實況與移時影音串流系統的實作與效能評估

學生：邱順胤　　　　　　　　　　　　　　　　　　　指導教授：張明峰　博士

國立交通大學資訊科學與工程研究所

摘要

點對點式網路架構目前廣泛應用於多媒體影音串流服務中，用以解決過去主從式網路架構所無法負荷的龐大流量。然而在點對點影音串流的蓬勃發展中，移時影音串流服務，也就是提供使用者觀看任意過去時間點上的影音串流服務的相關研究並不多。

於此篇論文中，我們提出了一個數學模型來評估移時影音串流服務的可行性，並且實作了一支援移時服務的點對點實況串流系統。此系統提出針對移時串流影音資料的分散式快取管理解決策略，並加強移時影音用戶的合作關係以分散對於整體系統的負擔。最後我們在 PlanetLab 平台上進行實驗，分析整體系統的效能以及特性。在我們實驗中全部移時用戶錯過少於 0.5%的影音片段，並且只有造成影音伺服器少於全部移時影音服務流量的 3%的負擔。而移時串流訂閱機制使移時用戶直接分享正在觀看的影音內容，其中有 30%的移時串流流量來自此種方式，並且降低了 15%原本需由實況用戶提供的移時串流流量。我們相信這將提供此類系統的一般性了解，幫助我們進一步的發展多媒體串流服務。

Implementation and Performance Evaluation of A P2P Live/Time-shifted Streaming System

Student：Shun-Yin Chiu   Advisor：Dr. Ming-Feng Chang

Institute of Computer Science and Engineering
National Chiao Tung University

ABSTRACT

Peer to peer (P2P) technologies have been applied to multimedia streaming services to solve the problem of heavy traffic flow on the servers in server-client architecture. However, there are few researches on the time-shifted streaming service, where viewers can choose an arbitrary offset of time to watch. In this thesis, we present a numeric model to analyze the feasibility conditions of the time-shifted streaming, and implement a P2P live and time-shifted streaming system. In this system, we propose a distributed cache management strategy for time-shifted streaming contents and enhance the cooperation among time-shifted peers to balance the traffic load in the system. Experiments were performed on PlanetLab to evaluate the system performance. Our experiment results show the feasibility of P2P time-shifted video streaming systems and the effectiveness of the proposed strategies. The video block missed rate of Time-shifter viewers is less than 0.5% and the server stress of time-shifted streams is less than 3% of all time-shifted traffic. By establishing supplier-subscriber relationship among time-shifted peers, more than 30% of the time-shifted streams are supplied by other time-shifted peers. This reduces 15% transmission load of time-shifted streams on live streaming peers.

誌謝

首先我要感謝指導教授張明峰老師，於碩士班的這兩年中，老師的指導及教誨，除了幫助我順利完成此篇論文外，更指引了我作研究應有的態度、縝密的思考模式以及獨立思考的能力，實在是獲益匪淺。

我也要感謝實驗室的同學，境余、亦彰，還有羽豪、睦倫和魏翔學弟，以及同學明辰、怡廷、日全、唯泰和佳慧，你們不只是一同努力的同伴，給予我碩士之路上奮鬥的動力，更為我在實驗室的生活增添了許多色彩，在這裡獻上由衷的感謝。

最後我要感謝我親愛的家人，感謝你們在我求學期間全心全意的支持，我才可以專心完成研究所的學業。

邱 順 胤 謹識於

國立交通大學資訊科學與工程研究所碩士班

中華民國一〇〇年七月二十四日星期日

Table of contents

List of figures

List of tables

Chapter 1

Introduction

With the increasing prevalence of broadband Internet access, multimedia streaming services have been very popular on the Internet in recent years. In the early developments of media streaming applications, client-server architecture suffers from scalability problems; as the number of simultaneous users increases, the servers are quickly overloaded [1]. Content delivery networks (CDNs) with strategically placed proxies have been developed to alleviate the load of the servers, but CDNs are too costly for general streaming applications [2]. IP multicast is probably the most bandwidth-efficient vehicle, but its deployment is very limited due to many practical issues, such as the lack of IP multicast supporting infrastructures and the lack incentives for network operators to carry streaming data traffic [3]. Application-level multicast, by constructing an overlay network with unicast connections between peers in the system, has been widely used to deal with the scalability issue in many Internet applications. Thus, streaming services based on P2P technologies are also very popular.

Current P2P multimedia streaming service researches can be classified into two categories: live streaming and VOD (video on demand) services. Live streaming is like a broadcast TV channel; it delivers the same video content to viewers simultaneously. On the other hand, a VOD is a video clip watched by viewers at different positions, and thus the video contents delivered to viewers are different. Recently there are few studies on time-shifted streaming services. A time-shifted streaming service allows viewer to choose an arbitrary offset of time to retrieve the streaming contents; it is like a VOD whose length keeps growing constantly.

Sachin Deshpande and Jeonghun Noh(2008) proposed a time-shifted and live streaming system (P2TSS) where peers cache part of the video contents to provide time-shifted stream.

