Title	Request-peer selection for load-balancing in P2P live streaming systems
Author(s)	Liu, N; Wen, Z; Yeung, LK; Lei, ZB
Citation	The IEEE Conference on Wireless Communications and Networking (WCNC 2012), Paris, France, 1-4 April 2012. In IEEE Wireless Communications and Networking Conference Proceedings, 2012, p. 3227-3232
Issued Date	2012
URL	http://hdl.handle.net/10722/165312
Rights	IEEE Wireless Communications and Networking Conference. Proceedings. Copyright © IEEE Communications Society.

Request-Peer Selection for Load-Balancing in P2P Live Streaming Systems

Nianwang Liu, Zheng Wen, Kwan L. Yeung
Department of Electrical and Electronic Engineering
The University of Hong Kong
Pokfulam, Hong Kong
{nwliu, wenzheng, kyeung}@eee.hku.hk

Zhibin Lei
Applied Science & Technology Research Institute (ASTRI)
Shatin, Hong Kong
lei@astri.org

Abstract—Unlike peer-to-peer (P2P) file sharing, P2P live streaming systems have to meet real-time playback constraints, which makes it very challenging yet crucial to maximize the peer uplink bandwidth utilization so as to deliver content pieces in time. In general, this is achieved by adopting tailor-made *piece selection* and *request-peer selection* algorithms. The design philosophy is to regulate the network traffic and to balance the load among peers. In this paper, we propose a new request-peer selection algorithm. In particular, a peer in the network estimates the service response time (SRT) between itself and each neighboring peer. An SRT is measured from when a data piece request is sent until the requested piece arrives. When a peer makes a piece request, the neighbor with smaller SRT and fewer data pieces would be favored among potential providers. This is because smaller SRT implies excess serving capacity and fewer data pieces suggests less piece requests received. We evaluate the performance of our request-peer selection algorithm through extensive packet level simulations. Our simulation results show that the traffic load in the network is better balanced in the sense that the difference of the normalized number of data packets uploaded by each peer is getting smaller and the number of repeated piece requests generated by each peer (due to request failure) is significantly reduced. We also found that the load of streaming server is reduced, and the overall quality of service, measured by playback continuity, startup delay etc, is improved as well.

I. INTRODUCTION

P2P live streaming, representing the state of the art technique to stream live media, has been attracting increasing attentions from both academia and industry (e.g. [1][2][3]). By utilizing the P2P infrastructure, the live streaming system can easily scale up to millions of users as the server load is largely distributed among peers in the network.

As compared to P2P file sharing system, the real-time playback constraint of live media poses challenges in designing efficient live streaming systems. Specifically, the video streaming largely relies on the collaborative piece exchange among peers in the network. It is essential for the peers to spread out the *rare* pieces as quickly as possible. The content variety incurred at peer neighborhood could help to maximize the uplink bandwidth utilization. On the other hand, the urgency of each piece should also be considered in order to meet playback constraint. To address this issue, researchers have mainly focused on designing efficient *piece selection algorithms*. For a given set of missing pieces, a piece selection algorithm decides which piece should a peer requests first [4]. The key insight is that piece request should take not only the content rarity but also the timeliness requirement into consideration [4][5].

In contrast to piece selection algorithm, fewer efforts have been spent on a subsequent yet equally important problem of request-peer selection [6]. For a selected data piece and a set of potential piece providers, the task of request-peer selection is to determine which neighbor/provider should be approached for the selected piece. Properly allocating the piece request to different neighbors would help to balance the load at each peer. This would help to ensure the neighbors of a peer always have enough bandwidth to serve incoming piece requests in time. This could also help to decrease the origin streaming server load.

