Prefetching Optimization in P2P VoD Applications

Yifeng He and Ling Guan
Department of Electrical and Computer Engineering,
Ryerson University, Toronto, Canada

Abstract

Most existing Peer-to-Peer (P2P) Video-on-Demand (VoD) systems have been designed and optimized for the sequential playback. In practice, users often want to seek to the positions they are interested in. Such frequent seeks raise a great challenge to the design of the prefetching scheme. In this paper, we propose an optimal prefetching scheme to minimize the expected seeking delay at every viewing position. The proposed prefetching scheme optimally determines which segments will be prefetched and cached, based on the segment access probability. Through extensive simulations, we demonstrate that the proposed prefetching scheme significantly reduces the seeking delay compared to the sequential prefetching scheme.

Index Terms

Peer-to-Peer (P2P) Video-on-Demand (VoD), seeking delay, prefetching scheme, optimization.

1. Introduction

Recently Peer-to-Peer (P2P) technology has become a promising approach to provide Video-on-Demand (VoD) service [1][2] to a huge number of the users over a global area. Most of the existing work on P2P-based VoD systems [3][4] has made an implicit assumption that a user who has joined a streaming session would keep on watching till it leaves or fails the session. This assumption excludes flexible Video Cassette Recording (VCR) functionalities from VoD systems to make the system design simple. Unfortunately, based on the analysis of a large amount of real user viewing logs, we found that users usually do not play the video successively and passively. Instead, users perform seeks quite frequently [5]. The frequent seeking behaviors have also been observed in GridCast [6], a real P2P VoD system deployed in China since 2006. The reasons for these seeking behaviors are: 1) some users may feel that the current segment is boring, so they jump away from it; 2) some users may not have sufficient time to watch the whole video and just want to browse the exciting or interesting segments.

In the P2P VoD applications, the frequent seeks pose a great challenge on the playback continuity, which can be measured with the seeking delay, defined as the interval between the time when a segment is requested and the time when the segment is ready for playback. Ideally, we would like to have a zero seeking delay, such that we can view any segment in the video anytime without any interruption. To achieve a zero seeking delay, each segment has to be fetched prior to its playback time. Prefetching scheme prefetches one or multiple segments while the current segment is being played. It turns out to increase the playback continuity [7]. Different prefetching schemes have been designed in P2P streaming applications. Sharma et al [7] have proposed a distributed prefetching protocol where peers prefetch and store portions of the streaming media ahead of their playback time. Shen et al [8] have examined a prefetching scheme in P2P VoD applications such that the client peer can tap the reservoir of the prefetched bits while searching for a replacement serving peer. However, these prefetching schemes assume that the peers do the sequential playback without any seek. In the VoD applications with random seeks, the next position that the peer will access may not be the segment next to the current segment, it will be probably any segment in the whole video. Therefore, the sequential prefetching scheme does not work well if the peers perform seeks frequently. The prefetching scheme in P2P VoD systems with frequent seeks needs to predict the access probability and then prefetch the appropriate segments.

The prefetching scheme considering user seeking behaviors is discussed in [5], in which an optimal prefetching scheme is developed to minimize the expected seeking distance. However, the prefetching optimization in [5] is based on a one-dimensional segment popularity $P(y)$, in which the popularity of segment $y$ is determined by the number of accesses to that segment. The one-dimensional segment popularity can not accurately capture the user seeking behaviors since it neglects the start position of the seek.

Different from previous work [7][8][9], we propose the concept of guided seeks. A guided seek is different from a random seek in that it is performed based on the segment access information, which is learned from the seeking statistics in the previous and/or concurrent sessions. Typically, the popular segments are visited more frequently, thus the access count of a segment approximately reflects its popularity. With guidance, users can jump to the desired positions in the video quickly and efficiently. For example, a user watching
Figure 1. The proposed framework for P2P VoD applications with guided seeks

We have proposed an optimal prefetching framework in P2P VoD applications with guided seeks. The proposed framework is shown in Fig. 1. It consists of two modules: the seeking statistics aggregation and the optimal prefetching scheme. In the seeking statistics aggregation, we represent the seeking statistics with the FM sketches [10] to prevent the message size from growing linearly. The peer retrieves a set of random neighbors from the tracker, and then gossips the hybrid sketches with them. Based on the aggregated seeking statistics, the peer estimates the segment access probability \( P(x, y) \), which is used for the design of the optimal prefetching scheme, and is also used to guide the user to perform efficient seeks. The module of the optimal prefetching scheme takes the segment access probability \( P(x, y) \) as input, and then determines the optimal segments for prefetching and the optimal cache replacement policy.

