COPACC: An Architecture of Cooperative Proxy-Client Caching System for On-Demand Media Streaming*

Alan T.S. Ip, Jiangchuan Liu, John C.S. Lui

Abstract

Proxy caching is a key technique to reduce transmission cost for on-demand multimedia streaming. The effectiveness of current caching schemes, however, is limited by the insufficient storage space and weak cooperations among proxies and their clients, particularly considering the high bandwidth demands from media objects. In this paper, we propose COPACC, a cooperative proxy-and-client caching system that addresses the above deficiencies. This innovative approach combines the advantages of both proxy caching and peer-to-peer client communications. It leverages the client-side caching to amplify the aggregated cache space and rely on dedicated proxies to effectively coordinate the communications. We propose a comprehensive suite of distributed protocols to facilitate the interactions among different network entities in COPACC. It also realizes a smart and cost-effective cache indexing, searching, and verifying scheme. Furthermore, we develop an efficient cache allocation algorithm for distributing video segments among the proxies and clients. The algorithm not only minimizes the aggregated transmission cost of the whole system, but also accommodates heterogeneous computation and storage constraints of proxies and clients. We have extensively evaluated the performance of COPACC under various network and end-system configurations. The results demonstrate that it achieves remarkably lower transmission cost as compared to pure proxy-based caching with limited storage space. On the other hand, it is much more robust than a pure peer-to-peer communication system in the presence of node failures. Meanwhile, its computation and control overheads are both kept in low levels.

Keywords: Media Streaming, Proxy caching, Peer-to-Peer caching, Media segmentat ion and Resource allocation

I. Introduction

For the past few years, we have witnessed the increasingly used streaming multimedia traffic on the Internet, and on-demand streaming for clients of asynchronous playback requests is amongst the most popular networked media services. Given its broad spectrum of applications, like NetTV and distance

*Alan T.S. Ip and John C.S. Lui are with the Computer Science & Engineering Department, The Chinese University of Hong Kong, Shatin, N.T., Hong Kong. Email:alanip@gmail.com, cslui@cse.cuhk.edu.hk. Jiangchuan Liu is with the School of Computer Science, Simon Fraser University, Vancouver, BC, Canada. Email:csljc@ieee.org.

learning, it has attracted much attention and many practical deployments have been proposed in recent years [1]. The limited server capacity and the unpredictable Internet environment, however, make it a challenging task to design and deploy an efficient and scalable on-demand media streaming service[2], [3], [4], [5].

To reduce server/network loads, an effective approach is to cache frequently used data at proxies that are close to clients [6], [7]. Streaming media, particularly those with asynchronous demands, could also benefit and could have a significant performance improvement from proxy caching given their static nature in content and highly localized access interests. However, media objects have high data rates requirements and long playback durations, which combined yield a huge caching resource. To illustrate, a one-hour standard MPEG-1 video has a volume of about 675 MB; several such large streams will quickly exhaust the cache space of a standalone proxy. As such, it is necessary to design partial caching algorithms or group proxies to enlarge cache space [6], [8], [9], [10]. There have been extensive studies toward these directions, but the storage space of existing proxies are still far from satisfactory, and thus remains a performance bottleneck of the whole system.

Another approach is to generalize the proxy functionalities into every client [11], [12]. Such a peer-to-peer communication paradigm allows economical clients to contribute their local storage spaces for streaming. Specifically, the video data originally provided by a server are spread among clients of asynchronous demands, and each client can store the full or partial versions of the video stream in its local cache. Then, one or more clients can collectively supply cached data to other clients, thus amplifying the system capacity with increasing suppliers over time. However, in contrast to the reliable and dedicated servers or proxies, these loosely-coupled autonomous end-hosts are not highly reliable since these end-hosts can fail or may leave the network without any notice. Given the requirement that a media playback lasts a long time and consumes huge resources, a pure peer-to-peer system may not provide the desirable information availability in the Internet environment. Another reason for not adopting the pure P2P approach is that there are no authoritative parties, it is also difficult to identify and penalize malicious clients that intentionally inject forged data.

In this paper, we propose COPACC, a novel cooperative proxy-client caching system that addresses the

above deficiencies. The innovative approach in COPACC *combines* the advantages of both proxy caching and peer-to-peer client communications. We leverage the client-side caching to amplify the aggregated cache space and rely on dedicated proxies to effectively coordinate the communications. We develop an efficient cache allocation algorithm that distributes video segments among the proxies and clients. The algorithm not only minimizes the aggregated transmission cost of the whole system, but also accommodates heterogeneous computation and storage constraints of proxies and clients. When multicast service is available, COPACC also makes effective use of multicast delivery in local regions, which further reduces the cost of the system.

In this work, we propose a comprehensive suite of distributed protocols to facilitate the interactions among different network entities. Most operations in this protocol suite are executed by dedicated proxies. As such, it is not only suitable for clients with limited computation power, but also resilient to client failures. We also embed an efficient indexing and searching algorithm for video contents cached across different proxies or clients, as well as a signature verification mechanism, which can effectively identify and block malicious clients.

The performance of COPACC is extensively evaluated under various network and end-system configurations. The results demonstrate that it achieves remarkably lower transmission cost as compared to proxy-based caching with limited storage space. On the other hand, with the assistance from dedicated proxies, it is much more robust than a pure peer-to-peer system. Its transmission cost only slightly increases when a large portion of clients fail, even though the clients contribute a significant fraction in the total cache space. Moreover, it scales well to larger networks, and the cost generally reduces when more proxies and clients cooperate with each other.

The balance of the paper is organized as follows. In Section II, we review the related work. The COPACC architecture and its parameters are presented in Section III. We derive efficient algorithms for cache allocation in Section IV, and describe the cooperative caching protocol in Section V. The performance of COPACC is extensively evaluated in Section VI. Finally, we conclude our work in Section VII.

II. **Related Work**

Proxy caching for media streaming has attracted much attention in the past decade, and numerous algorithms have been proposed in the literature, e.g., run-length caching [13], prefix caching [9], and segment caching [8], [10], [14]; see a comprehensive survey in [6]. Considering the static nature of video contents and their intensive I/O demands, many of the algorithms employ a semi-static caching approach, where popular video portions are cached over a relatively long time period, rather than dynamically saved or replaced in response to individual client requests. COPACC also advocates semi-static caching, and its cache allocation is closely related to the prefix-suffix partition and stream segmentation algorithms [15]. However, these studies generally focus on a single proxy case with no cooperation among proxies.

It is well-recognized that proxies grouped together can achieve better performance than independent standalone proxies [16], [17]. An example for media caching is MiddleMan [18], which operates a collection of proxies as a scalable cache cluster; media objects are segmented into equal-sized segments and stored across multiple proxies, where they can be replaced at a granularity of a segment. There are also several local proxies responsible to answer client requests by locating and relaying the segments. To achieve better load balance and fault tolerance, a Silo data layout is suggested in [19], which partitions a media object into segments of increasing sizes, stores more copies for popular segments, and yet guarantees at least one copy stored for each segment. Our work is motivated by these cooperative systems, and we enhance them by combining proxy caching and client-side caching, which greatly expands the aggregated cache storage with contributions from the less expensive clients.

On the other hand, peer-to-peer communications have recently become a popular alternative to the traditional client/server paradigm. There are a series of pioneer works on peer-to-peer streaming, e.g., PROMISE [12], ZIGZAG [20], and CoopNet [21], which have demonstrated the superior scalability of shifting all functionalities to end-hosts. Yet, we are aware that, in contrast to the reliable and dedicated servers or proxies, the loosely-coupled autonomous end-hosts can easily crash, leave without notice, or even refuse to share its own data. Given that a media playback lasts a long time and consumes huge resources, we believe that dedicated proxies could still play an important role in building high-quality media streaming systems, as suggested in [22], [23]. Different from COPACC which focuses on caching,

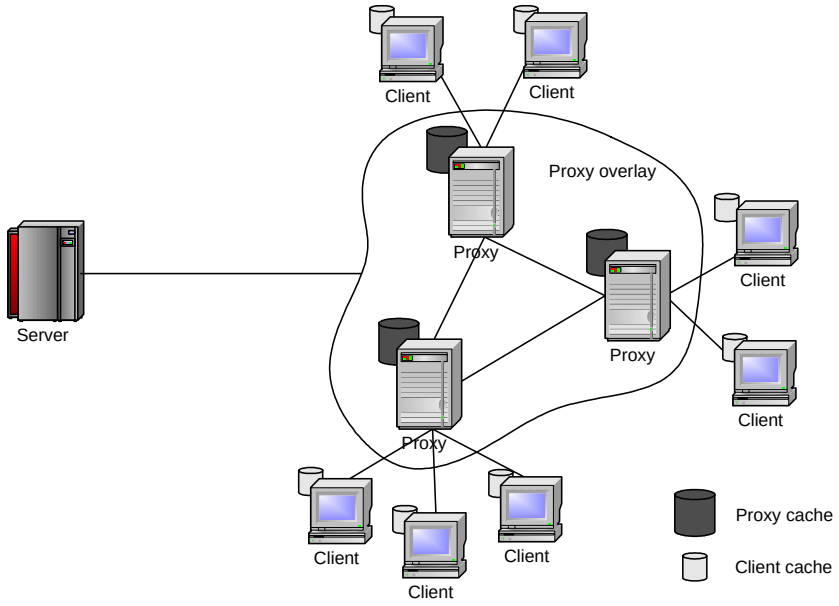

Fig. 1. The cooperative proxy-client caching architecture.

