Journal of Zhejiang University SCIENCE A
ISSN 1673-565X (Print); ISSN 1862-1775 (Online)
www.zju.edu.cn/jzus; www.springerlink.com
E-mail: jzus@zju.edu.cn

A P2P streaming service architecture with distributed caching*

GUO Pan-hong†1, YANG Yang[1], LI Xin-you[2]

([1]*Department of Computer Science and Technology, University of Science and Technology Beijing, Beijing 100083, China*)

([2]*China State Information Center, Beijing 100045, China*)

†E-mail: guopanhong@tom.com

Received Jun. 8, 2006; revision accepted Oct. 15, 2006

Abstract: Multimedia streaming served through peer-to-peer (P2P) networks is booming nowadays. However, the end-to-end streaming quality is generally unstable due to the variability of the state of serve-peers. On the other hand, proxy caching is a bandwidth-efficient scheme for streaming over the Internet, whereas it is a substantially expensive method needing dedicated powerful proxy servers. In this paper, we present a P2P cooperative streaming architecture combined with the advantages of both P2P networks and multimedia proxy caching techniques to improve the streaming quality of participating clients. In this framework, a client will simultaneously retrieve contents from the server and other peers that have viewed and cached the same title before. In the meantime, the client will also selectively cache the aggregated video content so as to serve still future clients. The associate protocol to facilitate the multi-path streaming and a distributed utility-based partial caching scheme are detailedly discussed. We demonstrate the effectiveness of this proposed architecture through extensive simulation experiments on large, Internet-like topologies.

Key words: Cooperative steaming, Peer-to-peer networks, Partial caching scheme, Distributed caching
doi:10.1631/jzus.2007.A0605 **Document code:** A **CLC number:** TP393; TP37

INTRODUCTION

To tackle the scalability issue of unicast-based media streaming architectures, hierarchy-based (tree-based) solutions were proposed in the literature, such as IP-multicast (Deering, 1991), application-level multicast(ALM) (Lee *et al*., 2001; Banerjee *et al*., 2002; Tran *et al*., 2003; Cui *et al*., 2004) and proxy-based structure (Liu and Xu, 2004; Liu *et al*., 2004). In the hierarchy of IP-multicast, the server acts as root-node and the clients act as the leaf-nodes. The intermediate nodes are routers that provide point to multipoint transmission through packet replication. The ALM tree purely consists of the server and the clients. In ALM, data are delivered over hierarchical multicast delivery tree out of a set of unicast-based connections among ordinary clients. The proxy-based hierarchy consists of the original server, multiple proxy servers and the clients. The proxy servers are dedicated devices that are usually powerful (e.g., huge disk volume, huge memory size, large I/O bandwidth, etc.) and stable, as compared with ordinary clients.

The IP-multicast has seen very slow deployment because of its high demand on router capability (Diot *et al*., 2000). The ALM has also seen very slow deployment because the tree is usually very unstable. In proxy-based hierarchy, the proxy servers are usually deployed on the edge of the Internet. It does not need the router support and is much more stable than ALM. Several works (Diot *et al*., 2000; Krishnamurthy *et al*., 2001; Akamai Technologies, 2000; Vakali and Pallis, 2003) have constructed content delivery networks (CDNs) by introducing a dedicated set of proxy servers so as to provide high performance media distribution services over the Internet. Despite the fact that proxy-based hierarchy has proven to be effective, it does not solve the scalability problem completely. The reason is simply that the deployment of proxy

* Project (Nos. 90412012 and 60673160) supported by the National Natural Science Foundation of China

servers, which must be powerful and stable online servers, is expensive. Presently, some research has been conducted to study the effectiveness of data sharing of P2P-based (Yim *et al*., 2001; Gopalakrishnan *et al*., 2004) or BitTorrent-like overlay systems (Qiu and Srikant, 2004; Pouwelse *et al*., 2005).

In this paper, we propose a peer-based cooperative streaming architecture with distributed caching for delivering media content effectively over the Internet. The basic idea there is to ask the concurrent peers to cooperate to improve the streaming quality mutually. The proposed architecture combines the strengths of many techniques such as multi-source/multi-path streaming, P2P overlay networks and distributed proxy caching. Specifically, we try to improve the streaming quality of a later client through the help of one or more earlier client(s) in the same neighborhood, given that there is no dedicated powerful proxy server. The proposed system is based on the layered scalable video coding source bit-stream. Assume an earlier client, Client A, stores the title at a specific quality (via distributed caching), a later client, Client B, collects the data cached by peer server (i.e., Client A) and, at the same time, requests more data from the original server (via multi-source streaming). As a result, Client B will enjoy better quality than Client A does. Client B, in turn, also caches its data (partially or entirely) so as to serve others later on. With the same logic, an even later client, Client C, will enjoy even better quality by the multi-path streaming from the original server, Clients A and B. The procedure is illustrated in Fig.1. Clearly, the quality of a certain title will improve progressively as more clients viewed it, because of more chances for cooperation. Note that the later client could be Client A/B if they want to watch the same title again.