If there is no peer can provide time-shifted contents, the time-shifted viewers retrieve from the server. However, time-shifted peers in P2TSS cannot share contents in their playback buffers; only the distributed stream cache (DSC) is shared. In this thesis, we implement a live streaming system that supports time-shifted streaming service. Moreover, time-shifted peers cooperate in pulling the desired contents from live streaming peers to reduce the load on live peers. We also performed experiments on the PlanetLab platform to evaluate our system performance.

The remaining part of this thesis is organized as follows. Chapter 2 describes the current work in P2P streaming studies related to our research. Chapter 3 presents the idea, design and implementation of our system in details. Chapter 4 presents an analytic model to estimate the feasibility of the time-shifted system. Chapter 5 presents the experiment setup, results, system performance. Finally, we give our conclusions in Chapter 6.

Chapter 2

Related Work

2.1 P2P live streaming

To provide P2P live streaming service, peers need to receive continuous delivery of video streams and may need to forward the streams to other peers. Peers can form an overlay structure to efficiently deliver video streams in a real-time fashion. CoopNet [6] adopts a hybrid model; a source node is responsible for maintaining a multi-tree overlay of stream delivery and asisting new peers to join. Using a multiple-description-coding (MDC) technique, each tree transmits a different MDC description. CoopNet is a complement to a client-server framework; the multi-tree overlay is only invoked when the video server is unable to handle the load imposed by clients.

In SplitStream [7], the video stream is split into multiple stripes and independent multicast trees are constructed to deliver a stripe on each tree. The multicast trees are constructed such that an interior node in one tree is a leaf node in all the remaining trees, the load of video forwarding can be evenly spread across all the peers. However, such node-disjointness is a property hard to achieve, especially in heterogeneous environments [8]. In GridMedia [4], a rendezvous point assists peers to join the overlay. A new peer first contacts the rendezvous point to obtain a list of peers on the overlay. Then, the new peer measures the end-to-end delay to each peer in the list and selects a number of peers as partners, with the probability of a peer being selected in inverse proportion to the end-to-end delay. This enables nearby peers to become partners, so that the latency of stream delivery can be reduced. In DONet/CoolStreaming [3], a peer first contacts an origin node and the origin node randomly selects a deputy peer and redirects the new peer to the deputy. The new peer can obtain a list of partner candidates from the deputy and establish partnership with these candidates. In addition, the video stream is divided into segments of uniform length, and the availability of segments in a peer's buffer is represented as a bitmap called Buffer Map (BM). Each peer continuously exchanges its BM with its partners, and pulls segments from its partners accordingly. The scheduling of segment pulling operations takes both availability and the partners' upload ability into consideration. The segment with the least number of available providers will be pulled first, from the partner with the highest available and sufficient bandwidth among the multiple potential providers, if any.

2.2 P2P VoD streaming

Video-on-Demand (VoD) service allows users to watch any video programs at any time. One of the design issues of P2P VoD service is what a peer should cache to alleviate the load of the VoD servers and how to locate and retrieve cached contents from other peers. In P2Cast [9], peers watching video clips within a short time interval form a session in a single-tree fashion. Each peer caches the beginning part of the video program and a new peer can be patched with the cached beginning part from its parent's cache. In P2Vod [10], peers form generations, where in each generation, peers have a synchronized buffer start. A new peer tries to join a generation, or form a new generation. A number of generations form a video session. If no peer in a session caches the first video block of the program, the session is closed to new peer;, and a new video session is created for new peers. Both P2Cast and P2Vod only support viewing from the beginning of VoD programs. oStream [11] allows peers to view from arbitrary positions of a program, but since oStream inserts new peers into the existing overlay, video disruption is noticeable on the child peers of the new peers.

BASS [12] uses BitTorrent protocol to distribute video contents, with the VoD server supporting emergency content dilivery. Their simulation results indicate that the server's load in terms of transmission bandwidth is reduced by 34% when the peers' average outgoing bandwidth is about the same as the video bit-rate. However, the required bandwidth from the server still increases linearly as the number of peer increases. PONDER [13] adopts a mesh-based overlay similar to BitTorrent, but a different delivery strategy to accommodate VoD service. While BitTorrent treats all data units, called chunks, with equal importance, PONDER partitions the video into equal sized sub-clips, each of which contains hundreds of chunks. The sub-clip close to the playback deadline is given a higher priority to download, so that the urgent data can be downloaded first. PONDER also gives up the tit-for-tat incentives of BitTorrent; peers are served based only on their needs without considering their contributions. This maximizes the probability that video contents can be downloaded before the playback time. PONDER achieves 70% saving of server bandwidth with users' average outgoing bandwidth being about 80% of video bit-rate, and up to 93% saving with users' average outgoing bandwidth being 112% of the video bit-rate.

