Computer Networks 56 (2012) 1103–1117

Contents lists available at SciVerse ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Measurement, modeling and enhancement of BitTorrent-based VoD system

Zhen Ma [a], Ke Xu [a,*], Jiangchuan Liu [b], Haiyang Wang [b]

[a] Department of Computer Science & Technology, Tsinghua University, Beijing 100084, China
[b] School of Computing Science, Simon Fraser University, British Columbia, Canada V5A 1S6

ARTICLE INFO

Article history:
Received 21 March 2011
Received in revised form 6 October 2011
Accepted 24 October 2011
Available online 20 December 2011

Keywords:
BitTorrent-based VoD streaming
Hybrid BitTorrent system
Sliding buffer window
Closest-ahead peers

ABSTRACT

BitTorrent is one of the most popular Peer-to-Peer (P2P) applications for file sharing over the Internet. Video files take up a large proportion of space among the BitTorrent shared files. Recently, BitTorrent has attracted researchers' interests, as an alternative method of providing video on demand (VoD) service. In this paper, we concentrate on enabling BitTorrent to support VoD service in existing swarms while maintaining the download efficiency of file-sharing users. We first examine the content properties of the BitTorrent system to explore the demands and challenges of VoD service in BitTorrent swarms. The efficiency of BitTorrent for various piece selection policies is then compared through measurement on PlanetLab. We also use an optimization mathematical model to analyze the hybrid BitTorrent system in which downloading peers and streaming peers coexist. Both measurement results and model analysis indicate the problem of system efficiency decline in the BitTorrent-based VoD systems, in comparison with the original BitTorrent file-sharing system. Our proposed approach, unlike existing strategies that are limited to changing the piece selection policy to allow BitTorrent to support streaming services, modifies both piece and peer selection policies to provide a "streaming while downloading" service in the BitTorrent system with downloading peers. For the peer selection policy, a CAP (Closest-Ahead Peers) method is applied to make better use of the peers' upload bandwidths. For the piece selection policy, a sliding window-based hybrid method that combines the rarest-first policy with the sequential policy is proposed. To demonstrate the performance of our proposed approach, an evaluation is made using various metrics on PlanetLab. The results show that our proposed method has higher throughput and better streaming continuity than the sequential policy and BiToS.

1. Introduction

The Peer-to-Peer (P2P) network has become a successful architecture for file sharing over the Internet. BitTorrent is one of the most popular P2P applications for file distribution. Video files take up a large proportion of the shared files. Currently, BitTorrent users increasingly prefer to enjoy videos while they are still downloading. Researchers have also reported a strong interest in using BitTorrent as an alternative to providing VoD service. According to the requirements of streaming systems, which consume a video's pieces sequentially from beginning to end, most research on BitTorrent-based streaming is based on BitTorrent protocol modifications, especially the piece selection mechanism modification.

In this paper, we focus on how to provide VoD service in existing BitTorrent swarms where streaming peers (the peers that watch videos during the downloading process) and downloading peers (the peers that simply download video files) coexist. We first examine the shared contents in the BitTorrent network with a large-scale measurement. The results show that video files are popular among BitTorrent swarms and that video contents in BitTorrent swarms are much larger than those in YouTube and PPVA, which

* Corresponding author.
E-mail addresses: mazhen@csnet1.cs.tsinghua.edu.cn (Z. Ma), xuke@tsinghua.edu.cn (K. Xu), jcliu@cs.sfu.ca (J. Liu), hwa17@cs.sfu.ca (H. Wang).

1389-1286/$ - see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2011.10.030

makes the BitTorrent-based streaming service promising but challenging. We also use a modified Ctorrent[1] (a command line-based BitTorrent client in FreeBSD) client with different piece selection policies to transfer files on PlanetLab and compare the performances of different policies. According to the trace analysis, an optimization model is proposed to analyze the BitTorrent-based VoD system with both *downloading* and *streaming* users. Both real trace and mathematical models indicate that the system throughput will decrease when the *sequential* piece selection mechanism is used instead of the *rarest-first* policy. Therefore, making BitTorrent more efficient while fulfilling the demands of streaming remains a challenging problem.

In light of the measurement results and model analysis, we find that, apart from piece selection policy, peer selection also greatly affects the efficiency[2] of a BitTorrent-based VoD system. Hence, our method modifies both piece and peer selection policies. For the peer selection policy, the CAP (Closest-Ahead Peers) method is applied to make better use of peers' upload bandwidth. Peers will request pieces from the closest-ahead peers whose *sequential* download process is just ahead of the requesting peer's.

For the piece selection policy, a sliding buffer window-based hybrid method that combines the *rarest-first* policy with the *sequential* policy is proposed in this paper. In the new policy, peers request pieces sequentially in the buffer window first in order to maintain fluent streaming; they then adopt the *rarest-first* policy after buffering enough pieces for play to obtain the rare pieces for sharing, adapt to *tit-for-tat*, and maintain piece diversity.

Based on the model analysis, we evaluate the system throughput and miss penalty in this hybrid BitTorrent-based VoD system with *downloading* and *streaming* peers. System throughput can be regarded as the inverse of the download completion time, while the miss penalty is the peers' total playback lag time, defined in Section 4. In addition, the startup delay, piece distribution and upload volume distribution of peers in the BitTorrent-based VoD system are analyzed in our evaluation. The performance of different policies is evaluated with modified Ctorrent on PlanetLab using various metrics. We also investigate the performance of our approach in different situations, such as situations with deferent window sizes and different swarm compositions.

The main contributions of this paper are as follows:

(1) A multi-objective optimization model is proposed to analyze the BitTorrent-based VoD system, in which *downloading* and *streaming peers* coexist for the first time. Model analysis results indicate that the system efficiency will decline in a BitTorrent-based VoD system as the number of *streaming peers* grows.
(2) As it has been shown that the modification of piece selection is not enough to guarantee the performance of a hybrid BitTorrent system with *downloading* and *streaming* peers, our proposed method modifies both piece and peer selection policies to provide better streaming service. Both peer and piece selection modifications are compatible with BitTorrent's built-in *tit-for-tat* mechanism.
(3) Extensive evaluation has been performed on PlanetLab to compare different policies in hybrid BitTorrent systems with both *downloading* and *streaming peers*. The evaluation results show that our method is better than the *sequential* policy and other existing work, such as BiToS [10].

The rest of this paper is organized as follows: In Section 2, related work on the subject is presented. Section 3 presents our measurement results. An optimization model is used to analyze a BitTorrent-based VoD system in Section 4. Our method is proposed in Section 5, and its evaluation is presented in Section 6. The paper is concluded in Section 7.

2. Related work

P2P file sharing and the P2P streaming service are two major applications of P2P technology, and their design and measurement have attracted significant attention. BitTorrent is the most popular of the P2P file sharing systems, and much research has been conducted to analyze BitTorrent-like file sharing networks, such as [5–8]. Meanwhile, researchers have also designed several peer-assisted streaming systems to support the VoD streaming service, such as PPLive [1], GridCast [2], Joost [4] and UUSee [3].

Much recent work has investigated supporting peer-assisted VoD services with the BitTorrent protocol [11]. The simplest method is to change BitTorrent's piece selection policy from the *rarest-first* policy to the *sequential* policy to meet the playback requirements. Parvez et al. [9] presented detailed fluid flow models for different piece selection policies and proposed three optional approaches. However, these approaches have not yet been evaluated. Fan et al. [23] used analytical models to analyze different schemes in a P2P VoD system and described a fundamental tradeoff that exists between system throughput, sequential downloading and system robustness, proving that no system can achieve all of them simultaneously. Our model is different from theirs in that we concentrate on providing P2P VoD service in BitTorrent systems where *downloading* and *streaming peers* coexist, not in a pure P2P VoD system. Unlike Fan et al. [23], who disabled *tit-for-tat*, we keep this core feature of BitTorrent and make our method compatible with it.

Apart from model analysis, some methods have been proposed to enable BitTorrent to support a streaming service efficiently. It is a common belief that the *sequential* download of pieces will reduce the swarm's piece diversity, resulting in poor system performance. Therefore, existing research has mainly concentrated on ways of meeting the *sequential* requirements of streaming while maintaining piece diversity. Carlsson and Eager [15] adopted a Zipf-based probabilistic piece selection policy. In addition to the probabilistic-based approaches, there is window-based work. Vlavianos et al. [10] proposed BiToS, which provides a higher download possibility p to pieces in a high priority set. Purvi and Jehan [12] prevented the peers from requesting pieces outside of a sliding window that contains pieces for the near future. Petri [19] proposed

[1] http://ctorrent.sourceforge.net.
[2] In this paper, we use the system throughput and user bandwidth utilization to indicate the BitTorrent system efficiency.

a stretching window algorithm to suit the *tit-for-tat* mechanism. Youmna Borghol et al. [18] presented an adaptive window-based piece selection policy, with each peer dynamically computing its window size. In addition to the probabilistic and window-based approaches, LiveBT [20] used a Most-wanted-Block-Download-First (MBDF) scheme and a fast rescheduling scheme to enable BitTorrent to support VoD streaming service. Our previous work [21] tried to solve this problem at the swarm level, rather than at the peer level. Unlike most of the abovementioned work that simply modifies the piece selection policy to allow BitTorrent to support streaming service, we modified not only the piece selection policy but also the peer selection policy to provide better streaming service in hybrid BitTorrent systems with both *downloading* and *streaming* peers. As for piece selection policy, which is different from existing window-based piece selection policies, we use different policies in and out of the window. The *sequential* policy is used in the buffer window, and the *rarest-first* policy is used outside it.