The module of the seeking statistics aggregation is discussed in our previous work [11]. The purpose of the module is to estimate the segment access probability \( P(x, y) \) using the seeking statistics of the peers who have watched the same video. In a centralized VoD system, a server can be used to collect the seeking statistics. However, it overloads the burden of the server and introduces a single point of failure. Nevertheless, collecting the seeking statistics in distributed P2P networks is a nontrivial task. There are two major challenges: 1) the message size grows linearly with the number of the collected seeking statistics; 2) the duplicate statistics cause an inaccuracy in the estimation of the segment access probability. Our previous work in [11] addressed the challenges by employing FM sketches [10] to represent the seeking statistics. Each peer aggregates the FM sketches by gossiping with the neighbors, and then estimates the segment access probability \( P(x, y) \), which is output to the module of the optimal prefetching scheme.

This paper focuses on the module of the optimal prefetching scheme in the proposed framework. Our contribution in this paper is that we optimize the prefetching scheme to minimize the expected seeking delay at each viewing position based on the segment access probability.

The remainder of this paper is organized as follows. In Section 2, we optimize the prefetching scheme to minimize the expected seeking delay. We present the simulation results in Section 3 and conclude the paper in Section 4.

2. Optimization of Prefetching Scheme

In this section, we first optimize the prefetched segments, and then optimize the cache replacement.

2.1. Optimize Prefetched Segments

The segment access probability is estimated from the seeking statistics of the peers [11]. By taking the segment access probability \( P(x, y) \) as input, we can design the optimal prefetching scheme.

Suppose that the peer is watching segment \( i \) currently. After finish viewing segment \( i \), this peer may seek to any other position. In order for the continuous playback, it can prefetch the appropriate segments during the viewing time.

The video has a Constant Bit Rate (CBR) of \( r_s \), and the video is divided into \( M \) segments. The viewing time of a segment is \( T \). Each segment has a fixed size \( s_{seg} = r_s T \). A segment is played right after it has been completely downloaded at this peer. We define a whole set \( Y \) as the set of all the segments in the video. The prefetching process when the peer is viewing segment \( i \) is illustrated in Fig. 2. The playback time of segment \( i \) is denoted by \( t_i \), \( \forall i \in Y \). At \( t_i \), segment \( i \) is ready for playback. However, the peer has been prefetching some other segments, which are useful for the future playback. The peer completes these ongoing downloads before it starts the next-round prefetching. So there is a gap between the prefetching time and the playback time of segment \( i \), which is called prefetching shift \( \theta_i \). After the peer finishes viewing segment \( i \), it will request a new segment (e.g., segment \( j \)) at time \( (t_i + T) \). The interval between the request time and the playback time of segment
Figure 2. Determine the prefetching scheme when segment \( i \) is being watched

\( j \) is denoted as \textit{seeking delay} \( \tau_j \) of segment \( j \). So the seeking delay is given by \( \tau_j = t_j - (t_i + T) \).

Each peer maintains a cache set \( B \) in its buffer, the downloaded segments are put into the cache set. The cached segments help to reduce the seeking delay. If all the segments are cached, the seeking delay is always zero no matter where the peer jumps. Moreover, the cached segments can be served to the other peers. In P2P applications, each peer is only willing to contribute a limited buffer resource. Therefore a peer can only cache a certain amount of the segments. Let \( \xi \) denote the maximum number of the segments in the cache set. The cache set is dynamically updated due to the limited buffer capacity. The cache replacement policy is completed by every prefetching time. We will describe the cache replacement policy in the following subsection.

The cache set during the interval from the prefetching time \((t_i + \theta_i)\) of segment \( i \) to the prefetching time of the next segment \((t_j + \theta_j)\) is denoted as \( B_i \). The unavailable set \( U \) contains the segments that are not in the cache set. The unavailable set is also varying with time. The unavailable set during the interval from the prefetching time \((t_i + \theta_i)\) of segment \( i \) to the prefetching time of the next segment \((t_j + \theta_j)\) is denoted as \( U_i \). So we have \( Y = B_i + U_i, \forall i \in Y \).