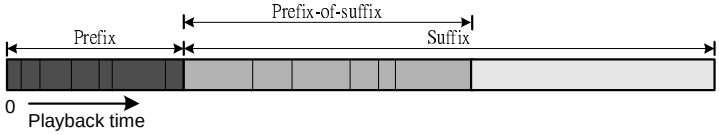

Fig. 2. Illustration of different portions of a video stream. The prefix is to be cached by proxies, while the prefix-of-suffix by clients

the key issue addressed in these studies is the optimal construction of an overlay structure. For storage allocation and management in a hybrid system, an optimal replication algorithm is proposed in [24], and a cooperative algorithm between a single proxy and its clients in a local area network is presented in [25]. COPACC complements them by considering a more general system with multiple cooperative proxies with client caching. A two-level hybrid architecture is exploited in [26], where an overlay network is used in the upper level to deliver videos from a central server to proxies and a collaborative-client network using loopback mechanism is applied in the lower level to transmit video data from proxy to clients. In loopback, cache is dynamically updated, which introduces an intensive disk I/O demand for the clients. Given that the video access pattern changes slowly, semi-static caching is adequate and it can be practically implemented. Moreover, Loopback concentrates on the collaboration between proxy and its clients only, but we also emphasize the importance of cooperative caching between proxies in reducing cost.

III. Overview of The Cooperative Proxy-Client Caching (COPACC) System

Fig. 1 depicts a generic architecture of COPACC. A cluster of proxies are logically connected through direct or indirect peer links to form a proxy overlay, and each of them serves as the *home proxy* for a set of local clients. We assume that proxies and their clients are closely located with relatively low communication costs, e.g., they could be in the same ISP domain or in the same metropolitan area. A server storing the repository of videos, however, is far away from them, and the remote communications incur much higher costs.

The video data are cached across both proxies and clients. We assume that the storage space of a proxy or a client is limited; the videos thus can be partially cached only, and there is always a full copy at the server. Specifically, as shown in Fig. 2, a video stream is partitioned into a *prefix* and a *suffix*, and the beginning part of the later is also referred to as the *prefix-of-suffix*. The proxies are responsible to cache the prefix of video, whereas the clients cache the prefix-of-suffix of video. Given that the initial part of a video stream is normally the mostly accessed, this setting reduces the initial playback latency; it also facilitates the multicast delivery with dynamic clients, as will be illustrated later. When a client expects to play a video, it first initiates a playback request to its home proxy, which intercepts the request and computes a streaming schedule: when and where to fetch which portion of the video. It then accordingly fetches the prefix, prefix-of-suffix, as well as the remaining part of suffix, and relays the incoming stream to the client. If needed, a proxy may also perform a verification operation, which detects forged video data through a simple signature mechanism.

Considering the video contents and their access patterns are relatively stable in several hours or even days, we advocate semi-static caching in COPACC. The cached contents are updated only when the system parameters have drastically changed, and a cache reconfiguration is then applied through a progressive cache filling mechanism.

There are two key issues to be addressed in the COPACC architecture:

- How to partition each video and allocate the prefixes and prefix-of-suffixes to different proxy and client caches? The objective is to minimize the total transmission cost of the COPACC system given

the video access patterns, the heterogeneous transmission costs, and the storage constraints.

- How to manage, search, and retrieve the cached data in different proxies and clients? These operations should be highly efficient so as to deploy COPACC in large-scale networks with intensive requests.

To address the above challenges, we present an efficient allocation algorithm as well as a comprehensive suite of cache management and search protocols in the next two sections. Before proceeding our discussions, we first list the notations and parameters for COPACC, which are also summarized in Table I.

We assume that there are H cooperative proxies, indexed from 1 through H, and proxy j serves as the home proxy for K_j local clients. The video repository at the server includes N Constant-Bit-Rate (CBR) videos, and video i has length L^i seconds and rate b^i bps, $i = 1, 2, ..., N$. The total average access rate at proxy j is λ_j, and the probability for accessing video i is f_j^i $(\sum_{i=0}^{N} f_j^i = 1)$. We assume such statistics are known *a priori*, or obtained through online monitoring.

For cache allocation, there is a basic unit of u, also called *cache grain*, which is a hardware or operating system constraint, e.g., the size of a disk block. The cache space for proxy j is s_j^p units, and that for client k of proxy j is $s_{j,k}^c$ units. The volume of video i is also represented as a number of units, i.e., $V^i = b^i L^i / u$ units. In practice, the aggregated cache space is less than the total volume of all the videos, i.e. $S^p + S^c \leq \sum_{i=1}^{N} V^i$, where $S^p = \sum_{j=1}^{H} s_j^p$ and $S^c = \sum_{j=1}^{H} \sum_{k=1}^{K_j} s_{j,k}^c$ are the total proxy cache size and total client cache size.

The cost for transmitting one unit of data from the server to a proxy is denoted by $w^{s \to p}$, and, similarly, the unit cost from proxy j to proxy k and that from proxy j to its own clients are represented by $w_{j,k}^{p \to p}$ and $w_j^{c \leftrightarrow p}$, respectively.

We use P^i to denote the prefix size (in units) of video i, and, Q^i, the prefix-of-suffix size. Both the prefix or prefix-of-suffix of a video are further partitioned into several segments and cached at a proxy or client. For video i, the size of a prefix segment cached in proxy j is represented by p_j^i, and the size of a prefix-of-suffix segment cached at the client k of proxy j is $q_{j,k}^i$. The segment sizes are to be determined by the cache allocation algorithm, and the exact positions of the segments are to be determined by the

cache organization protocol.

Parameter	Definition
N	Number of the videos
V^i	Volume of video i (in units)
H	Number of proxies
K	Number of clients
K_j	Number of local client attached to proxy j
s_j^p	Cache space of proxy j (in units)
S^p	Total cache space of all proxies (in units)
$s_{j,k}^c$	Cache space of client k of proxy j (in units)
S^c	Total cache space of all clients (in units)
λ_j	Total access rate at proxy j
f_j^i	Probability for accessing video i at proxy j
$w^{s\to p}$	Transmission cost per unit data from server to proxy
$w_{j,k}^{p\to p}$	Transmission cost per unit data from proxy j to proxy k
$w_j^{c\leftrightarrow p}$	Transmission cost per unit data from proxy j to its client
w^{in}	Internal cost per unit data of a proxy
P^i	Prefix size of videos i (in units)
p_j^i	Size of the prefix segment of video i cached in proxy j
Q^i	Prefix-of-suffix size of videos i (in units)
$q_{j,k}^i$	Size of the prefix-of-suffix segment of video i cached at client k of proxy j

TABLE I

PARAMETERS OF SYSTEM

IV. Optimal Cache Allocation (CAP)

The optimal cache allocation problem (CAP) in COPACC can be formulated as follows,

$$\begin{aligned}
&\mathbf{CAP}: \min Cost(\{p_j^i\},\{q_{j,k}^i\}),\\
&\text{s.t. } p_j^i, q_{j,k}^i \geq 0, j \in [1\ldots H], k \in [1\ldots K_j],\\
&\quad \textstyle\sum_{i=1}^{N} p_j^i \leq s_j^p,\\
&\quad \textstyle\sum_{i=1}^{N} q_{j,k}^i \leq s_{j,k}^c,\\
&\quad \textstyle\sum_{j=1}^{H} p_j^i + \sum_{j=1}^{H}\sum_{k=1}^{K_j} q_{j,k}^i \leq V^i,
\end{aligned}$$

where $Cost(\{p_j^i\},\{q_{j,k}^i\})$ is the function of the total transmission cost (per unit time) given allocation $\{p_j^i\}$ and $\{q_{j,k}^i\}$; the second and third constraints follow the cache space limit of proxy j and that of client k of proxy j, respectively; the forth constraint applies because we do not consider replication in this study. That is, the prefix and prefix-of-suffix are stored only once among the proxies and clients in the network. In this section, we start our discussion from a simple scenario of no cooperation between proxies, where the cache allocation for each proxy and its own clients can be examined independently. We derive an efficient optimal solution for this scenario, which is then extended to accommodate multiple cooperative proxies with client caching, i.e., a general COPACC system.

A. Single Proxy with Client Caching

As said, we focus on a single proxy and its clients, both of which contribute cache spaces, but there is no interactions with other proxies nor their clients. Since the transmission costs between this proxy and all its clients are identical, we refer to this system as a homogeneous cost system. We drop the proxy index (subscript j) from the relevant parameters for ease of exposition.

This homogeneous cost system has a nice property that the total transmission cost depends only on how the video streams are partitioned into prefixes and prefix-of-suffixes for caching. This is because all prefixes are to be cached at the single proxy, and any allocation of the prefix-of-suffix segments across the local clients yield the same cost due to the uniform cost for proxy-client transmissions. As such, we can combine the cache of all the clients to form an aggregated cache space S^c, and, to derive the minimum transmission cost, we only need to find the optimal values of $\{P^i\}$ and $\{Q^i\}$ subject to cache space constraints S^p and S^c.

We define an auxiliary cost function $C^i(P^i, Q^i)$, which is the cost for delivering video i with prefix size P^i and prefix-of-suffix size Q^i. Note that $Cost(\{p^i_j\}, \{q^i_{j,k}\})$ is now equal to $\sum_{i=1}^{N} C^i(P^i, Q^i)$ in this simple scenario. Moreover, minimizing it is equivalent to maximizing the cost saving against the system with no caching, i.e. maximizing $\sum_{i=1}^{N}[C^i(0,0) - C^i(P^i, Q^i)]$.

We use a dynamic programming approach to solve the problem. Let B be a three-dimensional matrix, where $B(i, t^p, t^c)$ represents the maximum cost saving for videos 1 through i $(1 \le i \le N)$, when t^p $(0 \le t^p \le S^p)$ units of proxy cache and t^c $(0 \le t^c \le S^c)$ units of client cache are used. We have

$$B(i, t^p, t^c) = \begin{cases} 0, \quad i = 0,\ 0 \le t^p \le S^p,\ 0 \le t^c \le S^c \\ \max\{B(i-1, t^p - v^p, t^c - v^c) + C^i(0,0) - C^i(v^p, v^c)\}, \\ \qquad\qquad 0 \le v^p \le t^p, 0 \le v^c \le t^c, v^p + v^c \le V^i. \end{cases}$$

The matrix can be filled in plane-order starting from $B(0,0,0)$ to $B(N, S^p, S^c)$, and the latter gives the maximum cost saving. The minimum total transmission cost is therefore $\sum_{i=1}^{N} C^i(0,0) - B(N, S^p, S^c)$, and the corresponding prefix and prefix-of-suffix partitioning can be obtained through backtracking the iterations.