Lucia D'Acunto et al. [25] extended a BitTorrent-like peer selection mechanism with techniques that allow peers to relax their reciprocity-based peer selection and upload pieces to more random nodes. Zhenhua Li et al. [24] designed an "Urgent Line" mechanism for every node to predict which pieces are likely to be missed and prefetch them. However, these two mechanisms were not designed for BitTorrent-based streaming service in hybrid swarms with both *downloading* and *streaming* peers. Yang et al. [16] proposed using LLP-P as a peer selection policy to better utilize peers' upload capacity and a DAS scheduling approach for the service scheduling problem in a BitTorrent-like VoD system. DAS changes peers' policy of deciding which request to serve, while our work changes the piece selection policy to decide which piece to request next. For the peer selection approach, contrary to LLP-P [16] and iPass [17], our approach is simple and easy to implement. We also make some changes to consider hybrid swarms with both *downloading* and *streaming* peers. Our work concentrates on providing VoD service in BitTorrent systems with both *downloading* and *streaming peers* coexisting, while Yang et al. [16]'s proposal failed to solve this problem.

Although great efforts have been made to develop a BitTorrent-based VoD system, until now, little progress has been made in the area of hybrid BitTorrent swarms that accommodate *downloading peers* and *streaming peers* simultaneously. Further, most of these works have not been evaluated with a real, large-scale BitTorrent system measurement. Considering the complexity of P2P systems, it is insufficient to evaluate P2P strategies with simulations alone. In this paper, a comparison of policy performances is conducted using measurement results from PlanetLab. BiToS [10] is chosen as a reference policy due to its easiness and clearness of implementation. The results of the evaluation presented in Section 6 show that our method performs better than BiToS.

3. Measurement

This section aims to investigate the demands and potential problems of the BitTorrent-based VoD system. We analyze the content properties of a BitTorrent system in Section 3.1, exploring the potential and challenges of the BitTorrent-based streaming service. In Section 3.2, we run real P2P file transference on PlanetLab using a modified Ctorrent, and compare the performances of the Ctorrent with different piece selection policies.

3.1. BitTorrent video file measurement

In this subsection, we focus on exploring the content properties of the BitTorrent system, as well as the potential and challenges of the BitTorrent-based streaming service. Although measurement work on BitTorrent [8] already exists, we believe that new measurement results will be more precise because the network environment changes rapidly. Our measurement results indicate that video files are popular in BitTorrent swarms, and video files now shared using BitTorrent are much larger than those shared in other VoD applications, such as YouTube [13] and PPVA [14]. This offers more opportunities and challenges for a BitTorrent-based VoD system.

This study analyzes ByrBT,[3] the most popular BitTorrent site in CERNET2[4] (The Second Generation China Education and Research Network). ByrBT has more than 48,000 registered users, contains 17,831 torrent files and uses a customized uTorrent[5] as the BitTorrent client. The torrent files and system logs were traced from its website on July 2nd, 2010.

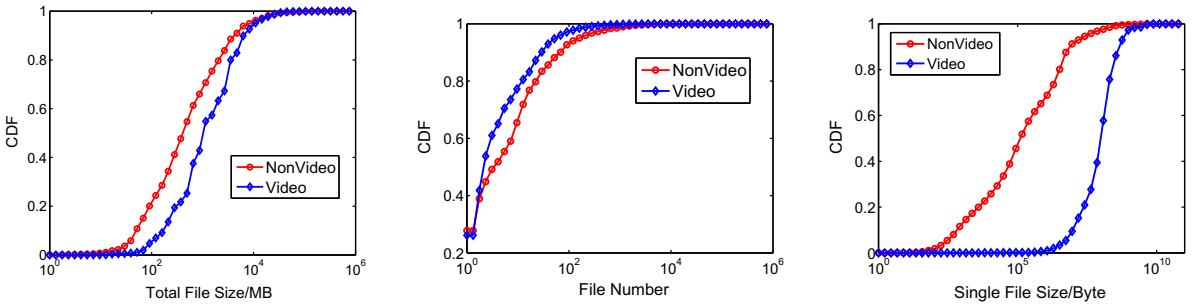
Among the 17,831 torrent files, 12,600 video torrents comprise 70% of the total. As we can see, video files are the most popular files shared in the BitTorrent system. Fig. 1(a) shows the cumulative distribution of the torrents' sharing file sizes. Video files are larger than non-video files on average (1405 MB median file size for video files and 549 MB for non-video files). The size of a BitTorrent shared video file is also larger than it was in 2008 [21] (696 MB median file size). The file distribution in BitTorrent swarms is shown in Fig. 1(b), providing evidence that multi-file content is very popular in both types of BitTorrent swarms and that fewer than 30% of torrents consist of a single file. Fig. 1(c) depicts the cumulative distribution of file sizes. The size of single non-video content files is relatively small, with a mean size of 32 MB; only 3.5% of non-video content files are larger than 100 MB. The median object size of video files is approximately 162 MB, and 65% of video content files are larger than 100 MB. Moreover, 6.6% of the video files have sizes larger than 1 GB, and the size of some Blu-ray videos reaches nearly 23 GB. In contrast, YouTube's median content size is just 8 MB [13]; that of PPVA is 15 MB [14]. YouTube[6] has a small median video size because most videos are user-generated short videos and because videos uploaded by standard account holders are limited to 15 min in duration. Furthermore, YouTube videos

[3] http://bt.byr.cn.

[4] CERNET2 (http://www.cernet2.edu.cn/) is the largest next-generation Internet backbone. It is the core network of the China next-generation Internet demonstration project CNGI and is the only nationwide academic network. To date, it is the world's largest native IPv6 backbone.

[5] http://www.utorrent.com.

[6] http://www.youtube.com.



(a) Total File Size Distribution (b) File Number Distribution (c) Single File Size Distribution

Fig. 1. BitTorrent file property measurement results.

are usually of low quality, with a resolution of 320 × 240 pixels and mono MP3 audio before 2008.[7] PPVA[8] uses P2P technology to accelerate video file downloading from video sites like Youku.[9] Some of the video files shared with PPVA are user-generated like those on YouTube. Other types of videos, such as movies, are usually split into several small video fragments; therefore, videos shared with PPVA are generally small in size. Because video files shared with BitTorrent are usually high in quality or high definition, videos shared with BitTorrent are usually much larger than those shared in YouTube and PPVA.

Therefore, there is considerable evidence that the majority of shared files are video files. Video files shared using BitTorrent are much larger than those shared in other VoD applications like YouTube [13] and PPVA [14]. We believe that as users' bandwidth grows, high-quality video files will increasingly be shared using the BitTorrent system and that BitTorrent-based streaming services will become more popular as well. A BitTorrent-based VoD system offers several advantages in providing high-quality VoD service compared to general P2P VoD applications like PPLive [1]. First, a BitTorrent-based VoD system is low in cost because it provides streaming service in existing BitTorrent swarms and requires no server support. In addition, high-or low-bandwidth users can download the same video files, and users who watch streaming videos and those who download them can share in the same swarm, making the BitTorrent-based VoD system more scalable than a general P2P VoD system. It should also be noted that without server support, the user's experience of a BitTorrent-based VoD system can be worse than that of a general P2P VoD system, especially for unpopular videos. Hence, providing good streaming service in a hybrid BitTorrent system is a challenging problem.

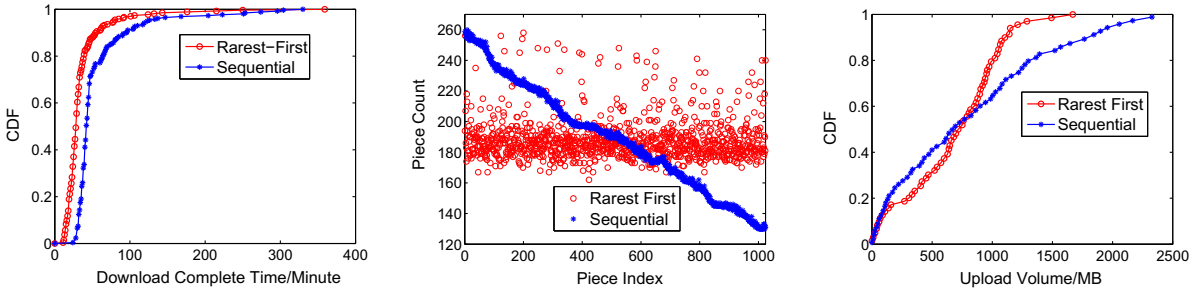
3.2. PlanetLab BitTorrent swarm measurement

Because it is frequently used by users sharing high-quality videos, there is a great demand for BitTorrent to provide streaming service. To meet the playback requirement, pieces should be requested using the *sequential* policy rather than the *rarest-first* policy. Although many analyses show that the *rarest-first* policy performs better than the *sequential* policy in BitTorrent file sharing, we will present the results of an efficiency comparison of these two piece selection policies based on the PlanetLab measurements in this subsection using the results of these measurements.

Two kinds of Ctorrent are used to run a real file transfer on PlanetLab. Unmodified Ctorrent uses the *rarest-first* policy as its piece selection policy, while modified Ctorrent uses the *sequential* policy. Two transfer rounds are measured using these two policies. The shared file is 1 GB with a piece size of 1 MB. Two hundred and fifty-four servers were used on PlanetLab between June 4th, 2010, and June 10th, 2010. One node is chosen as a tracker, and another is chosen as a seed. The rest of the nodes are the peers who begin downloading the shared files simultaneously. The transferred file is self-generated and poses no copyright concerns.