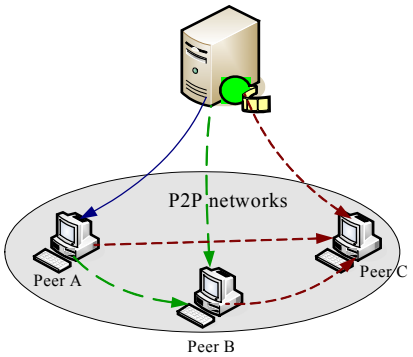

Fig.1 Illustration of peer-assisted quality enhancement

This is very useful for Video-on-Demanding (VOD) those hottest titles.

In this paper, we focus on the following two problems. The first problem is on the distributed multi-source based video streaming. In our architecture, except the original server, all the other sources are peers which are heterogeneous in many aspects including contents cached, bandwidth, and availability, etc. We solve this problem through a receiver-driven protocol based on a sliding-window mechanism. The second problem is about the distributed cache replacement scheme for each peer. In general, ordinary clients can hardly provide sufficient disk space to cache all media data they viewed. Therefore, contents must be selectively cached/replaced according to a certain replacement policy. However, existing replacement schemes for traditional proxies tend to make clients cache similar low quality contents. This is ill-suited to our multi-source streaming model. To address this problem, we proposed a novel utility-based cache replacement scheme to help client's cache content that has the potential of providing better performance in the future cooperation, taking into account both the characteristics of clients and their peers.

MULTI-SOURCE STREAMING PROTOCOL

In this section, we will discuss the distributed multi-path streaming protocol. To enable fine-grain cooperation among peers, we slice the FGS layers into roughly equal-sized logical layers (LLs). Each LL is further segmented into equal-sized pieces termed segments. Assume the playback duration for each segment is Δt. The segment is the minimum manipulation unit for delivering and caching in our framework. In this architecture, we adopt a receiver-driven protocol for client's cooperation. Namely, the receiver determines and instructs each peer server cooperative in a transmission policy based on the information collected from the supplying peers or its own measurement. Furthermore, a receiver will timely update its supplying peers' transmission policy by periodically monitoring the end-to-end network dynamics between them.

In our proposed architecture, every participating peer contributes some of its outbound bandwidth and storage to the system. Before making transmission

policies for all senders in a certain cooperation window, the receiver needs to contact each peer server X_i for collecting relevant information. For measuring the available bandwidth between C and X_i, the receiver, C, directly uses a TCP throughput function to calculate its expected bandwidth. One possible function is as follows (Handley *et al*., 2001):

$$B=\frac{s}{RTT\sqrt{2p/3}+RTO\cdot(3\sqrt{3p/8})\cdot p\cdot(1+32p^2)}. \quad (1)$$

This gives the TCP throughput B in bytes/s, as a function of the packet size s, round-trip time RTT, steady-state loss event rate p, and the TCP retransmit timeout value RTO. The following control loop is performed by each receiver:

Step 1: Measures or estimates P, RTT, and RTO.

Step 2: If it receives a Report Packet with a new rate vector, goto Step 3, else goto Step 1.

Step 3: Stores the rate vector in P and calculates B using Eq.(1).

Step 4: Goto Step 1.

Furthermore, the estimated available bandwidth is exponentially smoothed and discretized using as the unit bit rate of a logical layer. In other words, the channel bandwidth is represented as the number of segments it can deliver in time Δt in this paper. In addition, we also adopt the hysteresis algorithm (Chu *et al*., 2001) to tackle possible oscillations.

We developed a mathematical model to determine the optimal transmission policies for peer servers using the information collected above. The performance can be further enhanced with the pre-fetch technique. The protocol is implemented based on a sliding-window mechanism, and the length of cooperation window is WT, as illustrated in Fig.2. The receiver maintains a playback time. At playback time t_0, the receiver collects relevant information and decides a set of transmission policies for all its peer servers in the time interval $[t_0+T, t_0+T+WT]$, where T is the duration of contents that have been buffered. This interval is called cooperation window hereafter. Each peer server will be assigned an independent transmission policy that contains an ordered list of segments that it should send within the cooperation window. The cooperation window slides periodically as the playback proceeds. Thus, after WT, the receiver will make new transmission policies for the next cooperation window. New transmission policies will preempt those of old ones. That is, if a peer server receives a new transmission policy, it will ignore the old policy even though the old policy has not been finished and start to execute the new policy.

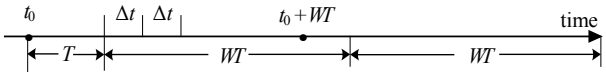

Fig.2 Illustration of sliding-window

Suppose the receiver C owns a set of N senders $P=\{P_1, P_2, \ldots, P_N\}$, and the corresponding estimated available bandwidths between C and P_i are $\{b_1, \ldots, b_N\}$. Let L be the maximum number of logical layers of the source bit stream, with the origin server denoted as P_0. Define a matrix $\boldsymbol{K}=\{k_{ij}^{\,d}\}$. Where $k_{ij}^{\,d}=1$ indicates that P_i has cached the corresponding segment of the jth logical layer within playback time interval $[t_0+T+(d-1)\Delta t, t_0+T+d\Delta t]$. This matrix defines the sets of segments of the multiple senders that C can retrieve from.