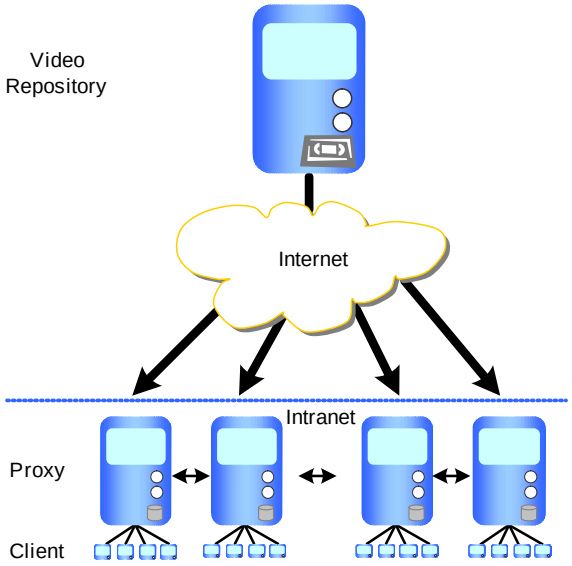

Fig. 3. A logical view of multi-proxy with client caching.

This dynamic programming algorithm has time complexity $O(N \cdot S^p \cdot S^c \cdot M)$, where $M = \max_{1 \le i \le N}(1+V^i)V^i/2$. It is applicable with arbitrary cost function $C^i(P^i, Q^i)$, which can be instantiated given a specific transmission scheme. As an example, assume both a server-to-client and a client-to-client transmissions are unicast-based and relayed by a proxy, $C^i(P^i, Q^i)$ can be derived as $\lambda f^i \cdot [w^{c \leftrightarrow p} P^i + 2w^{c \leftrightarrow p} Q^i + (w^{s \to p} + w^{c \leftrightarrow p})(V^i - P^i - Q^i) + w^{in}(P^i + Q^i)]$, where the first four terms in the second part respectively represent the costs for retrieving prefix, prefix-of-suffix, the remaining suffix, and the internal cost of the proxy, for each playback request. Note that w^{in} is the internal cost per unit data handled by the proxy. When there is no caching ($P^i = Q^i = 0$), we have $C^i(0,0) = V^i \lambda f^i \cdot (w^{s \to p} + w^{c \leftrightarrow p})$. In the end of this section, we further introduce multicast delivery to the system and derive the corresponding cost function.

The optimal cache allocation algorithm is practically feasible as it is executed off-line and the resulted allocation lasts for a relatively long period of time. Furthermore, the computation complexity of the algorithm can be reduced by using a large cache grain u. For storage overhead, a globally unique video ID as well as the start time of each segment is stored together with the video data, and this is already incorporated in the existing stream caching system.

B. Multiple Proxies with Client Caching

We now consider the case of multiple cooperative proxies with client caching. Fig. 3 offers a logical view of this general COPACC system, in which the segment of prefix and prefix-of-suffix of a video are placed across different proxies and their clients, respectively, and the transmission of a video stream thus involve interactions among several proxies and clients. Moreover, the unit transmission costs for the proxy-to-proxy and client-to-proxy links can be heterogeneous. The cache allocation problem (CAP) thus becomes much more complex than in the homogeneous cost system.

In fact, we formally prove that CAP is NP-hard in this general case (see *Appendix A*). We thus resort to a practically efficient heuristics, which consists of two phases: first, it partitions the prefix and prefix-of-suffix for each video; second, given the partitions, it allocates the segments of prefixes and prefix-of-suffixes to the proxies and clients.

1) Partitioning of prefix and prefix-of-suffix: In this phase, we calculate the optimal values of P^i and Q^i for each video, and, to achieve a computationally efficient solution, we do not address their allocation across the proxies and clients. Instead, we approximate the system by a single proxy system with aggregated proxy cache space S^p and aggregated client cache space S^c. Other parameters are approximated as follows: video access rate $\lambda = \sum_{j=1}^{H} \lambda_j$, access probability $f^i = (1/\lambda) \sum_{j=1}^{H} \lambda_j f_j^i$, unit transmission cost $w^{c \leftrightarrow p} = (1/K) \sum_{j=1}^{H} K_j w_j^{c \leftrightarrow p}$, and internal cost $w^{in} = (1/H^2) \sum_{j=1}^{H} \sum_{k=1}^{H} w_{j,k}^{p \rightarrow p}$, that is, we consider the cost for proxy-to-proxy transmissions as an internal cost, and assume $w_{j,k}^{p \rightarrow p}$ is 0 if $j = k$.

Given the above transformation, an approximate solution can be directly obtained using the dynamic programming algorithm for the homogeneous cost system.

2) Allocation to proxy and client caches: In this phase, we allocate the prefix and prefix-of-suffix to the proxies and clients so as to meet the storage constraints at each proxy and client. The objective is to minimize the average transmission cost, which is defined as the sum of average cost in delivering the prefix and prefix-of-suffix to the requested clients. Once P^i and Q^i are determined in the first phase, the allocation for prefixes to proxy caches is independent from that for prefix-of-suffixes to client caches, and vice versa. The reason is that the prefix and prefix-of-suffix are stored in different location, and they are

transmitted to the requested clients separately. Thus, the two allocation problems are independent to each other and we can solve them individually.

We first consider the allocation for prefixes. Let $W^p(i,j,p_j^i)$ be the transmission cost when the segment of size p_j^i from the prefix of video i is stored in proxy j. The problem for optimal prefix allocation is then formulated as

$$\begin{array}{ll}\mathbf{PA}: \min \sum_{i=1}^{N}\sum_{j=1}^{H} W^p(i,j,p_j^i) \\ s.t. \quad \sum_{j=1}^{H} p_j^i = P^i, \quad i \in [1\ldots N] \\ \qquad \sum_{i=1}^{N} p_j^i \le s_j^p, \quad j \in [1\ldots H].\end{array}$$

For unicast delivery, $W^p(i,j,p_j^i)$ can be instantiated as

$$W^p(i,j,p_j^i) = \sum_{j'=1}^{H} p_j^i \left[w_{j,j'}^{p\to p} + w_{j'}^{c\leftrightarrow p}\right] \lambda_{j'} f_{j'}^i.$$

Let $\bar{W}^p(i,j) = \sum_{j'=1}^{H} \left[w_{j,j'}^{p\to p} + w_{j'}^{c\leftrightarrow p}\right] \lambda_{j'} f_{j'}^i$, the optimization objective for problem **PA** can be re-written as $\min \sum_{i=1}^{N}\sum_{j=1}^{H} \bar{W}^p(i,j)\cdot p_j^i$. Note that, $\bar{W}^p(i,j)$ is independent of p_j^i, and can be viewed as the transmission cost when each unit prefix data of video i cached in proxy j. The above formulation for **PA** thus can be relaxed as a linear programming problem if p_j^i is not restricted to integers. In practice, this is generally viable, for a video stream that can be partitioned with fine-granularity, and the total data cached in any proxy is less than its maximum capacity for any optimal solution to the linear programming.

Similarly, we can formulate the optimal allocation problem for prefix-of-suffixes to be cached at clients as follows,

$$\begin{array}{ll}\mathbf{SA}: \min \sum_{i=1}^{N}\sum_{j=1}^{H}\sum_{k=1}^{K_j} W^c(i,j,k,q_{j,k}^i) \\ s.t. \quad \sum_{j=1}^{H}\sum_{k=1}^{K_j} q_{j,k}^i = Q^i, \quad i \in [1\ldots N] \\ \qquad \sum_{i=1}^{N} q_{j,k}^i \le s_{j,k}^c, \quad j \in [1\ldots H], k \in [1\ldots K_j]\end{array}$$

where $W^c(i,j,k,q_{j,k}^i)$ is the transmission cost when the segment of size $q_{j,k}^i$ from the prefix-of-suffix of video i is stored in client k of proxy j. For unicast delivery, $W^c(i,j,k,q_{j,k}^i)$ is given by

$$\sum_{j'=1}^{H} q_{j,k}^i [w_{j,j'}^{p\to p} + w_{j'}^{c\leftrightarrow p} + w_j^{c\leftrightarrow p}] \lambda_{j'} f_{j'}^i,$$

which can be re-written as $\bar{W}^c(i,j,k)\cdot q_{i,j}^k$ if we define $\bar{W}^c(i,j,k) = \sum_{j'=1}^{H} \left[w_{j,j'}^{p\to p} + w_{j'}^{c\leftrightarrow p} + w_j^{c\leftrightarrow p}\right] \lambda_{j'} f_{j'}^i$.

Obviously, both the cost function and the problem **SA** itself have similar structure as that of problem **PA**. The linear programming relaxation thus also applies. We will show later that such relaxation also holds for multicast delivery.

We also compared the performance between the optimal solution to CAP and the proposed heuristics solution. As there is no effective algorithm to solve CAP, we use the brute-force approach to find the cache allocation of a small problem, which contains 20 media streams, 2 proxies and 10 clients. We found that the solution obtained by our heuristics algorithm is comparable to the optimal solution, where the performance difference is only around 1%. Thus, our heuristics algorithm provides an efficient way to obtain a near-optimal solution.