Fig. 2(a) shows the distribution of peers' download completion times using both the *rarest-first* and *sequential* piece selection policies. Most peer download completion times are 20–100 min, but the peers using the *rarest-first* policy achieve faster downloads than the peers using the *sequential* policy. Piece finishing events and timestamps are traced in logs; we can obtain a snapshot of each peer's piece procession. Fig. 2(b) describes the swarm's piece distribution when half of the peers are finished downloading. Pieces are distributed uniformly when the *rarest-first* policy is used; while using the *sequential* policy, the piece count descends linearly as the pieces close to the end of the file. Therefore, the *sequential* policy reduces piece diversity. The piece selection policy also has a close connection with upload bandwidth usage. The *rarest-first* policy makes peers upload more, as a peer will be requested more frequently and upload more pieces when it has the rarest pieces. However, using the *sequential* download policy, the peers downloading more slowly will have fewer opportunities to upload pieces because they have no pieces requested by other peers. Fig. 2(c) shows the distribution of the peers' upload volume. The upload volume of the *rarest-first* peers is more dispersed with smaller variance than the *sequential* peers; hence, the *rarest-first* policy inspires more peers to upload and is more suitable for the *tit-for-tat* incentive mechanism.

[7] http://en.wikipedia.org/wiki/YouTube.
[8] http://www.ppacc.com/.
[9] http://www.youku.com.



(a) Download Completion Time Distribution (b) Piece Count Distribution (c) Upload Volume Distribution

Fig. 2. PlanetLab BitTorrent swarm measurement results: comparison of different policies.

The measurement results show that the *sequential* policy reduces the swarm's efficiency compared with the original *rarest-first* piece selection policy; it also increases download completion time and reduces upload bandwidth usage and piece diversity. Hence, simply changing the piece selection policy from *rarest-first* to *sequential* is insufficient to support streaming service. In the next section, we will illustrate the problem of efficiency decline with mathematical models.

4. Model analysis

According to the measurement results presented in Section 3, we can see that there is a great demand for BitTorrent-based VoD service, but a simple modification from a *rarest-first* to a *sequential* download policy will reduce the P2P system's efficiency. In this section, we use mathematical models to analyze the problem of efficiency decline.

4.1. Optimization model analysis

In this subsection, we use an optimization model to analyze a BitTorrent system with both *downloading* and *streaming* peers. We also use a Matlab-based simulation to compare the BitTorrent system's throughput when different weights and different peer compositions between *downloading* and *streaming* are used. The results show that the more weight that is given to the streaming service and the more peers that use the streaming service, the less throughput the BitTorrent system will have.

This study focuses on methods of providing VoD streaming service in existing swarms while maintaining the file-sharing users' download efficiency. In BitTorrent swarms, peers are classified into two categories: *downloading* peers ($M(t)$) that simply download video files and *streaming* ones ($N(t)$) that play the videos while downloading them. These two kinds of peers have different objectives: downloading peers pay more attention to improving throughput and reducing download completion time, while streaming peers attach more importance to continuity of play. We use a multi-objective optimization model to describe a BitTorrent system that shares a given video file while *downloading* and *streaming peers* coexist. Table 1 provides a summary of the main notations of the proposed model. For the purpose of simplification, VCR operations such as drag, pause, and stop are not supported in the model.

$$\text{maximize} \quad \sum_{t=0}^{T-1} \sum_{i \in M(t)} \sum_{j} \sum_{k} X_{ijk}(t)$$

$$\text{minimize} \quad \sum_{t=0}^{T-1} \sum_{i \in N(t)} \sum_{k} P_{ik}(t)$$

$$\text{subject to} \quad \sum_{k} \sum_{j} X_{ijk}(t) \leqslant D_i \tag{4.1.a}$$

$$\sum_{k} \sum_{i \in L(t)} X_{ijk}(t) \leqslant U_j \tag{4.1.b}$$

$$\sum_{t=0}^{T-1} \sum_{j} X_{ijk}(t) \leqslant 1 - G_{ik}(0) \tag{4.1.c}$$

$$X_{ijk}(t) \leqslant G_{jk}(t) \tag{4.1.d}$$

$$X_{ijk}(t) \in \{0,1\} \tag{4.1.e}$$

$$G_{ik}(t) \in \{0,1\} \tag{4.1.f}$$

$$\text{where} \quad P_{ik}(t) = \begin{cases} 0 & t \leqslant T_i + k\mu \\ 1 - G_{ik}(t) & t > T_i + k\mu \end{cases} \tag{4.1.g}$$

$$G_{ik}(t) = \sum_{t'=0}^{t-1} \sum_{j} X_{ijk}(t') + G_{ik}(0) \tag{4.1.h}$$

(4.1)

In the model, we use t to represent time interval $[t\tau, t\tau + \tau)$, $t = 0, 1, 2, \ldots$ We assume that peers start requesting new pieces at $t\tau$ and finish downloading those pieces in one time unit, τ. $X_{ijk}(t) \in \{0,1\}$ denotes whether peer i downloads piece k from peer or seed j during time interval t, and $G_{ik}(t) \in \{0,1\}$ represents whether peer i has downloaded piece k by time $t\tau$. Given one time period $[0, T\tau)$, our goal is to find the value of $X_{ijk}(t) \in \{0,1\}$ to maximize system throughput and user salification during this period. Hence, this optimization model is a zero-one programming model. Two objectives are taken into consideration. The first objective is to maximize the system throughput and minimize the download completion time of downloading users, and the other is to minimize miss penalties and ensure that *streaming peers* obtain smooth play. Model (4.1) illustrates the BitTorrent-based system. $G_{ik}(t)$ is the accumulated value of $X_{ijk}(t)$ during time interval $[0, t\tau)$. Its initial value at time 0 is shown in Eq. (4.1.h) for all $i \in L(t) \cup S(t)$ *and* (k), and $G_{ik}(t) = 1$ for all k, t if $i \in S(t)$. Miss penalty $P_{ik}(t)$

Table 1
Summary of main notations.

Symbol	Illustration
$N(t)$	*streaming peers* set at time t
$M(t)$	*downloading peers* set at time t
$L(t)$	Peers set, namely, $N(t) \cup M(t)$
$S(t)$	Seeds set at time t
U_i	Upload capacity of peer i
D_i	Download capacity of peer i
T_i	Start playing time of peer i
n	Piece number of the sharing file
γ	Piece size, here $\gamma = 1$ for simplicity
τ	Time unit
μ	Time slice that one piece plays
$X_{ijk}(t)$	Whether peer i downloads piece k from j in time interval t
$G_{ik}(t)$	Whether peer i finishes downloading piece k at time $t\tau$
$P_{ik}(t)$	Penalty of streaming peer i missing piece k in time interval t

is defined as a piecewise function in Eq. (4.1.g) for all $i \in N(t)$ *and* (k). The miss penalty of streaming peer i requesting piece k at time interval t is 0 before the piece's playback deadline and $1 - G_{ik}(t)$ after the deadline. The playback deadline of peer i for piece k is $T_i + k\mu$, which is the former k pieces' playtime since peer i started playing. If peer i finishes downloading piece k before its playback deadline, its miss penalty is only 0. In the long run, the cumulative miss penalty of peer i requesting piece k is the piece's lag time interval count.

Constraints (4.1.a) for all $i \in L(t)$ and (4.1.b) for all $j \in L(t) \cup S(t)$ guarantee that download and upload speeds will not exceed the peers' capacity. The uniqueness constraint (4.1.c) for all $i \in L(t) \cup S(t)$ *and* (k) ensures that each peer can download only one copy of a given piece. The source availability constraint (4.1.d) for all $i \in L(t)$, $j \in L(t) \cup S(t)$ *and* (k) [22] means that peer j can only upload piece k after it has finished downloading piece k.

The solution of model (4.1) is $X_{ijk}(t)$. Given peer i, it should find the appropriate j and k at time t for $X_{ijk}(t)$. The selection of j corresponds to peer selection policy, while the selection of k corresponds to piece selection policy. To achieve an optimum solution, both peer and piece selection policies should be optimized.

Many methods have been proposed to find a solution to a multi-objective optimization problem by constructing a single aggregate objective function, such as Weighted Linear Sum, Normal Boundary Intersection, and Normal Constraint.[10] In this model, both system throughput and miss penalty are integral multiples of piece size, so the *downloading* and *streaming* objective are of the same dimension. We use an intuitive weighted linear sum approach to construct a single aggregate objective function based on two objectives. The weighted linear sum of the two objectives is obtained by specifying scalar weights α and $1 - \alpha$ for the two objectives, respectively, as shown in (4.2). Scalar weights α and $1 - \alpha$ imply the relative importance of the *downloading peers*' throughput and the *streaming peers*' miss penalty in the BitTorrent system.