The download capacity of the receiving peer is denoted by \( b_d \). Each prefetched segment will be downloaded from the serving peers who are caching this segment. The receiving peer can retrieve the desired segment by downloading it from one or multiple serving peers. The upload capacity of segment \( i \) is denoted as \( o_i \). Due to peer dynamics, the upload capacity of a segment varies with time. The receiving peer needs to estimate the upload capacity of each unavailable segment periodically. In P2P VoD applications, the upload capacity of segment \( i \) is typically no larger than the download capacity, that is \( o_i \leq b_d, \forall i \in Y \).

The purpose of the prefetching scheme is to reduce the seeking delay. In the case without prefetching, the peer starts to download the requested segment after the seeking request is generated. Suppose that segment \( i \) is being watched, the expected seeking delay of the next requested segment \( (segment \ j) \) without prefetching is given by \( E(\tau_j) = \sum_{j \in U_i} P(j|i)s_{seg}/o_j \). If we prefetch some segments before the request time, the expected seeking delay can be reduced since some segments in the unavailable set have been downloaded completely or partially before the request time. Prefetching is performed between the prefetching time of segment \( i \) and the request time of segment \( j \), this interval is defined as \textit{prefetching interval}. All the segments in the unavailable set are the prefetching candidates. We allocate each of them a prefetching rate \( r_k \in U_i \). In P2P applications, the bandwidth bottleneck typically occurs at the access links, therefore we have \( r_k \leq o_k \)). We have assumed that \( o_k \leq b_d \), so the constraint \( r_k \leq o_k \) covers both constraints. The minimum required rate to completely download a segment during the prefetching interval is given by \( r_m = s_{seg}/(T - \theta_i) \). In order to fully utilize the download bandwidth, we limit the prefetching rate to no larger than \( r_m \), which is \( r_k \leq r_m \). Those segments allocated with a rate of \( r_m \) will be prefetched completely by the request time \((t_i + T)\) of segment \( j \), while those allocated with a rate less than \( r_m \) will be prefetched partially by the request time.

In the prefetching scheme, if the current segment is segment \( i \), we can calculate the expected seeking delay considering two different cases as follows.

1) If the requested segment \( j \) is in the cache set \( B_i \), the seeking delay of segment \( j \) will be 0. The probability of this case is \( P_1 = \sum_{k \in B_i} P(k|i) \).

2) If the requested segment \( j \) is in the unavailable set \( U_i \), the prefetched part of segment \( j \) during the prefetching interval is \( \tau_j(T - \theta_i) \), the remaining part of segment \( j \) will be downloaded at the upload capacity \( o_j \). Therefore, the seeking delay of segment \( j \) is given by \( \tau_j = s_{seg}/o_j \).

Then, the expected seeking delay is given by

\[
E(\tau_j) = \sum_{k \in U_i} \frac{P(k|i) s_{seg}/o_k}{o_k} \mu_k + \sum_{k \in B_i} \frac{P(k|i) r_k}{o_k} + \sum_{k \in B_i} \frac{P(k|i) s_{seg}/o_k}{o_k} \mu_k - \sum_{k \in U_i} \frac{P(k|i) r_k}{o_k} \mu_k.
\]

(1)

The first term \( \sum_{k \in Y} \frac{P(k|i) s_{seg}/o_k}{o_k} \mu_k \) in equation (1) represents the expected seeking delay without caching and prefetching. The second term \( \sum_{k \in B_i} \frac{P(k|i) r_k}{o_k} \mu_k \) represents the reduction of the expected seeking delay due to the cache set \( B_i \). The third term \( \sum_{k \in U_i} \frac{P(k|i) s_{seg}/o_k}{o_k} \mu_k \) represents the reduction of the expected seeking delay caused by the prefetched part of the segments in the unavailable set \( U_i \).

The \textit{prefetching optimization problem} is to minimize the expected seeking delay by optimally allocating the prefetching rate of each segment. The optimal allocation is given by

\[
\text{minimize} \quad E(\tau) \quad \text{s.t.} \quad \sum_{i \in Y} r_i \leq b_d, \quad \text{and} \quad \sum_{i \in Y} o_i \leq \sum_{i \in Y} s_{seg}.
\]
ing rate for each of the segments in the unavailable set. When the current segment is segment \(i\), the optimization problem can be mathematically formulated as follows.