C. Cost Function with Suffix Multicast

So far we have focused on unicast delivery only, and presented the corresponding cost functions. In this subsection, we further consider multicast delivery, which is known as an efficient vehicle for streaming to clients with requests close in time [15], [27]. However, though IP multicast has been widely adopted within ISP networks, its deployment over the global Internet remains confined. We thus assume multicast delivery at the path from a proxy to its local clients, but only unicast delivery from the server to a proxy or between two proxies. This assumption does not limit the deployment of COPACC since unicast delivery is always an alternative if local IP multicast is not supported. Yet, if IP multicast is enabled, the performance of COPACC can be improved. Furthermore, Application Layer Multicast (ALM) can also be applied in delivering data between the proxies and clients.

Even though multicast is only enabled at local paths, a proxy can still serve a series of requests from its local clients for the same video using a *suffix batching* technique. Specifically, assume the first request for video i arrives at time 0, the home proxy will fetch and relay the prefix of the video to this client through unicast, which takes $P^i u/b^i$ seconds; all the local requests arrive during interval $[0, P^i u/b^i]$ will then be batched with a single copy of the suffix for video i being multicast to all the requested clients. In other words, the batching window is of size $P^i u/b^i$.

We now derive the cost function $C^i(P^i, Q^i)$ for the case of single-proxy with client caching. We assume

that the video accesses follow a Poisson arrival, that is, the average number of requests arrived in the batching window for video i is $1+(P^iu/b^i)(\lambda f^i)$. The cost per request for multicasting the suffix to batch of clients is thus $[w^{c\to p}(V^i-P^i)+w^{c\to p}Q^i+w^{s\to p}(V^i-P^i-Q^i)+w^{in}Q^i]/[1+(P^iu/b^i)(\lambda f^i)]$. Since a prefix is always delivered using unicast, the cost function $C^i(P^i,Q^i)$ is then given by:

$$\lambda f^i\cdot\{\frac{w^{c\to p}(V^i-P^i)+w^{c\to p}Q^i+w^{s\to p}(V^i-P^i-Q^i)+w^{in}Q^i}{1+(P^iu/b^i)(\lambda f^i)}+(w^{c\to p}+w^{in})P^i\}.$$

Similarly, we can derive the cost function $\bar{W}^c(i,j,k)$ of problem **SA**. For a batching windows contains $1+(P^iu/b^i)(\lambda_{j'}f^i_{j'})$ requests from proxy j', we need only a single retrieval for the suffix distributed at client caches and the server. The cost function $\bar{W}^c(i,j,k)$ at proxy j is thus

$$\sum_{j'=1}^{H}\left[\frac{w_j^{c\to p}+w_{j,j'}^{p\to p}+w_{j'}^{c\to p}}{1+(P^iu/b^i)(\lambda_{j'}f^i_{j'})}\right]\lambda_{j'}f^i_{j'}.$$

Regarding the cost function $\bar{W}^p(i,j)$ of problem **PA**, it is exactly the same as that for unicast case because a prefix is delivery through unicast only. In addition, if $f_1^i=f_2^i=...=f_H^i$, we have the following observations for $\bar{W}^p(i,j)$:

- Given $i,i'\in[1\dots N]$, $\bar{W}^p(i,j)/\bar{W}^p(i',j)$ is a constant for any $j\in[1\dots H]$;
- Given $j,j'\in[1\dots H]$, $\bar{W}^p(i,j)/\bar{W}^p(i,j')$ is a constant for any $i\in[1\dots N]$.

Since clients often have common interests, it is likely that the distributions of video access probabilities are similar at different proxies, that is, $f_1^i=f_2^i=...=f_H^i$ holds. The above observation thus leads to an simpler yet optimal greedy algorithm for problem **PA**, as shown in Algorithm 1. Intuitively, this algorithm always cache the most expensive prefix into the cheapest proxy, so as to minimize the total transmission cost. Its complexity is $O(NlogN)$, which is generally lower than directly solving the linear programming problems (even if the simplex method [28] is used). A formal proof of the optimality of this greedy algorithm can be found in *Appendix B*.

Algorithm 1 Greedy prefix allocation

1: Sort proxies in ascending order of cost $\bar{W}^p(1, j)$;
   Store the results in j-List;
2: Sort videos in descending order of cost $\bar{W}^p(i, 1)$;
   Store the results in i-List;
3: $j^* \leftarrow$ first component of j-List;
   $i^* \leftarrow$ first component of i-List;
4: Cache as many units as possible for the prefix of video i^* to proxy j^*;
5: If proxy j^* has cache space left, then $i^* \leftarrow$ next component of i-List;
6: If prefix of video i^* has not been fully cached, then $j^* \leftarrow$ next component of j-List;
7: Repeat steps 4 to 6 until all prefixes are allocated.

V. Cooperative Proxy-Client Caching Protocol

As shown in Fig. 1, COPACC operates as a two-level overlay, where the first level consists of all the proxies, and the second level consists of each proxy and its own clients. The interactions among different entities in this two-level overlay are specified by a cooperative proxy-client caching protocol, which consists of three subprotocols: *cache allocation and organization*, *cache lookup and retrieval*, and *client access and integrity verification*. We now detail the operations, and address the practical issues toward realizing the COPACC system.

A. Cache Allocation and Organization

All the cache allocation and organization decisions are implemented in proxies. The protocol starts by establishing connections among the proxies, and an election algorithm is then executed to choose a coordinator. We currently employ the distributed Bully algorithm [29], which opts for the proxy of the highest computational power as the coordinator. The coordinator is responsible for collecting parameters from all other proxies and then running the optimal cache allocation algorithm described in the previous section.

Given $P^i = \sum_{j=1}^{H} p_j^i$ and $Q^i = \sum_{j=1}^{H} \sum_{k=1}^{K_j} q_{j,k}^i$, the interval of the prefix in video stream i is simply

$[0, P^i u/b^i]$ and that of prefix-of-suffix is $[P^i u/b^i, Q^i u/b^i]$. The coordinator should then determine the position of each segment to be allocated to proxies and clients in the prefix and prefix-of-suffix. Since the total transmission cost depends only on the segment size, COPACC employs a simple organization scheme: for prefix of video i, allocate segment of interval $[\sum_{m=1}^{j-1} p_m^i u/b^i,\ \sum_{m=1}^{j} p_m^i u/b^i]$ in the video stream to proxy j, and, for the prefix-of-suffix, allocate interval $[\sum_{m=1}^{j-1}\sum_{n=1}^{K_{j-1}} q_{m,n}^i u/b^i, \sum_{m=1}^{j}\sum_{n=1}^{K_j} q_{m,n}^i u/b^i]$ to the clients of proxy j, which further partitions this interval into segments to be cached in its local clients according to their cache spaces. Hence, the cache location of each interval of the stream can be easily calculated from $\{p_j^i\}$ and $\{q_{j,k}^i\}$. As the coordinator keeps a full copy of the allocations, a lookup request for the cache locations of a particular video stream can always be accomplished by contacting the coordinator. To balance the load of the proxies, the coordinator also distributes the lookup information uniformly to other proxies using a hash function $h(i)$; that is, for video i, a copy of its cache location information are kept by proxy $h(i)$ as well. Since the proxies are persistent and reliable nodes, even the simplest hashing like $h(i) = (i \ mod \ H) + 1$ will work well in COPACC. In other words, COPACC does not have to rely on a flooding-based search, nor a complex and costly distributed hash table (DHT), as in many peer-to-peer systems.

B. Cache Lookup and Retrieval

For each playback request for video i from a client, its home proxy discovers and retrieves the video data on behalf of its clients. This is accomplished by first issuing a *cache lookup* request $R_{lookup}[i]$, which, according to the cache organization, can be directly submitted to proxy $h(i)$. Upon receiving the location information from proxy $h(i)$, the initiated proxy then issues a series of cache retrieval requests, $R_{retrieval}[i]$, to corresponding proxies for retrieving and then relaying the segments cached at proxies or their clients. Finally, the un-cached part of the suffix is retrieved from the server.

When a proxy receives a retrieval request, it first checks whether the requested data has been cached. If cached, it will stream the data to the requested proxy; if not, it will retrieve the data from the server, store a copy in its own cache or its clients' cache, depending on whether the content belongs to a prefix or to a prefix-of-suffix, and then stream to the initiated proxy. This leads to a passive filling scheme with

1: **while** Receive a request **do**

2: **if** $R_{retrieval}[i]$ from local client **then**

3: Look up proxy $j = h(i) \rightarrow$ get $\{p_j^i\}$ and $\{q_j^i\}$

4: Send $R_{retrieval}[i]$ to proxy j for prefix of interval $[\sum_{m=1}^{j-1} p_m^i u/b^i, \sum_{m=1}^{j} p_m^i u/b^i]$

5: Send $R_{retrieval}[i]$ to proxy j for prefix-of-suffix of interval $[\sum_{m=1}^{j-1} q_m^i u/b^i, \sum_{m=1}^{j} q_m^i u/b^i]$

6: Retrieval remaining interval $[P^i + Q^i, L^i]$ from server

7: Relay the stream to the request client

8: **else if** $R_{retrieval}[i]$ for prefix of interval $[a, b]$ **then**

9: Prefix of interval $[a, b]$ not exist in proxy cache $\rightarrow$ retrieval from server and store in proxy cache

10: Send prefix of interval $[a, b]$ to requested proxy

11: **else if** $R_{retrieval}[i]$ for prefix-of-suffix of interval $[a, b]$ **then**

12: Prefix of interval $[a, b]$ not exist in the cache of any local client $\rightarrow$ retrieval from server and store in a local client's cache

13: Send prefix-of-suffix of interval $[a, b]$ to requested proxy

14: **else if** $R_{lookup}[i]$ from another proxy **then**

15: Reply $\{p_j^i\}$ and $\{q_j^i\}$

16: **end if**

17: **end while**

TABLE II

CACHE LOOKUP AND RETRIEVAL

no need for a synchronized global replacement, that is, an empty cache space (or one with an outdated cache allocation) will be filled up gradually following the requests from other proxies, which represents the updated allocation.