To begin, we divide the whole cooperation window into $WT/\Delta t$ sub-periods, each of which is Δt in time, and make optimal transmission policies for each sub-period independently. In other words, for the dth sub-period $[t_0+T+(d-1)\Delta t, t_0+T+d\Delta t]$, the receiver should determine which segments (may come from different logical layers) within the playback interval of each sender so as to maximize its aggregated effective number of segments while not violating the constraints on link bandwidth, layer dependency, and so on. This can be modeled as a zero-one integer programming problem. Let boolean variable $h_{ij}^{\,d}\in\{0, 1\}$ represent whether or not C will receive the corresponding segment of the jth logical layer from P_i in this sub-period, with $h_{ij}^{\,d}=1$ meaning C will. Let $J=\{1, 2, \ldots, L\}$ and $I=\{0, 1, \ldots, N\}$. The problem is then formulated as follows:

$$\text{Maximize} \sum_{i\in I, j\in J} x_{ij}, \quad (2)$$

$$\text{s.t.} \sum_{i\in I} h_{ij}\le 1, \ \forall j\in J, \quad (3)$$

$$\sum_{i\in I}(h_{ij}-h_{i(j+1)})\ge 0, \ \forall j\in J, \quad (4)$$

$$\sum_{j\in J} h_{ij}\le b_i, \ \forall i\in I, \quad (5)$$

$$h_{ij}^{d}\le k_{ij}^{d}, \ \forall i\in I; j\in J; d\in[1, WT/\Delta t]. \quad (6)$$

Eq.(2) is the objective function, which is to maximize the total number of segments the peer will receive in the sub-period. Eq.(3) indicates that duplicated segments from different senders should be avoided. Eq.(4) expresses the dependencies that must be maintained between segments from different logical layers. Eq.(5) is the constraint on bandwidth consumption. Finally, Eq.(6) guarantees the segment to be delivered should have been stored in the corresponding supplying peer's cache.

Fig.3 shows a simple example of this model. In the example, the total number of logical layers is 5. Although the bandwidth to the server is only 3, C still received all the 5 logical layers. More than that, 1/3 of the bandwidth from C to the server was saved. Hence, this example illustrates that our scheme can not only achieve higher quality but also alleviate the server load.

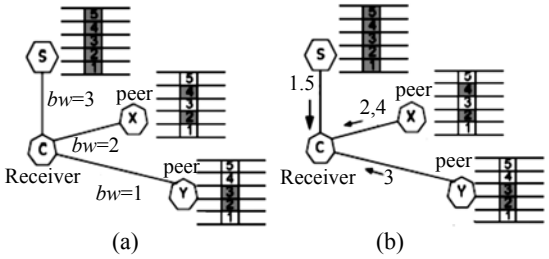

Fig.3 Example of transmission policy in a sub-period
(a) Topology; (b) Transmission policy

The performance can be further enhanced by using pre-fetch technique. By pre-fetch we mean that a peer server can use its redundant bandwidth (the portion of bandwidth exceeding the requirement for the current transmission task) to deliver in advance the segments that are scheduled in future sub-periods. We incorporate the pre-fetch technique into the above model by modifying Eq.(5) as follows:

$$\sum_{j\in J} h_{ij}^{d+1} \le b_i + br_i^{d+1}, \quad \forall i \in I, \tag{7}$$

where br_i^{d+1} represents the total redundant bandwidth for P_i in all previous d sub-periods $[t_0+T, t_0+T+d\Delta t]$, which is to be utilized to deliver future segments that were originally scheduled for $[t_0+T+d\Delta t, t_0+T+(d+1)\Delta t]$ sub-period in the current sub-period. In addition, br_i^{d+1} can also be calculated using

$$br_i^{d+1} = b_i + br_i^d - u_i^d = d \cdot b_i + \sum_{k=1}^{d} u_i^k + \Delta_i^1, \tag{8}$$

where u_i^k is the bandwidth consumed in sub-period $[t_0+T+k\Delta t, t_0+T+(k+1)\Delta t]$ and $\Delta_i^1 = 0$.

In general, the mathematic model above can be solved using branch and bound method (Papadimitriou and Steiglitz, 1982) easily because the number of peers, the maximum logical layers and the available bandwidth (normalized by the number of logical layers) among peers are relatively small in practice. In the case that the scale increases, it is still easy to find some simple heuristics to obtain a suboptimal solution. The final transmission policy for each peer server within the cooperation window is the collection of the segments it should send in all sub-periods.

