Exploring the Optimal Chunk Selection Policy for Data-Driven P2P Streaming Systems

Bridge Q. Zhao* John C.S. Lui* Dah-Ming Chiu+
*Computer Science & Eng. Department +Information Eng. Department
The Chinese University of Hong Kong, Shatin, N.T., Hong Kong
{qzhao, cslui} cse.cuhk.edu.hk dmchiu@ie.cuhk.edu.hk

Abstract

Data-driven P2P streaming systems can potentially provide good playback rate to a large number of viewers. One important design problem in such P2P systems is to determine the optimal chunk selection policy that provides high continuity playback under the server's upload capacity constraint. We present a general and unified mathematical framework to analyze a large class of chunk selection policies. The analytical framework is asymptotically exact when the number of viewers is large. More importantly, we provide some interesting observations on the optimal chunk selection policy: it is of $\vee$*-shaped and becomes more greedy as the upload capacity of the server increases. This insight helps content providers to deploy large scale streaming systems with a QoS-guarantee under a given cost constraint.*

1. Introduction

Video streaming is part of the basic service that we expect in the current Internet. There has been number of studies on how to provide streaming service using the client-server architecture and how to engineer streaming servers so as to provide the quality-of-service guarantees [1]. In recent years, the attention is on how to provide a *scalable* streaming service to a large number of viewers. To this end, IP multicast was proposed so that the server only needs to send a copy of video file and routers along the distribution network will relay all packets to different end users. However, due to security and deployment issues [2], IP multicast has not been widely deployed. Instead, people are using application layer multicast to deliver the video files to users.

Peer-to-peer (P2P) system is considered one form of application layer multicast. In particular, the data-driven model of P2P systems (e.g., BitTorrent) is shown to exhibit high scalability property: the service rate of the P2P system is proportional to the number of users. In the past few years, number of companies such as PPLive and UUSee are using the data-driven P2P approach to provide live-streaming or video-on-demand services [5]. The basic idea of the data-driven P2P streaming is for the server to organize the video content into a stream of "*chunks*" in playback order. The server uploads these stream of chunks to some randomly selected peers. Peers, in return, upload chunks that they possess and at the same time, request for chunks that they do not have. Given a large enough peer population, a sufficient number of neighbors and sufficient buffer size at each peer, the data-driven P2P streaming approach can potentially deliver a good playback performance for all peers.

For a P2P streaming service provider, the technical challenge is how to design a system that can provide *good playback continuity* to a *large population of viewers* and at the same time, reduce the operating cost of the system. In general, the operating cost includes (a) the number of servers needed to support a large viewing population, and (b) the amount of traffic uploaded by these servers since ISPs usually use the volume-based charging method for these streaming service providers.

At the heart of the P2P streaming protocol is the chunk selection algorithm: given a set of missing video chunks, which chunk should a peer request from its neighboring peers so as to enhance the system performance, e.g., playback continuity. In [14, 15], authors propose some *heuristic* chunk selection policies. These include (a) the greedy chunk selection, in which each peer requests the missing chunk with the most urgent playback deadline so as to maximize its own playback continuity, (b) the rarest chunk policy, in which each peer requests the missing chunk with the furthest playback deadline with the aim to maximize the rarest piece so as to improve the scalability of the system. The insight is that the priority for selecting a chunk depends on two factors: playback urgency (based on the local view of a peer's buffer), and distribution efficiency (based on the global scarcity of a chunk). A chunk policy based purely on playback urgency (i.e., the greedy chunk selection) can-

not scale; whereas a policy based purely on distribution efficiency (i.e., the rarest chunk selection) is asymptotically suboptimal as buffer becomes abundant compared to the peer population size. One fundamental question we seek to answer is to discover the *"structure" of the optimal chunk selection policy*. That is, given the server's upload capacity, the number of peers in the system and the buffer size of each peers, determine the chunk selection policy so as to maximize the average system playback continuity. The contributions of our work are as follows.

- Instead of focusing on few heuristic algorithms, we propose to study a large family of chunk selection policies, which we called the *priority-based chunk selection class*.
- We propose to use a general and unified analytical framework, the *density dependent jump Markov process* (DDJMP) [6] to analyze any chunk selection policy in the above mentioned class, and we prove that our framework is asymptotically exact when we scale up the system.
- We also propose the segmentation algorithm to reduce the computational complexity of evaluating the performance of a chunk selection policy. This algorithm facilitates us to explore the optimal chunk selection policy. This segmentation algorithm not only helps us to to efficiently find the optimal policy, but it also demonstrates the important influence of the server's upload capacity on the class of chunk selection policies that are likely to include the optimal: namely, the more the server is able to upload, the more the peers can afford to use the local greedy policy.
- We show that the optimal chunk selection policy has the *"$\vee$"-shaped structure*, while the worst chunk selection policy has the *"$\wedge$"-shaped structure*, where the shape refers to the priority of requesting missing chunks in the peer's buffer. Furthermore, the server upload capacity is monotonically beneficial to achieve higher playback continuity.
- We present a distributed adaptive algorithm so that the server can notify all peers about the optimal chunk selection policy whenever there is any change in the system parameters.

The balance of our paper is as follows. In Section 2, we provide the model of a data-driven P2P streaming system. In Section 3, we define the family of chunk selection policies we study and present the density dependent jump Markov process framework in analyzing any policy in the class. In Section 4, we explore the structure of the optimal chunk selection policy and state its properties. In Section 5, we present the segmentation algorithm to improve the computational efficiency of evaluating a given policy. In Section 6, we present an adaptive algorithm so that the server can notify the optimal chunk selection to all peers whenever there is any change in the system parameters. Experiments are carried out to illustrate the performance and robustness of our proposals. Related work is given in Section 7 and finally, Section 8 concludes.

2. System Models

In this section, we present the model of a data-driven P2P streaming system, as well as the buffering structure of each peer in receiving and displaying the video chunks.

2.1. Model of a P2P Streaming System

Consider a P2P live-streaming system which needs to serve M homogeneous peers. This P2P system has a logical server $\mathcal{S}$, which organizes the video content into stream of chunks in playback order. The server $\mathcal{S}$ has an upload bandwidth of C (in unit of bit per second). In order to view the live-streaming program, there is a playback rate requirement of r (in unit of bit per second). In this work, we consider a large scale P2P system wherein $C < Mr$. In other words, the server $\mathcal{S}$ can only support $f = \frac{C}{Mr} < 1$ fraction of peers. Therefore, peers need to collaborate with each other to maximize their chance of continual playback.

We model this large scale P2P live-streaming network as a discrete time system. At each time slot, the server $\mathcal{S}$ uploads one chunk of video to a fraction, f, of peers. Each chunk has a sequence number, starting from 1. Therefore, at time slot t, the server $\mathcal{S}$ randomly selects f peers and uploads the video chunk of sequence number t to these randomly selected peers.