2.3 live streaming with time-shifted streaming support

P2P time-shifted streaming services are not as popular as P2P live and P2P VOD streaming. In recent years, researches on P2P time-shifted streaming include LiveShift [14], P2TSS [15], DRPSS [16], Pseudo-DHT [17], and J-Tree [18]. LiveShift is a live streaming system based on a multiple layered tree overlay. When the buffer reaches a pre-defined size of one segment, the segment then is stored on a long-lasting storage and the peer adds a reference to the segment on the DHT. Although they presented a software prototype of the system, they did not show further detailed analysis of the system. P2TSS proposed a similar system architecture with two distributed cache algorithms: Initial Playback Position Caching (IPP) and Live Stream Position Caching (LSP) to determine which video block to be cached for others. Their simulation showed that P2TSS achieved low server stress with 7-15 connections to the server every hour. However they did not explain how long the connections to the server last. DRPSS and Pseudo-DHT are both based on P2TSS, but reduce the DHT searching cost. J-tree constructed independent multicast trees to deliver video content. Each multicast tree consists of peers whose playback positions are within 30 seconds after the root of the multicast tree. The roots of the multicast trees also forms a connected list with their playback positions in increasing order, so that each root forwards video blocks to its successor. The root of the first multicast tree receives video blocks from the video source. They achieve at most 1.5% block missing rate. However their simulation settings did not assume peers change viewing position in time-shifted streaming.

2.4 Kademlia DHT

Kademlia is a DHT system based on XOR metric. Each Kademlia node has a 160-bit identifier; each node chooses its identifier at random when joining the system. The keys used for the hash table mapping are also 160-bit identifiers. Given two identifiers *x* and *y*, the distance between them is the bitwise XOR (exclusive OR) result interpreted as an integer. The detailed operation will not be described here, but two major functions used in our system are PUT<*key*, *value*> and GET<*key*>. PUT<*key*, *value*> function stores the <*key*, *value*> pair on *K* nodes closest to the *key*, where *K* is a system parameter that can be adjusted. The GET<*key*> function retrieves the *value* associated with the previous PUT<*key*, *value*> have performed.

Chapter 3

System Design & Implementation

3.1 System overview

We intend to design a P2P streaming service that provides both live and time-shifted streams to viewers. Our system needs to cope with the following issues: the delivery of live streams, the caching and publishing of live streams, and the retrieval of time-shifted contents.

1. Live streaming framework

A live streaming framework provides a base for our system, because it supports live streaming service and the live streaming peers may need to store the received contents for the future use of time-shifted streaming viewers. Since many live streaming frameworks have been developed and comprehensively studied, we would not create a brand new live streaming system; instead, we adopted the design of the latest DONet/Coolstreaming with modifications to suit our needs.

2. Caching strategy

Live streaming peers need to cache the contents they have watched to support time-shifted streaming peers, and thus the caching strategy is an important issue. Since a live streaming peer may go off-line, and thus its cached contents cannot be retrieved by time-shifted peers. We will study how many replicas of live streams need to be cached to ensure that most time-shifted viewers can retrieve all the contents they need.

3. Time-shifted content retrieval

Time-shifted peers must first locate the cached contents in their interests before they can retrieve the contents. Kademlia [19-20] distributed hash table (DHT) is used for publishing and locating the cached contents with special care being taken to ensure that PUT operations do not over-write each other. In addition, time-shifted peers interested in the same video position would cooperate to retrieve the cached contents in order to reduce the transmission load on live peers caching the contents.

Figure 3-1 depicts the architecture of our system. The system consists of three types of nodes: a bootstrap server, channel providers and streaming viewers. The bootstrap server maintains a list of available channels and a partial list of participating peers of each channel, in order to bootstrap new peers. A channel provider is the source node of a streaming channel and registers its channel information with the bootstrap server. When a viewer joins the system, it first obtains the information of available channels and participating peers from the bootstrap server, and then retrieves the desired video contents for live or time-shifted streaming.