C. Client Access and Integrity Verification

The client-side operations are relatively simple, which can be easily implemented in economical but less powerful personal computers. In particular, a client is not involved in managing the overlay, nor determining cache allocation and organization. It simply reports its available spaces to its home proxy.

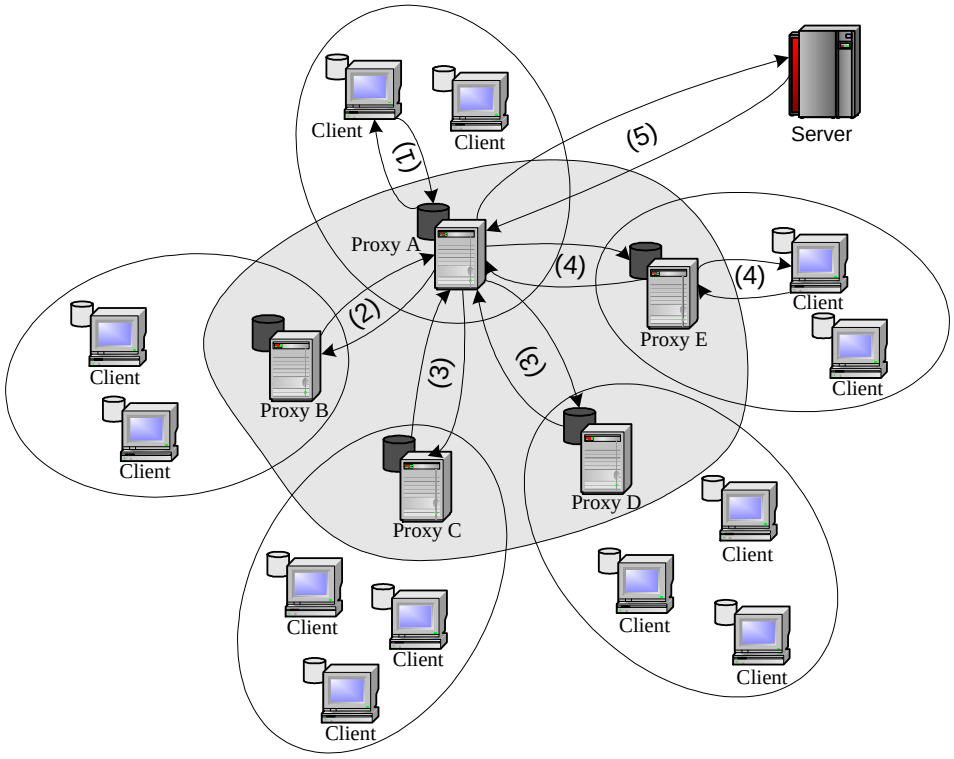

Fig. 4. An illustration of the cache lookup and retrieval operations. (1) client request to home proxy for video i; (2) location lookup request to proxy $B = h(i)$; (3) retrieve and relay prefix segments from proxy cache; (4) retrieve and relay prefix-of-suffix segments from clients; (5) retrieve and relay the remaining part of suffix from server.

The home proxy then determines and keeps the location for data cached in its local clients, and then instructs the clients for caching the data. Given that clients are not trustable, a flexible and adaptive mechanism is needed to verify the cached content. For the segment cached in the client, the home proxy save a signature of the copy, such as its SHA-1 hash value. The overhead of such an verification is relatively low. More importantly, note that the integrity verification can be enabled/disabled depending on the importance of the content, as in many peer-to-peer systems. The verification frequency can be adaptively set to control the verification overhead. A client contributes its cached data only upon a request from its home proxy. The home proxy will then relay the data to the proxy initiated the request, and if needed, verify the integrity of the data using the signature. As such, the system can easily identifies and blocks malicious clients.

VI. **Performance Evaluation**

In this section, we evaluate the performance of COPACC. We focus on the transmission cost reduced by introducing cooperative caching among proxies and clients. We are also interested in examining the robustness and scalability of this system, as well as identifying the key influential factors.

Unless otherwise specified, the following default settings are used in our evaluation. The video repository in the sever contains 100 CBR videos each of 512 Kbps rate. Their lengths are uniformly distributed in between 100 and 140 minutes; the mean (120 minutes) is a typical length of a movie. As suggested by existing studies on media access patterns, we assume the access probabilities of the videos follow a Zipf distribution with skew factor $\theta = 0.271$ [27]. The cache grain (unit) is set to the size of 2-minute video data. All the cache sizes discussed in this section are normalized by the total size of the video repository, and the transmission costs are normalized by the corresponding cost of a system with no cache. Therefore, our conclusions are also applicable to systems with proportionally scaled parameters.

A. Effectiveness of Cooperative Proxy and Client Caching

A primary design objective of COPACC is to reduce the transmission cost for streaming to clients of asynchronous requests. Hence, in the first set of experiments, we examine the cost reduction under various proxy and client configurations.

We assume there are 4 proxies cooperated with each other, and the client access rate at each proxy is 50 requests per minutes. The ratio between the unit transmission costs of different paths is set to $w^{s\to p} : w^{p\to p} : w^{c\leftrightarrow p} = 10 : 3 : 1$. Note that, this setting is indeed conservative as compared to that in many previous studies [15]. In addition, we are interested in the normalized transmission cost, which depends on this ratio, while not the exact value at each path.

Fig. 5 plots the transmission cost as a function of the total cache space in the system, where $S^p : S^c = 1 : 1$, i.e., the proxies and clients respectively contribute half of the total cache size. Not surprisingly, increasing the total space reduces transmission cost. With unicast, the cost decreases linearly, while with suffix multicast, it decreases much faster. When the total cache space is 0.2 (20% of the video repository),

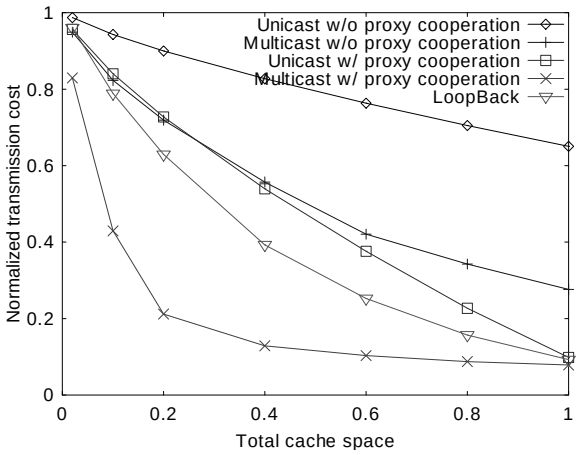

Fig. 5. Transmission cost as a function of the total proxy-client cache space. $S^p : S^c = 1 : 1$

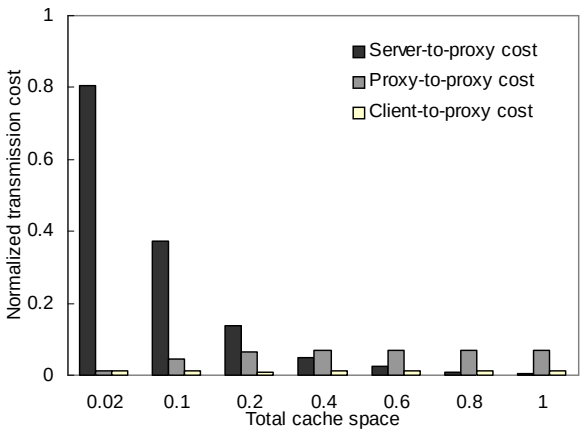

Fig. 6. Transmission cost at different paths with suffix mulitcast.

the cost with suffix multicast has been reduced to 0.2; in other words, a 20% cache space leads to a 80% cost reduction, which implies that batching the requests from local clients can avoid a significant amount of remote transmissions (server-to-proxy). This can also be verified by Fig. 6, which shows the cost due to server-to-proxy transmissions quickly decreases with an increase of the cache space, and becomes a minor part in the total transmission cost when the cache space is over 0.4.

In Fig. 5, we also show the cost when a proxy cooperates with its clients only, while not with other proxies. Clearly, the cost with cooperative proxies are much lower, particularly when multicast is also enabled in local paths. As such, in the following discussions, we focus on the results with cooperative proxies and multicast delivery only.

In addition, we compare COPACC with another hybrid caching system called Loopback [26]. Each client in Loopback dynamically caches a portion of video, and a forwarding ring is formed among the collaborative clients to distribute the video. The results shown in Fig. 5 illustrate that COPACC (with multicast) performs better than Loopback. In particular, when the total cache space is 0.2, the transmission cost in COPACC is one-third of that in Loopback. With the cooperation in deciding what to cache, COPACC utilizes the buffer space by caching distinct segments among the proxies and clients, and ,thus, achieves lower transmission cost.