In this paper, we focus on designing request-peer selection algorithms. The simplest approach is to pick up a potential provider randomly. It is interesting to note that contrary to the conventional wisdom, such a randomized algorithm does not balance the load among peers. According to the classic ball-and-bin model [7], randomized algorithm tends to overload some of the nodes with extremely high probability. In CoolStreaming [3], the peer with the highest uploading bandwidth is selected as piece provider. However, it is very difficult to predict each neighbor's uploading bandwidth *dedicated to* serve a particular peer. In the context of P2P video-on-demand (VoD) streaming, closest playback-point first (CPF) is proposed in [6], where peer sends the piece request to the neighbor with the closest playback-point with respect to itself. Since such peer pairs may have larger buffer window overlap, they can thus better utilize each other's uplink capacity for mutual piece sharing. But CPF is not suitable for P2P live streaming because live streaming playback is relatively synchronized and the playback-point difference among peers will be too small. Recently, an analytical model is constructed to study the load balancing performance in P2P streaming in [9]. To facilitate the analysis, it assumes that *all* neighbors have the piece a requesting peer wants so that the requesting peer can adaptively adjust the number of piece requests sent to each neighbor to balance among neighbors. To limit the total number of neighbors allowed, it finds a group of neighbors

978-1-4673-0437-5/12/$31.00 ©2012 IEEE

based on the measured service response time. Our work differs from [9] mainly in that: 1) we consider a more practical request-peer selection scenario where only a small subset of the neighbors has the piece selected by a peer; 2) we evaluate our algorithm through extensive *packet level* simulations and show the strength of our algorithm in terms of both server load deduction and quality of service. Note that the analytical model in [9] is verified by high level simulation with stronger assumptions.

The rest of the paper is organized as follows. In Section II, we present the proposed request-peer selection algorithm. In Section III, we introduce our packet level simulation setup as well as the performance comparison between our proposed algorithm and the randomized algorithm. Lastly, we conclude the paper in Section IV.

II. LOAD BALANCED REQUEST-PEER SELECTION

A peer in live streaming system maintains a data piece pre-fetch window which shifts/slides gradually as the playback point moves [3]. Without loss of generality, we assume a peer is only interested in getting the pieces in the pre-fetch window for smooth playback. (Note that the pre-fetch window of a VoD system is much bigger, and that of a file sharing system is the biggest, and covers every piece of the file.) As compared to P2P VoD streaming systems, peers in the live streaming system tend to have similar playback-point. Their pre-fetch windows tend to be significantly overlapped and this facilitates mutual piece exchange among peers. This also implies that more neighboring peers have the missing piece the requesting peer wants. Given a set of potential piece providers in the neighborhood, how to decide which neighbor should be contacted for retrieving the data piece already selected by the piece selection algorithm? From the requesting peer's point of view, a proper selection of piece provider can help to retrieve the missing piece in time for playback; otherwise the peer would either suffer from video quality degradation or experience a playback suspension. From the system's perspective, a proper piece provider selection can balance the traffic load in the network so that peers can better utilize their uplink capacity to deliver the most urgent pieces. Moreover, due to the more efficient peer upload bandwidth utilization, the uplink bandwidth consumption at the server can be reduced.

Due to simplicity and runtime efficiency, the randomized algorithm for request-peer selection is quite popular. But the analysis of the classic ball-and-bin model in [7] suggests that such a randomized scheme would overload some peers with extremely high probability when the network scales up. To take a closer look at the situation in the context of P2P live streaming system, we use the following example to illustrate this problem.

Let us consider the homogeneous case first, where peers: $Peer_1$, $Peer_2$ *and* $Peer_3$ are the neighbors of $Peer_k$ (please refer to Fig. 1.) and they all have data piece *j*. If the random algorithm is used by $Peer_k$ to select a supplier for piece *j*, the three neighbors would get equal opportunity to be selected:

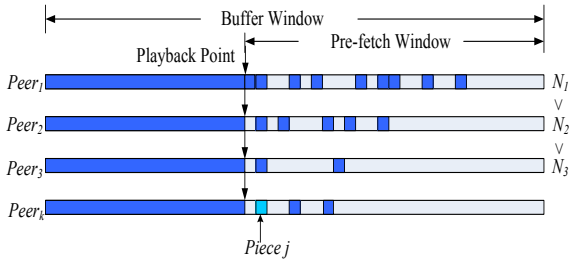

Figure 1. P2P Live Streaming

$$p_1 = p_2 = p_3 = \frac{1}{n},$$

Note: *n* is the number of neighbors having piece *j*; here *n*=3. It is obvious that smaller *n* would lead to greater p_i. That means a rare piece (and thus its owner) has a higher probability being requested/selected.