CACHE REPLACEMENT SCHEME

Segment-based caching has been proved as an effective way for multimedia caching. So in our proposed system, we still use segment-based cooperative caching scheme. A centralized P2P system should need an Index Server for coordination of distributed cache. The Index Server should be a reliable and always-on entity in the network. Definitely, the original server can act as an Index Server that maintains a subset of indices of media segments in the system for content location. A media segment index should contain a location list of peers, each of which caches a copy of the media segment, and the access information of this segment, which is used for replacement operations. We assume that each media object is expressed as a segment-link. The segment link is illustrated in Fig.4.

The segment locating is conducted in two steps: first to route the request to Index Server; and then to select a peer that caches a copy of the segment via the info of segment link which is maintained in Index Server. The selection of serving peer can be optimized according to the capacities and workloads of peers caching the required segments. Once the required segment is successfully located, the media streaming between the serving peer/proxy server and the requesting peer is established.

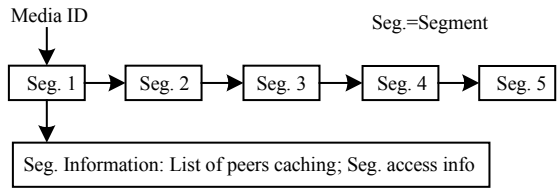

Fig.4 Segment link maintained in Index Server

Note that our proposed scheme for segment location is a centralized policy that all segment indexes are stored in dedicated server. Someone may wonder whether it would be a bottleneck because of intensive control overhead. As a matter of fact, in cooperative web caching, a critical issue is how to efficiently locate web pages at minimum communication costs. On the occasion of cooperative streaming caching, however, it is not a major concern as the bandwidth consumption for streaming objects is of orders of magnitude higher than that for object indexing and discovering.

Furthermore, we propose a distributed utility-based fine-grain cooperative cache replacement scheme. We evaluate each segment and assign a utility index for it. The utility index is essentially a measurement of the potential value that the segment, if cached, would help other peers. The objective of our replacement policy is then to maximize the utility indices of all the segments cached subject to the cache size constraint. Evidently, when the cache is full, the segments with the smallest utility indices should be replaced by newer segments with higher utility indices. Moreover, the utility index must be evaluated dynamically instead of being assigned a constant value. When a new segment arrives, an initial utility index is assigned, and then the index will be updated periodically until it is flushed out. In this way, the cache management can effectively handle the emergence of new hot titles.

The key of our cache replacement scheme is to measure the utility index for each segment. The calculation of utility index should take into consideration not only the properties of the local cache but also the status of those potentially cooperative caches. We define the utility index for a client, C, to cache a certain segment, g, as follows:

$$I(C,g)=W(C,g)^{k_1}+F(C,g)^{k_2}, \tag{9}$$

where $W(C, g)$ indicates the cost for C to obtain the segment g, $F(C, g)$ represents the estimated frequency at which the segment g will be retrieved by other clients in the future if it is cached by C. The two exponentials, k_1 and k_2, are introduced to balance the impacts of the two factors. For example, if we set k_1=0, the scheme degenerates to traditional LFU algorithm. If k_2 is set to 0, the segments are cached or replaced solely according to the retrieval cost.

The rationale of the two factors $W(C, g)$ and $F(C, g)$ is as follows. Intuitively, the utility index should be a function of cost savings of other future peers that the current peer will service. Without loss of generality, assume X is a client in the neighborhood of C and will be requesting the segment g. If C does not cache g, X will have to obtain g from the original server directly or from other peer servers from P_g, which represents the set of potential peer servers that have cached g. In this case, the cost for X to obtain g will be $W(X, g)$, which is close to $W(C, g)$ because the distance between X and C is much smaller than that between X and any other peer server or the original server. However, if C caches g, X will retrieve g from C because they are the closest. The cost associated is $W(X, C, g)$. The cost savings is then $W(X, g)-W(X, C, g)$. Since $W(X, C, g)$ is usually much smaller than $W(X, g)$ and $W(X, g)$ is close to $W(C, g)$, the cost savings is approximately equal to $W(C, g)$, which is readily collectible. On the other hand, if the title is hot, many clients in the neighborhood of C may also view it in future. That is, $F(C, g)$ is large. Obviously, caching g in C may lead to more cost savings because of g's potentially high retrieval frequency. Therefore, the utility index of a segment is directly related to the above two factors and is empirically defined as the product of them in our scheme.

To calculate the cost function $W(C, g)$, two observations need to be considered. Firstly, the number of peers having cached the segment g, i.e., $|P_g|$, has a direct impact on the cost. For instance, if there are already many cached copies of the segment, the cost savings for client C to cache g should be much smaller, as compared with the case where only few copies have been cached. Secondly, as stated above, the heterogeneities of peers have direct impact on the system performance of our architecture. Specifically, three heterogeneities that influence the service capacity of a peer server are considered. These hetero-

geneities include: (1) available bandwidth between the peer server and C; (2) availability of the peer server; (3) load of the peer server. A peer server with little available bandwidth, or is frequently offline, or heavily loaded can hardly provide useful (reliable) service. In our work, we consider all the above-mentioned factors and define the cost function $W(C, g)$ as follows:

$$W(C,g)=\frac{1}{\max\{V(P_g,C,g),V_0\}}. \quad (10)$$

And

$$V(P_g,C,g)=\sum_{p\in P_g}\{P_a(p)[1-P_L(p)]B(C,p)\}, \quad (11)$$

where $P_a(p)$ is the probability that the peer will be online, $P_L(p)$ is the probability that the peer is overloaded and cannot serve as a peer server, and $B(C, p)$ is the available bandwidth estimated and discretized. $P_a(p)$ is estimated by the peer itself as the ratio of average online time to total time elapsed. $P_L(p)$ is estimated similarly as the ratio of average load to the peer's maximum affordable load. Moreover, we set an empirical value, $1/V_0$, as the default value of $W(C, g)$ in case that P_g empty or all the peers in P_g fail to service the desired segment g. Note that $W(C, g)$ of each segment is calculated only once when it first arrives Client C. We do not update its value periodically because $W(C, g)$ is a measurement on an average sense. Moreover, frequent update of $W(C, g)$ will cause much protocol overhead since the calculation consumes many network resources to collect necessary relevant information.

$$F(C,g)=\begin{cases}\dfrac{1}{P_g}\sum_{p\in P_g}F(p,g), & P_g\neq\varnothing,\\ F_0, & \text{otherwise.}\end{cases} \quad (12)$$

To calculate $F(C, g)$, an initial value is first assigned using Eq.(12) when a new segment comes in. Eq.(12) indicates that, if the client discovers some peers have cached g, it will use the mean value of those peers' current frequency as the initial value. Otherwise, it takes a default value, F_0.

If a segment is already cached, its frequency will be updated periodically using Eq.(13) so as to match the popularity of the title. Specifically, we update the frequency in a smooth manner.

$$F'(C,g)=w_1F(C,g)+w_2\Delta_F(g) \text{ s.t. } w_1+w_2=1, \quad (13)$$

where $\Delta_F(g)$ represents the frequency at which the segment has been used to serve other peers during the update period Δ, w_1 and w_2 are the relative weights of frequency of the two different circumstances.

SIMULATIONS AND ANALYSIS

In this section, we present the simulation results to demonstrate the performance of the proposed architecture.

Simulation setup

In all the simulations, we use large hierarchical, Internet-like topologies, all with three levels. The top level consists of several Transit Domains, which represent large Internet Service Providers (ISPs). The middle level contains several stub domains, which represent small ISPs, campus networks, moderately sized enterprise networks, etc. (Each stub domain is connected to one of the Transit domains). At the bottom level, end hosts (peers) are connected to stub domains. The first two levels (router-level) contain transit routers and stub routers, which are generated using the GT-ITM tool (Zegura *et al*., 1996). We then randomly attach end hosts (peers) to stub routers with uniformed probability. The network topology for the presented results consists of 5 transit domains, each with 10 transit nodes, and a transit node is then connected to 8 stub domains, each with 4 stub nodes. The total number of nodes is thus 1650 nodes. Fig.5 shows an example of 100 nodes topologies generated by GT-ITM.

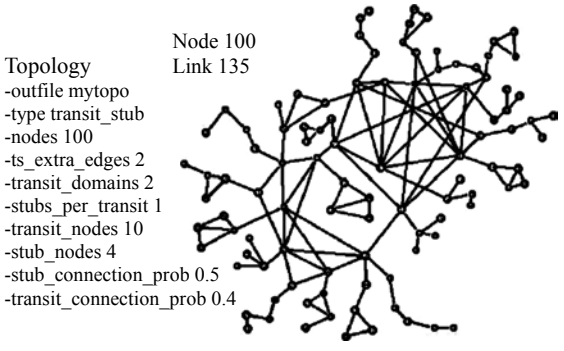

Fig.5 Example topology generated by GT-ITM

During the simulation, there are totally 500 videos published in the network, each with 512 kbps constant playback bit rate and 1 hour length and is divided into 8 logical layers. Each LL is roughly 64 kbps. In our simulations, an LL is further divided into a number of segments each with an interval of 10 s, while the length of the cooperation window is set to 2 min. That is, Δt=10 s and WT=120 s. In our simulations, we randomly select 500~1500 nodes to build an overlay network. In default, the number of participated clients is 700. Each client reserves a fix sized disk space (the default value is 500 MB) as its local cache. The maximum load (the maximum number of peers a client can support concurrently) is set to four. To simulate the probability that a peer is online, we utilized a two state Markov model. A client can be in two states: Inactive and Active. The transition of one client is independent of the states of other clients. The transition probability from Active to Inactive is randomly distributed in the range from once per 80 min to once per 160 min. The transition probability from Inactive to Active is also randomly distributed in the range from once per 40 min to once per 80 min.

For VOD service, client requests are generated as a Poisson process with arriving rate 15 requests/min. These requests are then uniformly distributed to those Active clients that are not viewing any title. This is reasonable because, in general, a client will not play more than one title simultaneously. The popularity of video titles follows a Zipf like distribution with a default skew factor 0.271. Table 1 summarizes the default values and the ranges of the parameters used in our simulations.