One important note is that for a P2P streaming service provider, it is not only important to provide an adequate service (e.g., sustainable playback rate) to these M peers, but at the same time, the provider wants to minimize its operating cost. Deploying and maintaining a large server contribute to the operating cost, moreover, the amount of traffic upload also contributes to the operating cost since an ISP usually charges a content provider based on its upload traffic based on the volume-based charging model. Therefore, the streaming service provider would like to support a large population under a given continuity playback requirement with a smallest value of f as much as possible.

Each peer needs to receive and buffer these video chunks from the P2P streaming system. To achieve this, each peer maintains a local buffer $\mathcal{B}$, which can cache up to n video chunks. $\mathcal{B}(1)$ is used to store the newest video chunk that the server $\mathcal{S}$ is uploading in the current time slot, while $\mathcal{B}(n)$ is used to store the oldest video chunk that is being played

back. In other words, when the server $\mathcal{S}$ is uploading chunk with sequence number k, and if $k \geq n-1$, then video chunk $k-n+1$ is the chunk being played back by that peer (provided that the video chunk is available in $\mathcal{B}(n)$). At the end of each time slot, the video chunk in $\mathcal{B}(n)$ will be discarded, and all chunks will be "*shifted left*" by one position: video chunk in $\mathcal{B}(i)$ will be shifted to $\mathcal{B}(i+1)$, for $i = 1, \ldots, n-1$. Figure 1 illustrates the dynamics of buffer $\mathcal{B}$. At the beginning of time slot t, the server $\mathcal{S}$ is filling in $\mathcal{B}(1)$ and video chunks are available in $\mathcal{B}(3)$, $\mathcal{B}(4)$, $\mathcal{B}(6)$ and $\mathcal{B}(7)$ respectively. Also, at time slot t, video chunk in $\mathcal{B}(7)$ is fed to the video player for playback. At the beginning of time slot $t+1$, all video chunks in $\mathcal{B}$ are shifted left by one position.

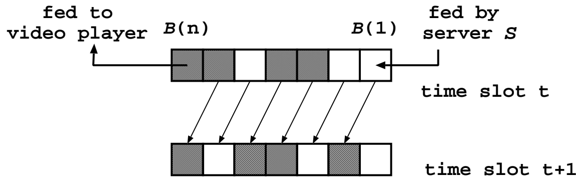


Figure 1: Buffer structure $\mathcal{B}$ with $n = 7$, server S fills $\mathcal{B}(1)$ at time slot t.

When a peer joins the P2P streaming system, its buffer $\mathcal{B}$ is initially empty. Eventually, each position of $\mathcal{B}$ may be filled, either by the server $\mathcal{S}$ or by other peers. The goal for each peer is to ensure the display buffer, namely $\mathcal{B}(n)$, is filled in as many times as possible so as to maximize its probability of continuous video playback. We define

$$\pi(i)=\text{Prob}[\mathcal{B}(i) \text{ is filled with video chunk}] \quad i=1,..,n. \tag{1}$$

Therefore, peers want to maximize $\pi(n)$ for continuous playback. Consider a client-server architecture (e.g., without the assistance of P2P technology). Since the server $\mathcal{S}$ randomly selects fraction of peers to upload, therefore $\pi(1) = f$. Due to the buffer shifting operation, it is not difficult to see that $\pi(n) = \pi(n-1) = \cdots = \pi(1)$, or

$$\pi(n) = f = C/(MR). \tag{2}$$

Remark: From the above equation, we can see that a client-server architecture has a scalability problem. When M is large and if a streaming service provider wants to have a reasonable value of $\pi(n)$, the service provider needs to ensure $C \approx Mr$. This translates to deploying many servers and uploading more traffic to viewers, which implies a very high operating cost, especially when one wants to support a large number of viewers. To alleviate this problem, one can rely on the P2P technology to fill in more video chunks in $\mathcal{B}$. Moreover, which video chunk to request to fill in to $\mathcal{B}$ (a.k.a, *chunk selection policy*) has a significant impact on the performance measure of continuous playback. In the following, we present a general framework to model and analyze a large class of chunk selection policies.

3. Chunk Selection Policies

The chunk selection is modeled as a *pull process*: at the beginning of the time slot, each peer randomly selects another peer and requests a video chunk. Since $\mathcal{B}(1)$ is used to cache the video chunk uploaded by the server $\mathcal{S}$, and $\mathcal{B}(n)$ is used as the playback buffer by the video player, therefore, a peer only needs to request for a missing video chunk in $\mathcal{B}(2)$ to $\mathcal{B}(n-1)$.

Note that this *pull model* has two implications. Firstly, a peer can be selected by multiple requesting peers. In this case, we assume the selected peer has sufficient upload capacity to satisfy all requests at one time slot. It is important to point out that when M is sufficiently large, the probability of being selected by many requesting peer is asymptotically small. Secondly, if the selected peer does not possess the requesting video chunk, the requesting peer loses the chance to download in this time slot. As we will illustrate, this simplification helps us to analyze a large family of chunk selection policies.

Since the downloading bandwidth of a peer is limited, when a peer has multiple missing chunks, it needs to decide which chunk to download. In this paper, we study the chunk selection policy which belongs to the *priority-based chunk selection class*.

Definition 1 (Priority-based Chunk Selection Policy) A priority-based chunk selection policy for buffer $\mathcal{B}$ with size n is represented by a permutation of length $n-2$, where the ith digit from the right is the relative selection priority of $B(i+1)$, and larger value implies higher priority. In each time slot, when a peer has multiple missing video chunks in $\mathcal{B}(2)$ to $\mathcal{B}(n-1)$, the peer always choose the missing chunk with the highest selection priority to download.

To illustrate, consider the example in Figure 1 in which the buffer $\mathcal{B}$ has size $n = 7$. Therefore, we can use 5 digits to present a priority-based chunk selection policy. Let say the chunk selection policy is 54321, which implies that $\mathcal{B}(6)$ has the highest selection priority while $\mathcal{B}(2)$ has the lowest selection priority. At time slot t, since the only missing video chunks are $\mathcal{B}(5)$ and $\mathcal{B}(2)$, therefore, this peer will request for the missing video in $\mathcal{B}(5)$, or in other words, the video chunk with the sequence number $t-4$.

Note that the above definition allows us to represent a large family of chunk selection policies. For example:
Random Chunk Selection: under this chunk selection policy, a peer requests to download any missing video chunk with *equal probability*. Note that this is the policy used for file distribution and it is shown to be very efficient [7]. For a buffer $\mathcal{B}$ with size n, the random chunk selection policy

is represented by $n-2$ digits of 1's. For example, when $n = 7$, this chunk selection policy is specified as 11111.

Greedy Chunk Selection Policy: under this chunk selection policy, a peer requests to download the missing video chunk that has the earliest playback deadline. The objective of this chunk selection policy is that each peer tends to maximize its probability of playback continuity. For a buffer $\mathcal{B}$ with size n, the greedy chunk selection policy is represented by $n-2$ digits with decreasing value from the left. For example, when $n = 7$, this chunk selection policy is specified as 54321.

