, and (iv) cache hit rate. After each video chunk was downloaded successfully, its selected bitrate and the current buffer fullness were recorded. For the video freeze time, we counted how many times the buffer fullness dropped below zero. In addition, the cache hit rate was defined as the number of cache hits at all intermediate CCN nodes, i.e., CCN1, CCN2, and CCN3 in **Fig. 5**, divided by the total number of interest packets generated by the HTTP Proxy.

6.1 Selection of SLA Parameters

To obtain robust values for α and β, we simulated an extreme case where the network bandwidth varied between 250 Kbps and 2 Mbps with intervals of 20 seconds. No

[5] A content-object corresponds exactly to an item of content that a CCN data packet carries.

intermediate caches were used. In this environment, we measured the number of video freeze (VF) events and the selected bitrate (SB) several times, while varying α from 16.0 to 19.0 and β from 13.0 to 16.0. Next, we normalized the results against the maximum value so the normalized values ranged from 0 to 100. For video freeze events, a smaller number was translated into a higher normalized value because a low number indicated better performance. We defined the parameter metric used to combine these two normalized results as follows.

$$Parameter\ metric = w \times (normalized\ VF) + (1-w) \times (normalized\ SB) \quad (4)$$

In this study, we used w = 0.7 as the number of video freeze events, which was an important performance metric. **Fig. 6** shows the parameter metric results, which indicate that the performance was the best when we used 17.5 seconds for α and 14.5 seconds for β. Based on the results, we used these values in the following experiments.

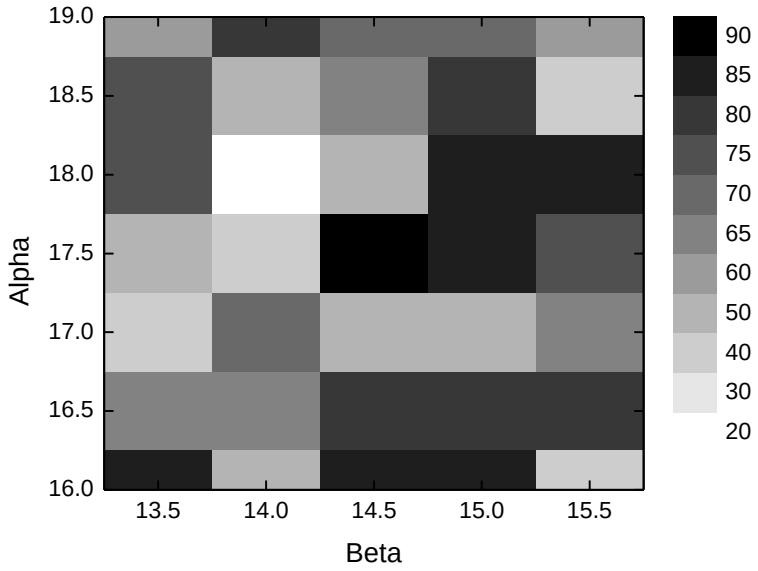


Fig. 6. Parameter metric with different values of α and β using the SLA scheme.

6.2 Video Freeze Time

Table 1 shows the video freeze counts when the bandwidth bottleneck was set to 300 Kbps and 500 Kbps. As shown in **Table 1**, the traditional RDA produced a longer freeze time because the cache size was smaller or the number of download trials was lower. For example, when the cache size was 2,000 and the bottleneck was 300 Kbps, which corresponded to 35% of the total interest packets, each download trial had a long freeze time. In particular, the traditional RDA attempted to select higher quality levels for the next chunk because it overestimated the network bandwidth, but the combination of (the small amount of) the cached data and the network bottleneck was not sufficient to support a higher bitrate encoding. For the third download trial, in particular, when the cache sizes were 2,500, 3,000, and 4,000, there were frequent overestimations, which produced the highest freeze times among these configurations. According to LFU, the data chunks that corresponded to the base layer and the

