Replication Algorithms to Retrieve Scalable Streaming Media over Content Delivery Networks

Zhou Su, Jiro Katto and Yasuhiko Yasuda

School of Science and Engineering, Waseda University
3-4-1, Ohkubo, Shinjuku-ku, Tokyo 169-0072, Japan
Contact Email: suzhou@aoni.waseda.jp

ABSTRACT

CDN (Content Delivery Networks) improves end-user performance by replicating web contents on a group of geographically distributed content servers. Replication Algorithm plays an important role in helping users to retrieve Web objects from the content servers. If a user can directly get the requested objects from the content server, he need not to contact the remote origin server and the user delay can be reduced. However, current replica strategies in CDN are to simply and repeatedly keep the complete replica of the original object on many content servers. This method has some disadvantages, including too much consumed server space and a waste of the storage cost. It is more serious for replicating some large-sized objects such as streaming media, which are being distributed over the Internet more and more.

In this paper, we discuss a replication strategy for scalable video streaming in CDN to reduce user response and storage cost. Based on theoretical analysis, assuming layered video coding, we propose a novel replication algorithm which deals with following three problems. (1) How many content servers should be selected to replicate a given video content? (2) For a single video content, how many layers should be kept in a given content server? (3) After selecting a group of content servers for each video content, how do we decide the replication priority for each content server? Simulation results show that the proposed algorithm can efficiently resolve the above problems, and provide much better performance than the conventional methods.

Categories and Subject Descriptors
H.3.3 **[Information Search and Retrieval**]: Retrieval models.

General Terms
Algorithms.

Keywords
Content Distribution Networks, Replication Algorithm, Scalable. Streaming, Web Performance, Network Traffic

1. INTRODUCTION

Replication strategies are about how to efficiently store the replica of Web objects in some content servers which have limited capacity. When a user 's requested object can be retrieved and provided by keeping its replica in a nearby content server, the user need not to contact the remote origin server. Then, the user delay can be reduced. Therefore, the appropriate placement of server replicas benefits content providers by reducing latency for their clients, and benefits ISPs by reducing bandwidth consumption and transmission cost.

The current content replication is to simply replicate the whole original data into several content servers. Disadvantages of this method are as follows: to repeatedly store the same large-sized object into different content servers consumes too much server space. Also, because some of content servers are not always requested by the clients, to keep replicas on these servers causes a waste of storage cost.

These problems become more serious for streaming media, which has several inherent properties. (1) The size of streaming media is usually larger than non-streaming files by orders of magnitude [10]. (2) User access behavior shows quite different characteristics. For example, clients often stop watching a stream without watching all of the parts [11]. (3) Different from conventional web objects, streaming media do not require to be delivered at once. Instead, streaming servers continuously send data packets to clients in a (quasi) synchronized manner on the Internet. For these reasons, the replication mechanisms developed for conventional web objects such as HTML files or images can't be efficiently applied to streaming media such as video and audio.

Recently, researchers found that scalable (layered) video streaming is appropriate for the Internet because of its better flexibility and functionality. In this paper, we therefore discuss a scheme to replicate scalable streaming contents in CDN. Figure 1 illustrates our CDN architecture, where different layers (quality) of streaming media are stored in a group of content servers according to stream/server popularity and server location. In this paper, our work focuses on how to replicate *different* layers of *different* streams into *different* content servers, in order to save system resources and improve user response time simultaneously.

We firstly carry out theoretical analysis of storage cost, capacity limit and access distribution of scalable video streaming. Based on this analysis, we then propose a replication algorithm in which not only popularities of streams and content servers but server location are considered. Through simulations, we check the performance of our proposal by changing related parameters. Simulation results show that our proposal can efficiently minimize user response time and network traffic. Our proposal is also stable against varying conditions even if access patterns are dynamically changed.

This paper is organized as follows: in Sect.2, related work with regard to stream replication algorithms is reviewed. In Sect.3, an overview of the proposed system architecture is provided, where scalable streaming is applied to replicate streaming media over CDN. Sect.4 presents the mathematical analysis of Web access, server storage cost and capacity limit, then our proposed algorithm is given later. Simulation results are given in Sect.5 and conclusions are shown in Sect.6.