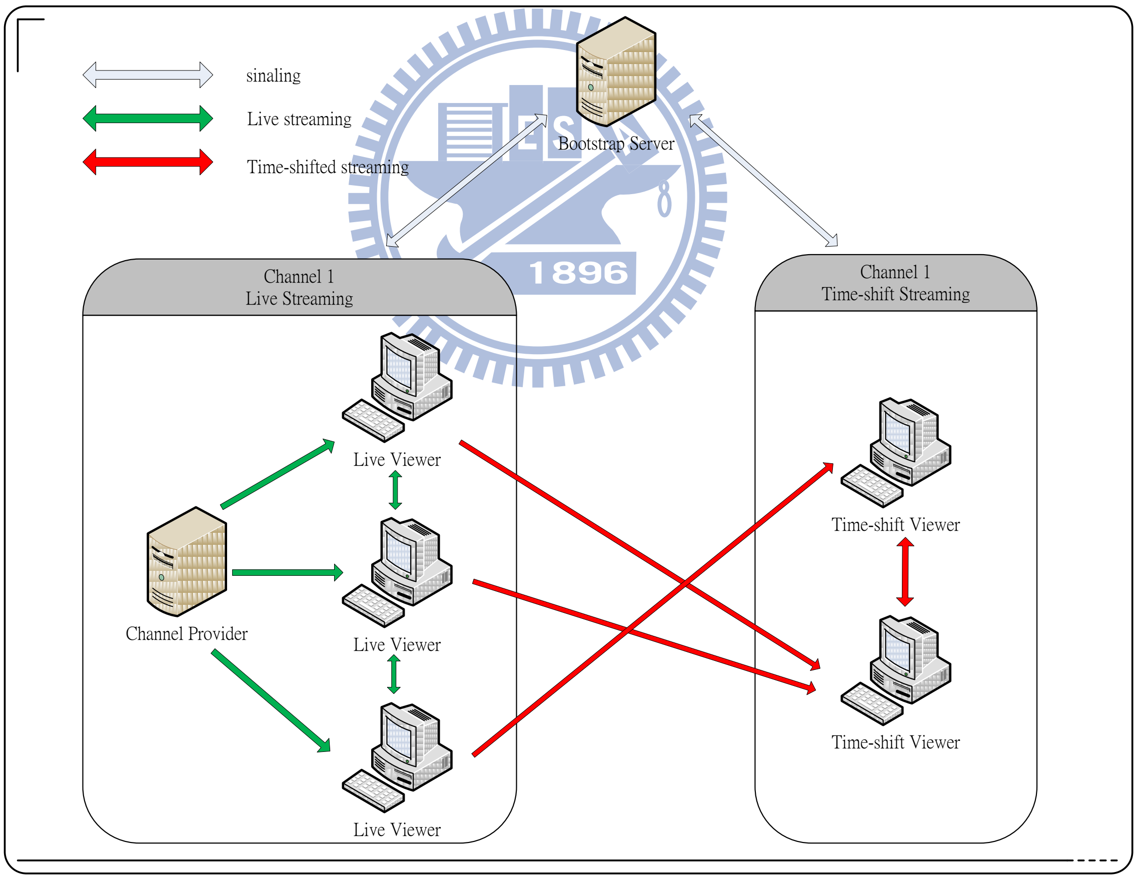


Figure 3-1 The system architecture.

3.2 The streaming transmission

The streaming transmission of our system is depicted in Figure 3-2. A video source generates continuous video contents of a channel. A channel provider encodes the video contents into a continuous stream of video packets and transmits the video packets to the viewers. At the video source, the video is encoded into UDP packets by a VLC media player [21]. The UDP packets are then sent to the channel provider via local loopback interface. The channel provider measures the duration of each packet and adds the duration and packet length for each received UDP packet. In addition, the channel provider packs continuous packets received in one second into a video block. Furthermore, in order to support time-shifted streaming, 10 consecutive video blocks, with the starting block's timestamp aligned to multiples of 10 seconds, are packed into a video file stored in local file system. The file is named after the information given by the channel provider, along with the timestamp of the first video block. For example, a video file with name "*ProviderName_Channel*1_20100620182520" stands for 10 blocks of *Channel* 1 provided by *ProviderName*, with timestamp 2010/06/20 18:25:20. Figure 3-3 shows the structures of a video block and a video file.

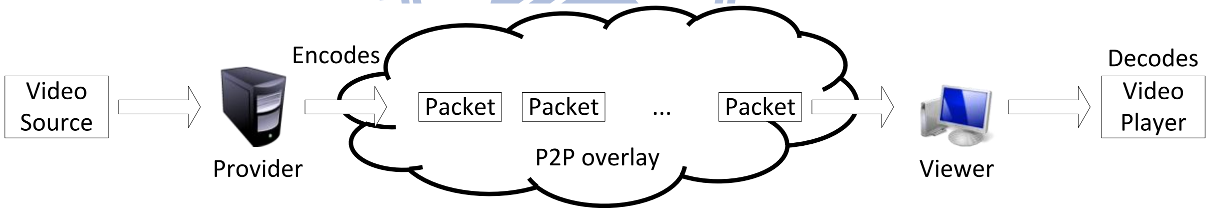


Figure 3-2 The streaming packet transmission.

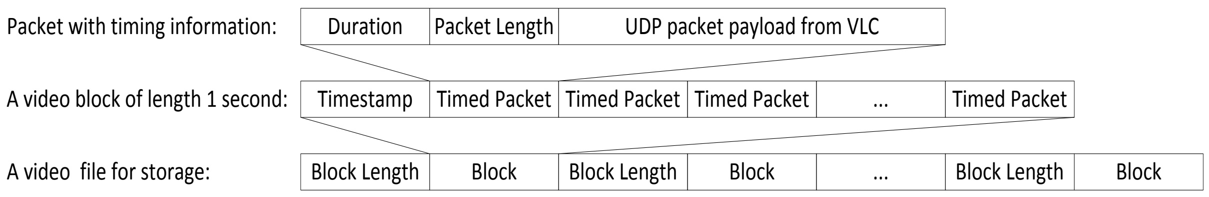


Figure 3-3 The structures of a video block and a video file.