Rarest First Selection Policy: under this chunk selection policy, a peer requests to download the missing video chunk that has the largest sequence number (or the video chunk that has just been pushed out by the server $\mathcal{S}$). This is a reasonable policy since this helps the rarest video chunk to spread faster in the P2P network and thereby improve the system scalability. For a buffer $\mathcal{B}$ with size n, the rarest chunk selection policy is represented by $n-2$ digits with increasing value. For example, when $n = 7$, this chunk selection policy is specified as 12345.

Family of Mixed Selection Policy: one can specify a chunk selection policy that combines the advantages of the greedy chunk selection and the rarest first selection. For example, when $n = 7$, we can specify a number of mixed selection policies, say 53124 or 42135. Note that the policy 53124 gives a higher weight to the chunk with the earliest playback deadline while the policy 42135 gives a higher weight to the chunk that has the highest sequence number. For both of these policies, the rarest chunk in $\mathcal{B}(2)$ and the video chunk with the earliest playback deadline in $\mathcal{B}(6)$ have a higher selection priority than other video chunks, while the chunk in the middle of the buffer (e.g., $\mathcal{B}(4)$ in this case) will have the lowest selection priority.

Note that the priority-based chunk selection represents a large family of policies. For a buffer $\mathcal{B}$ with size n, there are $(n-2)!$ permutations so there are at least $(n-2)!$ chunk selection policies. Given that there are a lot of chunk selection policies, what we need is a general and accurate modeling framework to evaluate and compare their performance. In the following, we discuss this modeling framework.

3.1. Modeling Framework for Chunk Selection Policies

The most direct approach to model any chunk selection policy is to use a discrete time Markov chain (DTMC). Let $\mathcal{C}_n$ be the set of all n-digits binary numbers that represent the buffer states, with the ith digit representing the state of $\mathcal{B}(i)$, e.g., if the ith digit is '1', it means the video chunk is available in $\mathcal{B}(i)$ and '0' otherwise. For example, if $n = 7$, then '1000001' represents that $\mathcal{B}(7)$ and $\mathcal{B}(1)$ have video chunks while other buffer cells do not have video chunk. Let $\boldsymbol{c}_k$ be the state of peer k, the state space of the DTMC is

$$S = \{(\boldsymbol{c}_1, \boldsymbol{c}_2, \ldots, \boldsymbol{c}_M) | \boldsymbol{c}_k \in \mathcal{C}_n, k = 1, \ldots, M\}. \quad (3)$$

A close examination of the state space S reveals that the number of states is $(2^n)^M$ for any chunk selection policy with buffer size n. This implies that the direct approach of using DTMC as a modeling framework has a huge storage and computational requirement, especially if we want to model a realistic system with a reasonable value of buffer size n and a relatively large umber of peers (e.g., $M \geq 500$).

Let us consider a different approach. Since $M \gg n$, instead of modeling the dynamics of *all* peers, we model the system dynamics with a given chunk selection policy as a *density dependent jump Markov process (DDJMP)* [6]. Let $x_c(t), c \in \mathcal{C}_n$ be the fraction of peers with buffer state c at time slot t. A buffer changes its state after downloading a chunk. In the downloading process, a peer has probability f to get the newest chunk from the main server, or it randomly chooses another peer and download the highest priority missing chunk which the selected peer has.

To illustrate, consider a system using a six-cell buffer and the rarest chunk selection policy. Assume the buffer state of a peer is 010100 at time slot t. If the peer gets the newest chunk from the main server, its buffer state will become 101010 (after shifting) in the next time slot. If it does not get the newest chunk and it selects a peer with a buffer state of 001110, then its buffer state will become 101100 in the next time slot. Note that due to the shifting, the lowest bit of all possible states in the system is always equal to zero.

Let $r(c)$ be the buffer state at the next time slot after a peer with buffer state c downloads the newest chunk from the main server. Let $s(c, c')$ be the buffer state at the next time slot after a peer with buffer state c downloads a chunk from another peer with buffer state c'. Then at time slot t, $x_c(t)f$ fraction of peers in state c download from the main server and switch to state $r(c)$, while $x_c(t)x_{c'}(t)(1-f)$ fraction of peers in state c download from another peer in state c' and switch to state $s(c, c')$. Summing all possible cases that could generate state c in the next time slot, we get the following equations that describes the system dynamics:

$$x_c(t+1) = f \sum_{r(k)=c} x_k(t) + (1-f) \sum_{s(i,j)=c} x_i(t)x_j(t), \quad c \in \mathcal{C}_n. \quad (4)$$

Let $t \to \infty$ and x_c be fraction of peers in state c when the system is stable, then for all $c \in \mathcal{C}_n$, we have:

$$x_c = f \sum_{r(k)=c} x_k + (1-f) \sum_{s(i,j)=c} x_i x_j, \quad c \in \mathcal{C}. \quad (5)$$

Let $b(c, i)$ be the ith bit of state c, then the probability that $\mathcal{B}(i)$ is filled with video chunk, which is denoted in

Equation (1), can be expressed as:

$$\pi(i) = \sum_{c:b(c,i)=1} x_c, \quad \text{for } i = 1, \ldots, n. \qquad (6)$$

To illustrate the DDJMP framework, we apply it to the following chunk selection policies:

• *Rarest first chunk selection policy with buffer size* $n = 4$. This system has eight possible states (since $B(1)$ is always 0): $0000, 0010, 0100, 0110, 1000, 1010, 1100, 1110$. Based on Equation (5), the fixed point equation are as follows:

$$x_{0010} = f \cdot x_{*000}, \quad x_{1010} = f \cdot x_{*100}$$
$$x_{0110} = f \cdot x_{*010}, \quad x_{1110} = f \cdot x_{*110}$$

where

$$\begin{aligned}
x_{0000} &= (1-f)(x_{*000}x_{*000}) \\
x_{1000} &= (1-f)(x_{*000}x_{*100} + x_{*100}x_{**00}) \\
x_{0100} &= (1-f)(x_{*010}x_{*0*0} + x_{*000}x_{**10}) \\
x_{1100} &= (1-f)(x_{*110}x_{***0} + x_{*010}x_{*1*0} + x_{*100}x_{**10})
\end{aligned}$$

with $\pi(1) = x_{***1}, \pi(2) = x_{**1*}, \pi(3) = x_{*1**}$ and $\pi(4) = x_{1***}$. Here we use the notation "$*$" to denote the sum of all possible cases. For example $x_{*010} = x_{0010} + x_{1010}$, $x_{*1*0} = x_{0100} + x_{0110} + x_{1100} + x_{1110}$,... etc. Note that one can determine the values of x's via standard numerical methods.