2. PREVIOUS WORK

We briefly discuss previous work as follows:

How to place replicas on multiple servers assuming a tree topology (i.e. a root is an origin server) has been discussed in [13] and [14]. Since the actual topology is not limited to be a tree, these approaches are not necessarily suitable for CDN. In [15], the authors presented an algorithm for placing replicas in a CDN. However, they assumed that all replicas act independently and cooperative schemes have not been studied.

In [12], replication decisions on a per-object granularity are discussed. A useful cost model for replication is presented. However, the replication object is just the normal Web object such as HTML and image files. Since the video streaming has distinct statistical properties and different user access patterns, their strategy cannot be directly applied. Furthermore, they just discussed relations between a given object and its related node. How to reduce user response time based on relations among CDN-nodes, nor how to efficiently replicate them among cooperative content servers had not been considered.

There have been many other published studies on streaming media. Sen et al showed that storing a prefix (i.e., the initial part) of a stream at the proxy can hide the potentially large initial start-up delays of work-ahead transmission schedule from the clients [16]. Lee et al proposed a scheme that provided users with “video summary” (a number of key-frames) before they download original stream files [17]. [8] and [18] studied distribution of layered encoded video through caches, and [19] discussed how to cache MPEG-2 video with a goal of video quality adjustment. However, all of these researches focused on how to replace or cache streaming contents in a single proxy cache. How to efficiently replicate and distribute streaming contents in a group of servers has not been mentioned.

We ourselves proposed an integrated pre-fetching and replacing algorithm for the hierarchical (graceful) image based on a cooperative proxy-server model, in which efficiency of hierarchical image caching was proved [23]. We also presented a scheme for stream caching by using hierarchically distributed proxies with adaptive segments assignment, in which “segment” meant a group of pictures [22]. This method clarified effectiveness of “local-scope” server cooperation with per-segment management. However, the former did not deal with scalable (hierarchical) video streaming, and the latter left the “global-scope” server cooperation for distributed streaming contents with per-layer management, instead of per-segment management, as a future work.

3. OVERVIEW OF CDN AND SCALABLE STREAMING

3.1 Content Delivery Networks

With the growth in popularity of the Internet and the wide availability of high-speed networks, Web content providers find that it is difficult to serve all users with low response time, especially in the face of high loads. In recent years, how to set up contents distribution networks to efficiently distribute stored information has become a major concern in the Internet.

In popular P2P networks, users can determine where different files can be downloaded with the help of a directory service [5][6]. Similar ideas are expanded as the concept of “overlay network” [9], where each connection in the overlay is mapped onto paths in the underlying physical network.

Content delivery networks (CDNs) appeared recently and are deploying quite rapidly [1]-[4]. Load balancing by request routing, efficient content delivery by locating edge servers near to clients and information exchange protocols among different CDN sites are developed. However, their concern is mainly placed on efficient delivery of static content, i.e. HTML files and images. Some CDN companies advocate their streaming caching support, but their technical details are not yet clarified nor verified.

3.2 Scalable Streaming

Researchers and engineers have argued that scalable (layered) video streaming is appropriate for the Internet because of its better flexibility and functionality [8]. Basically, scalable streaming assumes scalable coding, in which the original signal is coded into several layers, from the lowest (base) to highest (enhancement) layers. This scalable coding is supported by some video compression standards, such as *H.263* and *MPEG-2*. *MPEG-4* specifies an improved scalable coding method, referred as fine granular scalability (FGS) [7]. Then, the scalable streaming transmits each layer over different channels (different protocols, different error correction codes, or different paths).

There are some advantages in this scalable streaming.

Quality adaptive

The compression ratio of the scalable streaming data can be easily changed according to available bandwidth, e.g., when the network congestion or delay get worsen, the sender can send few number of layers to maintain the continuous playback at a receiver; conversely the number of layers can be increased when network conditions become better.

Save server resources

If we keep only some layers of the stream instead of the whole original one in the content server when this stream is not popular, then it can give other popular stream more space.