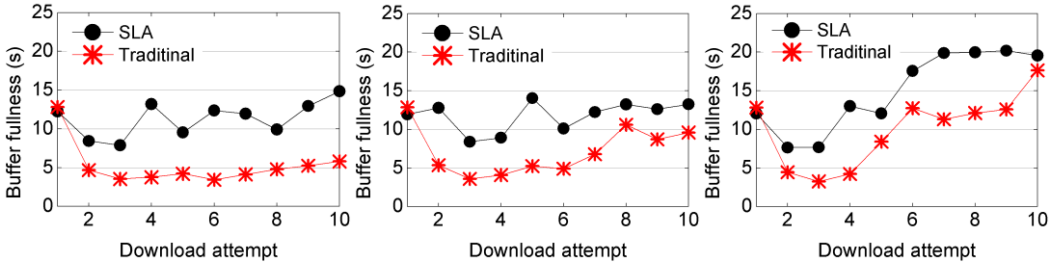
low quality enhancement layer were cached at the CCN nodes. By contrast, SLA used the drop timeout to avoid overestimation and there were almost no frozen scenes during the transmission of a single video chunk, depending on the current download state. Thus, SLA effectively alleviated the negative impact of overestimating the network bandwidth.

Table 1. Video freeze time (seconds)

Bottleneck	Cache Size	Scheme	Download attempt									
			1	2	3	4	5	6	7	8	9	10
300K	2000	Traditional	0	12	14	10	16	22	12	10	12	6
		SLA	0	0	0	0	0	0	0	0	0	0
	2500	Traditional	0	10	16	6	10	6	2	2	2	0
		SLA	0	0	0	0	0	4	0	0	0	0
	3000	Traditional	0	12	18	4	4	4	6	2	2	0
		SLA	0	0	2	0	0	0	0	0	0	0
	4000	Traditional	0	10	12	6	4	2	0	0	0	0
		SLA	0	0	0	0	0	0	0	0	0	0
500K	2000	Traditional	0	0	0	2	0	2	2	0	0	0
		SLA	0	0	0	0	0	0	0	0	0	0
	2500	Traditional	0	0	0	2	0	0	0	0	0	0
		SLA	0	0	0	0	0	0	0	0	0	0
	3000	Traditional	0	0	0	0	0	0	0	0	0	0
		SLA	0	0	0	0	0	0	0	0	0	0
	4000	Traditional	0	0	0	0	0	0	0	0	0	0
		SLA	0	0	0	0	0	0	0	0	0	0

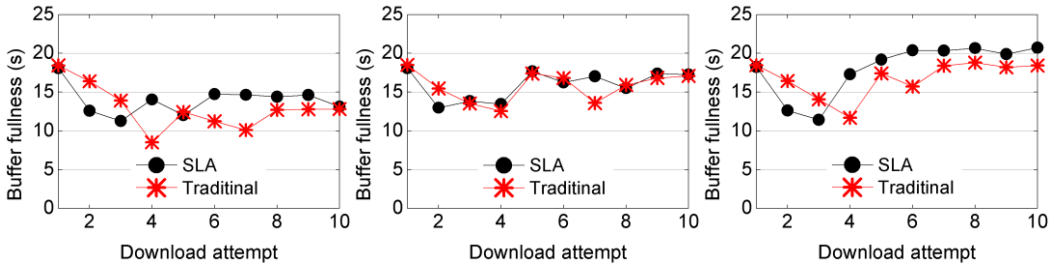
6.3 Buffer Fullness

The buffer fullness allows the absorption of fluctuations in the network bandwidth to provide better QoE. The SLA maintains a level of buffer fullness by setting the timeout for each hierarchy on the requested video chunk. As shown in **Fig. 7**, the buffer fullness was higher than the existing RDA in the second and later download trials for each cache size when the bandwidth bottleneck was set to 300 Kbps. Thus, higher buffer fullness was achieved even when a high dynamic network bandwidth was produced by cache hits. In all cases, the average deviation per trial was also smaller, which meant that the SLA was more stable. **Fig. 8** shows the case where the network bottleneck is set to 500 Kbps. Both RDAs had high buffer fullness because the throttling point was a slightly wider than that shown in **Fig. 7**. The download time of the top enhancement layer was shorter than the case shown in **Fig. 8** when the available network bandwidth was overestimated. Furthermore, the SLA selected different timeout policies depending on the current download state, so the buffer fullness differed in the following download trials. **Fig. 9** shows that there was a constant level of buffer fullness because of the wider network bottleneck and fewer changes in the bit rate. The number of hierarchies that were subject to timeouts was also small so there was almost no difference between the RDAs.