• *Greedy chunk selection with buffer size* $n = 4$. The system has eight states (again, because $\mathcal{B}(0)$ is always 0). Based on Equation (5), the fixed point equations are:

$$x_{0010} = f \cdot x_{*000}, \quad x_{1010} = f \cdot x_{*100}$$
$$x_{0110} = f \cdot x_{*010}, \quad x_{1110} = f \cdot x_{*110}$$

where

$$\begin{aligned}
x_{0000} &= (1-f)(x_{*000}x_{*000}) \\
x_{1000} &= (1-f)(x_{*000}x_{*1*0} + x_{*100}x_{**00}) \\
x_{0100} &= (1-f)(x_{*010}x_{*0*0} + x_{*000}x_{*010}) \\
x_{1100} &= (1-f)(x_{*110}x_{***0} + x_{*010}x_{*1*0} + x_{*100}x_{**10})
\end{aligned}$$

These equations are only different from those of the rarest first chunk selection policy at x_{1000} and x_{0100}.

• *Mixed chunk selection with buffer size of* $n = 5$. There are number of mixed selection policies, let us consider a particular mixed selection policy 312. For this system, there are 16 states and based on Equation (5), the fixed point equations are:

$$x_{00010} = f \cdot x_{*0000}, \quad x_{00110} = f \cdot x_{*0010}$$
$$x_{01010} = f \cdot x_{*0100}, \quad x_{01110} = f \cdot x_{*0110}$$
$$x_{10010} = f \cdot x_{*1000}, \quad x_{10110} = f \cdot x_{*1010}$$
$$x_{11010} = f \cdot x_{*1100}, \quad x_{11110} = f \cdot x_{*1110}$$

where

$$\begin{aligned}
x_{00000} &= (1-f)(x_{*0000}x_{*0000}) \\
x_{00100} &= (1-f)(x_{*0000}x_{*0*10} + x_{*0010}x_{*00*0}) \\
x_{01000} &= (1-f)(x_{*0000}x_{*0100} + x_{*0100}x_{*0*00}) \\
x_{01100} &= (1-f)(x_{*0010}x_{*01*0} + x_{*0100}x_{*0*10} \\
&\quad + x_{*0110}x_{*0**0}) \\
x_{10000} &= (1-f)(x_{*0000}x_{*1**0} + x_{*1000}x_{**000}) \\
x_{10100} &= (1-f)(x_{*0010}x_{*1**0} + x_{*1000}x_{***10} \\
&\quad + x_{*1010}x_{**0*0}) \\
x_{11000} &= (1-f)(x_{*0100}x_{*1**0} + x_{*1000}x_{**100} \\
&\quad + x_{*1100}x_{***00}) \\
x_{11100} &= (1-f)(x_{*0110}x_{*1**0} + x_{*1010}x_{**1*0} \\
&\quad + x_{*1100}x_{***10} + x_{*1110}x_{****0})
\end{aligned}$$

In essence, Eq. (4) is the density dependent jump Markov process and it is an approximation to the original Markov process. The right hand side of Eq. (4) gives the expectation of $x_c(t+1)$. Note that in the original Markov process, $x_c(t+1)$ has a non-zero variance which causes the Markov process to deviate from Eq. (4) for each time slot. However, when M is sufficiently large, the variance of $x_c(t+1)$ vanishes and Eq. (4) accurately describe the original Markov process. As a matter of fact, we have the follow result.

Theorem 1 The DDJMP described above converges almost surely, uniformly on all finite intervals $[0, T]$, to the solution of Eq.(4) when $M \to \infty$.

Proof: Please refer to Theorem 8.1 in [6] ∎

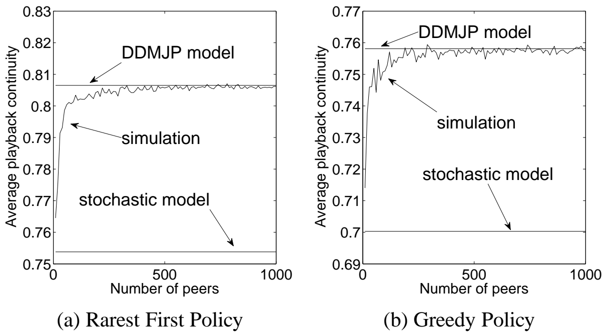
Corollary 1 The steady state probability vector $\boldsymbol{x} = \{x_c\}_{c \in \mathcal{C}}$ of the P2P streaming system is given by Eq. (5).

Let us illustrate the accuracy of using the DDJMP framework to model different chunk selection policies. Table 1 and Figure 2 compares the (1) simulation result, (2) our DDJMP model, and (3) the stochastic model given by [15] for the rarest First and the greedy chunk selection policies. In this study, the buffer size of $\mathcal{B}$ is set to $n = 8$ and the server's capacity can serve a fraction of $f = 0.1$ of users. In Table 1, the number of peers is $M = 1000$ and each row shows different value of $\pi(i)$, the probability that buffer $\mathcal{B}(i)$ is filled with video chunk, with the last row indicating $\pi(8)$, the probability of playback continuity. In Figure 2a and 2b, the horizontal axes are number of peers (M) and the vertical axes are the probability of playback continuity $\pi(n)$, or the probability that $\mathcal{B}(n)$ is filled with video chunk. In each figure, there are three curves: the simulation, the DDJMP model and the stochastic model in [15]. We can

see that (1) the DDJMP model is more accurate than the approach in [15], and (2) the results provided by the DDJMP model converges asymptotically when we increase the population size M. This agrees with our theoretical claim in Theorem 1.

$\pi(i)$	**RF sim**	**RF DDJMP**	**RF [15]**
$\pi(1)$	0.0000	0.0000	0.0000
$\pi(2)$	0.1000	0.1000	0.1000
$\pi(3)$	0.1807	0.1810	0.1810
$\pi(4)$	0.3074	0.3079	0.3024
$\pi(5)$	0.4696	0.4702	0.4496
$\pi(6)$	0.6245	0.6254	0.5858
$\pi(7)$	0.7355	0.7366	0.6863
$\pi(8)$	0.8058	0.8065	0.7538
$\pi(i)$	**GD sim**	**GD DDJMP**	**GD [15]**
$\pi(1)$	0.0000	0.0000	0.0000
$\pi(2)$	0.1000	0.1000	0.1000
$\pi(3)$	0.1375	0.1373	0.1296
$\pi(4)$	0.1879	0.1877	0.1715
$\pi(5)$	0.2600	0.2599	0.2330
$\pi(6)$	0.3688	0.3687	0.3272
$\pi(7)$	0.5342	0.5342	0.4759
$\pi(8)$	0.7576	0.7581	0.7003

Table 1: $\pi(i)$ for $M = 1000$, $n = 8$, $f = 0.1$, RF = Rarest First policy, GD = Greedy policy



(a) Rarest First Policy (b) Greedy Policy

Figure 2: Simulation & theoretical results. $n = 8$, $f = 0.1$

4. Exploring the Optimal Policy

The goal of this section is to explore the *optimal* chunk selection policy in the priority-based chunk selection class. In the previous section, we notice that $\pi(n)$, the probability of playback continuity of the streaming system, depends on multiple factors including (a) the server's upload capacity (which is represented by the fraction f in Equation (2)), (b) chunk selection policy s, (c) buffer size n and, (d) number of peers M. By Theorem 1, the system playback continuity $\pi(n)$ converges to a fixed value as $M \to \infty$, which can be solved using the DDJMP framework. Since this value only depends on n, s, f, we denote this asymptotic playback continuity as $C(n, s, f)$ and use it as the performance metric of policy s. Let l_s be the length of permutation s, then $n = l_s + 2$. So we may ignore the parameter n and write it as $C(s, f)$.