3.3 Live streaming framework based on DONet/Coolstreaming

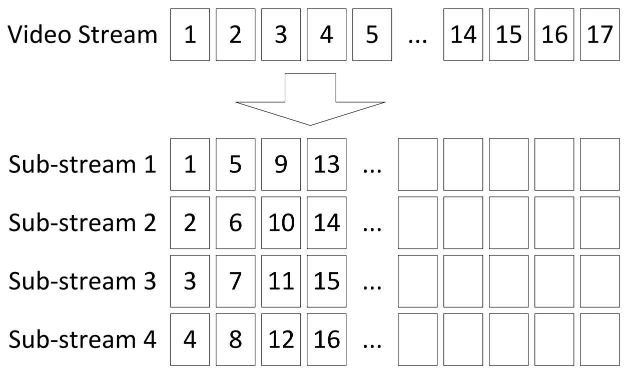
We adopted the design of the latest DONet/Coolstreaming as the live streaming framework to deliver live contents. For reader's interest, we briefly present the characteristics of the latest DONet/Coolstreaming, and our modifications.

1. Node hierarchy

Each live streaming peer maintains three levels of peers: members, partners and parents. Members are a subset of live-streaming peers watching the same channel as the peer. No connection is established between the peer and the members. Connections are established between partners to exchange the availabilities of video blocks. Parent-child relations are formed when connections are established for the transmission of video blocks. Apparently a peer's parents and children are a subset of its partners.

2. Multiple sub-streams

As described before, the video stream is encoded and packed into continuous video blocks, each of which is one-second long and time-stamped. The stream is also decomposed into *S* sub-streams, by grouping blocks whose timestamps have the same modulo of *S*. By dividing the stream into multiple sub-streams, each sub-stream can be retrieved from different parent peers independently, which means a peer can retrieve blocks from up to *S* peers. Figure 3-4 shows a video stream is divided into four sub-streams (i.e., *S*=4). In our implementation, the video stream is divided into 8 sub-streams.



Video Stream 1 2 3 4 5 ... 14 15 16 17
Sub-stream 1 1 5 9 13 ...
Sub-stream 2 2 6 10 14 ...
Sub-stream 3 3 7 11 15 ...
Sub-stream 4 4 8 12 16 ...

Figure 3-4 Sub-stream decomposition.

3. Joining procedure

When an new LS peer joins, it first contacts the bootstrap server and retrieves a list of available channels. After selecting a channel, the new peer retrieves a partial list of the active peers of the channel; the active peers become members known to the peer. Then the new peer randomly selects 24 of the members as its partners. Partners exchange their known members and the availabilities of video blocks periodically. Since each partner may receive the video sub-streams at different paces for different sub-streams, The new peer selects from its partners the fastest (with the largest timestamp) sub-stream of each sub-stream, and then sets initial playback position at the end of the slowest sub-stream among the *S* selected sub-streams. This would shorten the end-to-end delay since the fastest sub-streams are selected. After that, for each sub-stream, the new peer would subscribe the sub-stream from a partner whose pace is closest to the initial playback position. This would allow the peer to receive all sub-streams at about the same pace.

4. Hybrid push-pull mechanism

To form a parent-child relationship, a peer subscribes a sub-stream with a partner. When a partner receives a subscription message with a starting position of a sub-stream, the partner becomes the parent of the subscriber and stores the subscriber's IP address, a communication port number and a data port number in a sub-stream subscriber list. The parent starts sending to the subscriber the subscribed sub-stream starting from the requested position. The parent can be the provider or another live-streaming peer. The provider pushes a block to each subscriber whenever a new block is generated. A parent peer pushes a block to each subscriber whenever it receives a new block. Subscription contracts end when the subscriber re-selects its parents and sends an un-subscription message to its old parents.

5. Parent re-selection

As the subscription to a parent peer increases, the parent may be overloaded and lags in pushing blocks to its subscribers. A subscriber can detect such lagging by comparing block availabilities of its parents, or comparing block availabilities of itself and its partners. The subscriber measures the lagging of each sub-stream by comparing the position of each sub-stream with the average position of all sub-streams. If there is a sub-stream lagging over two blocks, which indicates the parent may be overloaded, the peer re-subscribes the most lagging sub-stream the partner whose position of the sub-stream is the nearest to the average position. As shown in the lower part of Figure 3-5, the peer compares the block availabilities in its buffer with a partner's buffer, and discovers that its sub-stream 2 is lagging behind the partner's sub-stream 2 by three blocks. The parent re-selection procedure is triggered, and a new parent is selected to provide the lagging sub-stream and the original subscription is cancelled. The new parent can be selected from the current partners or parents with better block availabilities.

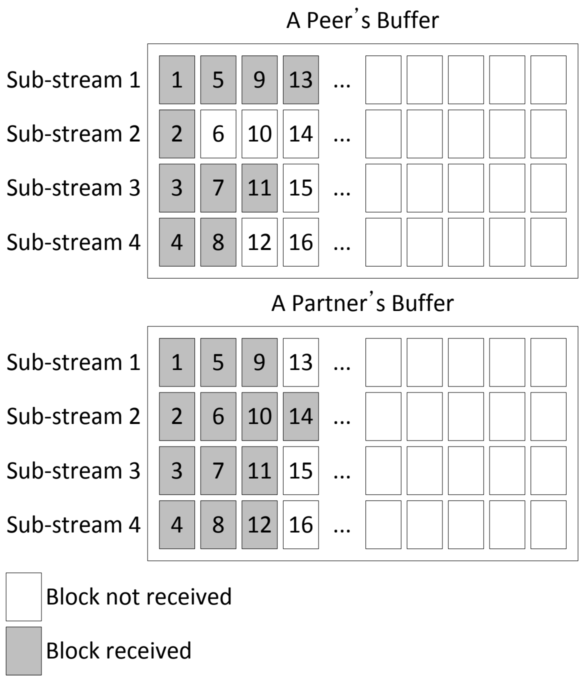


Figure 3-5 Comparing sub-stream block availabilities for parent re-selection.