Let L_i denote the load (i.e. the amount of requests received) on the uplink of peer $Peer_i$; and let N_i denote the total number of pieces in $Peer_i$'s pre-fetch window. We have

$$L_1 = p_{11} + p_{12} + p_{13} + \ldots + p_{1N1}$$
$$L_2 = p_{21} + p_{22} + p_{23} + \ldots + p_{2N2}$$
$$L_3 = p_{31} + p_{32} + p_{33} + \ldots + p_{3N3}$$

Since $N_1 > N_2 > N_3$, L_1 has the largest number of probability terms (i.e. the number of p_{1i} is N_1) while L_3 gets the smallest N_3 . As all the probability terms are positive, adding more terms would likely lead to bigger value, which in turn indicates heavier loading of the corresponding peer. Besides, the more pieces a peer buffers, the higher chance it would buffer some rare pieces and would thus have greater chance to be selected as piece provider.

Considering both the buffer occupancy of the pre-fetch window and the rarity of data pieces, it is very likely that: $L_1 > L_2 > L_3$. In other words, it is very likely that $Peer_1$ would be selected to serve more requests than $Peer_3$ and $Peer_2$, and thus becomes a hotspot. This uneven traffic load is not desirable. The uplink capacity of $Peer_3$ should be better utilized to deliver the two pieces in its pre-fetch window, while $Peer_1$ can focus on delivering pieces $Peer_3$ does not have. In doing so, the overall uplink bandwidth can be better utilized. From this example, we can see that the content availability at each peer plays an important role in request-peer selection.

In practice, peers in the P2P live streaming system have different upload/download bandwidths. In some cases the differences can be quite significant (e.g. ADSL *v.s.* Ethernet

based access technology). The randomized algorithm above neglects the differences in each peer's upload capability, which undermines load balancing. Let us reuse the previous example to illustrate this problem. Assume the upload bandwidth of each peer is U_1 , U_2, and U_3 , respectively. And $U_1 < U_2 < U_3$. With the randomized algorithm, $Peer_1$ gets the most requests to serve while its upload bandwidth is the least among the three potential providers, whereas $Peer_3$ with the largest upload capacity but receives the least requests. In this case, $Peer_1$ is overloaded while $Peer_3$'s upload bandwidth is underutilized. This example shows that the number of piece requests entertained by each peer should be proportional to its uplink capacity.

Based on the insights we obtained from the two examples above, a new request-peer selection algorithm is designed by taking both potential provider's content availability and upload capacity into consideration. The idea is that: potential providers with larger upload capacity and fewer data pieces should be given higher priority to be selected. Without loss of generality, let P_i be the probability of potential provider peer i to be selected as the piece supplier, and P_i is given by:

$$P_i = \frac{U_i}{\sum U_i} + \lambda \frac{\bar{N} - N_i}{\sum N_i} \quad (1)$$

In (1), U_i is the upload bandwidth of peer i dedicated to serve the requesting peer; N_i is the number of pieces in peer i's buffer; $\bar{N}$ is the mean value of all N_i; λ is a weighting factor that determines the relative importance of the two factors: data rarity and upload bandwidth.