To further identify the respective contributions of proxy caching and client caching, Fig. 7 depicts the

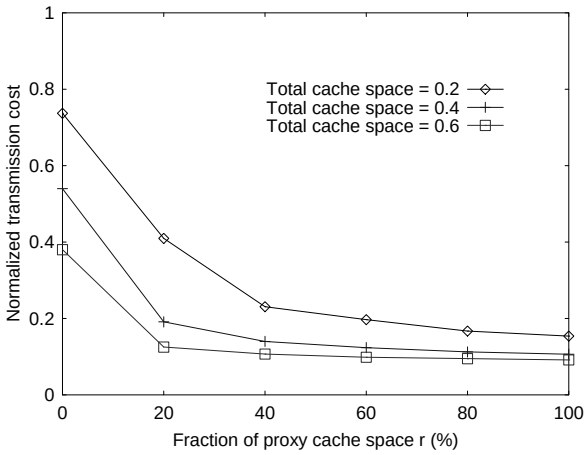

Fig. 7. Transmission cost versus the fraction of the proxy cache space in the total cache space. $r = S^p/(S^p + S^c) \times 100\%$

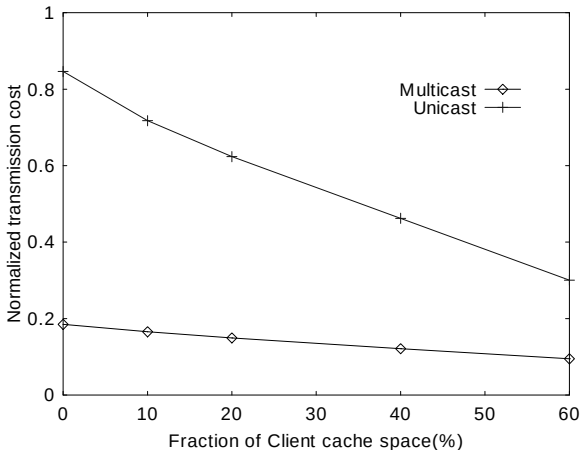

Fig. 8. Transmission cost versus the fraction of the client cache space.

transmission cost versus the fraction of the proxy cache space in the total cache space. We can see that the transmission cost reduces when the proxies contribute a higher fraction in the total cache space of the system. Intuitively, the more cache space contributed by proxies, the more direct transmissions among proxies for delivering a video stream, which generally incur lower costs, because the video data fetched from a client's cache have to be relayed by proxies as well. The best performance is thus achieved when all cache space is in the proxies. Nonetheless, it is often expensive to upgrade dedicated proxies and add more disk spaces. On the other hand, from Fig. 7, we find that, even if the proxy caches constitute a small part in the total cache space, a near optimal cost can still be achieved. As an example, when the total cache space is 0.6 and only 20% is from proxies, i.e., the total proxy cache space is only 0.12, the cost is already less than 0.13, which is quite close to the optimal value (around 0.1) when the fraction of proxy cache is 100%. In other words, client caching well complements proxy caching, making COPACC a very economical alternative to pure proxy caching.

We also examine the benefit of client caching in the system. Fig. 8 plots the transmission cost versus the fraction of the client cache space when the proxies contribute 10% of cache space. It shows that by using client cache, the transmission cost can be further reduced. For example, when the clients contribute 20% of cache space, the transmission cost is 20% less than the pure proxy caching system.

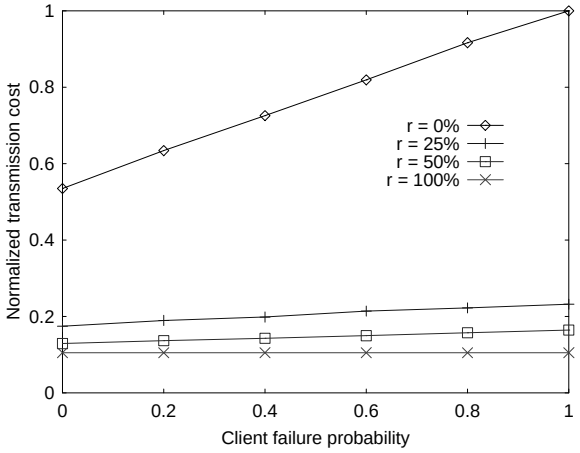

Fig. 9. Transmission cost versus client failure probability.

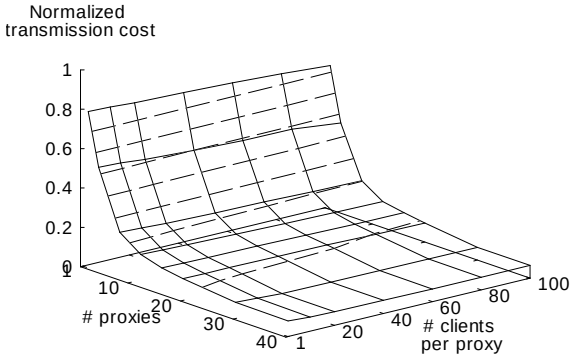

Fig. 10. Transmission cost with different numbers of proxies and clients.

B. Robustness

As in peer-to-peer streaming systems, the robustness in the presence of client failures is also a critical concern in COPACC. To evaluate this, we assume that each client has certain failure probability when its own cache is accessed, but the video access rate from all clients remains constant. In Fig. 9, we show the transmission cost as a function of different client failure probabilities. The total cache space of the system is 0.4, and we vary, r, the fraction of the total proxy cache space in the total cache space from 0% to 100%, which represents two extreme cases: when $r = 0\%$, COPACC degenerates to a pure peer-to-peer system, and, when $r = 100\%$, it degenerates to a pure proxy-based system.

We can see that, when there is no client failure, the costs for different r are quite close if there are certain cache spaces existing in proxies, and the pure proxy-based scheme is the best, which has been explained previously. More importantly, the cost of the pure proxy-based system remains unchanged when increasing client failures, and that for $0\% < r < 100\%$, or a normal COPACC system, is also very stable. For illustration, even if r is 25%, the transmission cost only slightly increases with an increase of failure probability; when the failure probability is 1, the cost remains a low as 0.22. This is because even if a suffix is to be fetched from the server in the presence of client failures, the overhead, shared by a batch of clients, is not excessive. To the contrary, the cost of the pure peer-to-peer system quickly increases and reaches 1 (the cost of a zero-cache system), when all clients fail. Such results demonstrate that the use of dedicated proxies with suffix batching remarkably improves the robustness and resilience of COPACC

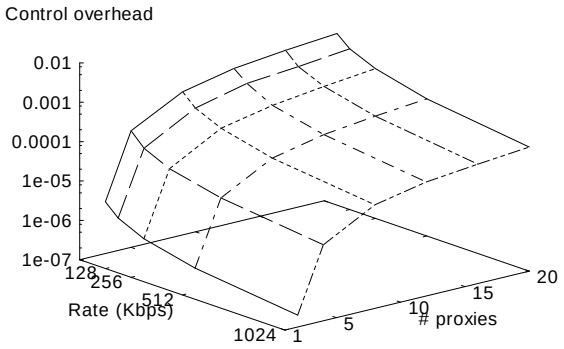

Fig. 11. Control overhead with different number of proxies and streaming rates.

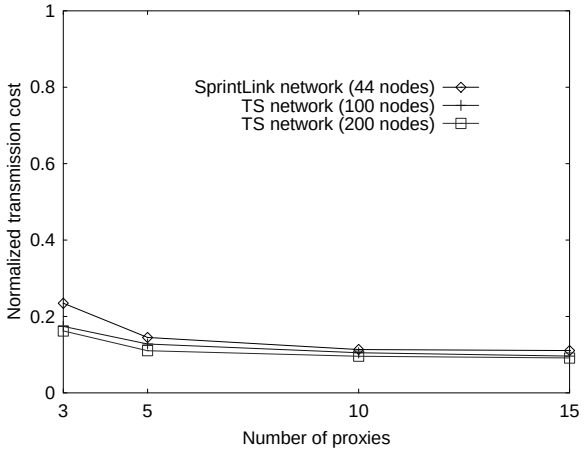

Fig. 12. Transmission cost as a function of the number of proxies under real and synthetic network topologies.

in the presence of client failures, even if the total proxy cache space is minor as compared to the total client cache space.

C. Scalability and Control Overhead

We further explore the scalability of COPACC with larger number of proxies and clients. Fig. 10 shows the total transmission costs for different number of proxies and clients. In this set of experiments, we increased the total number of videos to 1000. The cache space of each proxy, s_j^p, is set to 0.01, and that of each client, $s_{j,k}^c$, is 0.0005. The access rate from each client is set to 0.01 per minute. In other words, while a client joining the system contributes certain cache spaces, it also introduces more requests. Yet, we observe that the transmission cost slightly decreases with more clients, implying that client caching overcomes the increased loads. Note that the normalized cache space of each client is only 0.0005, or equivalently, the half size of one video, which can be easily accommodated by personal computers. With an increase of the number of proxies, we have observed a even more noticeable cost reduction, particularly when the number is changed from 1 to 10. This confirms that proxy cooperation is worth considerations.

Although the cache allocation algorithm is executed solely by a dedicated proxy, it is practical in a network with large number of nodes. It took less than a minute to solve a problem with 1000 videos, 20 proxies and 200 clients in our desktop computer. We expect the actual execution time in a dedicated powerful proxy can be much shorter. Moreover, the execution time can be further reduced by increasing

the cache grain.

The control overhead is also an important concern toward realizing COPACC. We define the overhead of COPACC as the traffic volume of control messages (election, allocation, lookup, and retrieval, etc.) over the total traffic volume, which obviously depends on the scale and streaming rate of the system. Note that the control message is piggybacked in the data packet, that is, no additional packet is generated for the control. Therefore, the control overhead of the system is kept in low level. In Fig. 11, we show the overhead with different number of proxies and streaming rates. The number of clients per proxy is set to 50. It can be seen that the overhead is reasonably low, which is less than 0.3% of the total traffic even with 20 proxies. In addition, the overhead decreases with higher streaming rates. This is mainly because the messages are quite short as compared to video segments, and most messages are locally exchanged.

D. Sensitivity to Network Topologies

So far, we focus on regular network topologies with identical transmission costs between proxies. We have also investigated the performance of our system under various synthetic and real network topologies. Fig 12 shows the costs under three representative topologies: the 44-node SprintLink network and the 100- and 200-node Transit-Stub (TS) networks. The SprintLink network, representing the topology of a typical backbone network in north America, is obtained from the Rocketfuel project at the University of Washington [30]. The TS network is synthesized by the GT-ITM topology generator [31], which attempts to reproduce the hierarchical structure of the Internet by composing interconnected transit and stub domains. For both topologies, we randomly place the given number of proxies to the network nodes, and set the link cost inversely proportional to the bandwidth of each link. A shortest-path routing is then used to determine the path between proxies, and the cost of a path is the sum of costs across all the link of this path. The server is connected to these proxies through a remote link: in SprintLink network, it is assumed to be in Asia, and in TS network, we manually set the unit transmission cost to 5 times the average cost between proxies.

