Understanding the Quality-of-Service of Ubiquitous BitTorrent-like Media Streaming

Zhenhua Li
School of Software and TNLIST
Tsinghua University
Beijing, China, 100084
lizhenhua1983@tsinghua.edu.cn

Tieying Zhang
Institute of Computing Technology
Chinese Academy of Sciences
Beijing, China, 100190
zhangtiey@software.ict.ac.cn

***Abstract*—As the two mainstream application scenarios of P2P (Peer-to-Peer) technique, P2P file sharing and P2P media streaming have presented a trend of convergence in recent years. The representative case is the *"ubiquitous BitTorrent-like media streaming"*, i.e., file-sharing peers and media-streaming peers coexist in the same BitTorrent swarm, exchanging and competing for data, while working in different manners. Considering the promising prospect if we can join ubiquitous BitTorrent swarms for media streaming (or says media multicast or broadcast) service, this topic is worth a preliminary investigation into its feasibility and influence. To this end, we propose a quantitative analytical model for evaluating the QoS (Quality-of-Service) of ubiquitous BitTorrent-like media streaming in this paper. Our model captures the most common and fundamental mechanisms (like the *sliding-window based piece selection*) adopted in various BitTorrent-like media streaming implementations, in order to better understand the key properties of QoS (such as playback continuity and startup/jump latency) and provide heuristics for optimizing the QoS of ubiquitous BitTorrent-like media streaming.**

***Index Terms*—BitTorrent (BT); media streaming; quality-of-service (QoS); modelling; optimization.**

I. Introduction

By means of its decentralized, resilient multicast/broadcast architecture and abundant utilization of Internet edge nodes/resources, P2P (peer-to-peer) technique has been widely utilized in various application scenarios for over a decade. The most common application scenarios are mainly two folds: 1) *File sharing*, such as BitTorrent [1], Gnutella [2], eDonkey/eMule [3], KaZaa [4], and so forth, among which BitTorrent is the most popular. 2) *Media streaming*, such as CoolStreaming [5], Skype [6], PPLive [7], UUSee [8], and so forth. Media streaming can be further classified into two modes: *live streaming* and *VoD* (*Video/voice-on-Demand*) *streaming*. Either mode is distinct from file sharing due to the stringent requirements on both data transfer bandwidth and latency.

Because P2P file sharing came into being earlier than P2P media streaming and its system architecture is easier to build, P2P file sharing generally has a larger user group than P2P media streaming, as well as richer contents accordingly. However, P2P file sharing and P2P media streaming basically work independently on today's Internet, with their respective network deployment, user group and file resources. In the past several years, signs have been emerging that the two mainstream application scenarios of P2P start to converge. People have realized that in P2P file sharing systems, there exist a large number of media files that can also be used for media streaming. Sometimes, this number may even exceed the number of media files in P2P streaming systems themselves. Of course, due to the stringent requirements of media streaming on both bandwidth and latency, the convergence of P2P file sharing and media streaming may still require extension, modification and optimization of the original data multicast/broadcast protocol(s) and system architecture(s).

As the representative case of the convergence of P2P file sharing and media streaming, *"ubiquitous BitTorrent-like media streaming"* has attracted wide attention from both industry and academia in recent years. It means file-sharing peers and media-streaming peers coexist in the same BitTorrent swarm, exchanging and competing for data, while working in different manners. In industry, it is often called the "view-as-download" BitTorrent mode, which has been supported by a number of popular commercial systems like BitComet [9], Xunlei [10], QQXuanfeng [11], Funshion [12], and so on. In academia, considerable researches, e.g., BASS [13], BiToS [14] and Toast [15], have been conducted on this novel topic.