[10] http://en.wikipedia.org/wiki/Multiobjective_optimization.

$$\text{maximize} \quad \alpha \sum_{t=0}^{T-1} \sum_{i \in M(t), j, k} X_{ijk}(t) - (1-\alpha) \sum_{t=0}^{T-1} \sum_{i \in N(t), j, k} P_{ik}(t) \tag{4.2}$$

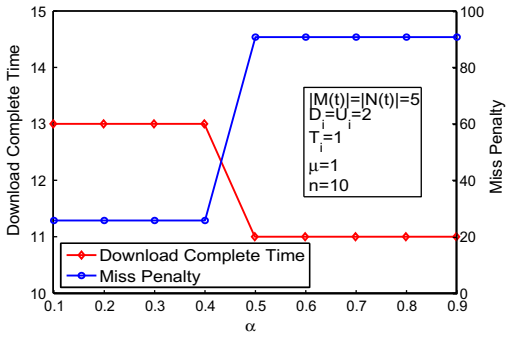
The simulation is built using the Matlab optimization toolbox to solve the zero-one programming problem. The variable number in (4.1) is $\Theta(T * |L(t)| * (|L(t)| + |S(t)|) * n)$, and the constraints number is of the same level. It is difficult to solve this large-scale problem to achieve a global optimum. Therefore, in the simulation, we obtain a greedy local optimum result in each time interval and then aggregate these local optimums to arrive at a local optimum for all time intervals. Although the local optimum is worse than the global optimum, it is closer to a practical situation in which a peer selects peers and pieces based on the existing information at any time, not as planned in the global optimum. The objective of the system in time interval t is stated in (4.3).

$$\text{maximize} \quad \alpha \sum_{i \in M(t), j, k} X_{ijk}(t) - (1-\alpha) \sum_{i \in N(t), j, k} P_{ik}(t) \tag{4.3}$$

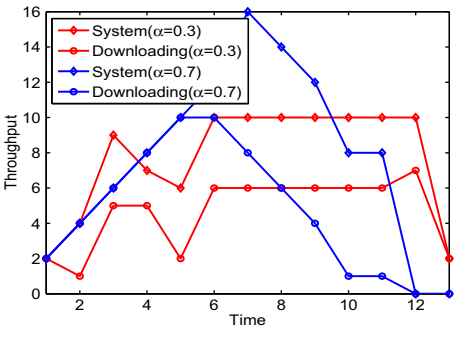
In the simulation, one seed exists in the swarm at the beginning, and all peers begin the downloading process simultaneously and continue until the file is completely downloaded. We use the download completion time to reflect the system throughput. The sizes of the swarm and file are restricted in the simulation due to the complexity of the problem. First, the effect of α is analyzed in a hybrid swarm with both *downloading* and *streaming peers*. Fig. 3(a) shows that the download completion time and miss penalty change as α changes when $|M(t)| = |N(t)| = 5$ and $n = 10$. As α grows, the download completion time decreases, while the miss penalty $\sum_{t=0}^{T-1} \sum_{i \in N(t), j, k} P_{ik}(t)$ grows. In Fig. 3(b), we present the system throughput and the *downloading peers*' throughput of every time interval t when the same swarm composition and different α are used. We can see that accumulated system throughput of the swarm with $\alpha = 0.7$ in $[0, T)$ $(T = 1, 2, \ldots, 13)$ is equal to or greater than that with $\alpha = 0.3$, as is the throughput of exclusively downloading peers. We also analyze the swarm composition's impact on system performance. Fig. 3(c) shows that the download completion time and miss penalty change as $|M(t)|$ (the *downloading peers* count) changes when $\alpha = 0.5$ and $|L(t)| = |M(t)| + |N(t)| = 10$. As $|M(t)|$ increases, the download completion time decreases, and the average miss penalty of the *streaming peers* $\frac{\sum_{i \in N(t), j, k} P_{ik}(t)}{|N(t)|}$ increases and then decreases. Fig. 3(d) presents the system throughput of every round and shows that the system throughput is larger when there are more *downloading peers*.

From the simulation result, we can see that the more weight that is given to the streaming service and the more peers that use the streaming service, the less throughput the system will have. It is obvious that a BitTorrent-based streaming service will encounter the problem of system efficiency decline, reduce the system throughput and increase the download completion time.

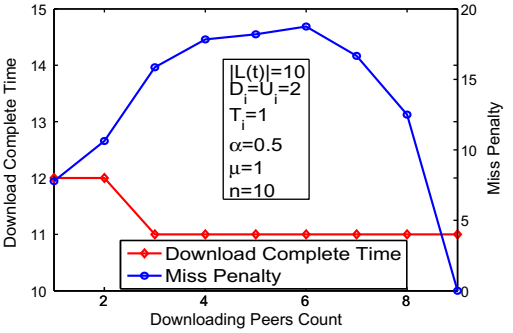
To support the streaming service, the piece selection policy needs to be changed. Different piece selection policies lead to different piece distributions, as shown in Fig. 2(b). We analyze the efficiency of BitTorrent when



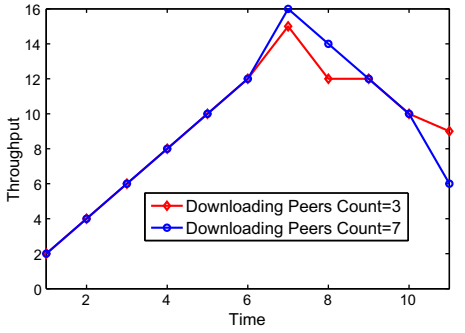
(a) Download Completion Time and Miss Penalty Changes as α Changes



(b) Throughput of Every Round for Different α



(c) Download Completion Time and Miss Penalty Changes as Swarm Composition Changes



(d) Throughput of Every Round for Different Swarm Composition

Fig. 3. Simulation result.

different piece selection policies are used in the next subsection.

4.2. Bandwidth usage analysis

Consider a swarm with $L = |L(t)|$ peers and $S = |S(t)|$ seeds sharing a file with n pieces. Each peer's download and upload capacity is $D_i = D$ and $U_i = U$, respectively. Assume that each piece is requested by the same number of peers. The download capacity of all peers for a given piece is then constant at $\frac{L*D}{n}$, and a piece's upload capacity is proportional to its count in the swarm. If the *rarest-first* policy is used in the swarm, the pieces are distributed uniformly [5]. Hence, the upload capacity of all peers and seeds for a given piece is also constant $\frac{(L+S)*U}{n}$. As for the *sequential* policy with the same count of the pieces in the swarm, the piece count descends linearly in the pieces close to the end of file. Piece k's count in the peers is $\frac{(n-k)*L}{n}$, as the peers requesting piece $k+1, k+2, \ldots, n$ have piece k. The total piece count in the peers is $\sum_{k=1}^{n} \frac{(n-k)*L}{n} = \frac{L*(n-1)}{2}$. Hence, the upload capacity of the peers for a given piece k is $\frac{2*(n-k)*L*U}{n*(n-1)}$. By adding the seeds' upload capacity, which is divided equally by the peers, we obtain the upload capacity of piece k, which is $\frac{2*(n-k)*L*U}{n*(n-1)} + \frac{S*U}{n}$. Because the upload capacity is usually the bottleneck, we assume that $U = \lambda D$ where $\lambda \in [0, \frac{N}{N+S}]$. For the *rarest-first* policy, the upload capacity of each piece is less than the download capacity; hence, the upload bandwidth can be used efficiently with the rarest-first policy.

$$\frac{2*(n-K)*L*U}{n*(n-1)} + \frac{S*U}{n} \leqslant \frac{L*D}{n} \tag{4.4}$$

For the *sequential* policy, we solve Eq. (4.4) and obtain $K \geqslant n - \frac{(L-\lambda S)(n-1)}{2\lambda L}$. For piece $k < K$, the upload capacity is larger than its download demand, and part of the upload bandwidth for the pieces is wasted. In total, $W = \sum_{k=1}^{K} \left(\frac{2*(n-k)*L*U}{n*(n-1)} + \frac{S*U}{n} - \frac{L*D}{n} \right)$ upload bandwidth is wasted. The upload bandwidth usage ratio of the *sequential* download can be defined as $1 - \frac{W}{(L+S)*U}$. Fig. 4 shows the upload bandwidth-wasting ratio of the *sequential* download as the peer number L changes and that the upload bandwidth-wasting rate decreases slightly as L increases. However, more than 14% of the upload bandwidth is still wasted. In a real system, the rare pieces $k > K$ will use part of the wasted upload bandwidth, but considerable bandwidth will still be wasted. Hence, we can see that the random peer selection and the *sequential* piece selection fail to make the best use of the peers' upload capacity and reduce the system's efficiency.

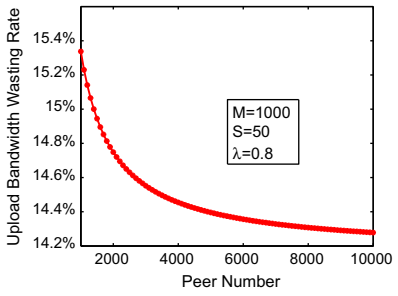


Fig. 4. Upload bandwidth wasting rate changes as peer count changes.

According to the model analysis and measurement results above, it is insufficient to support the streaming service using BitTorrent by simply changing the piece selection policy from the *rarest-first* policy to the *sequential* policy. The peer and piece selection policies should both be optimized. In the next section, our joint method will be presented.

5. Policy modification

In this section, we present our method to make BitTorrent support downloading and streaming service simultaneously. Our method concentrates on peer selection policy and piece selection policy optimization. The modified BitTorrent is compatible and can work together with unmodified BitTorrent clients.

5.1. Peer selection policy

Unmodified BitTorrent selects peers randomly to request pieces. In Section 4, we showed that the *sequential* piece selection and the *random* peer selection policies waste the peers' upload capacity and reduce the system's efficiency. In a streaming system, the former pieces are abundant while the latter pieces are rare. The random peer selection policy may choose older peers for former pieces and occupy the latter pieces' upload capacity while the newcomer peers are idle.