It can be seen that, under all the three network topologies, the transmission costs of COPACC are pretty low and generally decrease with an increase of the number of proxies. The performance under

the TS topology is slightly better, suggesting that COPACC works well with a hierarchical network structure, where local transmission cost is much lower than remote transmission cost. It is worth noting that SprintLink network also follows a hierarchical structure, but many low-level nodes are abstracted into a single nodes. Moreover, the proxies in our evaluation are randomly placed. We thus expect a even better performance when the proxies are strategically placed and cooperated with each other in closer distances.

VII. **Conclusion**

This paper has introduced COPACC, a novel cooperative proxy-client caching system that combines the best features of proxy caching and peer-to-peer communications. It leverages the client-side caching to amplify the aggregated cache space and relies on dedicated proxies to effectively coordinate the communications. We have developed an efficient cache allocation algorithm for distributing video segments among the proxies and clients. We have also proposed a comprehensive suite of protocols that facilitate the interactions among different network entities. It also enables smart and cost-effective cache indexing, searching, verifying operations in this hybrid caching system.

The performance of COPACC has been evaluated under various network and end-system configurations. Our key findings can be summarized as follows: 1) With an amplified total cache spaces, cooperative proxy-client caching significantly reduces the transmission cost for on-demand media streaming; 2) With the assistance from dedicated proxies, it is much more robust than a pure peer-to-peer system, even though the proxies may contribute only a small fraction of the total cache space; and 3) It scales well in larger network, and the cost generally reduces when more proxies and clients cooperate with each other.

We have demonstrated that COPACC works well for symmetric links. However, while many LAN technologies are symmetric, e.g., Ethernet, certain types, like ADSL, have restricted uploading speeds, which may reduce the effectiveness of COPACC. Firewalls could have the same effect. This problem has been a key obstacle to the success of many other peer-to-peer applications, and we expect that the potential solutions to these applications can help COPACC as well. We are currently investigating these issues. Other issues, like how to accurately estimate the system parameters or how to model the participation incentive[32], [33] and security issues[34], should be well addressed in practice. Besides, we

are also interested in studying how replication can be used to improve the robustness of COPACC under unexpected client leave.

APPENDIX

A. NP-Hardness of the CAP problem

In this appendix, we prove that the general optimal cache allocation problem (CAP) is NP-hard. We show this by transforming the optimal resource allocation problem (RAP) to CAP in polynomial time. It is known that RAP is NP-hard and its decision version is NP-complete [35].

In RAP, there are M kinds of resources to be allocated to N activities, indexed from 1 through N, and the total available quantity of resource $j(\in [1 \dots M])$ is N_j. The objective is to minimize the cost in allocating the resources to activities, which can be formulated as:

$$\begin{array}{l} \mathbf{RAP}: \min \sum_{i=1}^{N} f_i(\sum_{j=1}^{M} a_{ij} x_{ij}), \\ \text{s.t.} \sum_{i=1}^{N} x_{ij} \leq N_j, j = 1, 2, ..., m, \\ \quad x_{ij} \in Z^+, \end{array}$$

where x_{ij} is the quantity of resource j allocated to activity i, a_{ij} is the effectiveness for each unit of resource j allocated to activity i, and $f_i()$ is a convex and non-increasing cost function for activity i with given allocations.

Note that the resources and activities in RAP are analogous to the cache spaces and videos in CAP, respectively. Given an instance of RAP, we can create a CAP problem with the following settings: $s_j^p = N_j$,$s_{j,k}^c = 0$, and $p_j^i = x_{ij}$, $i \in [1...N], j \in [1...H], k \in [1...K_j]$. Since $Cost(\{p_j^i\}, \{q_{j,k}^i\})$ can be arbitrary function, we set it as $f_i(\sum_{j=1}^{M} a_{ij} p_j^i)$. We further set V^i to $\sum_{j=1}^{M} N_j$, such that the constraint $\sum_{j=1}^{H} p_j^i + \sum_{j=1}^{H} \sum_{k=1}^{K_j} q_{j,k}^i \leq V^i$ in CAP is always satisfied. Given this transformation, it is obvious that an optimal solution to CAP, p_j^i, leads to an optimal solution to RAP: $x_{ij} = p_j^i, i \in [1...N], j \in [1..H]$. Since transformation is in polynomial time, it follows that problem CAP is NP-hard. ■

B. Optimality of the Greedy Algorithm

In this appendix, we prove the optimality of the proposed greedy algorithm for **PA** with $f_1^i = f_2^i = ... = f_H^i$. We define the matrix of the unit transmission costs $\bar{W}^p(i,j)$ after executing step 1 through 2 as:

$$\begin{pmatrix} \bar{W}^p(1,1) & \bar{W}^p(1,2) & \dots & \dots & \bar{W}^p(1,H) \\ \bar{W}^p(2,1) & \bar{W}^p(2,2) & \dots & \dots & \bar{W}^p(2,H) \\ \vdots & \vdots & \bar{W}^p(i,j) & \ddots & \vdots \\ \bar{W}^p(N,1) & \bar{W}^p(N,2) & \dots & \dots & \bar{W}^p(N,H) \end{pmatrix}$$

where $\bar{W}^p(i,j) = \sum_{j'=1}^{H} \left[w_{j,j'}^{p \to p} + w_{j'}^{c \leftrightarrow p} \right] \lambda_{j'} f_{j'}^i$. Since $f_j^i = f_{j'}^i$ for all $j \neq j'$, we can drop subscript j of f_j^i and simplify the calculation of $\bar{W}^p(i,j)$ as $f^i \cdot \sum_{j'=1}^{H} \left[w_{j,j'}^{p \to p} + w_{j'}^{c \leftrightarrow p} \right] \lambda_{j'}$. We have the following two observations on $\bar{W}^p(i,j)$:

Observation 1. Given $i, \hat{i} \in [1 \dots N]$, $\bar{W}^p(i,j)/\bar{W}^p(\hat{i},j)$ is a constant for any $j \in [1 \dots H]$.

Proof: $\bar{W}^p(i,j)/\bar{W}^p(\hat{i},j)$

$= f^i \cdot \sum_{j'=1}^{H} \left[w_{j,j'}^{p \to p} + w_{j'}^{c \leftrightarrow p} \right] \lambda_{j'} / f^{\hat{i}} \cdot \sum_{j'=1}^{H} \left[w_{j,j'}^{p \to p} + w_{j'}^{c \leftrightarrow p} \right] \lambda_{j'}$

$= f^i / f^{\hat{i}}$

$= \bar{W}^p(i,\hat{j})/\bar{W}^p(\hat{i},\hat{j})$.

Observation 2. Given $j, \hat{j} \in [1 \dots H]$, $\bar{W}^p(i,j)/\bar{W}^p(i,\hat{j})$ is a constant for any $i \in [1 \dots N]$.

Proof: $\bar{W}^p(i,j)/\bar{W}^p(i,\hat{j})$

$= f^i \cdot \sum_{j'=1}^{H} \left[w_{j,j'}^{p \to p} + w_{j'}^{c \leftrightarrow p} \right] \lambda_{j'} / f^i \cdot \sum_{j'=1}^{H} \left[w_{\hat{j},j'}^{p \to p} + w_{j'}^{c \leftrightarrow p} \right] \lambda_{j'}$

$= \sum_{j'=1}^{H} \left[w_{j,j'}^{p \to p} + w_{j'}^{c \leftrightarrow p} \right] \lambda_{j'} / \sum_{j'=1}^{H} \left[w_{\hat{j},j'}^{p \to p} + w_{j'}^{c \leftrightarrow p} \right] \lambda_{j'}$

$= \bar{W}^p(\hat{i},j)/\bar{W}^p(\hat{i},\hat{j})$.

Note that the proxies are sorted in ascending order of cost $\bar{W}^p(1,j)$ and the videos are sorted in descending order of cost $\bar{W}^p(i,1)$ in the greedy algorithm, that is, $\bar{W}^p(i,j) \leq \bar{W}^p(i,j')$ for $j \leq j'$ and $\bar{W}^p(i,j) \geq \bar{W}^p(i',j)$ for $i \leq i'$. We then have another two observations:

Observation 3. $\bar{W}^p(i,j) - \bar{W}^p(i,j-\hat{j}) \leq \bar{W}^p(i-\hat{i},j) - \bar{W}^p(i-\hat{i},j-\hat{j})$ for $\hat{i} \in [1 \ldots (i-1)]$ and $\hat{j} \in [1 \ldots (j-1)]$.

Proof: From observation 1, we have $W^p(i,j)/\bar{W}^p(i-\hat{i},j) = \bar{W}^p(i,j-\hat{j})/\bar{W}^p(i-\hat{i},j-\hat{j}) = a$, where a is a constant. This is equivalent to $\bar{W}^p(i,j) = a \cdot \bar{W}^p(i-\hat{i},j)$ and $\bar{W}^p(i,j-\hat{j}) = a \cdot \bar{W}^p(i-\hat{i},j-\hat{j})$. Here, $0 \leq a \leq 1$ because $\bar{W}^p(i,j) \leq \bar{W}^p(i-\hat{i},j)$ for $\hat{i} \geq 0$. It follows that

$$\begin{aligned}
&\bar{W}^p(i,j) - \bar{W}^p(i,j-\hat{j}) \\
&= a \cdot \bar{W}^p(i-\hat{i},j) - a \cdot \bar{W}^p(i-\hat{i},j-\hat{j}) \\
&\leq \bar{W}^p(i-\hat{i},j) - \bar{W}^p(i-\hat{i},j-\hat{j}).
\end{aligned}$$

Observation 4. $\bar{W}^p(i,j+\hat{j}) - \bar{W}^p(i,j) \geq \bar{W}^p(i+\hat{i},j+\hat{j}) - \bar{W}^p(i+\hat{i},j)$ for $\hat{i} \in [1 \ldots (N-i)]$ and $\hat{j} \in [1 \ldots (H-j)]$.