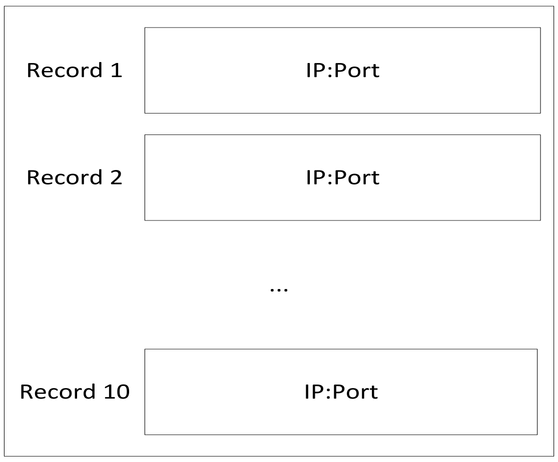


Figure 3-6 The owner list of each video file.

3.4 Distributed cache management strategy

Live streaming peers need to cache streaming contents for TS viewers in the future. The goal of our distributed cache management strategy is to effectively keep a desired number of replicas of video files for TS peers. The strategy is composed of two parts: content caching based on probability and publishing policy.

1. Caching based on probability

To distribute the responsibility of caching streaming contents and keep a desired number of replicas in the system, we adopted a probability algorithm to determine whether a file should be cached or not. Assume that the system wants to keep R replicas, and the system has N live streaming peers. A simple way to do it is that each node should cache the received content with a probability of R/N. Since R is a given, the discovery of N is the issue here.

To estimate N, first, a local knowledge based on the design of DONet/Coolstreaming is used. Since each peer keeps connections with its partners and parents, which are active LS peers. It is clear that N must be no less than the number of partners plus the number of parents. In addition, the number of the current active LS peers can be obtained by a modified peer-startup procedure. When a peer joins the system, heartbeat messages are periodically sent to the bootstrap server to update the member list maintained by the server, and the number of active LS peers can piggybacked to the peer in the reply messages. With the two values, N is selected as the larger one of the two. The local knowledge helps the peer to react fast to the change of active peers, especially when the size of viewers is small, since they would form an almost fully-connected mesh structure, and the number of the current active viewers helps the peer to make better decisions when the size of viewers becomes larger.

2. Content publishing policy

After a video file is collected for future time-shifted viewers, the peer publishes the ownership information of the file on the DHT. In addition, the channel provider caches all video contents but never publishes the ownership information. The channel provider would act as a backup source; its cached contents can only be accessed at emergency. For example, when a block is 5 seconds to the time-shifted playback deadline but had not been received, or when no owner information of a video file is published on the DHT. Since the system would keep multiple replicas for each video file, the published record on the DHT is a list of <IP: Port > tuples. Fig.3-6 depicts the data structure of the owner list. The owner list contains records of the IP:Port of the owners of a video file, and it can be retrieved by the hashed file name as a key GET(hash(filename)).

When a peer wants to update an owner list, it first tries to get the list from the DHT. If the size of the list is less than the desired number of replicas, the peer adds its IP address and a port number to the list, and puts the list back to the DHT. However, in this way the accesses of the DHT from the peers are not coordinated, which means a published record may be overwritten by another peer. This is a well-known write-after-write data hazard, and will be referred to as publishing collision.

To deal with the publishing collisions, each peer would perform an indirect DHT publishing to avoid such collisions. The publishing of the owner list of each video file on the DHT is performed by an LS peer chosen by the bootstrap server. First, the bootstrap server randomly chooses an LS peer from the partial peer list to be the collector responsible for publishing the owner list of the current video file. The bootstrap server puts a record <key(video-filename), value(the chosen peer's IP and port number)> on the DHT, i.e., the collector (the chosen peer) is the first on the owner list. Since a video file 10-sec. long, this is done every ten seconds. Second, an LS peer who wants to publish a video file needs to first check (get) the owner list of the file from the DHT. The LS peer would also check if the number of owners is less than *R* (the desired number of replicas). If so, the LS peer sends a publishing request to the collector, the first peer on the owner list. Third, the collector receives the publishing requests from LS peers, and sequentially updates the owner list on the DHT according to the requests. The publishing request message sent through a TCP connection contains only the file name to be published, since the owner's future contact IP:Port is the same as the TCP connection.

The collectors are requested to report their publishing results to the bootstrap server. If the owner list on the DHT does not match the local owner list, the collector reports to the bootstrap server, and the bootstrap server will not choose the peer as a collector. This enable the bootstrap server to have a more reliable list of collectors.