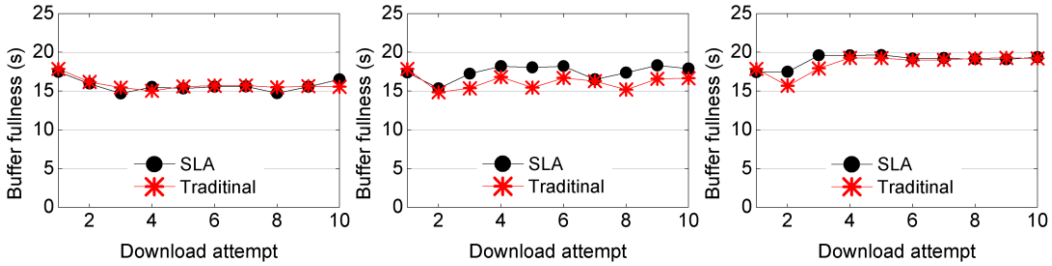
(a) Cache size = 2,000 (b) Cache size = 2,500 (c) Cache size = 3,000

Fig. 7. Average buffer fullness (bottleneck bandwidth = 300 Kbps).



(a) Cache size = 2,000 (b) Cache size = 2,500 (c) Cache size = 3,000

Fig. 8. Average buffer fullness (bottleneck bandwidth = 500Kbps).



(a) Cache size = 2,000 (b) Cache size = 2,500 (c) Cache size = 3,000

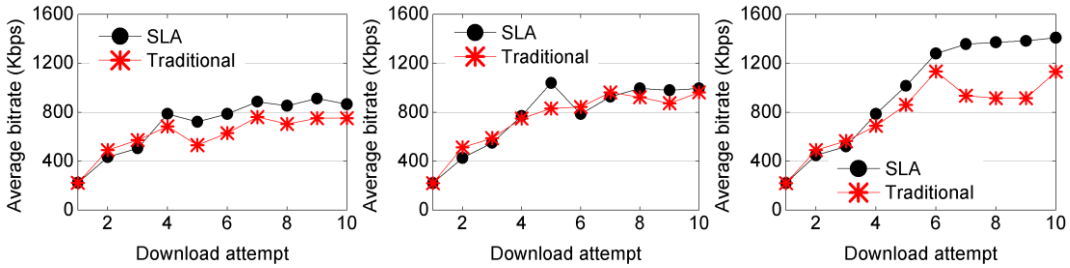
Fig. 9. Average buffer fullness (bottleneck bandwidth = 1,000Kbps).

6.4 Video Bitrate

Fig. 10–12 show the average selected bitrates with different bottleneck bandwidths. As described in Section 5, the cache size was set to 2,000, 2,500, or 3,000 content-objects and, using these values, the CCN router could store approximately 35%, 44%, or 52%, respectively of the total interest packets generated during the delivery of a single video. In each of these figures, the average bitrates for the first download attempt were the same for both schemes because we reset the caches at the start of each experiment.