Proof: From observation 1, we have $\bar{W}^p(i+\hat{i},j)/\bar{W}^p(i,j) = \bar{W}^p(i+\hat{i},j+\hat{j})/\bar{W}^p(i,j+\hat{j}) = b$, where b is a constant. This is equivalent to $\bar{W}^p(i+\hat{i},j) = b \cdot \bar{W}^p(i,j)$ and $\bar{W}^p(i+\hat{i},j+\hat{j}) = b \cdot \bar{W}^p(i,j+\hat{j})$. Here, $0 \leq b \leq 1$ because $\bar{W}^p(i+\hat{i},j) \leq \bar{W}^p(i,j)$ for $\hat{i} \geq 0$. It follows that

$$\begin{aligned}
&\bar{W}^p(i,j+\hat{j}) - \bar{W}^p(i,j) \\
&\geq b \cdot \bar{W}^p(i,j+\hat{j}) - b \cdot \bar{W}^p(i,j) \\
&= \bar{W}^p(i+\hat{i},j+\hat{j}) - \bar{W}^p(i+\hat{i},j)
\end{aligned}$$

The above two observations imply that swapping one unit data of video i in proxy j with that of video i' ($i' \in [1 \ldots (i-1)]$) in proxy j' ($j' \in [1 \ldots (j-1)]$) yields the same or higher total cost, and, similarly, swapping one unit data of video i in proxy j with that of video i' ($i' \in [(i+1) \ldots N]$) in proxy j' ($j' \in [(j+1) \ldots H]$) yields the same or higher cost. As the prefixes are fully packed to the proxies and there is no space left, the solution given by the greedy algorithm is optimal.

■

Acknowledgement: The authors like to thank the referees for their insightful and useful comments for improving the quality and presentation of this paper. This research was supported in part by the RGC Grant CUHK/4186/03E.

REFERENCES

[1] D. Wu, Y. T. Hou, and Y.-Q. Zhang, "Transporting Real-time Video over the Internet: Challenges and Approaches," *Proceedings of the IEEE*, vol. 88, no. 12, Dec. 2000.

[2] L. Golubchik, J. C. Lui, and R. R. Muntz, "Adaptive Piggybacking: A Novel Technique for Data Sharing in Video-On-Demand Storage Servers," *ACM Journal of Multimedia Systems*, vol. 4, no. 3, June 1996.

[3] S. Lau, J. C. Lui, and L. Golubchik, "Merging video streams in a multimedia storage server: complexity and heuristics," *ACM Multimedia Systems*, vol. 6, no. 1, pp. 29–42, Jan. 1998.

[4] S. Lau and J. C. Lui, "Scheduling and data layout policies for a near-line multimedia storage architecture," *ACM Multimedia Systems*, vol. 5, no. 5, pp. 310–323, Sept. 1997.

[5] P. W. Lie, J. C. Lui, and L. Golubchik, "Threshold-based dynamic replication in large-scale Video-on-Demand systems," *Multimedia Tools and Applications*, vol. 11, no. 1, pp. 35–62, May 2000.

[6] J. Liu and J. Xu, "Proxy Caching for Media Streaming over the Internet," *IEEE Communications*, vol. 42, no. 8, Aug. 2004.

[7] J. Wang, "A Survey of Web Caching Schemes for the Internet," *ACM Computer Communication Review (CCR)*, vol. 29, no. 5, Oct. 1999.

[8] S. Chen, B. Shen, S. Wee, and X. Zhang, "Designs of High Quality Streaming Proxy Systems," in *Proc. IEEE INFOCOM'04*, Hong Kong, Mar. 2004.

[9] S. Sen, J. Rexford, and D. Towsley, "Proxy Prefix Caching for Multimedia Streams," in *Proc. IEEE INFOCOM'99*, New York, NY, Mar. 1999.

[10] K.-L. Wu, P. S. Yu, and J. L. Wolf, "Segment-Based Proxy Caching of Multimedia Streams," in *Proc. 10th international conference on World Wide Web (WWW-10)*, Hong Kong, May 2001.

[11] Y. Cui, B. Li, and K. Nahrstedt, "oStream: Asynchronous Streaming Multicast in Application-Layer Overlay Networks," *IEEE JSAC*, vol. 22, no. 1, Jan. 2004.

[12] M. Hefeeda, A. Habib, B. Botev, D. Xu, and B. Bhargava, "PROMISE: Peer-to-peer Media Streaming using CollectCast," in *Proc. ACM Multimedia*, Nov. 2003.

[13] S. Chen, B. Shen, Y. Yan, S. Basu, and X. Zhang, "SRB: Shared Running Buffers in Proxy to Exploit Memory Locality of Multiple Streaming Media Sessions," in *Proc. 24th International Conference on Distributed Computing Systems (ICDCS'04)*, Tokyo, Japan, Mar. 2004.

[14] Z. Miao and A. Ortega, "Scalable Proxy Caching of Video Under Storage Constraints," *IEEE JSAC*, vol. 20, no. 7, pp. 1315–1327, Sept. 2002.

[15] B. Wang, S. Sen, M. Adler, and D. Towsley, "Optimal Proxy Cache Allocation for Efficient Streaming Media Distribution," in *Proc. IEEE INFOCOM'02*, New York, Jun. 2002.

[16] S. G. Dykes and K. A. Robbins, "A Viability Analysis of Cooperative Proxy Caching," in *Proc. IEEE INFOCOM'01*, Apr. 2001.

[17] M. Hofmann, T. E. Ng, K. Guo, S. Paul, and H. Zhang, "Caching Techniques for Streaming Multimedia over the Internet," *Technical Report*, Apr. 1999, Bell Labs.

[18] S. Acharya and B. Smith, "Middleman: A Video Caching Proxy Server," in *Proc. ACM NOSSDAV'00*, Jun. 2000.

[19] Y. Chae, K. Guo, M. M. Buddhikot, S. Suri, and E. W. Zegura, "Silo, Rainbow, and Caching Token: Schemes for Scalable, Fault Tolerant Stream Caching," *IEEE JSAC*, vol. 20, no. 7, pp. 1328–1344, Sept. 2002.

[20] D. A. Tran, K. A. Hua, and T. Do, "ZIGZAG: An Efficient Peer-to-peer Scheme for Media Streaming," in *Proc. IEEE INFOCOM'03*, San Francisco, CA, USA, Apr. 2003.

[21] V. N. Padmanabhan, H. J. Wang, P. A. Chou, and K. Sripanidkulchai, "Distributing Streaming Media Content Using Cooperative Networking," in *Proc. ACM NOSSDAV'02*, May 2002.

[22] Y. Chawathe, S. McCanne, and E. Brewer, "An Architecture for Internet Content Distribution as an Infrastructure Service," *http://www.cs.berkeley.edu/yatin/papers/scattercast.ps*.

[23] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. H. Katz, and J. D. Kubiatowicz, "Bayeux: An Architecture for Scalable and Fault-tolerant Wide-area Data Dissemination," in *Proc. ACM NOSSDAV'01*, NY, Jun. 2001.

[24] M. M. Hefeeda, B. K. Bhargava, and D. K.-Y. Yau, "A Hybrid Architecture for Cost-Effective On-Demand Media Streaming," *Computer Networks*, vol. 44, no. 3, pp. 353–382, Feb. 2004.

[25] L. Guo, S. Chen, S. Ren, X. Chen, and S. Jiang, "PROP: A Scalable and Reliable P2P Assisted Proxy Streaming System," in *Proc. 24th International Conference on Distributed Computing Systems (ICDCS'04)*, Tokyo, Japan, Mar. 2004.

[26] E. Kusmierek, Y. Dong, and D. Du, "Loopback: Exploiting Collaborative Clients for Large-Scale Streaming," in *SPIE Conference on Multimedia Computing and Networking (MMCN'5)*, Jan. 2005.

[27] C. C. Aggarwal, J. L. Wolf, and P. S. Yu, "On Optimal Batching Policies for Video-on-Demand Storage Servers," in *Proc. IEEE International Conference on Multimedia Computing and Systems (ICMCS'96)*, Jun. 1996.

[28] G. B. Dantzig, "Linear Programming and Extensions," *Princeton University Press*, 1963.

[29] H. Garcia-Molina, "Elections in A Distributed Computing System," *IEEE Transactions on Computers*, vol. 31, no. 1, pp. 48–59, Jan. 1982.

[30] N. Spring, R. Mahajan, and D. Wetherall, "Measuring ISP Topologies with Rocketfuel," in *Proc. ACM SIGCOMM'02*, Aug. 2002.

[31] E. W. Zegura, K. Calvert, and S. Bhattacharjee, "How to Model an Internetwork," in *Proc. IEEE INFOCOM'96*, SF, CA, Mar. 1996.

[32] T. Ma, C. Lee, J. C. Lui, and D. K. Yau, "Incentive and Service Differentiation in P2P Networks: A Game Theoretic Approach," *IEEE/ACM Transactions on Networking*, to appear.

[33] ——, "A Game Theoretic Approach to Provide Incentive and Service Differentiation in P2P Networks," in *ACM Sigmetrics/Performance Confernece*, New York, USA, June 2004.

[34] S. Yeung, J. C. Lui, and D. K. Yau, "A Multi-key Secure Multimedia Proxy Using Asymmetric Reversible Parametric Sequences: Theory, Design, and Implementation," *IEEE Transactions on Multimedia*, vol. 7, no. 2, April 2005.

[35] N. Katoh, T. Ibaraki, and H. Mine, "Notes on the Problem of the Allocation of Resources to Activities in Discrete Quantities," *Journal of Operational Research Society*, vol. 31, pp. 595–598, 1980.