Considering the promising prospect if we can join ubiquitous BitTorrent swarms for media streaming (or says media multicast or broadcast) service, this topic is worth a preliminary investigation into its feasibility and influence. As depicted in Fig. 1, ubiquitous BitTorrent-like media streaming has the following major features:

- *Complicated working environments*. In a ubiquitous BitTorrent swarm, three kinds of users coexist: 1) *Seeds*, i.e., the users who possess the whole file and only upload data to others. 2) *Random leechers*, i.e., the users who retrieve data pieces in a random (in fact "rarest-first" [16]) order like traditional BitTorrent users. 3) *Sequential leechers*, i.e., the users who retrieve data pieces in a sequential order like common media streaming users. They exchange and compete for data with each other but work in different manners, in particular in terms of the *piece selection strategy* and the *peer connection* (or says "peer unchoking" [16]) *strategy*.
- *Full file cache and VoD streaming*. Since the BitTorrent

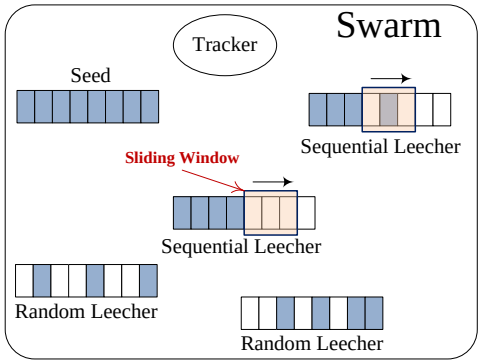

Fig. 1. A simple demo of ubiquitous BitTorrent-like media streaming. Inside the data swarm, all peers (including seeds, random leechers and sequential leechers) share the same file but work in different manners.

protocol is initially designed for file sharing, full file is (or will be) cached in each user. This is different from the partial file cache strategy employed by common P2P streaming systems. Besides, there are usually (sometimes necessarily) a certain number of seeds in a BitTorrent swarm. Thus, the full file cache and the existence of seeds make the BitTorrent-like media streaming better suited for VoD streaming rather than live streaming. Although in theory, BitTorrent-like media streaming can be live [17], supporting live streaming requires much more changes of the original BitTorrent protocol [14], which may impose negative impact on the working efficiency of BitTorrent. Therefore, in this paper we mainly focus on ubiquitous BitTorrent-like VoD streaming.

Researchers have designed various schemes to implement BitTorrent-like media streaming services, such as BASS [13], BiToS [14], Toast [15] and Shah et al. [18]. Each scheme has its advantages and disadvantages, and in fact some schemes employ an identical mechanism or design component. The key problem lies in that we lack a comprehensive analytical framework, through which we can understand which scheme performs better in a specific scenario, especially in the scenario of ubiquitous BitTorrent-like media streaming.

To this end, we propose a quantitative analytical model for evaluating the QoS (quality-of-service) of ubiquitous BitTorrent-like media streaming in this paper. Our model captures the most common and fundamental mechanisms (including the *sliding-window based piece selection*, the *differentiated bandwidth competition among leechers*, etc.) adopted in various BitTorrent-like media streaming implementations, in order to better understand the key properties of QoS (such as the *playback continuity* and *startup/jump latency*) and provide helpful heuristics for optimizing the QoS of ubiquitous BitTorrent-like media streaming.

The remainder of this paper is organized as follows. Section II reviews the related work. In Section III we present a quantitative analytical model for evaluating the QoS of ubiquitous BitTorrent-like media streaming, as well as several helpful heuristics for optimizing the QoS. Finally, we conclude the paper and discuss about the *pros* and *cons* of our work in Section IV.

II. Related Work

To our knowledge, the earliest work on BitTorrent-like media streaming is BASS (BitTorrent-Assisted Streaming System) [13], which proposes a hybrid system architecture combining sequential piece download from streaming servers and random piece download from peers. A client downloads pieces randomly (in fact in a "rarest-first" way) from its neighboring peers, excluding those overdue pieces locating in front of the current playback point. Meanwhile, the client downloads pieces sequentially from streaming servers, skipping the pieces that have been downloaded or are being downloaded. In a word, BASS utilizes BitTorrent swarms as the assistance of streaming servers. BASS may not scale well except when streaming servers are powerful enough in upload capacity.