Table 1 List of simulation parameters

Parameter	Default	Range
Number of participating clients	700	500~1 500
Mean outbound bandwidth	4	N/A
BW (clients, servers) (kbps)	200	N/A
Media length (min)	60	30~90
Media rate (kbps)	512	N/A
Δt (s)	10	N/A
WT (s)	120	N/A
Arrive rate (requests/min)	15	N/A
Zipf skew factor	0.271	0.1~0.9
Cache size (MB)	500	200~1 000
Number of medias	500	N/A
Max load of peer	4	N/A

Simulation results

One target of our simulations is to test the performance improvement of our multi-path cooperative streaming models presented in Section 2. The other target is to demonstrate the effect of our distributed utility-based fine-grained cooperative cache replacement (DUFC) policy. We compare DUFC with the LRU-based fine-grain cache replacement scheme (LRUF). LRUF replaces the least recently used segments firstly and adopts the fine-grain replacement pattern introduced in (Rejaie *et al*., 1999).

Note that, were it not for the proposed peer-assisted cooperative streaming architecture, all the clients would receive at most a quality of 200 kbps, which already reaches the bottleneck bandwidth of its connection to the server. With the proposed architecture, no matter what cache replacement schemes are adopted, the overall performances are improved drastically. For example, the average quality of all the titles is at least doubled, as shown in Fig.6. For the hottest titles, the quality improved several times, as shown in Fig.8.

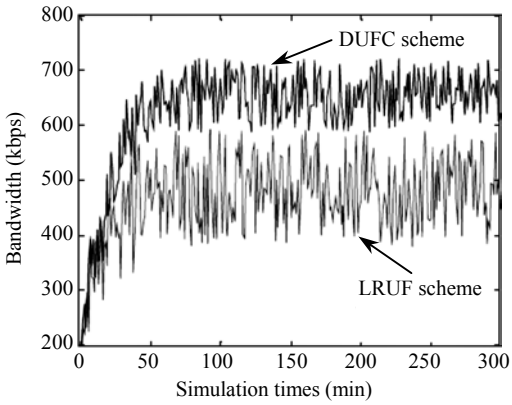

Fig.6 Average quality of all requests

We simulated our proposed system using DUFC and LRUF respectively. All the parameters take default values from Table 1. The results are shown in Fig.6, where the x-axis represents the simulation time and the y-axis is the average quality of all requests generated with the metric of bit rate experienced.

As shown in Fig.6, although the average quality in the initial period of simulations is low, it keeps on improving with time and converges into a high and steady value finally (after about 50 min for LRUF and 75 min for DUFC), because the caches of most peers were not fully filled in the early time of simulations.

During this period, it was hard for a client to find peers for cooperation. As time went on, more and more data got cached in the cluster of peers, which increased the opportunities for peers' cooperation. However, because of the limit of the cache size, the quality will not increase infinitely but fluctuate around a steady value. Another obvious observation is that DUFC results in a higher quality enhancement (over 25%) than LRUF. This is because DUFC can make more efficient use of limited disk space, by caching more useful segments that improve the co-operation efficiency among peers in return.

We further explain the conclusions above using Fig.7 and Fig.8, where the *x*-axes represent the title number sorted by their popularity from high to low, while the *y*-axes represent respectively the average number of peers that have cached the title and average quality (bandwidth) of requesting for the corresponding title. For the sake of clear presentation, we list the top 300 titles only.