Definition 2 (Optimal/Worst Chunk Selection Policy) Let Π_n be the set of all n-permutations. The "optimal chunk selection policy" for buffer size n and initial fraction f is defined as $opt(n, f) = \text{argmax}_{s\in\Pi_{n-2}} C(s, f)$. Conversely, the "worst chunk selection policy" for buffer size n and initial fraction f is defined as $wst(n, f) = \text{argmin}_{s\in\Pi_{n-2}} C(s, f)$.

In other words, the optimal policy is the one that reaches the highest playback continuity while the worst policy is the one reaches the lowest playback continuity. One possible way to find the optimal (worst) chunk selection policy for buffer size n and initial fraction f is to enumerate all $(n - 2)!$ chunk selection policies and compute their respective $C(s, f)$. The policy which has the maximal (minimal) $C(s, f)$ is the optimal (worst) chunk selection policy. Obviously we need an efficient approach to overcome this problem. But before we formally present a computationally efficient method to find the optimal/worst chunk selection policy, let us first define some terminologies.

Definition 3 (Greediness) A transposition (i, j) is an operation on a priority-based chunk selection policy. In essence, it swaps the priority of $B(i + 1)$ and $B(j + 1)$ while the priority of the others remain the same as before. A transposition is greedy if in the exchange, the higher buffer position results in higher priority. We say the chunk selection policy s is greedier than the chunk selection policy r if we can obtain s after some finite greedy transpositions on r.

For example, the transposition $(1, 3)$ on policy 2134 results in the policy 2431, and this transposition is greedy. The policy 4213 is greedier than the policy 2134 since we can obtain the former by three greedy transpositions on the later, namely: (1,2), (2,3) and (3,4).

Definition 4 (Concatenation operator $\cdot$) Let $\mathcal{P}_s(i)$ be the priority of the ith cell of policy counting from left to right in policy s and l_s be its length. Policy $s \cdot r$ is defined such that $l_{s\cdot r} = l_s + l_r$, $\mathcal{P}_{s\cdot r}(i) = \mathcal{P}_s(i)$ for $l_r < i \le l_r + l_s$ and $\mathcal{P}_{s\cdot r}(i) = \mathcal{P}_r(i) + l_r$ for $1 \le i \le l_r$.

According to the above definition, the chunk selection policy $s \cdot r$ can be split into two segments, where the upper one has the same relative priority as s and the lower one has the same relative priority as r. Moreover, the lower segment

has higher priority than the upper segment. For example, let $s = 4321$, $r = 1234$, then $s \cdot r = 43215678$.

We carry out the sequence of experiments to explore the optimal and worst chunk selection policies for $n = 6, 7$ and 8 under different server's uploading limit f. Table 2-4 depict the asymptotic probability of playback continuity $\pi(n)$ for the optimal policies as well as the worst chunk selection policy.

f	opt. cont.	opt. policy	worst cont.	worst policy
0.04	0.4076	1234	0.3699	4321
0.19	0.7393	2134	0.7169	4321
0.26	0.7831	3124	0.7688	2431
0.32	0.8080	4123	0.7964	1432
0.39	0.8295	4213	0.8192	1342
0.50	0.8522	4312	0.8454	1243
0.95	0.9589	4321	0.9588	1234

Table 2: Optimal and worst policy for $n = 6$

f	opt. cont.	opt. policy	worst cont.	worst policy
0.02	0.4050	12345	0.3466	54321
0.10	0.7397	21345	0.6833	54321
0.14	0.7843	31245	0.7445	54321
0.17	0.8064	41235	0.7747	35421
0.21	0.8281	42135	0.8019	25431
0.29	0.8563	52134	0.8349	13542
0.35	0.8699	53124	0.8508	12543
0.40	0.8779	53214	0.8612	12543
0.41	0.8793	54123	0.8631	12453
0.45	0.8844	54213	0.8699	12453
0.55	0.8938	54312	0.8847	12354
0.95	0.9608	54321	0.9608	12345

Table 3: Optimal and worst policy for $n = 7$

Based on these experiments, we have the following observations:

Observation 1 *The optimal chunk selection policy is of* $\vee$*-shaped. That is, for the optimal chunk selection policy, let* $B(k)$ *be the buffer cell which has the lowest priority, then priority increases as the position moves away from* $B(k)$*. Conversely, the worst chunk selection policy is of* $\wedge$*-shaped. That is, for the worst policy, let* $B(k)$ *be the buffer cell with the highest priority, then priority decreases as the position moves away from* $B(k)$*.* Table 2-4 illustrate Observation 1. For example, from Table 4, when $f = 0.15$ and $n = 8$, the optimal chunk selection policy is '521346', which is of $\vee$-shape, while the worst chunk selection policy is '365421', which is of $\wedge$-shape. The importance of this observation is that it can restrict the search space for finding the optimal policy from $(n-2)!$ to 2^{n-2}.

f	opt. cont.	opt. policy	worst cont.	worst policy
0.01	0.4038	123456	0.3251	654321
0.05	0.7364	213456	0.6369	654321
0.07	0.7809	312456	0.6982	654321
0.09	0.8089	412356	0.7411	654321
0.11	0.8287	421356	0.7730	654321
0.15	0.8556	521346	0.8131	365421
0.18	0.8692	531246	0.8326	265431
0.19	0.8731	631245	0.8373	146532
0.25	0.8904	641235	0.8587	136542
0.27	0.8946	642135	0.8641	125643
0.33	0.9037	652134	0.8768	124653
0.38	0.9092	653124	0.8851	124653
0.47	0.9153	653214	0.8968	123564
0.49	0.9162	654213	0.8990	123564
0.61	0.9201	654312	0.9114	123465
0.96	0.9684	654321	0.9684	123456

Table 4: Optimal and worst policy for $n = 8$

Observation 2 *As the initial fraction* f *increases, the optimal policy becomes more greedier while the worst policy becomes less greedy.* Observation 2 is intuitive because when the server has a high upload capacity (or high value of f), chunk scarcity is rare and so the chunk selection policy should download those chunks which have earliest playback deadlines.

Observation 3 *The optimal policy* w *for buffer size* n *that reaches the playback continuity of* π *has the form* $s \cdot r$ *for sufficiently large* n*. Here* s *is a chunk selection policy which depends on* π *but is independent of* n*, and* r *is the Rarest First policy of length* $l_w - l_s$*.* Observation 3 provides a way to extend the optimal chunk selection policy for a small buffer to the optimal policy for a larger buffer. For example, the optimal policies that reach playback continuity of $\pi = 0.81$ for $n = 6, 7, 8$ are 4123, 41235 and 412356 respectively, and they share the same upper segment policy of 4123. Based on the Observation 3, the optimal chunk selection policy that reaches the playback continuity of 0.81 for buffer length 11 should be 412356789. Again, the importance of this observation is that it can help us to easily determine the optimal chunk selection policy for a particular system configuration (e.g., n and f). In the following section, we provide a computational efficient approach to find the optimal/worst chunk selection policy for large n.