Efficient for caching

More kinds of streaming data can be kept in a cache, when necessary. Higher layers (enhancement layers) can be discarded with keeping lower layers (base layers) instead of removing whole data when there is no sufficient space in the cache.

3.3 Scalable Streaming in CDN

In this paper we discuss scalable streaming in CDN. In our system, we assume the original streams are encoded into several layers according hierarchical encoding format. The layered streams are then replicated into content servers in CDN. When a client requests one video stream, he can request either only base layers or complete layers (consisting of base layers and enhancement layers). The main work in this paper is to propose a hierarchical replication algorithm to replicate different layers of different streams into different servers in order to improve whole system performance of CDN.

4. THEORETICAL ANALYSIS

In this section, we give an analysis of replicating scalable streaming media over the Content Delivery Network. Firstly, the theory analysis of storage capacity, storage cost and average hop count are presented from Sect.4.1 to Sect.4.3, respectively. Then, based on the above results of theory analysis, How to reduce the user response time under the same constraints is discussed in Sect.4.4. The proposed replication algorithm is introduced in Sect.4.5.Finally, how to reduce the computational complexity is discussed in Sect.4.6.

4.1 Definition of Storage Capacity

We assume that each content server is located in a different administrative domain, such as autonomous system (AS). Let S_i (bytes) denote storage capacity of a server in domain i (i=1,…, I), and λ_i (bytes/second) denote an aggregate request rate from clients to the server. We assume that there are J streaming contents in our CDN, and each content is encoded by Q layers. Let us define a parameter $B_{q,j}$ as data size of the q-th layer (q=1,…,Q) of stream j (j=1,…, J), and define a parameter $P_{q,j}$ as a request probability for the q-th layer of stream j (i.e. stream popularity), respectively. We define a parameter $X_{i,q,j}$, which takes a binary value of

$$X_{i,q,j} = 1 \text{ (if the } q\text{-th layer of stream } j \text{ is stored in server } i\text{)},$$
$$X_{i,q,j} = 0 \text{ (otherwise).} \tag{1}$$

We also define a three-dimensional matrix $\boldsymbol{X}$ of which element is $X_{i,q,j}$, that represents a placement pattern of streaming contents. Storage capacity S_i is then constrained by defined parameters,

$$\sum_q \sum_j B_{q,j} \cdot X_{i,q,j} \le S_i \qquad (i = 1, \Lambda, I) \tag{2}$$

In this paper, our goal is to choose an optimum parameter set $\{X_{i,q,j}\}$in order to minimize user response time under this capacity constraint of each server and the cost constraint of whole CDN defined in next subsection.

4.2 Definition of Storage Cost

We then consider a storage cost of streaming contents in a server, which is associated with how much and how long the storage is used (such a "storage utility" model is in fact being offered by some companies, in which one can get storage whenever he needs and pays only for what he uses). If we take into consideration the cost of data redundancy for fault tolerance (e.g., by means of mirroring or error checking/correcting overheads) and the cost of server streams (which may be a function of the used storage), the storage cost would be higher.

Let us calculate the storage cost when we store the q-th layer of stream j in server i . We assume that each stream will be storaged in the server for a fixed period, and then cached contents will be updated periodically. Let M_i (\$/byte) denote storage cost for a given server i, and α represent a total expense limit of our CDN. Then, we can get:

$$\sum_i \sum_q \sum_j M_i \cdot B_{q,j} \cdot X_{i,q,j} \le \alpha \qquad (i = 1, \Lambda, I) \tag{3}$$

A parameter M_i can be considered as a priority parameter (or weighting factor) of each content server. Note that, though Eq.(3) can be considered as generalization of storage capacity in Eq.(2), it is evaluated for whole CDN, instead of each server.

4.3 Definition of Average Hop Count

Let us define an average hop count $T_i(\boldsymbol{X})$, that represents a traverse of requests from server i to a server having a requested content by

$$T_i(\boldsymbol{X}) = \sum_q \sum_j P_{q,j} \cdot D_{i,s(q,j)}(\boldsymbol{X}) \tag{4}$$

where a parameter $D_{i,s(q,j)}(\boldsymbol{X})$ denotes the shortest distance (hop count) from domain i to server $s(q, j)$ storing the q-th layer of stream j under the placement $\boldsymbol{X}$. This nearest copy is stored either in an origin server of the stream or in other servers where the layer has been replicated. We assume that clients are always redirected to the nearest copy (i.e. the optimum server) by some "request routing" mechanisms, though their details are out of scope of this paper. Note also that a parameter $D_{i,s(q,j)}(\boldsymbol{X})$ may deonote other QoS parameters such as delays and throughputs.