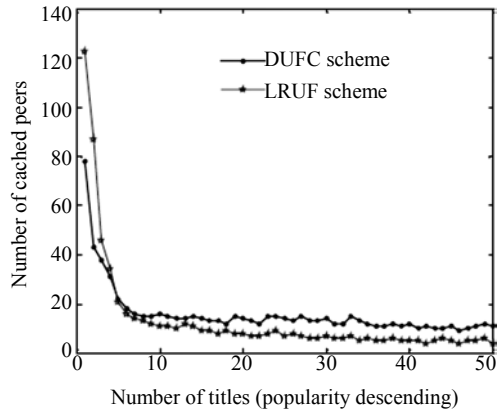

Fig.7 Number of peers that have cached the titles

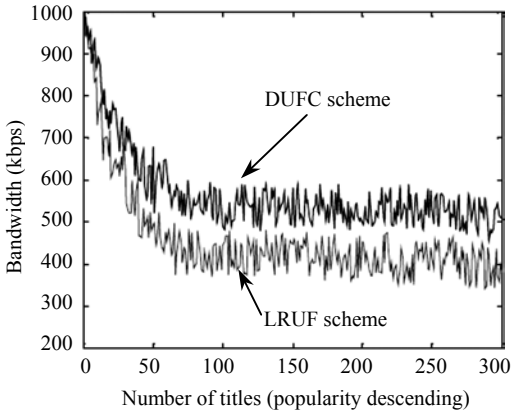

Fig.8 Average quality of different titles

Fig.7 shows that most titles except several hottest ones are cached by more peers in DUFC than in LRUF. In other words, DUFC cached less redundant copies of the hottest titles and donated the disk space saved to other titles. As a result, the quality of most titles improved significantly because more peers might have cached those titles for promoting multi-path cooperative streaming, as shown in Fig.8. Moreover, we found that the average quality of those hottest titles still greatly improved (Fig.8) even though the number of cached copies of those titles indeed dropped in DUFC (Fig.7), as compared with LRUF. The reasons are: (1) they still got cached by many (probably enough) peers, e.g., more than 5 peers cached title 5; (2) different peers might cache non-overlapped segments of a certain title thus render it possible for multi-source cooperative streaming, while in LRUF, most peers tended to cache similar segments from few lower logical layers.

The main concerning issue of the proposed mechanism is the network overhead because there are frequent information exchanges between peers. In our proposed system, the video stream is partitioned into many segments, each node periodically exchange segments' availability information with partners, and then schedules which segment to be fetched from which partner accordingly. Usually, the number of partners is a key factor of the control overhead. Fig.9 depicts the normalized control overhead as a function of the average number of partners in a stable environment, i.e., the lifetime of each node equals to the playback duration of streaming, typically as 60 min. In this experiment, we define the control overhead as

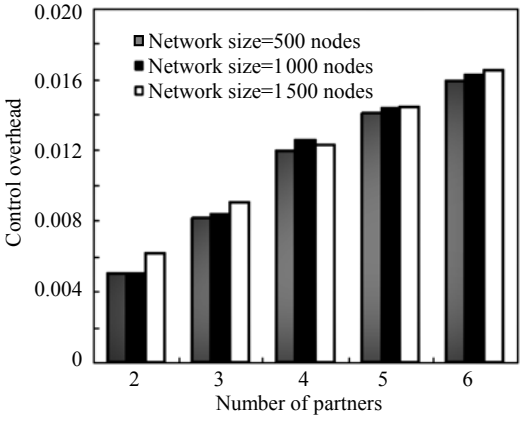

Fig.9 The control overhead as a function of the number of partners

the ratio of control traffic volume over video traffic volume at each node. The source bit stream is composed of 3 logical layers, with mean value of 64 kpbs bandwidth for each layer. The figure shows that the overhead in the system is trivial compared with video content traffic (1.62% with MAX value). Definitely, the control overhead increases with a larger number of partners, but with the increasing of overlay network size, the control overhead is kept invariability on the whole.

There are several parameters that may directly affect the performance of the proposed architecture. In Fig.10, we demonstrate that the average quality improves as the local cache size of each peer increases. We also include the special cases in which peer's cache size is zero, which is equivalent to traditional server/client topology situation since no client can cooperate with others. This is obvious because larger cache can store more segments, which makes the multi-source cooperative streaming scheme work more effectively, and then get better stream quality. Furthermore, Fig.8 also reveals that DUFC yields better performance than LRUF in all conditions. For a specific desired average quality, DUFC requires a much smaller cache space than LRUF does. The high efficiency character of DUFC has a profound impact on the system cost as the storage is still a substantial expense for a personal computer.

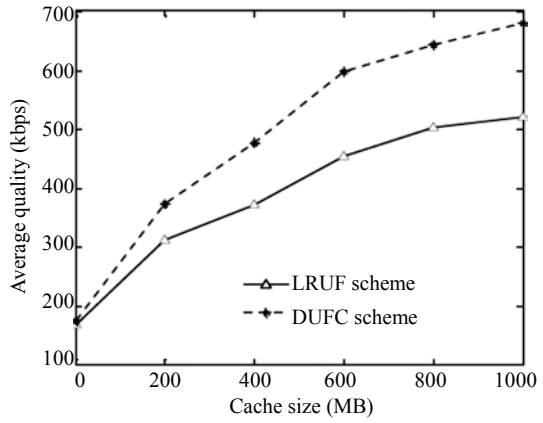

Fig.10 Impact of local cache size on the average quality

CONCLUSION

In this paper, we propose a cost-effective cooperative streaming architecture for heterogeneous peer-to-peer networks. In this architecture, published videos are encoded into multi-layered source bit stream and split into many segments, which are distributed to overlay peers. The main design philosophy is that outbound bandwidths from multiple peers who have viewed and cached the same title before can be aggregated to serve a single video streaming request. We propose the protocol for the multi-source cooperative streaming and the distributed utility-based fine-grain cooperative cache replacement scheme. We conducted extensive simulation experiments on large, Internet-like topologies by GT-ITM topology generator. Simulation results demonstrate that the proposed architecture bring in significant quality enhancement. The average quality of all the titles is more than doubled. For the hottest titles, the quality improved several times. In our future work, we plan to implement our prototype and deploy it over PlanetLab (http://www.planet-lab.org), which can evaluate its performance in the real Internet environment.