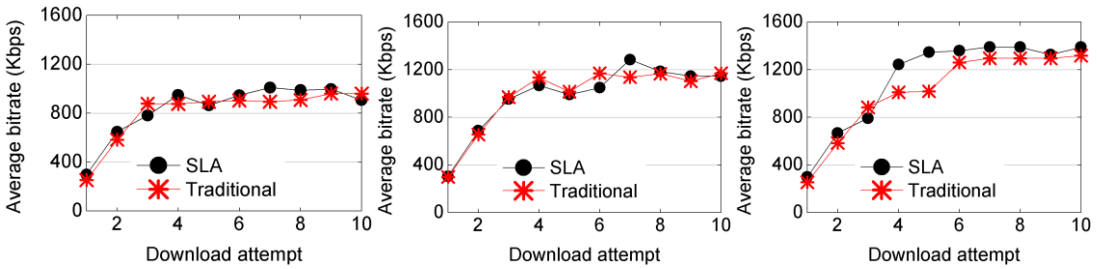
Fig. 10(a)–(c) show that the SLA delivered similar or better performance in terms of the average bitrate compared with the traditional scheme when the bottleneck bandwidth was 300 Kbps. In particular, **Fig. 10(c)** shows that the average bitrates with the traditional scheme at the 7^{th}, 8^{th}, and 9^{th} attempts were exceptionally low whereas with SLA, the curve increasing continuously as the download attempts increased. This is because the SLA dropped the highest quality enhancement layers based on the drop timeout, even when the cache hit rate was high, and it used two CCN caches (i.e., the two CCN nodes before the bottleneck link) so most of the base layers and lower enhancement layers were cached evenly over the two CCN nodes. In the remaining time before the timeout, the client downloaded the highest quality enhancement

layers from the server located behind the bottleneck link. Overall, the average bitrates with SLA shown in **Fig. 10(a)–(c)** converged to 0.8, 1.0, and 1.4 Mbps, respectively, as the download attempts increased. **Fig. 11** shows that the average bitrates were saturated at 1.0, 1.2, and 1.4 Mbps when the cache sizes were 2,000, 2,500, and 3,000, respectively. Thus, the client selected higher quality levels on average compared with **Fig. 10**, because the bottleneck bandwidth increased to 500 Kbps. When the bottleneck bandwidth was 1,000 Kbps, the high quality enhancement layers not present in the local caches were downloaded rapidly from the server, even when the download bandwidth was overestimated incorrectly on the client. Thus, drop timeouts occurred rarely with SLA because the bottleneck bandwidth (1,000 Kbps) covered the lower four quality levels (from 221 to 1,050 Kbps) directly. Therefore, there was no significant difference between the schemes in terms of the average bitrate, as shown in **Fig. 11**.



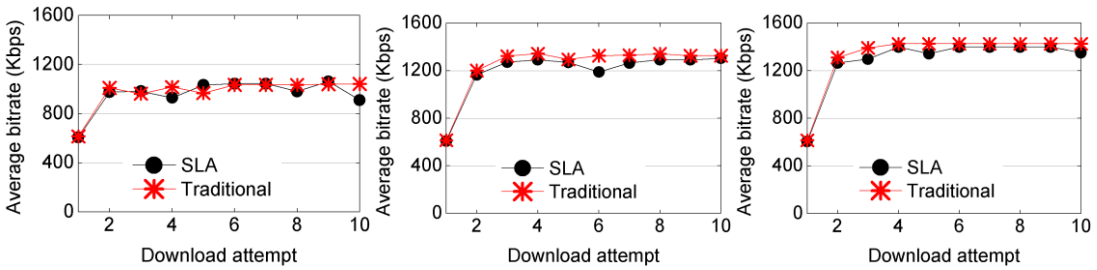
(a) Cache size = 2,000 (b) Cache size = 2,500 (c) Cache size = 3,000

Fig. 10. Average bitrate (bottleneck bandwidth = 300 Kbps).



(a) Cache size = 2,000 (b) Cache size = 2,500 (c) Cache size = 3,000

Fig. 11. Average bitrate (bottleneck bandwidth = 500 Kbps).