Since the upload bandwidth at a peer is shared among all potential network applications running on the end host, it is very difficult (if not impossible) to determine U_i , the upload bandwidth dedicated to serving each neighboring peer. Besides, U_i is time varying with the network load. We take an alternative approach. We adopt the service response time (SRT) instead of U_i . SRT is defined as the time duration from a data piece request is sent to the moment the requested piece is received. It is a good indication of whether a neighbor is overloaded or not. Therefore, the probability of peer P_i is selected by our request-peer selection algorithm becomes:

$$P_i = \frac{(\frac{1}{\tau_i})^2}{\sum(\frac{1}{\tau_i})^2} + \lambda \frac{\bar{N} - N_i}{\sum N_i} \quad (2)$$

τ_i is the expected service response time of peer i. To smooth out the fluctuation, we take the moving average of each response time sample. More specifically, each τ_i is calculated according to the following formula:

$$\tau = \alpha * \tau + (1 - \alpha) * \tau_{sample} \quad (3)$$

where α is a constant between 0 and 1, by adjusting the value of α we can adjust the weighting of the old sampled τ and the latest τ_{sample}. According to our simulation results, we found that 0.7 is a good value for α as more emphasis is paid on the historical value and big spike introduced by the fresh sample is avoided.

Note that peers in the system need to keep records of all its neighbors' service response time τ each time when a piece is successfully downloaded, the neighbor's SRT is updated according to (3). In addition, the content availability information of each neighbor is derived from the periodic buffer map exchange among peers. While making the request-peer selection, the requesting peer selects a piece provider according to the probability in (2).

Note that a specific peer may have different service response times to different peers due to the heterogeneity in the network. The response time based request-peer selection helps to stabilize the system. Specifically, a peer with smaller SRT would be favored by its neighbors thus serves more requests. Along with the gradual increase of the request queue size, the peer's response time will increase (as detected by its neighbors). Such an increase will lead to the decrease of piece requests generated from its neighbors thus protect the peer from being overloaded.

III. PERFORMANCE EVALUATION

A. Performance Metrics

For each peer in the system, we use three metrics to capture its quality of experience: the start-up delay, the total caching time; and the restart count [6]. Start-up delay is the time duration from the instance when a peer joins the system (i.e. approaching the tracker for a list of active peers) to the moment the peer has retrieved sufficient pieces and starts playback. The total caching time captures the total amount of time during which a peer suspends its playback waiting for the missing pieces to arrive. In our simulations, we adopt the simple playback control scheme as discussed in [10]. Briefly, a peer would suspend its playback whenever the next piece to play is not available for certain time duration (set to 3 seconds in our simulation), after which the peer will resume its playback from the next available piece in the local buffer window. (Note that the 3 seconds waiting also contributes to the total caching time measure.) For the worst case that there is no subsequent piece available in the local buffer for playback, the peer quits and rejoins the system by contacting the tracker again. We use the restart count to capture the total number of such events.

In order to investigate whether the traffic load is well balanced among peers, we closely monitor two metrics at each peer: the number of data packets sent, and the number of repeated piece requests sent. The former captures the total number of data packets sent by a peer during the session. And the latter is used to trace the piece request timeout events. Note that a piece request timer (3 sec.) is

Table I. SIMULATION PARAMETERS

Simulation Time	500 sec.
Peer Join Speed	10 peers/sec.
Number of Peers	300
Number of Packets per Piece	94
Avg. Piece Size	58KB(456Kbps)
Peer bandwidth Set I (Down/Up)	10/2 Mbps
Peer bandwidth Set II (Down/Up)	10/0.5 Mbps
Server Bandwidth (Down/Up)	20/20 Mbps
Max. Piece requests to one peer	3
λ	0.9
α	0.7