We propose CAP (Closest-Ahead Peers) as the peer selection policy to make the best use of the streaming *peers*' upload bandwidth. In this method, each peer sorts its own connected streaming *peers* according to its *sequential* download process. Peers obtain the other peers' downloading processes based on bitfield and the *HAVE* message. CAP then picks one peer randomly from its closest-ahead peers to request the next *sequential* piece, and the closest-ahead peers' newly downloaded pieces will be used in the upload. Similarly to YNP in [16], we select several peers to avoid overloading a few peers. The difference in the CAP used in [16] is that their policy requests from the least-loaded peers, while ours requests from the closest-ahead peers. Our policy requires no extra information interaction with trackers and other peers but obtains a similar effect based only on the existing information. Another advantage is that peers using CAP can obtain continuous uploads from only a few closest-ahead peers, and a more fluent playback experience will be achieved with less fluctuation in peer connection.

Algorithm 1. Peer Selection for Peer i to Request Piece k

In $M_i(t)$, $N_i(t)$, $S_i(t)$, $G_{jk}(t)$, $Q_j(t)$
function *find_max* finds peer with largest $Q_j(t)$ in a given set.
Output p: the chosen peer to send request for piece k;

```
for p ∈ M_i(t) do
    if G_pk(t) == 1 then
        return p
    end
end
min ← Infinity, max ← Infinity, CAP ← {}
for j ∈ N_i(t) ∪ S_i(t) do
    if Q_p(t) ⩾ k and (Q_p(t) < max or |CAP| < N) then
        CAP ← CAP ∪ {p} − {max}
        if |CAP| == N then
            max = find_max(CAP)
        end
    end
end
return p ← select a peer randomly in CAP
```

In practical situations, *streaming peers* and *downloading peers* coexist in the BitTorrent swarm. To better utilize their upload bandwidth, we expand our CAP method for used in hybrid swarms with both *downloading peers* and *streaming peers*. Peer i splits its neighbors (also referred to as peers) into three sets according to their bitfields, i.e., streaming set $N_i(t)$, downloading set $M_i(t)$ and seed set $S_i(t)$. Neighbor j's piece holdings $G_{jk}(t)$ and the *sequential* download process $Q_j(t)$ (if it is a streaming peer) are also available to peer i based on j's bitfield. The peer selection policy for peer i requesting piece k at time t is shown in Algorithm 1. When peer i wants to request piece k, if there are neighbors with piece k in the downloading set that are not busy with other requests, peer i will request piece k from a random neighbor with piece k in its downloading set. Otherwise, peer i will use CAP to select a peer in the streaming set to obtain that piece. If both downloading set and streaming set are unavailable, peer i requests piece k from seeds set randomly.

To achieve a larger download rate and better playback continuity, the peers will exclude low-bandwidth and heavy-loaded connected peers periodically, obtain new peers, and update the downloading set, the streaming set and the seeds set simultaneously. Hence, serious load balancing problems will not occur. This policy is compatible with the *tit-for-tat* incentive mechanism. Because *streaming peers* first request pieces from downloading sets, the formerly downloaded pieces of the *streaming peers* can be used in the upload for requests from peers in the downloading set because the peers using the *rarest-first* policy do not have all the former pieces. However, a peer cannot upload these pieces to its closest-ahead streaming peers

because newcomer peers have few pieces for the older peers. This problem can be addressed with our piece selection policy, described in the next subsection.

5.2. Piece selection policy

The original piece selection policy of BitTorrent is the *rarest-first* policy. A peer keeps track of each piece's count among the peers it connects and selects the piece with the lowest count to request. To support the streaming service, the pieces near the play point should be downloaded first, and *sequential* download is required while the video is playing. However, as shown in Sections 3 and 4, *sequential* download will significantly decrease the system's efficiency. Therefore, we propose a sliding window-based hybrid policy that combines the *rarest-first* and the *sequential* policies to avoid a considerable degradation of efficiency while supporting the streaming service.

In the proposed policy, two sliding windows are maintained while downloading:

Buffer window: it contains an appropriate number of pieces close to the play point to buffer for smooth playback. The buffer window size is b ($b = 10$ in Fig. 5), and it contains b continuous pieces from the next playing piece.

Request window: it contains only the requesting pieces in the buffer window. The request window never slides out of the buffer window. The window size is $d = \min\{c - r, b - f\}$, where c ($c = 5$ in Fig. 5) is the maximum number of pieces that a peer can request simultaneously, r is the number of requested but not downloaded pieces chosen by the *rarest-first* policy, and f is the number of downloaded pieces in the buffer window.

The buffer window slides when a piece is played. If users perform seek operations, the buffer window will slide as well. When a piece in the request window is finished or missed, the request window slides and changes its size to $d = \min\{c - r, b - f\}$. Simultaneously, a peer will select $c - d$ pieces from the buffer window to download using the *rarest-first* policy. Hence, the pieces in the buffer window will be requested sequentially, and the other pieces will be requested using the *rarest-first* policy when the buffer window is full.

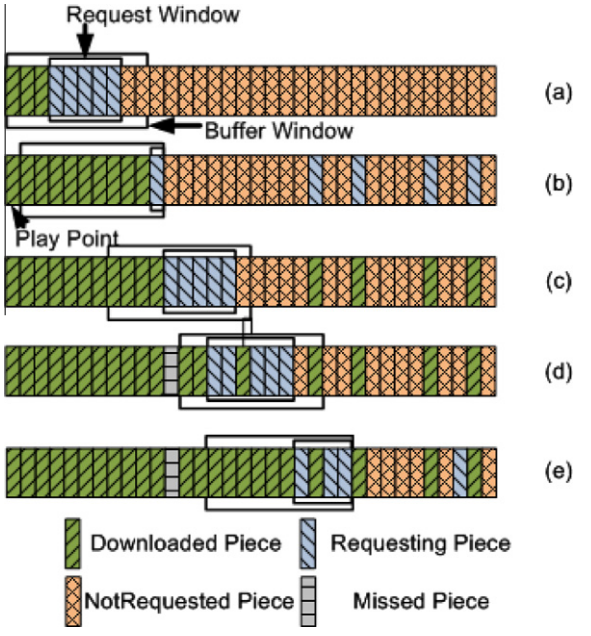


Fig. 5. Sliding window based hybrid piece selection policy.

Fig. 5 shows how the sliding window-based hybrid method works. In Fig. 5(a), before starting playback, the peer first downloads the pieces into the buffer window until the buffer window is full. All the requested pieces are in the buffer window. The startup delay is the time used to fill the buffer window. When the buffer window is full, the peer starts playback, and the buffer window slides, as shown in Fig. 5(b). The peer requests the newly added piece as the buffer window slides, and the request window size d is reduced to $b - f = 1$. The peer then requests other $c - d = 4$ pieces using the *rarest-first* policy. In Fig. 5(c), as the video plays and the buffer window slides, the buffer window is no longer full. Next, all the pieces are requested using the *sequential* policy in the buffer window, and the request window is enlarged to $c - r = 5$. During the playback process, pieces may be missed, as shown in Fig. 5(d). When the buffer window slides, the pieces in the buffer window will be downloaded first. The pieces in the buffer window will be downloaded using the *rarest-first* policy. The downloaded pieces using the *rarest-first* policy have a larger probability of being used to respond to other peers' requests; they help the peer upload more pieces so that it can obtain more downloads from other peers based on *tit-for-tat* policies. It should be noted that our method only works when the peers' download rate is greater than the video's bitrate. If this is not the case, the peer simply requests the pieces sequentially.

In addition, video on demand systems need to provide users with a large subset of VCR functionalities, including pause, fast forward, fast rewind, slow forward, slow rewind, and jump to previous/future frame.[11] It is easy for our method to provide these functionalities simply by changing the play point according to the VCR operation. The buffer window slides as the video fast forwards or jumps.

The buffer size b is crucial to the system's performance. If the buffer window size is too large, it becomes a *sequential* download; if it is too small, the video will not play smoothly, and user satisfaction will decrease. If a peer's normal download rate is D ($D \geqslant \frac{1}{\mu}$, to support streaming service), the peer's download rate decreases suddenly to D'($D' = \lambda D$, $\lambda \leqslant 1$). The rest time for smooth video play $t = \frac{b*\mu}{\frac{1}{\mu} - \frac{D'}{\gamma}}$ is the most important performance index of the buffer window, while γ is the piece size. The continuous time to play is in proportion to the buffer window size b and in inverse proportion to the download rate D'. The buffer window size b can be determined by the user's bandwidth, and peers with lower bandwidths require larger buffer windows to obtain better playback continuity.

[11] http://en.wikipedia.org/wiki/Video_on_demand.

However, for simplicity, we use a constant buffer window size in this evaluation.

In this section, we modified BitTorrent's peer and piece selection policies to make the BitTorrent system support downloading and streaming service simultaneously. It should be noted that these modifications are only suitable for *streaming peers*. The *downloading peers* will use unmodified BitTorrent.

6. Performance evaluation

This section evaluates various aspects of the performance of our proposed method using PlanetLab. Although there will be differences between the residential network and PlanetLab, we still treat PlanetLab as a useful testbed for P2P applications, considering the difficulty of deploying a P2P application on a real network, especially for the limited purpose of comparing policies. We first modified Ctorrent with the different policies. Four different kinds of Ctorrent clients are used:

(1) Unmodified Ctorrent using the *rarest-first* policy.
(2) Modified Ctorrent using the *sequential* policy.
(3) Modified Ctorrent using BiToS [10] with $p = 0.8$ and a high-priority set size of 60.
(4) Modified Ctorrent using our proposed peer and piece selection policy with a buffer window size of 60. "Sliding Window" is used to indicate our method.