(a) Cache size = 2,000 (b) Cache size = 2,500 (c) Cache size = 3,000

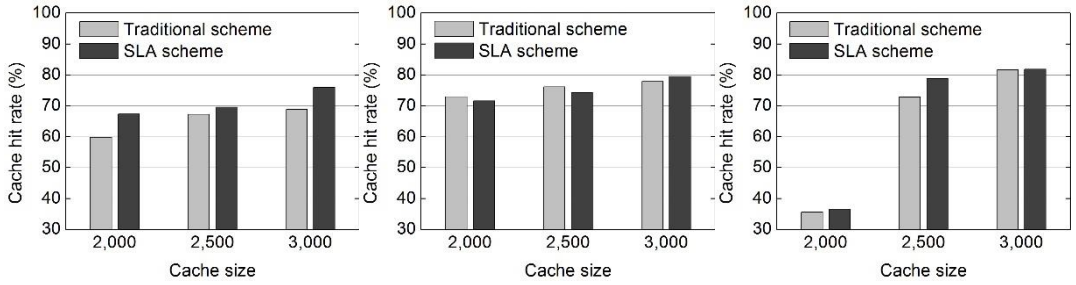
Fig. 12. Average bitrate (bottleneck bandwidth = 1,000 Kbps).

6.5 Cache Hit Rate

Fig. 13 shows the cache hit rates at the CCN nodes. **Fig. 13(a)** shows that the SLA performed better in terms of the cache hit rate when the bottleneck bandwidth was relatively small. Our

scheme requested a smaller number of enhancement layers several times so those chunks were well positioned in the caches with LFU, whereas the traditional scheme caused cache pollution by unnecessarily requesting high quality video chunks. This also resulted in frequent video freezes, as shown in Section 6.1. **Fig. 13(b)** shows that there was a higher cache hit rate on average with all cache sizes compared with **Fig. 13(a)**. It appears that both schemes utilized the caches well with the increased network bandwidth and they exhibited similar performance in this environment, as shown in **Fig. 10** and **Fig. 13(b)**.

Fig. 13(c) shows that the cache hit rate was very low at <40% when the cache size was 2,000. This was mainly because the client tried to request the highest quality video chunks based on the large network bandwidth. The cache size was not sufficient to cover such high quality video chunks (i.e., large chunks) and some chunks were removed continuously from the cache during video streaming, thereby resulting in a low cache hit rate. This phenomenon was mitigated by increasing the cache size, as shown in the figure, and we can see that the SLA had a higher hit rate when the cache size was 2,500. The traditional scheme immediately requested the highest bitrate from the start of the video stream, whereas SLA attempted to adjust the requested bitrate so the lower enhancement layers were cached steadily, rather than caching the highest quality enhancement layer. This resulted in a higher cache hit rate in **Fig. 13(c)** and a slightly lower bitrate in **Fig. 12(b)–(c)**.



(a) Bottleneck = 300Kbps (b) Bottleneck = 500Kbps (c) Bottleneck = 1,000Kbps

Fig. 13. Comparison of the cache hit rate and cache size.

7. Conclusion

In this study, we developed a new bitrate adaptation algorithm for SVC-HAS, which comprised bitrate selection and SLA. Our main finding was that the SVC-HAS client experienced frequent video freeze events in CCN environments. This was because the bitrate selection algorithm sometimes overestimated the network bandwidth because the base layers of video chunks were downloaded from the cache of the local CCN routers. Therefore, the client unnecessarily downloaded enhancement layers that were not cached, thereby draining the video buffer close to zero. To address this problem, we introduced a drop timer, which stopped the downloading of additional enhancement layers in a timely manner when the timer expired and moved onto the next chunk request. We implemented this method in the SVC-HAS client and deployed it in our CCN testbed for the performance evaluation. Our experiments confirmed that the SLA reduced the video freeze time significantly. We also found that our scheme delivered similar or better performance in terms of the average buffer fullness, average bitrate, and cache hit rate with different network parameters, network bandwidths, and cache sizes.

References

[1] Microsoft Smooth Streaming. http://www.iis.net/download/smoothstreaming.

[2] Apple. HTTP Live Streaming. http://developer.apple.com/resources/http-streaming.

[3] Adobe. HTTP Dynamic Streaming on the Adobe Flash Platform. http://www.adobe.com/products/httpdynamicstreaming.

[4] T. Stockhammer, P. Fröjdh, I. Sodagar, S. Rhyu, "Information technology MPEG systems technologies part 6: Dynamic adaptive streaming over HTTP (DASH)," ISO/IEC, MPEG Draft International Standard, 2011.