Peers who cannot access the DHT request the channel provider to publish all of their cached files. The provider does not perform the indirect DHT publishing, but directly publishes the requested file name by appending the owner to the owner list on the DHT.

3.5 Time-shifted streaming

A TS peer can retrieve time-shifted video content by per-block pulling from other peers, or by subscribing with TS peers. Using per-block pulling, the TS peer finds the owner list of video blocks, and pulls video blocks one by one from one of the owners, which can be LS or TS peers. Using subscription, the TS peer finds a TS peer whose playback buffer contains the needed video blocks, and subscribes for subsequent video blocks.

1. Per-block pulling

After a TS peer selects playback position, the name of the video file containing the video content is known. The TS peer hashes the file name to a key and obtains the owner list of the file by querying the DHT. Then the TS peer tries to pull up to 3 blocks per second, each from a randomly selected owner. When the TS peer has buffered 20 blocks, i.e. 20-second video, it starts to play back, or it is forced to play back after it starts to pull video blocks for 30 seconds. After playing back, the peer pulls 2 blocks per second until un-played blocks occupied half of the playback buffer, and then it pulls 1 block per second. For emergency handling, un-received blocks that is 10 seconds to the playback deadline are pulled from the channel provider. In addition, when no owner list is found on the DHT, the TS peer also pulls blocks from the channel provider. To retrieve video blocks smoothly, the owner list of the next 10 seconds from the current pulling position should be retrieved in advance. For example, the file of timestamp 10-19 should be retrieved when the file of timestamp 0-9 is pulled.

The video contents in the playback buffers of TS peers can be shared with other TS peers. As a TS peer pulls new blocks into its playback buffer, the old blocks are shifted out of because of the limited size of the playback buffer. We would like the keep the playback position at the middle of the playback buffer, so that half of the buffer are watched blocks that can be shared with other TS peers whose playback positions are older and the other half are real buffer for blocks to be played back.

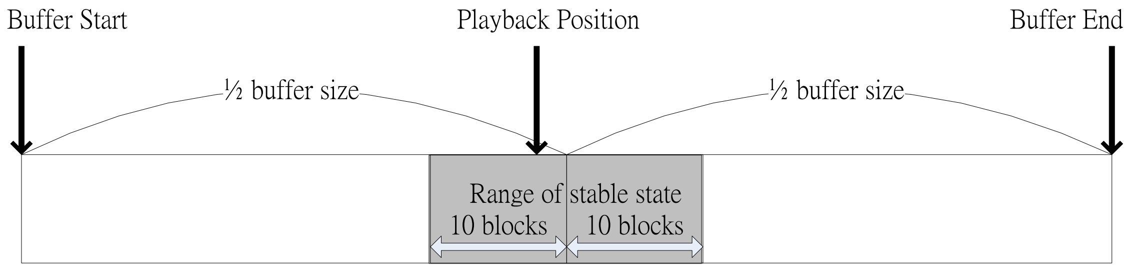


Figure 3-7 The stable state of a TS peer's buffer

2. Time-shifted stream subscription

TS Peers with overlapping playback buffers can share video blocks with each other. TS peers are partitioned into groups; peers in each group may share blocks in their playback buffers. The current playback position of each TS peer can be estimated by its initial playback position and the lapse of time after the playback. Using the estimated playback position of each TS peer, the bootstrap server can instruct TS peers with playback positions within a short distance to form a group. The median of a group is defined to be the average playback positions of all peers in the group. If a new TS peer's initial playback position is within 8 minutes of a group median, the bootstrap server instructs the peer join the group. Otherwise, the bootstrap server creates a new group for the peer. The peer receives from the bootstrap server a member list of the group. Peers in a group exchange playback buffer range (the buffer start and end) and the member list every 10 seconds.

With the exchanges of buffer range information, peers can check if their playback buffers are overlapping or not. After reaching the stable state, two peers with overlapping buffers can form a supplier-subscriber relation. The peer with an older playback position subscribes time-shifted stream with the other peer. Once the subscription is accepted, the supplier pushes time-shifted video blocks to the subscriber one block per second. If more than two peers have overlapping buffers, each peer would subscribe with the one whose buffer head is closest to its own. This means each supplier serves at most one subscriber at a time, and the supplier-subscriber relation of two peers will be rearranged when a new peer joins in the middle of them. The new peer would subscribe with the original supplier, and the original supplier cancels the contract with the original subscriber. The original subscriber will find that the new peer is with the closest buffer head and then subscribe with it. In subscription mode, the emergency handling for un-received blocks is the same as in that of per-block pulling.

3. Switching between DHT per-block pulling and time-shifted stream subscription

A new TS peer would first start in per-block pulling mode to retrieve initial playback blocks. After it reaches the stable state and finds a suitable supplier, it sends a subscription message to the potential supplier, and stops per-block pulling thread if the subscription is accepted. When a peer is served by a supplier, it stops requesting buffer range information with its group member, but still replies buffer range requests.