5. Segmentation Method

To find the optimal and worst policy, it is necessary for us to compute $C(s, f)$. However, for a given value of n, the number of states in the DDJMP model is 2^{n-2}, therefore, the large number of states in the DDJMP framework

is still computationally expensive. In here , we present the segmentation method which is computationally efficient to solve and gives a very good approximation to $C(s, f)$. In essence, the segmentation method is a divide-and-conquer approach to estimate $C(s, f)$.

5.1. Segmentation Approach

We assume a buffer using the chunk selection policy s_B can be split into two segments so that any priority-based policy in the lower segment has a higher downloading priority than any cell in the higher segment. We denote the lower segment by L and the higher segment by H. For convenience, we add one hypothetic cell at the end of L to hold the piece shifted out of L and another at the beginning of H to hold the piece to be shifted into H. After adding these hypothetic cells, both H and L has the similar structure as a video staging buffer. In fact, as we will show later, we can model them as separate buffers by the DDJMP model. Let s_H, s_L be the chunk selection policies of H and L respectively, then we have $s_B = s_H \cdot s_L$. For example, the buffer of size $n = 10$ shown in Figure 3 using policy 43215678 can be split into two buffers H and L both of size 6 (Figure 3). The lower segment L uses policy $s_L = 1234$ and the higher segment H uses policy $s_H = 4321$.

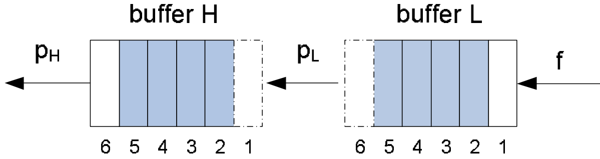


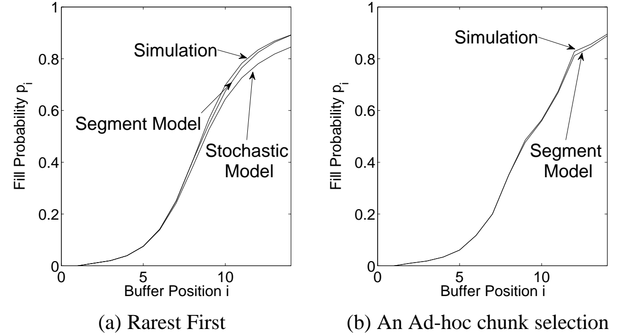
Figure 3: Splitting the buffer of size $n = 10$.

To model each segment separately, we have to fully account their mutual influence. Since cells in L have higher priorities than cells in H, H can never take away the downloading bandwidth from L, thus the activities in the H have no effect on L. Therefore, we can separate L from the whole buffer and model it accurately using DDJMP. The continuity of buffer L is $p_L = C(s_L, f)$.

Now we look at the higher segment H. The lower buffer L can affect H in two aspects. First, it loads up $H(2)$ with probability p_L at the beginning of each time slot. Secondly, L, with higher priority, preempts certain downloading probability from $H(2)$ to $H(l_{s_H} + 1)$. If we assume the correlation between H and L is small, the download probability took away by L is just p_L, which is the probability that downloading happens in L in a time slot. These two effects can also be fully accounted for by replacing L with the hypothetic $H(1)$ with downloading probability p_L in each time slot. Now we can concentrate on the analysis for cells in H. We can take the whole buffer B as the buffer H with the initial fraction p_L, and the buffer continuity is $p_H = C(s_H, p_L)$. Since $l_{s_H}, l_{s_L} \leq l_{s_B}$, the state space cardinality of the corresponding DDJMP models of the lower and higher buffer is much smaller than the original problem, therefore, p_H is much easier to compute than $p_B = C(s_B, f)$. We can summarize the segmentation method with the following proposition:

Proposition 1 Let s and r be two priority-based chunk selection policies, then $C(s \cdot r, f) = C(s, C(r, f))$.

Figure 4 compares the performance measure based on (1) the segmentation method, (2) simulation and (3) the stochastic model in [15]. The horizontal axes are the buffer positions from $i = 1$ to n. The vertical axes are the probability that $B(i)$ is filled, or $\pi(i)$. Figure 4a shows the result for the Rarest First Policy on a buffer with size $n = 14$. To apply the segment model, we split the buffer into two segments of size 8, both using the Rarest First policy. There are three curves which correspond to the simulation, the segment model and stochastic model in [15] respectively. Figure 4b shows the result for an ad-hoc piece selection policy $(3, 1, 6, 4, 2, 5, 11, 7, 12, 9, 10, 8)$ on a buffer of size $n = 14$. To apply the segment model, again we split it into two buffers of size 8. The lower segment uses policy 516342 while the higher one uses policy 316425. There are two curves which correspond to the simulation and the segmentation method respectively. From these two figures, we can see that the segmentation method is efficient and it provides more accurate results than the stochastic model [15].



(a) Rarest First (b) An Ad-hoc chunk selection

Figure 4: Evaluation for the Segment Model. $M = 10000$, $f = 0.01, n = 14$.

6. Adaptive Chunk Selection

In the previous section, we show that the optimal chunk selection policy depends on the server's upload capacity, which is represented by $f = \frac{C}{Mr}$, as well as the buffer size

n. Since the number of peers can be time varying, which implies that the optimal chunk selection can change also, therefore, we need an efficient protocol so the P2P streaming system can update the optimal chunk selection policy to all peers.

6.1. Estimate number of viewing peers M

Let us first consider how to estimate M, the number of peers of the video streaming session, in a distributed fashion. The value of M will increase whenever there is any new peer arrival, and will decrease whenever there is any peer departure. Note the there are two cases for a departure: a *normal* departure wherein a peer decides to leave the system, or an *abnormal* departure wherein a peer leaves the P2P streaming session due the software or network failure.

For the arrival event, a peer usually contacts the *tracker* so as to obtain the IP addresses of peers which are viewing the same video session. Therefore, the tracker can update the value of M whenever there is a new peer arrival. For normal departure, a peer usually informs the tracker so the tracker can update the value of M. However, the tracker cannot account for those peers which leave the P2P system due to abnormal departure. One can overcome this problem by using the distributed hash table (DHT). In particular, a tracker uses the IP address (or any unique ID) of a peer and assigns a DHT value to a peer upon its arrival. Given these DHT values, the tracker can organize peers in a logical ring (similar to Chord [13]) and each peer has a unique position in this logical structure. For each peer, say a, the tracker can assign a set of neighboring peers $\mathcal{N}_a$ for peer a to handle. All peers in $\mathcal{N}_a$ need to periodically send keep-alive messages to peer a. If peer a does not receive a keep-alive message from a given neighboring peer, peer a will inform the tracker so the tracker can update the value of M. Note that the above procedure is very lightweight and can be carried out in a fully distributed fashion. The pseudo-codes for the tracker and peers in estimating M are listed below.