\[
\text{minimize } E(r_j) = \sum_{k \in U_j} \frac{P(k|i)}{\theta_k} r_k \sum_{k \in B} \frac{P(k|i)}{\theta_k} \frac{r_k}{\theta_k} - \sum_{k \in U_j} \frac{P(k|i)}{\theta_k} r_k (T - \theta_k)
\]

subject to \(0 \leq r_k \leq \min\{o_k, r_m\}, \forall k \in U_i, \sum_{k \in U_i} r_k \leq b_d, \)

(2)

where \(r_k\) is the prefetching rate of segment \(k\) in the unavailable set \(U_i\), \(o_k\) is the upload capacity of segment \(k\), and \(r_m\) is the minimum rate to complete the download of a segment within the prefetching interval, and \(b_d\) is the download capacity of the receiving peer.

We define an Access-probability to Upload-capacity Ratio (AUR) as \(q_{k|i} = P(k|i)/o_k\), then the prefetching optimization problem is equivalent to the following:

\[
\text{maximize } (T - \theta_i) \sum_{k \in U_i} q_{k|i} r_k
\]

subject to \(0 \leq r_k \leq \min\{o_k, r_m\}, \forall k \in U_i, \sum_{k \in U_i} r_k \leq b_d, \)

(3)

The above problem is a Linear Programming (LP) problem, which can be solved efficiently using simplex method or interior point method [12]. By solving problem (3), we obtain the optimal prefetching rate for each segment in the unavailable set \(U_i\). If a segment is allocated with a zero prefetching rate, it will not be scheduled for prefetching.

Carefully observing problem (3), we find that the objective function is a weighted sum of the prefetching rate of each segment in the unavailable set. In order to maximize the objective function, the segment that has a larger AUR \(q_{k|i}\) should be allocated with a larger rate within the constrained range. With this in mind, we derive a greedy rate allocation algorithm as follows.

1) Sort the segments in the unavailable set with respect to AUR in descending order;
2) Starting from the segment with the largest value of AUR, allocate as large prefetching rate as possible within the constrained range to each segment in the unavailable set until the download capacity is used up.

**Theorem 1:** The rates allocated with the proposed greedy algorithm are the optimal solution to problem (3).

**Proof:**

Let \(\chi_k = \min\{o_k, r_m\}\). In the proposed greedy algorithm, the unavailable set \(U_i\) consists of three subsets: 1) \(U^1_i\), in which \(r_k = \chi_k, \forall k \in U^1_i\); 2) \(U^2_i\), in which \(0 \leq r_k < \chi_k, \forall k \in U^2_i\), there is only one segment \(k\) in subset \(U^2_i\) with prefetching rate \(r_k = b_d - \sum_{k \in U^1_i} \chi_k\); and 3) \(U^3_i\), in which \(r_k = 0, \forall k \in U^3_i\). In the proposed algorithm, the following relations hold: 1) \(U_i = U^1_i + U^2_i + U^3_i\), 2) \(\sum_{k \in U^1_i} \chi_k + r_k = b_d, 3) q_{k|i} \geq q_{k|i}, \forall k \in U^1_i, \) and 4) \(q_{k|i} \geq q_{k|i}, \forall k \in U^3_i\).

We compare the objective value \(f_a\) of problem (3) under the proposed greedy algorithm with the objective value \(f_a\) under any other allocation scheme. In the proposed algorithm, \(f_a = (T - \theta_i) (\sum_{k \in U^1_i} q_{k|i} \chi_k + q_{k|i} r_k)\). In the other allocation scheme, \(f_a = (T - \theta_i) (\sum_{k \in U^1_i} q_{k|i} \chi_k + \sum_{k \in U^2_i} q_{k|i} r_k + \sum_{k \in U^3_i} r_k).

Then \(f_a - f_a = (T - \theta_i) (\sum_{k \in U^1_i} q_{k|i} \chi_k + q_{k|i} r_k - (\sum_{k \in U^1_i} q_{k|i} \chi_k + \sum_{k \in U^2_i} q_{k|i} r_k + \sum_{k \in U^3_i} r_k)) = (T - \theta_i) (\sum_{k \in U^1_i} q_{k|i} \chi_k + q_{k|i} r_k - (\sum_{k \in U^1_i} q_{k|i} \chi_k + \sum_{k \in U^3_i} r_k)).\)

After some algebraic manipulation:

\[
q_{k|i} \geq q_{k|i}, \forall k \in U^1_i, \text{ and } q_{k|i} \geq q_{k|i}, \forall k \in U^3_i.
\]

Therefore the rates allocated with the proposed greedy algorithm maximize the objective function of problem (3). In other words, the rates allocated with the proposed greedy algorithm are the optimal solution to problem (3).