Let $\Lambda = \sum_i \lambda_i$ be the total request rate from all the domains. Let $T(\boldsymbol{X})$ be the average number of hops from all the domains weighted by "server popularity" λ_i / Λ . This is given by

$$\begin{aligned} T(\boldsymbol{X}) &= \frac{1}{\Lambda}\sum_i \lambda_i T_i(\boldsymbol{X}) \\ &= \frac{1}{\Lambda}\sum_i\sum_q\sum_j \lambda_i P_{q,j}\cdot D_{i,s(q,j)}(\boldsymbol{X}) \\ &= \sum_i\sum_q\sum_j S_{i,q,j}\cdot D_{i,s(q,j)}(\boldsymbol{X}) \end{aligned} \tag{5}$$

where

$$S_{i,q,j} = \frac{\lambda_i}{\Lambda}\cdot P_{q,j}\,. \tag{6}$$

Eq.(6) represents a request probability weighted by aggregation ratio to server *i* (i.e. joint probability of stream popularity and server popularity). Here, the placement ***X*** should be subjected to two constraints. One constraint is capacity limitation specified in Eq.(2), and another constraint is storage cost defined in Eq.(3).

4.4 Minimization of Traversed Hop Count

An average hop count that a request must traverse almost reflects the download time of an object and can be used as an indicator of user perceived latency. Some papers utilize this traversed hop count as an important and stable criterion to evaluate the performance of CDN or P2P networks [6] [14]. In this paper, our goal is to optimally choose the $X_{i,q,j}$ to provide requested streams to clients as quickly as possible under the constraints of node capacity and storage cost. For these constraints, we already formulated Eq.(2) and Eq.(3). Then, the main problem is how to actually reduce hop counts under these constraints.

If the *q*-th layer of stream *j* is replicated in server *i*, we can obtain the averagely reduced hop count $\Delta T_{i,q,j}$ as follows:

$$\Delta T_{i,q,j} = \frac{1}{\Delta}\cdot P_{q,j}\cdot\sum_k \lambda_k\cdot\left\{D_{k,o(q,j)}(\boldsymbol{X}_o) - D_{k,i}(\boldsymbol{X})\right\}, \tag{7}$$

where $\boldsymbol{X}_0$ is an initial placement of streaming contents, in which all the streams are stored in their origin servers only, that are denoted by $o(q, j)$, and ***X*** is their current placement after keeping a replica of the *q*-th layer of stream *j* into server *i*. A parameter $D_{k,o(q,j)}(\boldsymbol{X}_0)$ represents the shortest distance from server *k* to the origin server $o(q, j)$ under placement X_0, and a parameter $D_{k,i}(\boldsymbol{X})$ does the shortest distance from server *k* to server *i* having replication under placement ***X***, respectively. The proof of Eq.(7) is given in Appendix.

4.5 Proposed Algorithm

Here, we present a replication algorithm to resolve following three problems: (1) How do we decide a replication priority of each streaming content which needs to be replicated? (2) How many servers should be selected to replicate a given streaming content? (3) For a given streaming content, how many layers should be selected for it to be replicated? We fomulated these problems into an optimization problem: under the constraint of Eq.(2) and Eq.(3), minimize the averaged hop count of Eq.(7). To solve this, we give an iterative algorithm as follows.

1) For a given streaming content with layered representation (the *q*-th layer of stream *j*), $\Delta T_{i,q,j}$ in Eq.(7) is calculated. This is also provided to each content server as a "replication priority".

2) A "layer-server pair" which has the highest $\Delta T_{i,q,j}$ is picked up and a layer (*q, j*) is stored in server *i*. This action results in a new placement ***X***. $\Delta T_{i,q,j}$ is recalculated under this new placement and a new layer-server pair with the highest $\Delta T_{i,q,j}$ is selected.