Unlike the method proposed by [23], which disables *tit-for-tat*, we keep the built-in incentive mechanism to make the BitTorrent VoD system robust against selfish behaviors. We use the Ctorrent clients to share a given file on PlanetLab for several rounds. In each round, only one kind of Ctorrent is used in the swarm, except in hybrid swarms, in which unmodified and modified Ctorrents are used on different peers. The shared file is 1 GB in size with a piece size of 1 MB. Five hundred and seventy-nine nodes are added to our slice, and approximately 480 nodes work properly. The nodes' bandwidth can be set using PlanetLab[12] or Ctorrent commands. However, we use the default setting for simplicity. One node (cs-planetlab2.cs.surrey.sfu.ca) is chosen as a tracker, and another (pl1.pku.edu.cn) is chosen as a seed; the rest of the nodes are peers downloading the shared files simultaneously. To make all the nodes start downloading at the same time, we first start Ctorrent on each node then start the tracker. After the nodes finish downloading the video, they remain in the swarms. The evaluation was conducted between September 5th, 2010, and October 15th, 2010. To exclude the influence of daily background traffic fluctuations, we begin transferring and obtain the traced logs at the same time of day (22:00(GMT+8) and 10:00(GMT+8) the next day, respectively) for each round.

According to the model analysis of Section 4, the following metrics are used to evaluate the performances of different policies.

Download Completion Time: The time required for a peer to download a whole file. We use this time to indicate the system throughput. The less time used, the higher the system throughput is. In hybrid swarms with coexisting *downloading* and *streaming peers*, we concentrate on the download completion time of the *downloading peers*. In streaming swarms with only *streaming peers*, although the download completion time of the *streaming peers* has no direct connection with user satisfaction, we still analyze it to demonstrate the system's throughput.

Miss Penalty: The peers' miss penalty reflects their playback continuity, which directly affects the user's experience. In the experiment, we assume that the play time for one piece is $\mu = 5$ s, so the video's playback rate is 1.6 Mbps. Clients will start playing after the first b (buffer window size) pieces are downloaded. The required download completion time t_r of peer i for piece k can then be determined, which is the start time added to the former k pieces' playtime. Given the recorded download completion time t_d of piece k, the miss penalty of peer i for piece k is $\max\{0, t_d - t_r\}$. The total miss penalty of one peer is the sum of the miss penalties of all the pieces.

In addition to these two metrics, the following three metrics are used to analyze the performance of the BitTorrent-based VoD system.

Startup Delay: The time used to download the first k pieces is defined as the startup delay, which is another factor with a close connection to user satisfaction.

Upload Volume Distribution: We use the upload volume distribution to check whether the policy is suitable for the built-in *tit-for-tat* mechanism. If the peers' upload volumes are decentralized with small variances, all peers can contribute to one another. This policy *cooperates well with tit-for-tat*.

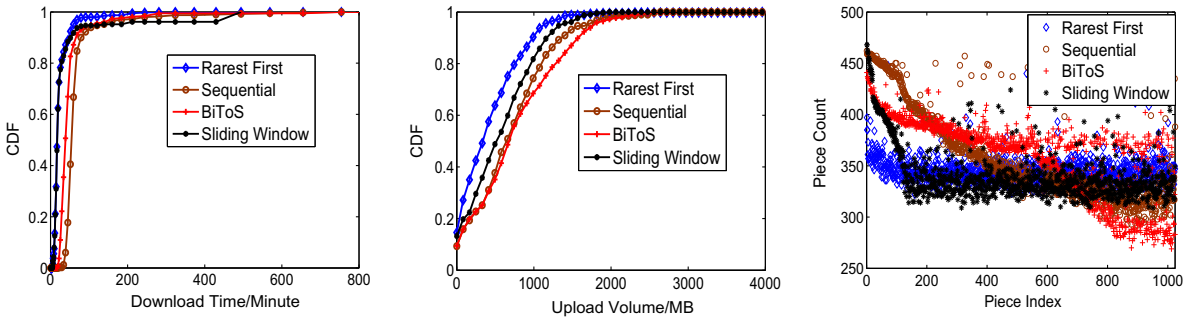
Piece Distribution: We take one snapshot of the download process when half of the peers finish downloading to demonstrate the system piece diversity.

6.1. Policies comparison

We first compare the performance of different policies in swarms in which all peers are streaming peers and join the swarm simultaneously.

Fig. 6(a) shows the download completion time distribution. Peers using the *rarest-first* policy have the lowest download completion times, while peers with the *sequential* download policy have the most. Eighty-eight percent of peers using the *rarest-first* policy will finish their download in less than 50 min; only 16% of the *sequential* peers will do so. The peers with our method and BiToS's download completion time fall in between, but our method can finish downloading in a much shorter time than BiToS for high-bandwidth peers who would be potential streaming service users. In Fig. 6(b), the peers' upload volume distribution is presented. The cumulative distribution curves of the *rarest-first* policy and our method are steeper than the other two curves with a smaller variance; our method's improved fairness and efficiency will inspire more peers to upload. Fig. 6(c) presents the piece distribution results. Pieces are uniformly distributed when the *rarest-first* policy is used, while the *sequential* download has a decreasing bias-like piece distribution. The piece distribution of BiToS has a smaller slope than that of the *sequential* download,

[12] http://www.planet-lab.org/doc/BandwidthLimits.



(a) Download Completion Time Distribution (b) Upload Volume Distribution (c) Piece Count Distribution

Fig. 6. PlanetLab measurement result for different policies.

but the counts of the different pieces differ greatly. With our proposed method, only the beginning pieces have a decreasing bias-like piece distribution, and the latter pieces are distributed uniformly. Therefore, our method can maintain the diversity of the pieces.

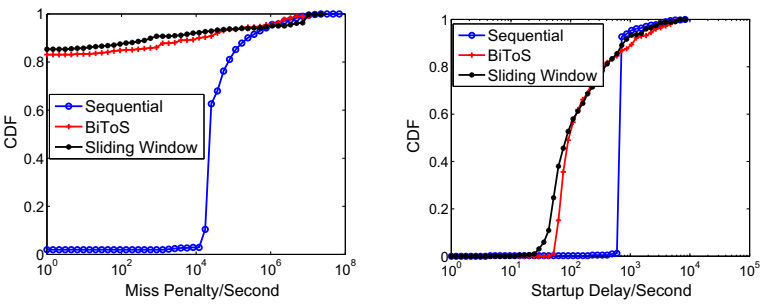
In addition to system throughput, the streaming service performance is also analyzed. Because the *rarest-first* policy is not suitable for streaming service, we exclude it from the figures. Fig. 7(a) shows the miss penalties of different policies. The *sequential* policy has the worst effect, with the largest miss penalties. Our proposed method is better than BiToS. The miss penalty of the *sequential* policy is around 10,000 s. Using our method, 85% of peers have a miss penalty of less than 100 s, which is acceptable for a 5120 s video. Therefore, the bandwidth of the PlanetLab network is sufficient for a BitTorrent-based VoD service. The startup delay is presented in Fig. 7(b). The startup delay is the time used to download the first $b(b = 60)$ pieces. The startup delay of the *sequential* download policy is in the vicinity of 800 s, while more than 60% of peers' startup times using our method are less than 100 s. BiToS has a similar performance. In summary, our method is much better than the *sequential* policy and BiToS to support streaming service.

In practice, *streaming* and *downloading peers* coexist in BitTorrent swarms. We next analyze the performance of different policies in hybrid swarms with both streaming and *downloading peers*. In the hybrid swarm, 50% of peers are downloading peers using unmodified CTorrent, while others are streaming peers using CTorrent with different policies. Fig. 8(a) shows the *downloading peers*' download completion time distributions. Eighty-six percent of *downloading peers* finish the download process in fewer than 60 min with our method, while only 66% do so with BiToS and 49% do so in the *sequential* swarms. Therefore, it is obvious that our method is much better than the *sequential* policy and BiToS. In Fig. 8(b), the miss penalty of *streaming peers* using the different policies is presented. The miss penalty of 70% of *streaming peers* using the *sequential* policy and our method is less than 100 s, while only 40% of peers' miss penalty is less than 100 s with BiToS. Compared with Fig. 7(a), we can see that the miss penalty of *streaming peers* in a hybrid swarm is larger than that in pure streaming swarms. Fig. 8(c) shows the *streaming peers*' startup delay. Although there are more *streaming peers* that start up in fewer than 100 s in swarms using the *sequential* policy than in swarms using our method, our method with a median startup delay of 168 s is faster than the *sequential* policy, with a median of 478 s, and BiToS, with 1258 s.

Based on the above results, it is obvious that the effect of our method is better than the *sequential* policy and BiToS in both system throughput and streaming service requirements.