[5] T. Stockhammer, "Dynamic Adaptive Streaming over HTTP–: Standards and Design Principles," in *Proc. of ACM MMSys*, 2011. Article (CrossRef Link)

[6] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, R. L. Braynard, "Networking Named Content," in *Proc. of ACM CoNEXT*, 2009. Article (CrossRef Link)

[7] R. Huysegems, B. De Vleeschauwer, T. Wu, W. Van Leekwijck, "SVC-based HTTP Adaptive Streaming," *Bell Labs Technical Journal*, vol. 16, no. 4, pp. 25-41, March, 2012. Article (CrossRef Link)

[8] Y. Sanchez, T. Schierl, C. Hellge, T. Wiegand, D. Hong, D. De Vleeschauwer, W. Van Leekwijck, Y. Le Louédec, "Improved caching for HTTP-based Video on Demand using Scalable Video Coding," in *Proc. of IEEE CCNC*, 2011. Article (CrossRef Link)

[9] Y. Sanchez, T. Schierl, C. Hellge, T. Wiegand, D. Hong, D. De Vleeschauwer, W. Van Leekwijck, Y. Le Louédec, "iDASH: improved dynamic adaptive streaming over HTTP using scalable video coding," in *Proc. of ACM MMSys*, 2011. Article (CrossRef Link)

[10] T. Wiegand, G. Sullivan, H. Schwarz, M. Wien, "ISO/IEC 14496-10: 2005/AMD3: Scalable video coding," International Standardization Organization, 2007.

[11] CCNx. http://www.ccnx.org.

[12] H. Schwarz, D. Marpe, T. Wiegand, "Overview of the Scalable Video Coding Extension of the H. 264/AVC Standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 17, no. 9, pp. 1103-1120, September, 2007. Article (CrossRef Link)

[13] O. Abboud, K. Pussep, A. Kovacevic, R. Steinmetz, "Quality Adaptive Peer-to-Peer Streaming Using Scalable Video Coding," in *Proc. of Wired-Wireless Multimedia Networks and Services Management*, 2009. Article (CrossRef Link)

[14] C. H. Ke, "myEvalSVC-an Integrated Simulation Framework for Evaluation of H. 264/SVC Transmission," *KSII Transactions on Internet and Information Systems*, vol. 6, no. 1, pp. 379-394, January, 2012. Article (CrossRef Link)

[15] M. Mushtaq, T. Ahmed, "Smooth Video Delivery for SVC based Media Streaming over P2P Networks," in *Proc. of IEEE CCNC*, 2008. Article (CrossRef Link)

[16] T. Schierl, Y. S. de la Fuente, R. Globisch, C. Hellge, T. Wiegand, "Priority-based Media Delivery using SVC with RTP and HTTP streaming," *Multimedia Tools and Applications*, vol. 55, no. 2, pp. 227-246, November, 2011. Article (CrossRef Link)

[17] D. Cheriton, M. Gritter, "TRIAD: A New Next-Generation Internet Architecture," 2000.

[18] M. Caesar, T. Condie, J. Kannan, K. Lakshminarayanan, I. Stoica, "ROFL: Routing on Flat Labels," in *Proc. of ACM SIGCOMM*, 2006. Article (CrossRef Link)

[19] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, S. Surana, "Internet Indirection Infrastructure," *IEEE/ACM Transactions on Networking*, vol. 12, no. 2, pp. 205-218, April, 2004. Article (CrossRef Link)

[20] C. Kim, M. Caesar, J. Rexford, "Floodless in SEATTLE: A Scalable Ethernet Architecture for Large Enterprises," in *Proc. of ACM SIGCOMM*, 2008. Article (CrossRef Link)

[21] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. Kim, S. Shenker, I. Stoica, "A Data-Oriented (and Beyond) Network Architecture," in *Proc. of ACM SIGCOMM*, 2008. Article (CrossRef Link)

[22] Y. Sanchez, T. Schierl, C. Hellge, T. Wiegand, D. Hong, D. De Vleeschauwer, W. Van Leekwijck, Y. Le Louédec, "Efficient HTTP-based streaming using Scalable Video Coding," *Signal*

Processing: Image Communication, vol. 27, no. 4, pp. 329-342, April, 2011. Article (CrossRef Link)

[23] D. Hong, D. De Vleeschauwer, F. Baccelli, "A chunk-based caching algorithm for streaming video," *NET-COOP 2010 - 4th Workshop on Network Control and Optimization*, 2010. http://hal.archives-ouvertes.fr/inria-00597186/

[24] Smooth Streaming Module for Apache. http://smoothstreaming.code-shop.com/trac/wiki/Mod-Smooth-Streaming-Apache.