3) The above process is iteratively carried out until either server capacity constraint in Eq.(2) or storage cost constraint in Eq.(3) exceeds its limit.

Here, we give some explanation about our algorithm. Because the replication priority $\Delta T_{i,q,j}$ represents how many hop counts can be reduced after storing a layer in a server, the average hop count can be minimized if we decide the replication order according to Eq.(7). Because the system selects the layer-server pair under the constraint of storage capacity and storage cost, higher priority layers of streams will be firstly stored. Higher priority servers will be also given a priority to store layers. When the constraint exceeds the limit, content replication will be stopped. At that time, for a given streaming content, how many layers should be replicated and how many content servers should be selected are decided simultaneously.

4.6 Consideration on Computational Complexity

Computational complexity of our replication algorithm mainly depends on the size of three-dimensional matrix ***X***, that is given by $I\times Q\times J$. Therefore, when there are numerous streams in a server (when *J* is large), it is unrealistic to manage all streams' replication to other servers. Also, since the scale of CDN is being increased recently (since *I* becomes larger), to manage all content servers' replication will cause a great amount of computational complexity in our CDN.

Fortunately, previous researches showed that the distribution of web requests from a fixed group of users follows a Zipf-like distribution. It has been proved that most web requests to a server are for a very small set of objects, for example top 10 %. Because of this property, it is enough to only manage the aforesaid set of streams. Other recent studies also showed that client load is heavily skewed towards popular servers. In [1], it had been found that 80% of the requests to streaming media were served by only top 4% most popular servers. Therefore, to reduce computation complexity of our CDN, it is suggested that we only need to care about popular servers and popular streams.

Let F_i and F_j represent access frequency of server *i* and stream *j* in last period, respectively. Then we apply

- If F_i or F_j exceeds a pre-specified threshold, the system will execute the iterative replication algorithm (matrix operation) for the coming period.
- If not, the system just keeps access records and does not carry out any calculation.

The threshold is an access frequency of the stream (or the server) of which ranking is Top 10, for example. Accordingly, the system only needs to manage a very small set of streams and servers, leading to reduction of computation complexity.

5. PERFORMANCE EVALUATION

In this section numerical results will be presented by simulation experiments to validate the proposed algorithm.

5.1 Simulation Conditions

In simulation experiments, we assume following conditions as an example. There are 21 nodes (servers) in our network simulator. Among these nodes, there are 15 original servers and 6 content cache servers, respectively. Physical distances among nodes are decided at random. As for the streaming contents, there are 1000 different streams with the rate of 384kbps. The length of each stream is uniformly distributed from one minute to ten minutes. Each stream is encoded into 2 constant bit rate layers [3]. In our simulation, clients often stop watching a streaming content after playback. The position where a client stops watching a stream is decided at random. Several researchers have observed that the distribution of web request from a fixed group of users follows a Zipf distribution, which states that the relative probability of a request for the *i*'th most popular page is proportional to Ω/r_i^{α}. Besides, the value of α , a parameter of Zipf distribution, varies from trace to trace, ranging from 0.64 to 0.83 [20][21]. Client requests arrive according to a Poisson process. For each stream, a client requests either a base layer only or a complete video consisting of a base layer and an enhancement layer. All clients are always redirected to the closest server without failure of request routing. The total request times in the simulations are 10000.

There are four replication algorithms we will study:

- *LRU (Least Recently Used) Policy*
- *LFU (Least Frequently Used) Policy*
- *Hierarchical Caching* [12]
- *Proposal*

The former two are conventional ones, in which we assigned *LRU* or *LFU* to one whole stream, corresponding to conventional CDN. Note that *LRU* and *LFU* can also be utilized for other content management such as cache replacement. The *Hierarchical Caching* [12], which can be used for scalable stream replication, was firstly proposed for hierarchical image databases. The original idea of this algorithm is to store the most accessed layer of different images (streams) in a cache server based on client's different access patterns. Then, we compared these four algorithms.