On the contrary, BiToS [14] utilizes streaming servers to assist BitTorrent swarms so as to improve the system scalability. In order to balance the timeliness and diversity of pieces, BiToS employs the sliding-window based piece selection strategy that restricts the original "rarest-first" piece selection strategy of BitTorrent within a sliding window. This sliding-window based strategy is simple but practical, requiring minor modification of the original BitTorrent protocol. Thus, it is widely adopted by many subsequent BitTorrent-like media streaming systems such as Toast (Torrent Assisted Streaming) [15], Shah et al. [18] and so forth.

The above two schemes can be unified into a more general model, that is to say, we should balance the *timeliness* of sequential piece selection and the *diversity* of random piece selection by providing "*differentiated request priorities*" to random pieces and sequential pieces. Then the problem becomes: how to assign the aforementioned request priorities? Through simulations, Carlsson et al. [19] point out that priority assignment should follow the Zipf distribution [20]. Specifically, the request priority for a piece should be inversely proportional to l^{θ}, where θ is a constant ($\theta > 1$) and the "urgent distance" $l =$ the sequential number of the piece $-$ the sequential number of the first missing piece in the sliding window.

The works [17] and [21] analyze the QoS of BitTorrent-like media streaming in theory. Tewari et al. [17] indicate that the working efficiency of live BitTorrent streaming depends on two parameters: peer group size and the number of pieces available for sharing. Besides, enabling live streaming on top of BitTorrent swarms requires significant changes of the original BitTorrent protocol. Parvez et al. [21] establish a comprehensive QoS model for BitTorrent-like VoD streaming. They focus on the effect of different piece selection strategies including rarest-first, in-order and their variants. Nevertheless, neither [17] nor [21] considers the sliding-window based piece selection strategy, the core mechanism adopted by most of today's real-world BitTorrent-like media streaming systems.

"private BitTorrent" (PT) and "bundling BitTorrent".

III. QoS Modeling and Optimization Heuristics

This section starts from a basic model of BitTorrent-like media streaming that clarifies the major notations and key mechanisms, followed by modelling of the playback continuity and the startup/jump latency.

A. Basic Model

Consider a BitTorrent swarm at time t containing N users in total: S seeds and L leechers. The leechers are further divided into L_s sequential leechers and L_r random leechers. Obviously, $N = S + L$ and $L = L_s + L_r$. This swarm shares a video file f consisting of M pieces, and the video playback rate of f is r. For convenience, r is normalized to 1.0 in our model.

As mentioned in Section II, the essential difference among various BitTorrent-like media streaming systems lies in the piece selection strategy. At present, the "*sliding-window based piece selection strategy*" is the most popular and thus our model also adopts this strategy. A sequential leecher only requests for the missing pieces in its sliding window (as shown in Fig. 1). The sliding window has a limited and usually constant size W ($W \ll M$), and its beginning always locates at the first missing piece. Inside the sliding window, piece selection depends on the rarity, urgency or some other metrics of a piece. It is impossible while unnecessary to model all these metrics; instead, we make a simple but reasonable assumption that piece selection inside the sliding window is approximately random [21]. On the contrary, a random leecher does not have a sliding window so it can request for any piece in f.

To guarantee a smooth startup of the video playback, a *startup cache window* is also necessary — the video playback cannot start until the startup cache window is filled up. The size of the startup cache window (W_s) is often set as a constant or proportional to the video size. The situation of the *jump cache window* (with the size W_j) is similar. Typically, W_s is set as the size of 10 seconds' video and $W_j = W_s$.

Due to the stringent requirements of sequential leechers on data transfer bandwidth and latency, BitTorrent-like media streaming systems usually modify the original peer connection (and bandwidth competition) strategy of BitTorrent [1]. First, the maximum number of peer connections is increased to larger than 5 in order to improve the data sharing level and downloading parallelism. Second, the original "tit-for-tat" peer unchocking strategy of BitTorrent is relaxed so that a newly joining sequential leecher can get a rapid playback startup — in our model, every seed or leecher fairly allocates its upload rate to the requesters, regardless of how much it gets from the requesters.

Regarding to a leecher i in the swarm, its download rate is denoted as D_i and its upload rate is U_i. The average download rate of all the users in the swarm is denoted as D and the average upload rate is U. For ease of reference, we list the basic notations used in our model as well as their brief descriptions in TABLE I.