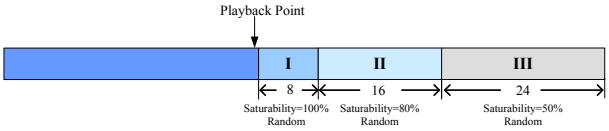

Figure 2. Piece selection strategy used in simulation

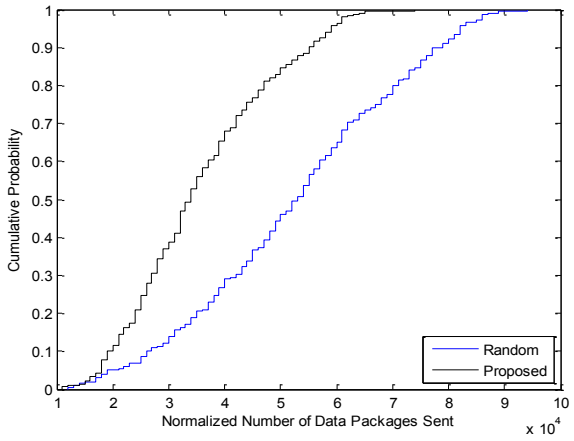

Figure 3. Normalized number of packets sent

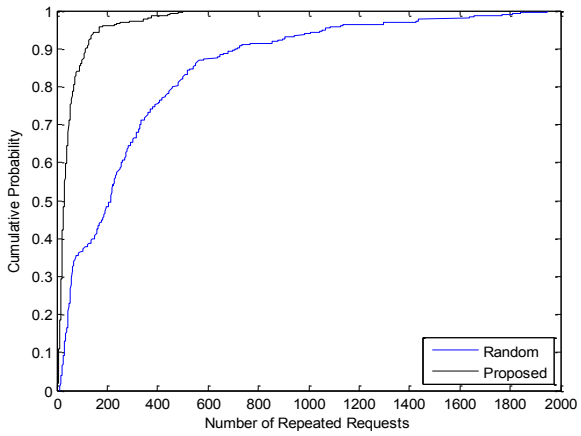

Figure 4. Number of repeated requests

triggered whenever a piece request is sent. On the expiry of the timer and the piece is not completely received, the requesting peer generates another (repeated) request for the same piece but to a different neighbor. When the traffic load among peers is balanced, the number of data packets sent by different peers would be comparable, which can be demonstrated as a steeper slope in the cumulative distribution function (CDF) curve (e.g. Fig.3). Moreover, when peers are properly loaded, large portion of the piece requests can be entertained in time and the number of repeated piece requests should be small.

From the system's point of view, balance the traffic load helps to boost the uplink bandwidth utilization of peers in the network from which the server will benefit in saving its uplink bandwidth. To this end, we record the server uplink bandwidth consumption every 10 seconds.

To show the efficiency of our proposed algorithm, we compare it with the randomized request-peer selection algorithm.

B. Simulation Setup and Results

We evaluate the proposed request-peer selection algorithm through simulations based on a simulator from ASTRI [11]. The simulator is built on NS2 and simulates packet-level detail of a P2P live streaming network. The simulator is well tested and is used both for academic research and industrial deployments [6][12].

To focus only on the request-peer selection strategy, unless otherwise stated, the following generalized system framework is used throughout the simulation: (1) One streaming source that continuously encodes the live video content with the average streaming bit rate of 456Kbps stays in the network throughout the simulation. The encoded stream is segmented into multiple pieces each contains the data for one second playback; (2) Each peer in the network keeps a list of neighbors with which it periodically exchanges their buffer maps; (3) A simple *section based* piece selection strategy is used where the prefetching window, with a size of 48 pieces/seconds, is divided into three sections each with different size and saturability, as shown in Fig. 2. A peer randomly selects a piece from a lower numbered section to request until the saturability of that section is satisfied before retrieving a piece from the next section. A peer never requests a piece beyond the request window; (4) At any time, each peer can have up to N pending/on-going piece requests (set to 10 in our simulation) and among them, no more than D requests to the same peer (set to 3 in our simulation); (5) At any time, a peer can serve piece requests on their order of arrivals. (6) Each peer starts video playback after receiving the first sixteen pieces and resumes the playback according to playback policy listed in sub-section A; (7) Peers join the streaming session with an uniformly distributed arrival rate of 10 peers per second until all 300 peers are in the system. They would not leave the network throughout the simulation; other parameters used in the simulation are summarized in Table I.