5.2 Simulation Results

In Figure 1, average hop counts are used to evaluate system performance. We define the average hop counts as a ratio of the number of traversed hop counts to the number of total request times. Average hop counts reflect content delivery time and thus can be used as an indicator of user perceived latency.

From this figure, we can find that the proposed algorithm achieved the fewest hop counts than the other ones. Although the *Hierarchical Caching* shows better performance than *LRU* and *LFU*, its hop counts are still fewer than our proposal because it doesn't exploit server location (hop counts between servers). We can also find that the proposed method substantially reduces the average hop counts by almost 50% compared with *LFU*. The reason is because the proposal decides whether a layer of one stream should be replicated in a content server by considering many aspects such as server popularity, stream request frequency (stream popularity) and server location (hop counts). However, *LFU* only considers stream request frequency. This reason also results in an effective utilization of system resources when the parameter of Zipf distribution is increased. For our proposal, the number of hop counts keeps being decreased with the parameter of Zipf distribution increased.

In Figure 2, since network traffic reflects bandwidth consumption and transmission cost, which is related to user perceived latency, we evaluate the effect of network traffic among different servers when the parameter of Zipf distribution is changed. From this figure, we can find that the proposed algorithm performs best and reduces network traffic most since this algorithm takes both content popularity and server location into consideration. It also verifies that algorithms with low hop counts lead to reduction of network traffics.

In Figure 4, a hit ratio is shown with respect to the parameter of Zipf distribution. We define the hit ratio in CDN as a usual manner; when a client requests a content and the content is available in a server which is located near to the client, this client needs not to wait for the requested content to be delivered from the original server, so called a *hit*. Under this measure, the proposed algorithm also performs best against the other three methods; *LRU*, *LFU* and *Hierarchical Caching*.

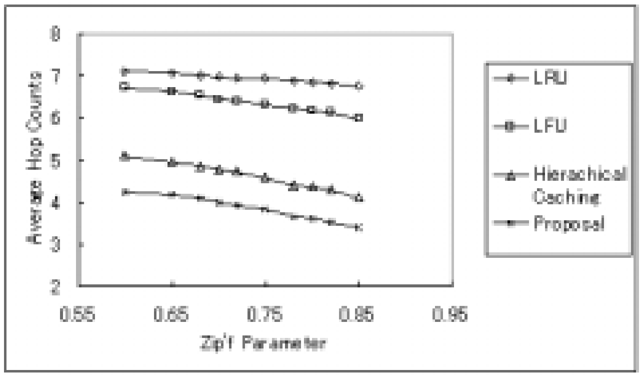

Fig. 1: Average As Hops under Different Zipf Parameters

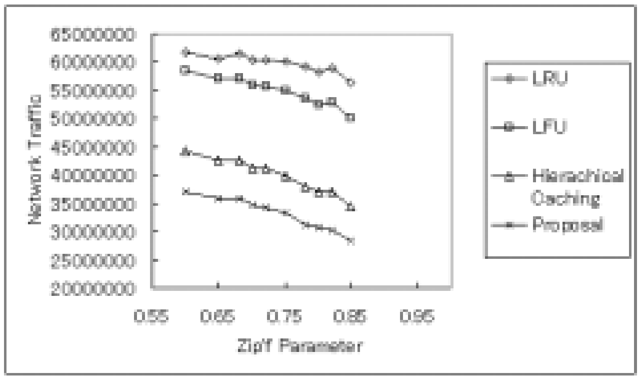

Fig. 2: Network Traffic under Different Zipf Parameters

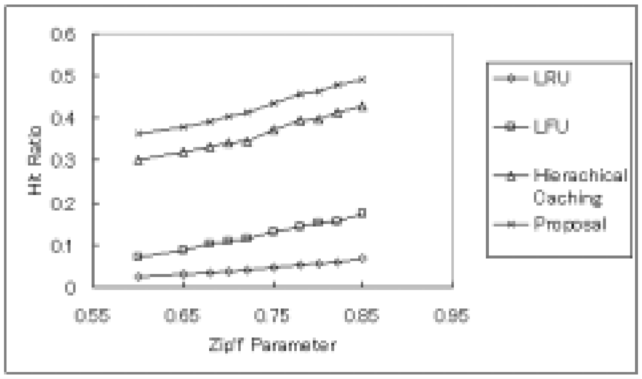

Fig. 3: Hit Ratio under Different Zipf Parameters

In conclusions from these figures, it is proved that the proposed algorithm achieves the best performance under various measures; hop counts, network traffic and cache hit ratio.