TABLE I
BASIC NOTATIONS.

notation	**description** (in a BitTorrent-like swarm)
N	total number of users.
S	total number of seeds.
L	total number of leechers.
L_s	total number of sequential leechers.
L_r	total number of random leechers.
M	total number of pieces in the shared video file.
m	current sliding-window position of a sequential leecher.
W	size of the sliding window.
W_s	size of the startup cache window.
W_j	size of the jump cache window.
r	video playback rate.
$D_i(t), D_i(m)$	download rate of a leecher i at time t (or piece m).
D	average download rate of all the users.
U	average upload rate of all the users.

B. Modeling the Playback Continuity

Playback continuity is the most critical QoS property of Internet media streaming. For most users, discontinuous playback leads to much worse user experience than long startup/jump latency, low video resolution (equal to low video playback rate), and so forth. For a sequential leecher i, suppose its sliding window is locating between the m-th piece (*piece*$_m$) and the $(m+W-1)$-th piece (*piece*$_{m+W-1}$) at this moment. Now we evaluate how many users are expected to own at least one piece in the sliding window: {*piece*$_m$, *piece*$_{m+1}$, $\cdots$, *piece*$_{m+W-1}$}. These users are the potential data suppliers of the leecher i. Below, we calculate the expected data supply from seeds, sequential leechers and random leechers, respectively.

Data supply from seeds. There are S seeds in the swarm. A seed is the potential supplier of all leechers, and all the L leechers compete for its upload rate. Therefore, the leecher i is expected to obtain a share of upload rate $\frac{U}{L}$ from a seed, and the total upload rate got from all the S seeds will be:

$$U_s = \frac{U \cdot S}{L}. \tag{1}$$

Data supply from sequential leechers. There are L_s sequential leechers in the swarm. Generally speaking, their sliding-window positions should uniformly distribute across the M pieces. Then we can expect there are $N_m = L_s(1 - \frac{m-W}{M})$ sequential leechers that own at least one piece in the leecher i's sliding window: {*piece*$_m$, *piece*$_{m+1}$, $\cdots$, *piece*$_{m+W-1}$}. For one of the N_m sequential leechers (denoted as Leecher-1), suppose its current sliding window begins at *piece*$_{m+k}$ (k can be negative). All the L_r random leechers compete for its upload rate, and meanwhile there are $L_s \cdot \frac{m+k+W-1}{M}$ sequential leechers competing for its upload rate. The total number of competitors for Leecher-1's upload rate is $L_r + L_s \cdot \frac{m+k+W-1}{M}$. Therefore, the leecher i is expected to acquire a share of upload rate $\frac{U}{L_r + L_s \cdot \frac{m+k+W-1}{M}}$ from Leecher-1, and the total upload rate got from all the N_m sequential leechers will be:

$$U_{ls} = \sum_{k=1-W}^{M-m} \frac{L_s}{M} \cdot \frac{U}{L_r + L_s \cdot \frac{m+k+W-1}{M}}. \tag{2}$$

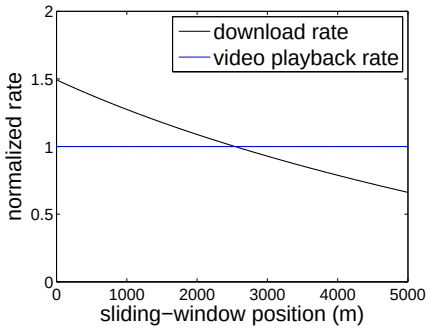

Fig. 2. Evolution of $D_i(m)$ as the sliding window moves forward.

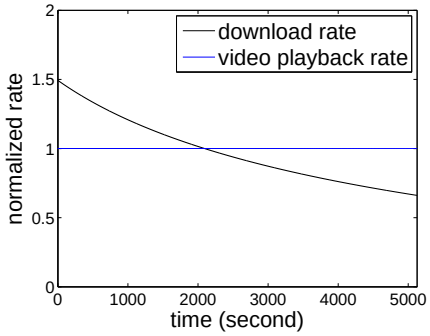

Fig. 3. Evolution of $D_i(t)$ as the time goes by.