We define a prefetched set \(F_i\) containing those segments allocated with a positive prefetching rate. In the prefetched set \(F_i\), we categorize those segments allocated with a rate of \(r_m\) into a complete set \(A_i\), and the other segments with a rate less than \(r_m\) into an ongoing set \(G_i\). We have \(F_i = A_i + G_i\). The optimal rate allocation algorithm determines which segments will be prefetched and what is the rate for each prefetched segment. The unavailable set is varying with time, hence the optimal prefetching scheme is performed at every prefetching time.

2.2. Optimize Cache Replacement

At the request time \((t_i + T)\), the peer requests the next segment (e.g., segment \(j\)). At this time, the peer determines the new cache set \(B_j\). The construction of the new cache set can be formulated as an optimization problem: to maximize the reduction of the expected seeking delay given that the current position is segment \(j\) by choosing the optimal segments from the old cache set \(B_i\) and the prefetched set \(F_i\), subject to the constraint of the cache capacity \(\xi\).

Mathematically, it is formulated as follows.

\[
\text{maximize } \sum_{k \in B_j} \frac{P(k|i) \sum_{k \in B_j} \chi_k}{\theta_k}
\]

subject to \(|B_j| \leq \xi, B_j \subseteq (B_i + F_i), \)

(4)

where \(B_j\) is the new cache set, \(|B_j|\) denotes the number of the segments in cache set \(B_j\), \(B_i\) is the old cache set and \(F_i\) is the prefetched set when segment \(i\) is being watched.

We can find the optimal solution to problem (4) by choosing \(\xi\) segments that have the largest value of AUR \(q_{k|i}\) from the union set of \(B_i\) and \(F_i\) to constitute the new cache set \(B_j\). Comparing the new cache set \(B_j\) with the old cache set \(B_i\), we know which segments should be ejected from the new cache set, and which segments will be filled into it. Therefore the optimal solution to problem (4) actually provides the optimal cache replacement policy. If the newly
3. Simulations

In the simulations, we choose a video clip of 30 minutes. The video is evenly divided into 90 segments. Each segment has a fixed duration of 20 seconds. We randomly generate the seeking behaviors of 1000 sessions as follows. At the current segment $i$, a peer will seek forward (including sequential playback) with a forward probability uniformly distributed between 0.75 and 0.95. Otherwise, it will seek backward. The popularity of the segments in the forward set or backward set follows a Zipf distribution with a shape parameter uniformly distributed between 0.7 and 1.4. The segment with a higher popularity is more likely to be chosen as the destination segment. The peers participating in this video session collect the seeking statistics from the 1000 sessions and estimate the segment access probability $P(x, y)$, which is used to optimize the prefetching scheme. In the default setting, the cache capacity is 15 segments, the upload capacity for each segment is uniformly distributed between 1000 Kbps and 1400 Kbps, and the download capacity at the receiving peer is 1600 Kbps. The playback rate of the video is 500 Kbps.

We compare the seeking delay in two prefetching schemes: 1) the proposed prefetching scheme, as described in Section 2; and 2) the sequential prefetching scheme, in which the peer prefetches the segments next to the current segment sequentially, and the segments closest to the current viewing position are cached in priority.

The seeking behaviors can be classified into the sequential playback from the current segment $x$ to the next segment $(x + 1)$, and the non-sequential seeks from the current segment $x$ to segment $y$ while $y \neq x + 1$. In the conditional access probability at current segment $x$, the next segment $(x + 1)$ always has the highest probability among all the segments. Therefore the next segment $(x + 1)$ will be prefetched prior to the other segments, which guarantees the continuity of the sequential playback. We focus on the examination of the seeking delay incurred by the non-sequential seeks. We define the average seeking delay as the average of the seeking delays of the non-sequential seeks in a session. Fig. 3(a) shows the average seeking delays in 100 sessions. The proposed scheme reduces the average seeking delay by 4.0 seconds in average over the sequential prefetching scheme. We define the playback continuity as 

$$c_p = \frac{T_{play}}{(T_{play}+T_{seek})},$$

where $T_{play}$ is the total playback time, and $T_{seek}$ is the total seeking delay. If all the seeking delays are 0, the playback continuity will be 1, indicating a fluent playback without any gap. Fig. 3(b) shows the comparison of the average playback continuity among 100 sessions between the proposed prefetching scheme and the sequential prefetching scheme. The proposed prefetching scheme improves the playback continuity compared to the sequential prefetching scheme because the proposed scheme reduces the total seeking delay $T_{seek}$. This improvement is more significant when the number of non-sequential seeks is increased.