Pseudo Code for Tracker:

```
1.   while(true){
2.   msg = wait_for_message_from_peers();
3.   if (msg == arrival of a new peer a ) {
4.     M++;
5.     Compute the DHT's ID for this new peer a;
6.     Compute the neighborhood N_a;
7.     Send ID and N_a to peer a;
8.     for (each peer b ∈ N_a)
9.         update peer b of its N_b by including peer a;
10   } /* end if for new peer arrival */
11.  if (msg == departure of a normal peer a) {
12.    M−−;
13.    for (each peer b where a ∈ N_b)
14.        update peer b of its N_b by excluding peer a;
15.  } /* end if for normal peer departure */
16.  if (msg == abnormal departure of peer a) {
17.    M−−;
18.    for (each peer b where a ∈ N_b)
19.        update peer b its N_b by excluding peer a;
20.  } /* end for abnormal peer departure */
21.  } /* end while */
```

Pseudo Code for peer a:

```
1.   Upon joining the system {
2.     msg = wait_for_message_from_tracker;
3.     ID = extract_ID (msg);
4.     N_a = extract_neighborhood_set (msg);
5.     for (each peer b ∈ N_a) {
6.       periodically send keep-alive message to peer b;
7.     } /* end for */
8.   } /* end for newly join peer */

9.   if (receive message from track to exclude peer b){
10.    remove b from N_a;
11.    disable sending keep-alive message to peer b;}

12.  if (no keep-alive message from peer b)
13.    send message to tracker to indicate the
             abnormal departure of b;
```

6.2. Updating the optimal Chunk Selection Policy

The next task is how to inform all peers about the new optimal chunk selection policy. From the above discussion, the system has an estimate of M, which implies that we now have an estimate of $f = C/(Mr)$. Given the value of f and the peer's buffer size n, the system can easily look up the optimal chunk selection policy, which we denote as s^*. Note that the optimal chunk selection policy for different values of f and n can be pre-computed offline. As we discussed in Section 4 and 5, one can use the DDJMP framework and the segmentation method to efficiently obtain the optimal chunk selection s^*.

To efficiently inform all peers about the latest chunk selection s^*. We consider the following approach: the streaming server will send the new chunk selection s^*, together will a timestamp t, to a fraction f of peers. When a peer, say a, receives this update message, it can compare with its current chunk selection policy and the associated timestamp. If the timestamp of the new message is larger than the timestamp of its current chunk selection, then the peer will use the received s^*, and then relay this update message to all its neighboring peers in $\mathcal{N}_a$. Else, peer a will suppress

relaying the new message to its neighbors. In essence, this is a *control gossip protocol* to broadcast the latest chunk selection s^* to all peers.

Let us analyze the time it takes to relay the new chunk selection s^* to all peers. Assume peers learn about the new policy s^* via gossip and peers receive gossips at a rate of 1 per time slot. If the gossip contains a policy with a newer time stamp, then the peer will switch to this policy. Let the serve pushes out the new chunk selection policy s^* to f fraction of peers at time $t = 0$. Let $x(t)$ be the fraction of peers using the new policy s^* at time t. Then on average, peers send out $x(t)M$ gossips altogether. Since there are $1 - x(t)$ fraction of peers using the old chunk selection policy and the receiver is randomly selected, each gossip will change $1 - x(t)$ peers on average. So in this current time unit, there will be $x(t)M(1 - x(t))$ peers, or fraction $x(t)(1 - x(t))$, switching to the new policy s^*. Thus we have:

$$\frac{dx}{dt} = x(t)(1 - x(t)).$$

Solving the above equation, we have:

$$x(t) = \frac{e^{t-\tau}}{1 + e^{t-\tau}}, \quad \text{where } \tau = \ln(f^{-1} - 1). \qquad (7)$$

Here τ is the time needed for half of all peers to receive the update. By Equation (7), when f is very small and x is very close to one, the time needed for the update to reach x fraction of users is

$$t = \tau + \ln \frac{x}{1 - x} \approx - \ln f - \ln(1 - x). \qquad (8)$$

In the following, we carry out a set of experiments to quantify the merits of our proposed algorithms.

Experiment 1: Performance comparison of different chunk selection policies: In Figure 5, we compare the performance of different chunk selection policies under different buffer lengths. The horizontal axis is the buffer length (n) and the vertical axis is the average playback continuity ($\pi(n)$). There are five curves correspond to the rarest first, random, greedy, the optimal and the worst chunk selection policies at each system configuration. In this experiment, there are 5000 peers and the initial fraction $f = 0.0002$. We can see that there is a big performance gap between the optimal and the worst policy. We also notice that the performance of the greedy policy is very close to that of the worst policy at all buffer lengths. This indicates that there is a need for peers to collaborate and not to focus purely on its local performance measure.

Experiment 2: Effectiveness of Adaptive Chunk Selection: In this experiment, we want to see the effectiveness of the proposed adaptive algorithms in subsection 6.1 and 6.2. We consider a P2P live streaming system with $M = 1000$, $n = 8$ and $f = 0.18$. Figure 6a illustrates the average

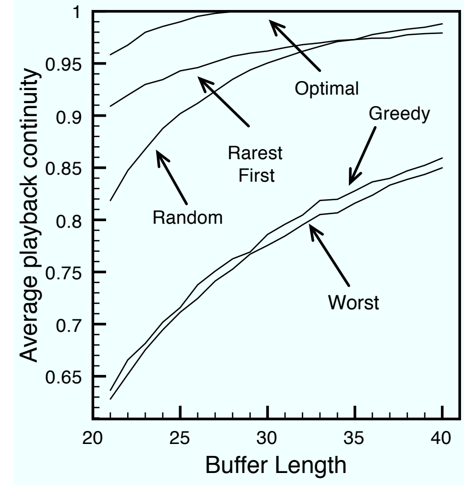


Figure 5: Comparison of five chunk selection policies.

playback continuity at different time points. Before time slot 2000, all peers use the rarest first chunk selection policy. Then the system switches to the optimal policy 531246 at time $t = 2000$. Figure 6b magnifies around the switching point. We see that the system adapts to the new policy within 10 time slots only. This shows the effectiveness and how quick the system can relay the optimal chunk selection to all M peers.

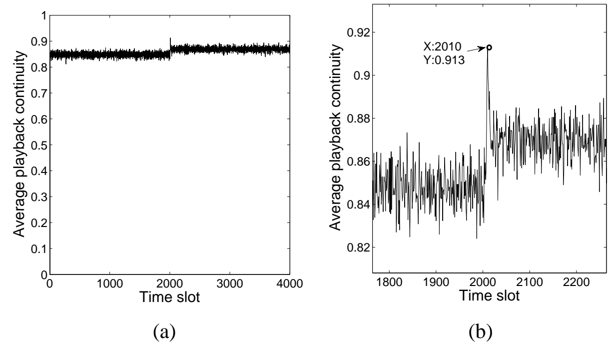


Figure 6: Policy switching delay, $n = 8$.