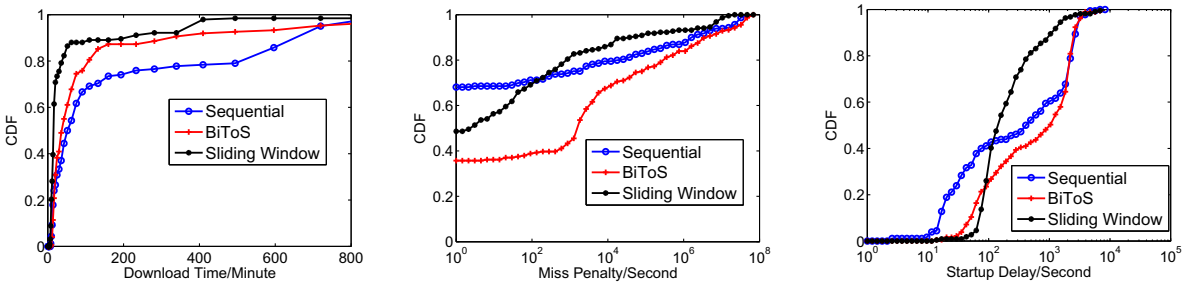
6.2. Dynamic situation discussion

The above experiments consider only the flash-crowd situation, in which all peers join the swarm simulta-



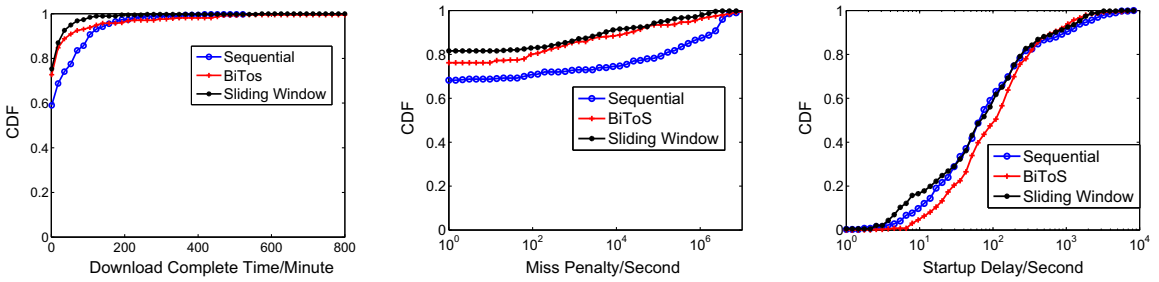
(a) Miss Penalty Distribution (b) Startup Delay Distribution

Fig. 7. PlanetLab measurement result for different policies.



(a) Download Completion Time Distribution (b) Miss Penalty Distribution (c) Startup Delay Distribution

Fig. 8. PlanetLab measurement result for different policies in a hybrid swarm.



(a) Download Completion Time Distribution (b) Miss Penalty Distribution (c) Startup Delay Distribution

Fig. 9. PlanetLab measurement result for dynamic scenario 1.

neously. This subsection will discuss a dynamic situation in which peers join and leave the swarm dynamically. For simplicity, we consider only two scenarios: (1) peers join uniformly, and seeds never leave the swarms; and (2) peers join uniformly, and seeds leave the swarms immediately after watching a video. In the first scenario, one seed exists in the swarms at the beginning, and 5 peers join the system every 3 min without leaving the swarm. In the second one, 5 peers join the system every 3 min and leave the system immediately after watching a video. The peers will stay in the system for $max\{T_d, T_w\}$ in total, where T_d is a peer's downloading completion time, T_w is the video's play time and $T_w = \mu * n = 5 * 1024 = 5120$ s.

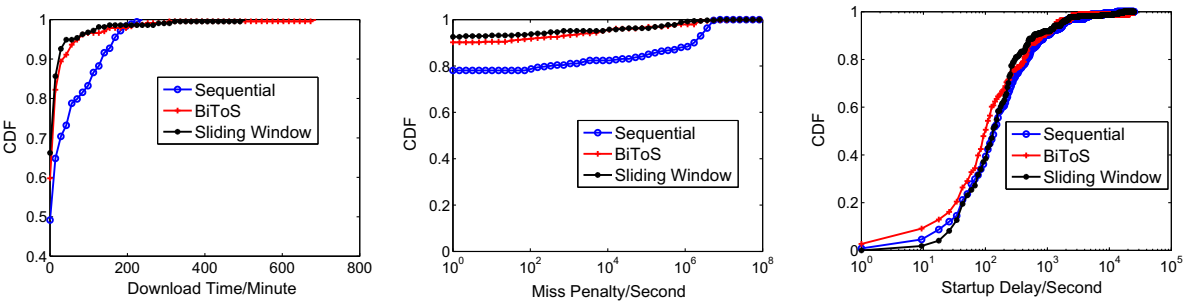
Fig. 9(a)–(c) show the download completion time, miss penalty and startup delay of scenario 1, respectively, while Fig. 10(a)–(c)[13] show the result of scenario 2. The evaluation results show that our policy and BiToS have much smaller download completion times and miss penalties than the *sequential* policy in both scenarios and that our method is slightly better than BiToS. For the startup delay, the three policies have similar results, making it difficult to distinguish between them using this criterion. In summary, the results of these two scenarios provide evidence that our method is much better than the *sequential* download policy and slightly better than BiToS.

6.3. Buffer window size discussion

The size of buffer window b is critical to the effect of our method. A buffer window that is too small leads to worse play fluency, while a buffer window that is too large degrades to the *sequential* download policy. We ran our modified Ctorrent with different buffer window sizes in different transfer rounds and compared their performances.

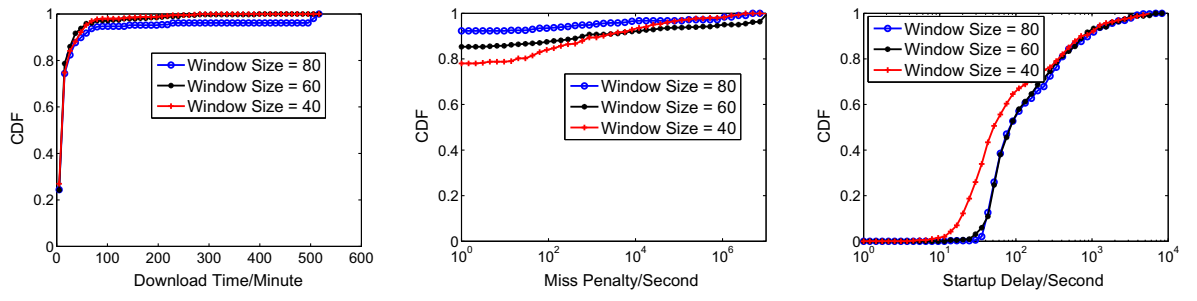
Fig. 11(a) presents the download completion time distribution when windows of different sizes are used. We can see that for high bandwidth users, the download completion time is slightly different when the buffer window size changes. Because it takes only a short time to fill in the buffer window, most pieces are downloaded using the *rarest-first* policy. However, for low-bandwidth end users, the larger the buffer window size is, the more time is used to download the whole file because a bigger window size means that more pieces are downloaded using the *sequential* download policy. Fig. 11(b) and (c) show the miss penalty and startup delay when using different buffer window sizes. These results show that a larger

[13] It should be noted that this simulation is a measurement using PlanetLab between September 1th 2011 and September 5th 2011. The result cannot be used as a comparison with previous results because of the differences in the network conditions. However, we believe it is sufficient to distinguish the different policies' performances.



(a) Download Completion Time Distribution (b) Miss Penalty Distribution (c) Startup Delay Distribution

Fig. 10. PlanetLab measurement result for dynamic scenario 2.



(a) Download Completion Time Distribution (b) Miss Penalty Distribution (c) Startup Delay Distribution

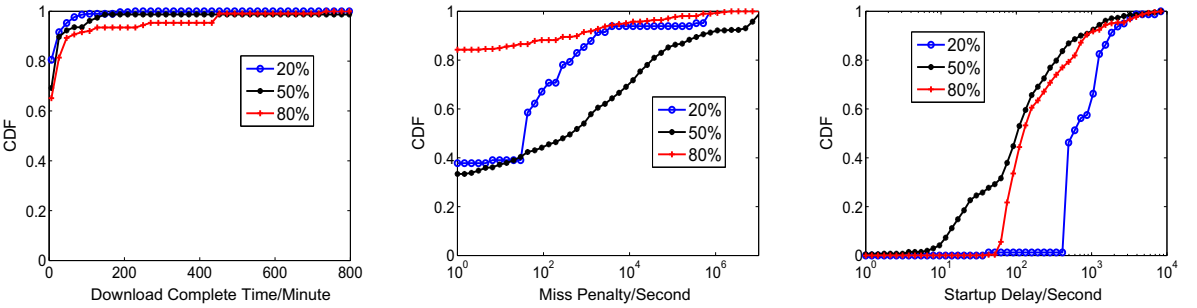
Fig. 11. PlanetLab measurement result for different window sizes.

window size results in smaller miss penalties and a larger startup delay.

6.4. Swarm composition discussion

The performance in hybrid swarms with different swarm compositions is also measured on PlanetLab. In hybrid swarms, *downloading peers* use an unmodified BitTorrent client, while *streaming peers* use a modified BitTorrent with our method. Different compositions of BitTorrent swarms with these two kinds of peers are discussed. Three situations are considered in our experiment: (1) 80% of peers use the *rarest-first* policy, while 20% use our method; (2) 50% use the *rarest-first*, while 50% use our method; and (3) 20% use the *rarest-first*, while the rest 80% use our method. These three situations are indicated by "20%", "50%" and "80%", respectively. We then compare the download completion times of the *downloading peers* and the miss penalty and startup delay of the *streaming peers* in different swarm compositions.

Fig. 12(a) shows the download completion time distributions of the *downloading peers*. It is obvious that the more peers that use the *rarest-first* policy, the less time is used for the *downloading peers* to finish downloads, as



(a) Download Completion Time Distribution (b) Miss Penalty Distribution (c) Startup Delay Distribution

Fig. 12. PlanetLab measurement result for different swarm composition.

the *rarest-first* policy will promote system throughput. Fig. 12(b) and (c) present the streaming performance of different swarm compositions. We can see that the swarms with 20% of peers or 80% of peers using our method have a smaller miss penalty than those with 50% of peers using our method, similar to the simulation result in Fig. 3(c).