Data supply from random leechers. There are L_r random leechers in the swarm. The available pieces in a random leecher should approximately uniformly distribute across the M pieces. For a random leecher that owns j pieces, it has a probability of $1-(1-\frac{W}{M})^j$ to own at least one piece in the leecher i's sliding window: $\{piece_m, piece_{m+1}, \cdots, piece_{m+W-1}\}$. Thus, we can expect there are $N_r = \sum_{j=1}^{M} \frac{L_r}{M} \cdot (1-(1-\frac{W}{M})^j)$ random leechers being the potential suppliers for the leecher i. For one of the N_r random leechers (denoted as Leecher-2), usually all the L leechers compete for its upload rate (the only exception is for a random leecher with very small j, but this should be rare). The leecher i is expected to acquire a share of upload rate $\frac{U}{L}$ from Leecher-2, and the total upload rate got from all the N_r random leechers will be:

$$U_{lr} = \frac{U \cdot N_r}{L}. \tag{3}$$

To sum up, there exist $S+N_m+N_r$ potential data suppliers for the sequential leecher i at this moment, and the total upload rate got from all the potential suppliers is expected to be:

$$D_i(m) = U_s + U_{ls} + U_{lr}. \tag{4}$$

As a result, we can draw the evolution curve of $D_i(m)$ as the beginning position of the sliding window (m) moves from 0 to M in Fig. 2. The parameters are configured as follows: $M = 5000$, $r = 1$, $W = 10$, $S = 10$, $L_s = 50$, $L_r = 50$, $L = L_s + L_r = 100$ and $U = 1.2$. They represent a common 5000-second video which is divided into 5000 pieces – one second, one piece. The video is shared among 10 seeds, 50 sequential leechers and 50 random leechers. The video playback rate r is normalized to 1 and the average upload rate of users U is normalized accordingly. The sliding window contains 10 pieces.

We can easily deduce the evolution curve of $D_i(t)$ as the time goes by from the evolution curve of $D_i(m)$ by using the following fact: when the beginning of the sliding window locates at $piece_m$, the time for the leecher i to download $piece_m$ is $\frac{1}{D_i(m)}$. Thus, the evolution curve of $D_i(t)$ corresponding to Fig. 2 is plotted in Fig. 3. The evolution behaviors of $D_i(m)$ and $D_i(t)$ are mainly consistent. As a whole, we can draw the following conclusions:

- During the time period from t_1 to t_2 ($t_1 < t_2$), the playback continuity of the leecher i can be statistically taken as:

$$\min\{\frac{W_{surplus} + \int_{t_1}^{t_2} D_i(t)\,\mathrm{d}t}{r(t_2 - t_1)}, 1\}, \tag{5}$$

  where $W_{surplus}$ is the surplus data volume at t_1. The above equation can be used to describe the playback continuity of the leecher i in a long enough period of time. However, it cannot tell the accurate playback continuity in a short period, because the specific playback continuity in a short period is tightly relevant to many unpredictable factors, such as the download order of adjacent pieces, network bandwidth fluctuation, and so forth. In practice, long-period playback continuity makes more sense than short-period playback continuity.
- For a common sequential leecher, usually its download rate declines as time goes by (i.e., as the sliding window moves forward), because a "late" piece usually owns fewer potential suppliers than an "early" piece. However, when the sliding window moves to late pieces and the download rate falls below the video playback rate, the playback continuity can still be very high, since a certain amount of surplus data volume has accumulated during the period when $D_i > r$. The surplus can offset the deficit when $D_i < r$ for some time.

The download rate evolution in Fig. 2 is unfit for VoD streaming, in particular for jump operations. About half of the jump operations ($m > 2500$) would suffer from discontinuous playback. Meanwhile, there exists considerable download rate surplus when $m < 2500$. Consequently, if we modify the allocation strategy of peers' upload rates, the download rate evolution curve in Fig. 2 will be less "steep". In another words, the download rate surplus when $m < 2500$ should be reduced to compensate the deficit when $m > 2500$, so that the overall playback continuity after all jump operations will be improved. Specifically, we can optimize the allocation strategy of peers' upload rates with the following heuristic:

- *Optimization Heuristic 1*: When several sequential leechers compete for the upload rate of the same user (no matter this user is a seed, a sequential leecher or a random

leecher), the requests for late pieces should be allocated with more upload rate.