6. CONCLUSIONS

This paper discussed how to optimally replicate scalable streaming contents onto CDN servers and presented an efficient scheme to replicate them without wasting content servers' resources. Based on mathematical analysis, we proposed a novel algorithm to minimize average hop counts over traversed domains where the scalable streams are delivered. Our proposal dealt with not only popularities of streams and servers but also server location. We then compared our proposal with other conventional methods using computer simulations. Simulation results showed that the proposed method substantially reduced average number of hop counts by 50% compared with the conventional ones. The network traffic could be also efficiently reduced at the same time. It was also verified that our proposal was robust against varying conditions even if client access patterns are dynamically changed.

There are a number of works to be done as further researches. Precise design of control packets exchanged among servers and its quantitative overhead evaluation should be carried out from the practical viewpoint. Furthermore, theoretical modeling and replication algorithms should be expanded and sophisticated, in order to lead us to a new content delivery paradigm including P2P (peer-to-peer) architecture, multicasting (IP-multicast and application-layer multicast) and multiple description coding instead of layered coding.

Appendix

In this appendix, we give a mathematical formulation of Eq.(7). Firstly, assume that the q-th layer of stream j is originally stored in server $o(q, j)$. When a client sends a request for this layer (q, j) to server $o(q, j)$ via a certain server k, the hop counts during the delivery from server k to server $o(q, j)$ is given by

$$T_{k,o(q,j),q,j}(\boldsymbol{X}_0) = \frac{1}{\Lambda}\lambda_k \cdot P_{q,j} \cdot D_{k,o(q,j)}(\boldsymbol{X}_0) \qquad \text{(A-1)}$$

which is an element of Eq.(5) when $\boldsymbol{X} = \boldsymbol{X}_0$. $D_{k,o(q,j)}(\boldsymbol{X}_0)$ is the shortest distance from server k to server $o(q, j)$ under the initial placement pattern $\boldsymbol{X}_0$ (no cache) and $\Lambda = \sum_i \lambda_i$ is the total request rate from all the domains. This sums up to the total hop counts of requests to the q-th layer of stream j from all servers, given by

$$T_{o(q,j),q,j}(\boldsymbol{X}_0) = \sum_k T_{k,o(q,j),q,j}(\boldsymbol{X}_0) = \frac{1}{\Lambda}\sum_k \lambda_k \cdot P_{q,j} \cdot D_{k,o(q,j)}(\boldsymbol{X}_0) \qquad \text{(A-2)}$$

Similarly, when the layer (q, j) is replicated onto a certain content server i which is not its original server, the total hop counts to layer (q, j) is given by

$$T_{i,q,j}(\boldsymbol{X}) = \frac{1}{\Lambda}\sum_k \lambda_k \cdot P_{q,j} \cdot D_{k,i}(\boldsymbol{X}) \qquad \text{(A-3)}$$

where $\boldsymbol{X}$ is the placement pattern after replication to the content server i. $D_{k,i}(\boldsymbol{X})$ means the shortest distance from server k to server i under the placement pattern $\boldsymbol{X}$. Finally, we can calculate the reduced hop counts (i.e. replication priority) $\Delta T_{i,q,j}$ by taking a difference of Eq.(A-2) and Eq.(A-3), then Eq.(7) is achieved.

7. REFERENCES

[1] J.Kangasharju and K.W. Ross," Performance Evaluation of Redirection Schemes in Content Distribution Networks", The 5th International Web Caching and Content Delivery Workshop, May 2000.

[2] A.Beck and M. Hofmann, "Enabling the Internet to Delivery Content-Oriented Services" Proc. The 6th International Web Caching and Content Distribution, Boston, USA Jun 2001.