The seeking delay is dependent on both the upload capacity and download capacity. In the P2P VoD applications, the segment is served by one or multiple serving peers who are buffering it. Therefore the upload capacity of a segment is different from segment to segment. We vary the average upload capacity from 500 Kbps to 1600 Kbps, and compare the average seeking delay between two prefetching schemes in Fig. 4(a). When the upload capacity is increased, the average seeking delay is reduced in both schemes. The proposed prefetching scheme outperforms the sequential prefetching scheme by 4.88 seconds in average. A larger download capacity allows the peer to prefetch...
more segments, thus reducing the seeking delay. As seen
in Fig. 4(b), the seeking delay in both schemes is reduced
when the download capacity is increased from 800 Kbps
to 2400 Kbps. Among different download capacity, the
average seeking delay in the proposed prefetching scheme
is 2.61 - 4.32 seconds smaller than that in the sequential
prefetching scheme. Fig. 4(b) also shows a tradeoff between
the average seeking delay and the prefetching bandwidth
usage. The playback rate of the video is 500 kbps. When the
download bandwidth is 800 kbps, the user spends 300 kbps
in prefetching extra segments, achieving an average seeking
delay of 4.90 seconds. When the download bandwidth is
2400 kbps, the user can spend 1900 kbps in prefetching extra
segments, thus obtaining a much smaller average seeking
delay of 2.59 seconds.

In practice, some users may not follow the guidance. In
order to study the performance under various user behaviors,
we introduce a follow factor, defined as the ratio between
the number of the seeks following the guidance and the
total number of seeks. A follow factor of 1 means that all
the seeks follow the guidance, while a follow factor of 0
means that all of the seeks are random, totally ignoring
the guidance. We evaluate the average playback continuity
among 100 sessions, in which the average number of non-
sequential seeks is 20. As shown in Fig. 5, the average
playback continuity in the two prefetching schemes are
almost the same when all the seeks are random (follow
factor is 0). When the number of the seeks following
guidance is increased, the average playback continuity in the
proposed prefetching scheme is increased greatly, reaching
0.942 when the follow factor is 1. On the other hand, the
average playback continuity in the sequential prefetching
scheme remains basically at the same level regardless of the
variation of the follow factor.

4. Conclusion

In P2P VoD applications, users may frequently seek to
the positions they are interested in. The frequent seeks raise
a great challenge to the playback continuity. Prefetching
scheme can increases the playback continuity by fetching
one or multiple future segments prior to the playback. This
paper proposes an optimal prefetching scheme, in which we
optimize the prefetched segments and the cache replacement
based on the segment access probability to minimize the
expected seeking delay in P2P VoD applications. The simu-
lations results demonstrate that the proposed prefetching
scheme can significantly reduce the seeking delay compared
to the sequential prefetching scheme.

References

Fault Tolerant Video-on-Demand Streaming in Peer-to-Peer
Environment,” Proc. of IEEE ICC, vol. 25, no. 1, pp. 119-130,
for Low-Cost On-Demand P2P Media Streaming,” Proc. of
QShine, Aug. 2006.
Streaming Solution Based On A Receiver-Driven Streaming
“PROMISE: Peer-to-Peer Media Streaming Using Collect-
for Random Seek Support in Peer-to-Peer Streaming Ap-
of a peer-to-peer video-on-demand system,” in Proc. of IPTPS,
Feb. 2007.
prefetching protocol for scalable asynchronous multicast in
P2P systems,” Proc. of IEEE INFOCOM, vol. 2, pp. 1139-
the Design of Prefetching Strategies in a Peer-Driven Video-
2006.
storage for peer-to-peer interactive video streaming,” IEEE
Journal on Selected Areas in Communications, vol. 25, no.
[10] P. Flajolet and G. N. Martin, “Probabilistic counting algo-
rithms for data base applications,” Journal of Computer and
prefetching scheme for P2P VoD applications with frequent
[12] R. J. Vanderbei, Linear programming: foundations and exten-