Suppose a competing leecher is allocated with a share of upload rate u under the common (fair) allocation strategy, it will get a different share of upload rate $f(m) \cdot u$ when *Optimization Heuristic 1* is used. Here $f(m)$ must obey the following rules:

$$\begin{cases} f(1) < 1,\ f(M) > 1; \\ \text{For every } m < n,\ f(m) < f(n); \\ \frac{\sum_{m=1}^{M} f(m)}{M} = 1. \end{cases}$$

As a simple example, if we adopt a linear distribution of $f(m)$:

$$f(1) = \frac{1}{2},\ f(M) = \frac{3}{2};\quad f(m) = a \cdot m + b; \tag{6}$$

then the two constant parameters a, b should be:

$$a = \frac{1}{M-1},\quad b = \frac{1}{2} - \frac{1}{M-1}.$$

- *Optimization Heuristic 1'*: Alternatively, the system designers/administrators can manually deploy more duplicates for late pieces over the network.

Another possible optimization heuristic is to improve the download rate at every m, rather than to balance the download rates at small m and big m:

- *Optimization Heuristic 2*: When several leechers compete for the upload rate of the same user (no matter this user is a seed, a sequential leecher or a random leecher), the requests from sequential leechers should be allocated with more upload rate.

Nevertheless, this optimization method impairs the download rate of random leechers, so it must be used very cautiously.

C. Modeling the Startup/Jump latency

To guarantee a smooth startup/jump of the video playback, a startup/jump cache window is adopted by nearly all the popular Internet VoD systems. In fact, startup latency can be seen as a special case of jump latency when the jump position $m = 0$. Thus, in this subsection we only talk about jump latency.

Jump latency denotes the time for a sequential leecher to fill up its jump cache window. Here we do not consider the time for a user to obtain the required peer list and neighboring peers' "bitmaps" after he issues a jump request, because such time depends on specific system implementations. Suppose the size of the jump cache window is W_j. W_j is usually statically set as a constant, or $W_j = \alpha M$ where α is a constant ($\alpha \ll 1$). In the previous subsection, we have deduced the download rate $D_i(m)$. Since $W_j \ll M$, the download rate during the jump stage can be approximately taken as $D_i(m)$. m is the sequential number of the piece to which the user i jumps. Then the jump latency is around $\frac{W_j}{D_i(m)}$.

However, we observe that the usual statical setting of W_j is either larger or smaller than necessary in most cases, because it does not take the dynamics of download rate into consideration. So we suggest using a dynamic jump window setting strategy, in which W_j is dynamically set according to the download rate $D_i(m)$. It can be further divided into the following two strategies:

- *Pessimistic dynamic jump window setting strategy* assumes that pieces inside the sliding window are always downloaded in the worst order, as shown in Fig. 4. Here "worst" means the pieces inside the sliding window are downloaded in a descending order. The requirement of the pessimistic strategy is: after the current jump window is filled up, even if the pieces inside the subsequent k sliding windows (k is a small constant integer and usually we set $k = 1$) are downloaded in the worst order, continuous playback should be guaranteed. This requirement can be formalized as:
$$\begin{cases} \frac{W_j}{r} \geq \frac{W}{D_i(m)}, \\ \frac{W_j+W}{r} \geq \frac{2W}{D_i(m)}, \\ \cdots, \\ \frac{W_j+(k-1)W}{r} \geq \frac{kW}{D_i(m)}. \end{cases}$$
- *Optimistic dynamic jump window setting strategy* assumes that the pieces inside the sliding window are downloaded in a random order. Inside the sliding window, suppose the number of downloaded *consecutive* pieces from the beginning of the sliding window is n. Consider the evolution of n in the subsequent sliding window after the jump window is filled up. Parvez et al. [21] have proved that following the optimistic strategy, the growth rate of n gradually increases as the time goes by. When the growth rate ($\frac{\mathrm{d}n}{\mathrm{d}p}$) exceeds the video playback rate (r), we can expect continuous playback. Here p is the number of downloaded pieces in the sliding window. According to [21], the expected value of n is $n = \frac{p}{W-p+1}$, and the growth rate is $\frac{\mathrm{d}n}{\mathrm{d}p} = \frac{W+1}{(W-p+1)^2}$. Let $\frac{\mathrm{d}n}{\mathrm{d}p} = r$, we get $p = W + 1 - \sqrt{\frac{W+1}{r}}$. Thus, the requirement on W_j can be formalized as:
$$\begin{cases} \frac{W_j}{r} \geq \frac{W+1-\sqrt{\frac{W+1}{r}}}{D_i(m)}, \\ \frac{W_j+W}{r} \geq \frac{2W+1-\sqrt{\frac{W+1}{r}}}{D_i(m)}, \\ \cdots, \\ \frac{W_j+(k-1)W}{r} \geq \frac{kW+1-\sqrt{\frac{W+1}{r}}}{D_i(m)}. \end{cases}$$

Based on the above analysis, we can get an optimization heuristic for the startup/jump latency:

- *Optimization Heuristic 3*: Since the pieces inside a sliding window are mostly downloaded in a random order, we suggest using the optimistic dynamic jump window setting strategy to set the size of the jump window.

IV. Conclusion and Discussions

P2P file sharing and P2P media streaming have presented a trend of convergence in recent years. As the representative case, ubiquitous BitTorrent-like media streaming is worth a preliminary investigation into its feasibility and influence. Thereby, in this paper we propose a quantitative analytical model for evaluating the QoS (such as playback continuity and startup/jump latency) of ubiquitous BitTorrent-like media

Fig. 4. Worst download order of pieces. T denotes a constant period of time and the blue pieces have been obtained.

streaming. Our model also provides helpful heuristics for optimizing the QoS.

Now we discuss about the *pros* and *cons* of our model. First, although our model clarifies the statistical relationship between the QoS and the main parameters of a ubiquitous BitTorrent-like media streaming system, it cannot describe the accurate QoS in a short period of time or at a time point, because the specific QoS in a short period of time or at a time point depends on many unpredictable factors. For the designer or administrator of a system, the statistical QoS in a long period is generally more substantial and valuable.

Second, our model does not describe the join and leave behaviors of peers. Instead, we assume that a server monitors the scales of the seeds, sequential leechers and random leechers in each swarm. As a matter of fact, this assumption is practically used in popular real-world streaming systems like the large-scale PPLive VoD system [7]. Finally, our model and its proposed optimization heuristics still need to be checked by using real-world measurements.

REFERENCES

[1] B. Cohen. "Incentives build robustness in BitTorrent," Proceedings of the 1st Workshop on Economics of Peer-to-Peer systems (P2PEcon), Berkeley, CA, June 5 - 6, 2003.

[2] M. Ripeanu. "Peer-to-peer architecture case study: Gnutella network," Proceedings of the 1st International Conference on Peer-to-Peer Computing (P2P), Linkoping, Sweden, August 27 - 29, 2001.

[3] K. Tutschku. "A measurement-based traffic profile of the eDonkey filesharing service," Proceedings of the 5th International Workshop on Passive and Active Network Measurement (PAM), Antibes Juan-les-Pins, France, April 19 - 20, 2004.

[4] N. Leibowitz, M. Ripeanu, and A. Wierzbicki. "Deconstructing the KaZaa network," Proceedings of the 3rd IEEE Workshop on Internet Applications (WIAPP), San Jose, CA, June 23 - 24, 2003.

[5] X. Zhang, J. Liu, B. Li, and T. Yum. "CoolStreaming/DONet: A data-driven overlay network for efficient live media streaming," Proceedings of the 24th IEEE International Conference on Computer Communications (INFOCOM), Miami, FL, March 13 - 17, 2005.