[3] Adero, <URL: http: www. adero. com>

[4] Akamai, <URL: http: www. akamai. com>

[5] Napster, < URL: http://www.napster.com>

[6] L. Gao, "On Inferring Automonous System Relationships in the Internet '' IEEE Global Internet, 2002, Nov.

[7] H M. Radha, M.V.D. Schaar, and Y. Chen, “The MPEG-4 Fine Grained Scalable Video Coding Method for Multimedia Streaming Over IP ”, IEEE Trans. Multimedia Vol.3, No.1, pp.53-67, March 2001.

[8] J.Kangasharju, F.Hartanto, M.Reisslein, K.W. Ross, “Distributing Layered Encoded Video through Caches”, IEEE Transactions on Computers, vol. 51, n. 6, pp. 622-636, June 2002.

[9] J.Byers, J.Considine, and M.Mitzenmacher "Informed Content Delivery Across Adaptive Overlay Networks"_ SIGCOMM 2002, Pittsburgh, PA, Aug.2002.

[10] M. Chesire, A. Wolman, G.M.Voelker, and H.M.Levy, “Measurement and Analysis of a Stream Media Workload”, USITIS’01, San Francisco, CA, Mar.2001.

[11] S. Acharya and B. Smith and P. Parnes, “Characterizing User Access To Videos on the Videos On the World Wide Web’’ SPIE/ACM MMCN 2000, San Jose, CA, Jan 2000.

[12] J.Kangasharju, J.Roberts, K.W.Ross, “Object replication strategies in Content Distribution Networks”, Computer Communications, 25 2002

[13] I. Cidon, S. Kutten, and R. Soffer. Optimal allocation of electronic content. In Proceedings of IEEE Infocom, Anchorage, AK, April 22{26, 2001. 609} P. Krishnan, D. Raz, and Y. Shavitt. The cache location problem. IEEE/ACM Transactions on Networking, pp 568-582, October 2000.

[14] B. Li, M. J. Golin, G. F. Italiano, and X. Deng. On the optimal placement of web proxies in the internet. In Proceedings of IEEE Infocom, New York, NY, pp 21-25, March 1999.

[15] L. Qiu, V. N. Padmanabhan, and G. M. Voelker. On the placement of web server replicas. In Proceedings of IEEE Infocom, Anchorage, AK, pp 22-26, April 2001.

[16] S.Sen, J.Rexford,and D.Towsley, ” Proxy Prefix Caching for Multimedia Streams,” IEEE INFOCOM99, N.Y, Mar.1999.

[17] Sung-Ju Lee, Wei-Ying Ma, and Bo Shen ”An Interactive Video Delivery and Caching System Using Video Summarization”, WCW*2001,* Boston, MA, June 2001

[18] R. Rejaie, H. Yu, M. Handley, and D. Estrin, “Multimedia proxy caching mechanism for quality adaptive streaming applications in the Internet,” *IEEE INFOCOM 2000*, March 2000.

[19] M.Sasabe, N.Wakamiya, M.Murata, and H.Miyahara, "Proxy caching mechanisms with video quality adjustment", SPIE ITCom Feb. 2001.

[20] L. Breslao, P. Cao, L. Fan, G. Phillips, and S. Shenker “Web Caching and Zip-like Distributions: Evidence and Implications” Proc. IEEE INFOCOM’99, New York, April, 1999.

[21] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "On the Implications of Zipf's Law for Web Caching", In 3rd International WWW Caching Workshop, June 1998.

[22] Z.Su, J.Katto, T.Nishikawa, M.Murakami and Y.Yasuda, “Stream Caching Using Hierarchically Distributed Proxies with Adaptive Segments Assignment”, IEICE Trans on Commun, Vol.E86-B, No.6, Jun. 2003, pp 1859-1869

[23] Z.Su, T.Washizawa, J.Katto, and Y.Yasuda, “Integrated Pre-fetching and Caching Algorithm for Graceful Image Caching”, IEICE Trans on Commun, Vol.E86-B, No.9, Sep. 2003, pp 2753-2763

[24] Q.Lv, P.Cao, E.Cohen, K.Li, S.Shenker: “Search and Replication in Unstructured Peer-to-Peer Networks”. ICS 2002