Experiment 3: Adaptiveness of different chunk selection policies: In Figure 7, we show the dynamics of different policies in a P2P live streaming system where the number of peers M changes with time. The horizontal axis is the time slot and the vertical axis is the average playback continuity. In this experiment, a new peer joins the system every two time slots and the server uploads is fixed at $f = 1/M$. Initially, the system has $M = 100$ peers. We simulate this P2P system for 5000 time slots so the peer number M increases steadily from 100 to 2600 and f decreases from 0.01 to 0.0004. The buffer length is set to 21. There are

three curves which are for non-adaptive, adaptive and the rarest first selection policy respectively. The non-adaptive policy is the optimal policy at time $t = 0$ and it remains to use this policy even when the number of peers is changing. The adaptive policy is the optimal selection policy at *each time slot*. We can see that as M increases and f decreases, the average playback continuity of the non-adaptive policy keeps deteriorating while the adaptive policy and the rarest first policy remain relative stable. We also see that the adaptive optimal policy has a significant performance gain over the other two policies.

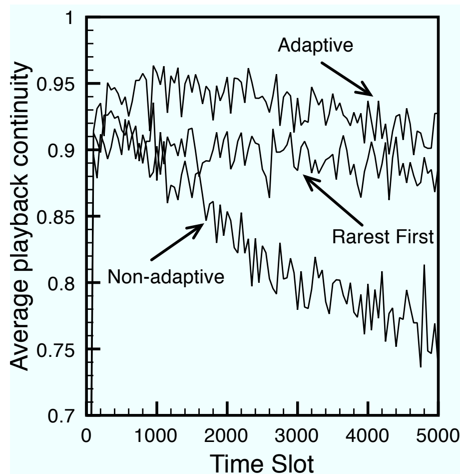


Figure 7: Dynamics of adaptive policies.

7. Related Work

There are number of recent work on P2P live streaming systems [3, 4, 9–11] wherein authors discuss the incentive issues, network-wide quality, time-shifting operations and advantages of the data-driven architecture. Several studies have explored the chunk selection policies for data-driven P2P streaming systems. In [14], authors propose to choose the chunk from the current *high-priority set* or from the *remaining set*. While authors in [12] propose to use the greedy chunk selection. Simulations were carried out to justify the above heuristics.

Closely related to this work is the simple stochastic model proposed in [15]. Authors proposed several heuristics algorithms. However, due to the inner buffer correlation, the model in [15] is not accurate enough to explore the property of optimality policy. In this work, we not only provide a *generalization* of the above model by allowing a variable server's upload capacity, but we also provide a more accurate analytical framework and shows its asymptotic correctness. More importantly, the density dependent jump Markov process (DDJMP) allows us to analyze a large class of chunk selection policies and derive optimality structure.

DDJMP is discussed in [6] and has been applied to P2P systems [7] to model BitTorrent file distribution systems. We apply this methodology to P2P streaming systems which has different chunk selection structures. There are some existing work which discuss various chunk selection policies. In [8], authors notice that chunk scarcity and urgency are two important factors. They consider a class of policies whose chunk priority is decided by a weighting faction that mixes these two factors. They also show via simulation that their optimal policy also exhibits the $\vee$-shaped structure while we use analytical approach to explore a large design space and derive optimal/worst structure.

8. Conclusion

In this paper, we provide an asymptotically exact analytical framework to analyze a large family of chunk selection policies for data-driver P2P streaming systems. We study the large design space of priority-based chunk selection policies observe some interesting properties of the optimal and worst policy. In particular, the optimal policy is of $\vee$-shaped and becomes more greedy as the upload capacity of the server increases. For a given continual playback probability, the structure of the optimal policy is also fixed as a concatenation of a policy independent of buffer size and the rarest First policy. This work provides some insight on the properties of the optimal policies and it allows streaming service providers to tradeoff between playback continuity and operating cost of deploying the service.

References

[1] C. F. Chou, L. Golubchik, and J. C. S. Lui. Design of scalable continuous media servers with dynamic replication. *Journla of Multimedia Tools and Applications*, pages 181–212, 17(2–3), 2002.

[2] C. Diot, B. N. Levine, B. Lyles, H. Kassem, and D. Balensiefen. Deployment issues for the ip multicast service and architecture. *IEEE Network*, pages 78–88, 14(1), 2000.

[3] F. V. Hecht, T. Bocek, C. Morariu, D. Hausheer, and B. Stiller. Liveshift: Peer-to-peer live streaming with distributed time-shifting. In *Peer-to-Peer Computing*, pages 187–188, 2008.

[4] X. Hei, Y. Liu, and K. W. Ross. Inferring network-wide quality in p2p live streaming system. In *IEEE JSAC*, pages 1640–1654, 2007.

[5] Y. Huang, T. Z. J. Fu, D. M. Chiu, J. C. S. Lui, and C. Huang. Challenges, Design and Analysis of a Large-scale P2P VoD System. In *ACM Sigcomm*, 2008.

[6] T. Kurtz. Approximation of population processes. *CBMS-NSF Regional Conference Series in Applied Mathematics.*, 1981.

[7] M. Lin, B. Fan, J. C. S. Lui, and D.-M. Chiu. Stochastic Analysis of File-Swaring Systems. *Performance Evaluation*, 64(9-12):856–875, 2007.

[8] Z. Liu, Y. Shen, S. S. Panwar, K. W. Ross, and Y. Wang. Using layered video to provide incentives in p2p live streaming. In *P2P-TV '07: Proceedings of the 2007 workshop on Peer-to-peer streaming and IP-TV*. ACM, 2007.

[9] Z. Liu, Y. Shen, S. S. Panwar, K. W. Ross, and Y. Wing. P2p video live streaming with mdc: Providing incentives for redistribution. In *ICME*, pages 48–51, 2007.

[10] N. Magharei and R. Rejaie. Prime: Peer-to-peer receiver-driven mesh-based streaming. In *INFOCOM*, pages 1415–1423, 2007.

[11] F. Pianese, D. Perino, J. Keller, and E. W. Biersack. Pulse: An adaptive, incentive-based, unstructured p2p live streaming system. In *IEEE Transactions on Multimedia*, pages 9(8), 1645–1660, 2007.

[12] P. Shah and J. F. Pairs. Peer-to-Peer Multimedia Streaming Using BitTorrent. In *IEEE IPCCC*, 2007.

[13] I. Stoica, D. Kiben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Transcations on Networking*, pages 17–32, 11(1), 2003.

[14] A. Vlavianos, M. Iliofotou, and M. Faloutsos. Bitos: Enhancing Bittorrent for Supporting Streaming Applications. In *9th IEEE Global Internet Symposium*, 2006.

[15] Y. Zhou, D. M. Chiu, and J. C. S. Lui. A Simple Model for Analyzing P2P Streaming Protocols. *IEEE International Conference on Network Protocols (ICNP)*, 2007.