[6] S. Baset, and H. Schulzrinne. "An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol," Proceedings of the 25th IEEE International Conference on Computer Communications (INFOCOM), Barcelona, Spain, April 23 - 29, 2006.

[7] Y. Huang, T. Fu, D.M. Chiu, J. Lui, and C. Huang, "Challenges, design and analysis of a large-scale p2p-vod system," Proceedings of the ACM SIGCOMM 2008 Conference on Data communication (SIGCOMM), Seattle, WA, August 17 - 22, 2008.

[8] Z. Liu, C. Wu, B. Li, and S. Zhao. "UUSee: Large-Scale Operational On-Demand Streaming with Random Network Coding," Proceedings of the 29th IEEE International Conference on Computer Communications (INFOCOM), Riverside, CA, March 14 - 19, 2010.

[9] BitComet web site. http://www.bitcomet.com.

[10] P. Dhungel, K. Ross, M. Steiner, Y. Tian, and X. Hei. "Xunlei: Peer-Assisted Download Acceleration on a Massive Scale," Proceedings of the 13th Passive and Active Measurement conference (PAM), Vienna, Austria, March 12 - 14, 2012, pp. 231 - 241.

[11] Z. Li, Y. Huang, G. Liu, F. Wang, Y. Liu, Z.-L. Zhang, and Y. Dai. "Challenges, Designs and Performances of Large-scale Open-P2SP Content Distribution," Preprint, IEEE Transactions on Parallel and Distributed Systems (TPDS), 2012, DOI Bookmark: http://doi.ieeecomputersociety.org/10.1109/TPDS.2012.252.

[12] Funshion web site. http://www.funshion.com.

[13] C. Dana, D. Li, and D. Harrison, "BASS: BitTorrent assisted streaming system for video-on-demand," Proceedings of the 7th International Workshop on Multimedia Signal Processing (MMSP), Shanghai, China, October 30 - November 2, 2005.

[14] A. Vlavianos, M. Iliofotou, and M. Faloutsos, "BiToS: Enhancing BitTorrent for supporting streaming applications," Proceedings of the 25th IEEE International Conference on Computer Communications (INFOCOM), Barcelona, Spain, April 23 - 29, 2006, pp. 1 - 6.

[15] Y.R. Choe, D.L. Schuff, J.M. Dyaberi, and V.S. Pai, "Improving VoD server efficiency with bittorrent," Proceedings of the 15th ACM International Conference on Multimedia (MM), Augsburg, Germany, September 24 - 29, 2007.

[16] A. Legout, G. Urvoy-Keller, and P. Michiardi. "Rarest first and choke algorithms are enough," Proceedings of the ACM Internet Measurement Conference (IMC), Rio de Janeiro, Brazil, October 25 - 27, 2006, pp. 203 - 216.

[17] S. Tewari, and L. Kleinrock, "Analytical Model for BitTorrent-Based Live Video Streaming," Proceedings of the 4th IEEE Consumer Communications and Networking Conference (CCNC), Las Vegas, NV, January 11 - 13, 2007, pp. 976 - 980.

[18] P. Shah, and J.F. Paris, "Peer-to-Peer Multimedia Streaming Using BitTorrent," Proceedings of the 26th IEEE Internationa Performance, Computing, and Communications Conference (IPCCC), New Orleans, Louisiana, April 11 - 13, 2007, pp. 340 - 347.

[19] N. Carlsson, and D.L. Eager, "Peer-assisted on-demand streaming of stored media using BitTorrent-like protocols," Proceedings of the IFIP-TC6 conference on Ad Hoc and sensor networks, wireless networks, next generation internet (Networking), Atlanta, Georgia, May 14 - 18, 2007, pp. 570 - 581.

[20] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and Zipf-like distributions: Evidence and implications," Proceedings of the 18th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM), New York, NY, March 21 - 25, 1999, pp. 126 - 134.

[21] N. Parvez, C. Williamson, A. Mahanti, and N. Carlsson, "Analysis of bittorrent-like protocols for on-demand stored media streaming," Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS), Annapolis, Maryland, June 2 - 6, 2008, pp. 301 - 312.