However, for the startup delay, a composition with 50% of peers using our method will perform the best, while a composition with 20% of peers using our method has the worst effect.

In this section, we evaluate the performance of our method by comparing it with other policies, such as the *sequential* policy and BiToS in streaming swarms and hybrid swarms. In addition to the flash-crowd situation, we compare different policies in a situation in which peers join a swarm dynamically. The results show that our method is better than the *sequential* policy and BiToS in both system throughput and playback continuity. We also discuss the system's performance in situations with different window sizes and the swarm composition's impact on system performance.

7. Conclusion and future work

In this paper, we explored methods of providing VoD service in existing BitTorrent swarms while maintaining the download efficiency of the file-sharing users. We examined the contents of BitTorrent swarms, especially video files. The measurement results show that there is an increasing demand for a BitTorrent-based streaming service. Based on the real system measurement results and the model analysis, we found that in hybrid BitTorrent systems with both *downloading* and *streaming peers*, system efficiency would decline if we only modified the piece selection policy from the *rarest-first* policy to the *sequential* one. We modified both piece and peer selection policies to create a BitTorrent support streaming service while maintaining the throughput of downloading service. We proposed a sliding window-based hybrid piece selection policy that combines the *rarest-first* and the *sequential* policies, and the peer selection is also optimized with CAP. The evaluation results obtained on PlanetLab show that the proposed method has a higher performance than the *sequential* policy and BiToS.

In our method and evaluation, the buffer window size and startup buffer size are constant; we believe it is best if they are dynamically adaptive to the peers' properties, such as network bandwidth. Therefore, the determination of the best buffer size is a challenge for future work. Our evaluative work is mainly based on flash-crowd situations, but in reality, a steady situation with peers joining and leaving is more common. However, a steady situation is hard to measure or simulate; hence, making a more accurate evaluation or deploying our strategy in a real application will be part of our future work. In addition, we have only considered situations in which peers download a single file, while peers usually download multiple files at the same time. Therefore, making the BitTorrent support streaming service better in a multi-swarm situation will be discussed in the future. Finally, we only modified BitTorrent for *streaming peers* to support streaming service. However, whether the downloading peers' client should be modified for swarms to provide better streaming service remains an interesting question.

Acknowledgments

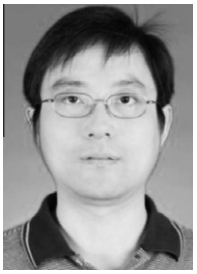
We would like to thank Professor Hongying Liu, the reviewers and the editors for their constructive comments. This work was supported in part by NSFC Project (61170292, 60970104), 973 Project of China (2009CB320501), 863 Project of China (2008AA01A323, 2008AA01A326) and Program for New Century Excellent Talents in University.

References

[1] Y. Huang, T.Z.J. Fu, D.M. Chiu, J. Lui, C. Huang, Challenges, design and analysis of a large-scale p2p-vod system, in: ACM SIGCOMM, 2008.
[2] B. Cheng, L. Stein, H. Jin, X. Liao, Z. Zhang, GridCast: improving peer sharing for P2P VoD, in: ACM Transactions on Multimedia Computing, Communications, and Applications (TOMCCAP), vol 4, issue no. 4, article 26, 2008.
[3] Z. Liu, C. Wu, B. Li, S. Zhao, Uusee: Large-scale operational on-demand streaming with random network coding, in: IEEE INFOCOM, 2010.
[4] Jun Lei, Lei Shi, Xiaoming Fu, An experimental analysis of Joost peer-to-peer VoD service, in: Peer-to-Peer Networking and Applications, vol 3, no. 4, 2010, pp. 351–362.
[5] D. Qiu, R. Srikant, Modeling and performance analysis of BitTorrent-like peer-to-peer networks, in: ACM SIGCOMM, 2004.
[6] A.R. Bharambe, C. Herley, V.N. Padmanabhan, Analyzing and improving BitTorrent performance, in: IEEE INFOCOM, 2006.
[7] A.L.H. Chow, L. Golubchik, V. Misra, BitTorrent: an extensible heterogeneous model, in: IEEE INFOCOM, 2009.
[8] Lei Guo, Songqing Chen, Zhen Xiao, Enhua Tan, Xiaoning Ding, Xiaodong Zhang, Measurements, analysis, and modeling of BitTorrent-like systems, in: ACM IMC, 2005.
[9] N. Parvez, C. Williamson, A. Mahanti, N. Carlsson, Analysis of bittorrent-like protocols for on-demand stored media streaming, in: ACM SIGMETRICS, 2008.
[10] A. Vlavianos, M. Iliofotou, M. Faloutsos, BiToS: Enhancing BitTorrent for supporting streaming applications, in: IEEE INFOCOM Global Internet Workshop, 2006.
[11] Y.R. Choe, D.L. Schuff, J.M. Dyaberi, V.S. Pai, Improving VoD server efficiency with bittorrent, in: ACM Multimedia, 2007.
[12] P. Shah, J.F. Pâris, Peer-to-peer multimedia streaming using BitTorrent, in: IEEE IPCCC, 2007.
[13] P. Gill, M. Arlitt, Z. Li, A. Mahanti, Youtube traffic characterization: A view from the edge, in: ACM IMC, 2007.
[14] K. Xu, H. Li, J. Liu, W. Zhu, W. Wang, PPVA: A universal and transparent peer-to-peer accelerator for interactive online video sharing, in: IEEE IWQOS, 2010.
[15] N. Carlsson, D. Eager, Peer-assisted on-demand streaming of stored media using BitTorrent-like protocols, in: IFIP NETWORKING, 2007.
[16] Y. Yang, A. Chow, L. Golubchik, D. Bragg, Improving QoS in bittorrent-like vod systems, in: IEEE INFOCOM, 2010.
[17] C. Liang, Z. Fu, Y. Liu, C.W. Wu, ipass: Incentivized peer-assisted system for asynchronous streaming, in: IEEE INFOCOM, 2009.
[18] Y. Borghol, S. Ardon, N. Carlsson, A. Mahanti, Toward efficient on-demand streaming with BitTorrent, in: IFIP NETWORKING, 2010.
[19] P. Savolainen, N. Raatikainen, S. Tarkoma, Windowing bittorrent for video-on-demand: Not all is lost with tit-for-tat, in: IEEE GLOBECOM, 2008.
[20] J. Lv, X. Cheng, Q. Jiang, J. Ye, T. Zhang, S. Lin, L. Wang, LiveBT: Providing video-on-demand streaming service over BitTorrent systems, in: PDCAT, 2007.
[21] H. Wang, J. Liu, K. Xu, Measurement and enhancement of BitTorrent-based video file swarming, Peer-to-Peer Networking and Applications, 3, Springer, 2010. no. 3, pp. 1–17.
[22] J. Mundinger, R. Weber, G. Weiss, Optimal scheduling of peer-to-peer file dissemination, Journal of Scheduling, 11, Springer, 2008. no. 2, pp. 105–120.

[23] B. Fan, D.G. Andersen, M. Kaminsky, K. Papagiannaki, Balancing throughput, robustness, and in-order delivery in P2P VoD, in: ACM CoNext, 2010.
[24] Z. Li, J. Cao, G. Chen, ContinuStreaming: Achieving high playback continuity of Gossip-based Peer-to-Peer streaming, in: IPDPS, 2008.
[25] L. D'Acunto, N. Andrade, J. Pouwelse, H. Sips, Peer selection strategies for improved QoS in heterogeneous BitTorrent-like VoD systems, in: IEEE ISM, 2010.

Zhen Ma received the BEng degree in computer science from Beijing University of Posts and Telecommunications, Beijing China. He is currently a master student in the Department of Computer Science of Tsinghua University. His research interests include P2P and overlay network.



Ke Xu received the BS, MS, and PhD degrees in computer science from Tsinghua University, China, in 1996, 1998, and 2001, respectively. Currently he is a full professor in the Department of Computer Science of Tsinghua University. His research interests include next generation Internet, traffic management, switch and router architecture, and Internet of Things. He is a senior member of the IEEE and a member of the ACM.

Jiangchuan Liu (S'01-M'03-SM'08) received the BEng degree (cum laude) from Tsinghua University, Beijing, China, in 1999, and the PhD degree from The Hong Kong University of Science and Technology, in 2003, both in computer science. He was a recipient of Microsoft Research Fellowship (2000), a recipient of Hong Kong Young Scientist Award (2003), and a co-inventor of one European patent and two US patents. He coauthored the Best Student Paper of the IWQoS'08 and the Best Paper (2009) of the IEEE Multimedia Communications Technical Committee (MMTC). He is currently an associate professor in the School of Computing Science, Simon Fraser University, British Columbia, Canada, and was an assistant professor in the Department of Computer Science and Engineering at The Chinese University of Hong Kong from 2003 to 2004. His research interests include multimedia systems and networks, wireless ad hoc and sensor networks, and peer-to-peer and overlay networks. He is an associate editor of the IEEE Transactions on Multimedia, and an editor of the IEEE Communications Surveys and Tutorials. He is a senior member of the IEEE and a member of the Sigma Xi.

Haiyang Wang is currently a Ph.D Student in the School of Computing Science, Simon Fraser University, British Columbia, Canada. He is working in the Multimedia and Wireless Networking Group and his research interests include peer-to-peer networks, multimedia systems/networks, IP routing and QOS.