References

Akamai Technologies, 2000. FreeFlow Overview. Http://www.cs.washington.edu/homes/ratul/akamai/freeflow.pdf

Banerjee, S., Bhattacharjee, B., Christopher, K., 2002. Scalable Application Layer Multicast. Proc. ACM SIGCOMM. Pittsburgh, USA, p.205-220.

Chu, Y., Rao, S.G., Seshan, S., Zhang, H., 2001. Enabling Conferencing Applications on the Internet Using an Overlay Multicast Architecture. Proc. ACM SIGCOMM, **31**(4):55-68. [doi:10.1145/964723.383064]

Cui, Y., Li, B.C., Nahrstedt, K., 2004. Stream: asynchronous streaming multicast in application-layer overlay networks. *IEEE J. Selected Areas Commun.*, **22**(1):91-106. [doi:10.1109/JSAC.2003.818799]

Deering, S.E., 1991. Multicast Routing in a Datagram Internetwork. PhD Thesis, Stanford University.

Diot, C., Levine, B., Lyles, B., Kassem, H., Balensiefen, D., 2000. Deployment issues for the IP multicast service and architecture. *IEEE Network*, **14**(1):78-88. [doi:10.1109/65.819174]

Gopalakrishnan, V., Silaghi, B., Bhattacharjee, B., Keleher, P., 2004. Adaptive Replication in Peer-to-Peer Systems. Proc. 24th International Conference on Distributed Computing Systems (ICDCS'04), p.360-369. [doi:10.1109/ICDCS.2004.1281601]

Handley, M., Pahdye, J., Floyd, S., 2001. TCP Friendly Rate Control (TFRC): Protocol Specification. IETF Internet Draft, draft-ietf-tsvwg-tfrc-02.

Krishnamurthy, B., Wills, C.E., Zhang, Y., 2001. On the Use and Performance of Content Distribution Networks. Proc. SIGCOMM IMW. San Francisco, p.169-182.

Lee, K.W., Ha, S., Li, J.R., Bharghavan, V., 2001. An Application-level Multicast Architecture for Multimedia Communications. Proc. 8th ACM International Conference on Multimedia. Los Angeles, p.398-400.

Liu, J., Xu , J., 2004. Proxy caching for media streaming over the Internet. *IEEE Communications Magazine—Special Issue on Proxy Support for Streaming Internet*, **42**(8): 88-94.

Liu, J., Chu, X., Xu, J., 2004. Proxy Cache Management of Fine-grained Scalable Video Streaming. Proc. IEEE INFOCOM. Hong Kong, China, p.1490-1500.

Papadimitriou, C.H., Steiglitz, H., 1982. Combinatorial Optimization: Algorithms and Complexity. Englewood Cliffs. Prentice Hall, New Jersey.

Pouwelse, J., Garbacki, P., Epema, D., Sips, H.J., 2005. The BitTorrent P2P File-sharing System: Measurements and Analysis. Proc. International Workshop on Peer-to-Peer Systems. Ithaca, New York, USA, p.134-142.

Qiu, D., Srikant, R., 2004. Modeling and Performance Analysis of BitTorrent-like Peer-to-Peer Networks. Proc. ACM SIGCOMM, **34**(4):367-378. [doi:10.1145/1030194.1015508]

Rejaie, R., Handley, M., Yu, H., Estrin, D., 1999. Proxy Caching Mechanisms for Multimedia Playback Streams in the Internet. Proc. 4th International Web Caching Workshop. San Diego, CA, p.100-111.

Tran, D.A., Hua, K.A., Do, T.T., 2003. Zigzag: An Efficient Peer-to-Peer Scheme for Media Streaming. Proc. IEEE INFOCOM. San Francisco, USA, p.1283-1292.

Vakali, A., Pallis, G., 2003. Content delivery networks: current status and trends. *IEEE Internet Computing*, **7**(6):68-74. [doi:10.1109/MIC.2003.1250586]

Yim, J., Kim, G.Y., Son, Y.S., 2001. Cooperative Caching Framework of VOD Using P2P Technology. Proc. 11th International Packet Video Workshop. Kyongju, Korea.

Zegura, E.W., Calvert, K., Bhattacharjee, S., 1996. How to Model an Internetwork. Proc. IEEE INFOCOM'96. San Francisco, CA, p.594-602.

Editor-in-Chief: Wei YANG
ISSN 1673-565X (Print); ISSN 1862-1775 (Online), monthly

Journal of Zhejiang University
SCIENCE A

www.zju.edu.cn/jzus; www.springerlink.com
jzus@zju.edu.cn

***JZUS-A* focuses on "Applied Physics & Engineering"**

➢ Welcome Your Contributions to *JZUS-A*

Journal of Zhejiang University SCIENCE A warmly and sincerely welcomes scientists all over the world to contribute Reviews, Articles and Science Letters focused on **Applied Physics & Engineering**. Especially, Science Letters (3~4 pages) would be published as soon as about 30 days (Note: detailed research articles can still be published in the professional journals in the future after Science Letters is published by *JZUS-A*).