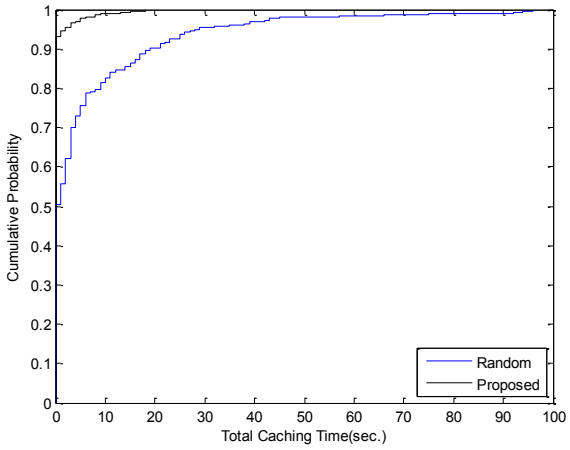

Figure 5. Total caching time

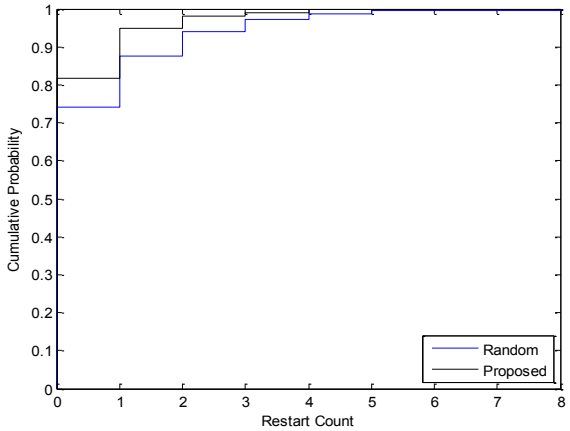

Figure 6. Restart count

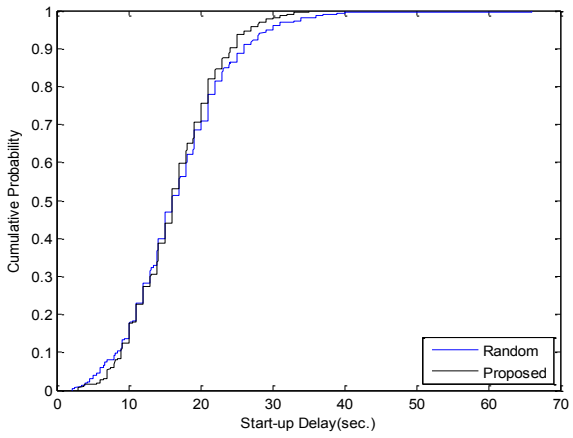

Figure 7. Start-up delay

We consider the heterogeneous scenario, where peers' download bandwidth are the same, while half of them are of higher uploading capacity and the other half are of lower upload bandwidth. Peers in the network form a star topology and the one way propagation delay between a peer and the central router is set to 5ms. The CDF curve of different metrics collected at each peer is used to demonstrate the performance difference and the corresponding average values are summarized in Table II.

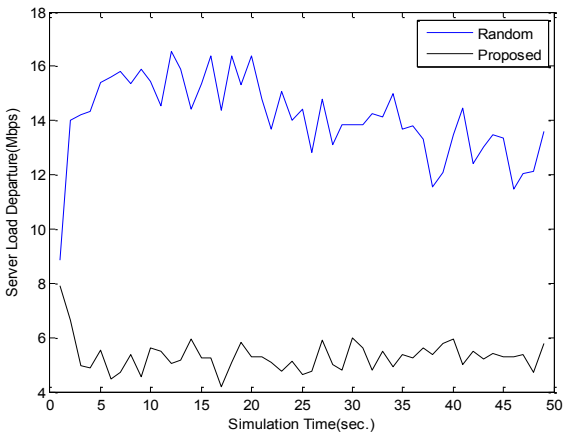

Figure 8. Server load

Table II. AVERAGE VALUE COMPARISON

	Random	Proposed
Start-Up Delay(Sec.)	14.97	15.27
Restart Count	0.49	0.26
No. of Repeated Requests	304.10	53.17
Total Caching Time(Sec.)	6.23	0.37