[25] IIS Smooth Streaming HD Sample Content. http://www.microsoft.com/download/en/details.aspx?id=18199.

[26] J. Reichel, H. Schwarz, M. Wien, "Joint scalable video model 9 (JSVM-9)," IEC JTC1/SC29/WG11 and ITU-T SG16 Q.6, Doc. JVT-V202, 2007.

[27] Dummynet. http://info.iet.unipi.it/luigi/dummynet/

[28] S. Podlipnig, L. Böszörmenyi, "A Survey of Web Cache Replacement Strategies," *ACM Computing Surveys*, vol. 35, no. 4, pp. 374-398, December, 2003. Article (CrossRef Link)

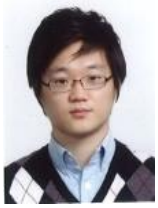
[29] S. Akhshabi, A. C. Begen, and C. Dovrolis, "An Experimental Evaluation of Rate-Adaptation Algorithms in Adaptive Streaming over HTTP," in *Proc. of ACM MMSys*, 2011. Article (CrossRef Link)

[30] T. Andelin, V. Chetty, D. Harbaugh, S. Warnick, and D. Zappala, "Quality Selection for Dynamic Adaptive Streaming over HTTP with Scalable Video Coding," in *Proc. of ACM MMSys*, 2012. Article (CrossRef Link)

[31] J. Famaey, S. Latre, N. Bouten, W. V. de Meerssche, B. D. Vleeschauwer, W. V. Leekwijck, and F. D. Turck, "On the merits of SVC-based HTTP Adaptive Streaming," in *Proc. of IFIP/IEEE International Symposium on Integrated Network Management*, 2013.

JungHwan Lee received a B.S. degree in computer science from Sungkyul University, Korea in 2007 and M.S degree computer science from Korea University. He is currently a Ph.D. candidate in the Department of Computer Science and Engineering at Korea University, Seoul, Korea. His research interests include scalable video codec, contents centric network and adaptive video streaming.

Jaehyun Hwang received the B.S. degree in computer science from Catholic University of Korea, Seoul, Korea in 2003, and the M.S. and Ph.D. in computer science from Korea University, Seoul, Korea in 2005 and 2010, respectively. His research backgrounds are mainly in TCP, focusing on a flexible TCP structure, advanced TCP flavors and their performance. Since September 2010, he has been with the networking research domain at Bell Labs, Alcatel-Lucent, Seoul, Korea as a Member of Technical Staff. His current research interests include data center networks, HTTP Adaptive Streaming, and Content-Centric Networking.



Nakjung Choi received the B.S. and Ph.D. degrees in computer science and engineering from Seoul National University (SNU), Seoul, Korea, in 2002 and 2009, respectively. From September 2009 to April 2010, he was a postdoctoral research fellow in the Multimedia and Mobile Communications Laboratory, SNU. Since April 2010, he is a member of technical staff at Bell Labs, Alcatel-Lucent, Korea. His research interests are mobile networks, software defined networking, information centric networking, and green networking.

Chuck Yoo received BS and MS degrees in electronic engineering from Seoul National University, Seoul, Korea, and an MS degree and PhD in computer science from University of Michigan. He worked as a researcher in Sun Microsystems Laboratory from 1990 to 1995. He is now a professor in the College of Information and Communications, Korea University, Seoul, Korea since 1995. His research interests include operating systems, embedded system, virtualization, and multimedia streaming. Prof. Yoo is a member of the IEEE, the IEEE Computer Society, and the ACM.