Since peers in our simulations have different upload bandwidth, we normalized the number of data packets sent by each peer by its upload bandwidth and depict its CDF curve in Fig. 3. We can find that our request-peer selection algorithm shows a much steeper slope than the randomized algorithm. That shows the difference of number of data packets sent (per unit upload bandwidth) is smaller among peers using our algorithm. In other words, this indicates that the piece requests are evenly distributed among peers.

According to the number of repeated piece requests performance shown in Fig. 4, it is can be easily seen that by adopting our proposed request-peer selection algorithm, peers in the network generate much fewer repeated requests which is only about one sixth of the randomized algorithm.

For the perceived playback quality, we examine the total caching time, restart count and start-up delay, which are shown in Fig. 5, Fig. 6 and Fig. 7, respectively. We can see that with our algorithm, each peer spends less time in caching, and experiences less restarts. It is understandable that there is no obvious improvement in start-up delay. This is because a request-peer selection does not directly address the startup performance.

Fig. 8 shows our proposed algorithm reduces the server load significantly as compared to the random algorithm.

This can be attributed to the increase of the peer upload bandwidth utilization when traffic is well balanced.

IV. CONCLUSION

In this paper, we proposed an efficient request-peer selection algorithm for P2P live streaming system. In our algorithm, the probability of peers to be selected as the piece provider is calculated according to their content availabilities and their loading measured by service response time. Through extensive simulations, we showed that our algorithm distributes the traffic load more evenly among peers. As a result, the peers' uplink bandwidth is better utilized and the streaming server load is reduced, meanwhile, the quality of experience is also improved.

REFERENCES

[1] PPLive: http://www.pptv.com/.

[2] PPstream: http://www.ppstream.com/.

[3] X. Zhang, J. Liu, B. Li, and T. S. P. Yum, "Coolstreaming/donet: A data-driven overlay network for efficient live media streaming," INFOCOM, 2005.

[4] Y. Zhou, D.M. Chiu, J.C.S. Lui, "A simple model for analyzing P2P streming protocols," Proc. ICNP, 2007.

[5] K. W. Hwang, V. Misra, and D. Rubenstein, "Stored media streaming in bittorrent-like p2p networks," *Tech Report, Columbia University, NY*,no. cucs-024-08, 2008.

[6] Z. Wen, N.W. Liu, K. L. Yeung and Z. B. Lei, "Closest playback-point first: a new peer selection algorithm for p2p vod systems," to appear in GLOBECOM 2011

[7] M. Mitzenmacher, "The power of two choices in randomized load balancing," IEEE Transactions on Parallel and Distributed Systems, vol.12, no.10, pp. 1094-1104, Oct., 2001.

[8] K.Graffi, S.Kaune, K.Pussep, A.Kovacevic and R. Steinmetz, "Load balancing for multimedia streaming in heterogeneous peer-to-peer systems," Proc. of NOSSDAV, 2008.

[9] Y. Wang, T.Z.J.Fu and D.M. Chiu, "Analysis of load balancing algorithms in p2p streaming," Proc. SIAM, 2008

[10] TZJ Fu, DM Chiu and ZB Lei, "Designing QoE Experiments to Evaluate Peer-to-Peer Streaming applications," VCIP 2010, July 2010

[11] ARSTRI: http:// www.astri.org/.

[12] J. Huang, G.Cheng, J. Liu, and D.M.Chiu et.all, "A simulation tool for the design and provisioning of p2p assisted content distribution plateforms," under review, A copy is available at: http://personal.ie.cuhk.edu.hk/~dmchiu/references/P2P_Simulation.pdf.