If a subscriber has not received video contents from its supplier for 3 seconds, it sends an un-subscription message to the supplier and starts per-block pulling. This is designed to react to unexpected behaviors on the supplier's side, such as, the shortage of network bandwidth shortage or disgraceful leaving. The supplier sends an un-subscription message to the subscriber when it changes its playback position or when another peer with a closer buffer head subscribes. Whenever a time-shifted peer switches from time-shifted subscription to per-block pulling, it needs to keep per-block pulling for at least a cycle time (10 sec.) of buffer range exchange to obtain buffer range information.

When a peer is in time-shifted stream subscription mode, it also continuously retrieves the current owner list and the next owner list as in per-block pulling mode to achieve smooth switching. Otherwise it may suffer from a longer startup delay when switching to per-block pulling mode.

3.6 System architecture

Figure 3-7 the block diagram of a channel provider. A VLC player encodes the original video stream into packet streams, and the provider packs the packets into video blocks and put the video blocks in a streaming buffer for serving LS peers. In addition, the video blocks are packed into video files and stored in local file system to serve emergent requests from TS peers. Streaming transmissions are carried out over TCP connections to avoid data losses in the network layer.

Figure 3-8 depicts the block diagram of a viewer node. The received video blocks are put in the playback buffer for playback, generating block availabilities and potential block transmission to other peers. To share video content among peers, the buffered video blocks can be transmitted to other peers for live streaming or time-shifted streaming. The live stream can only be provided from the playback buffer, while the time-shifted stream can be provided from both the playback buffer and the video files stored in the file system. The member, partner and parent management handle peer hierarchy and the exchange of video block availabilities for LS peers, while only member management is performed by TS peers. The sub-stream management puts the received video blocks at the right places in the playback buffer, and fetches the right video blocks to transmit for sub-stream subscription. The video file management and cache and publish policy cache the buffered content into the file system and publish the contents to the DHT. Publishing messages are transmitted over UDP packets. The DHT also takes the responsibility of obtaining block availability published by block owners in time-shift streaming.

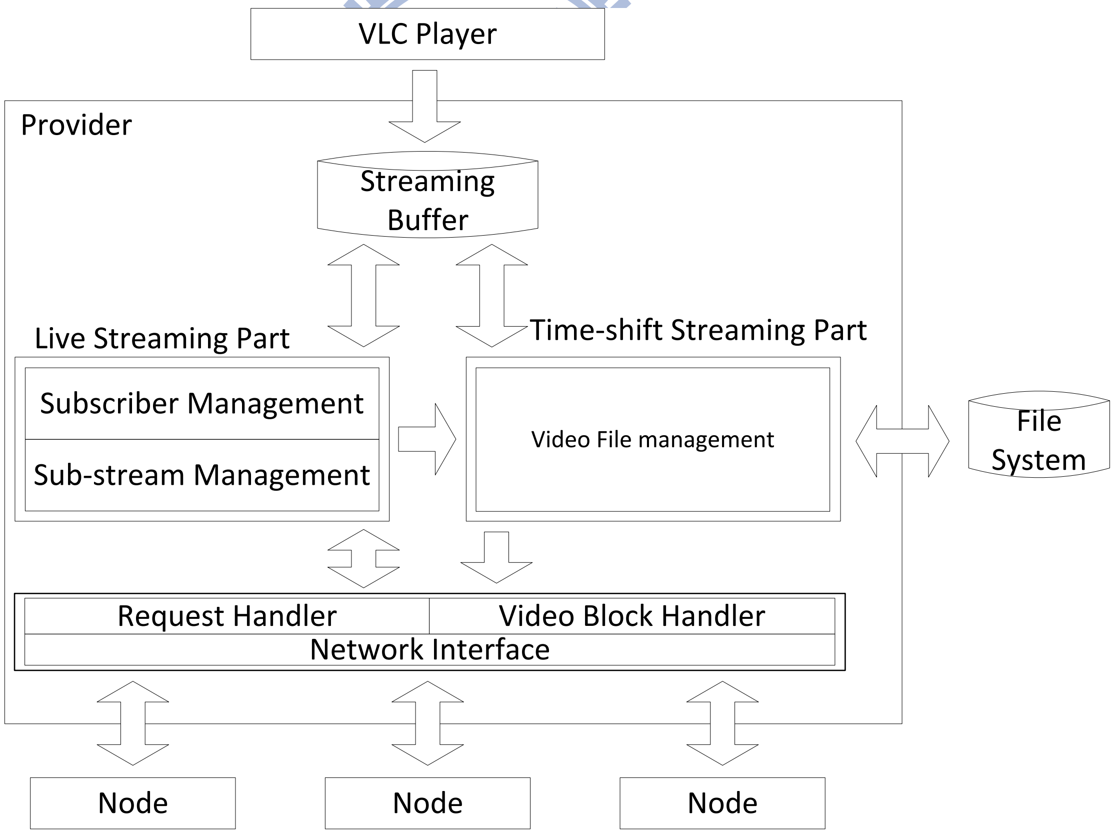


Figure 3-8 The block diagram of a channel provider

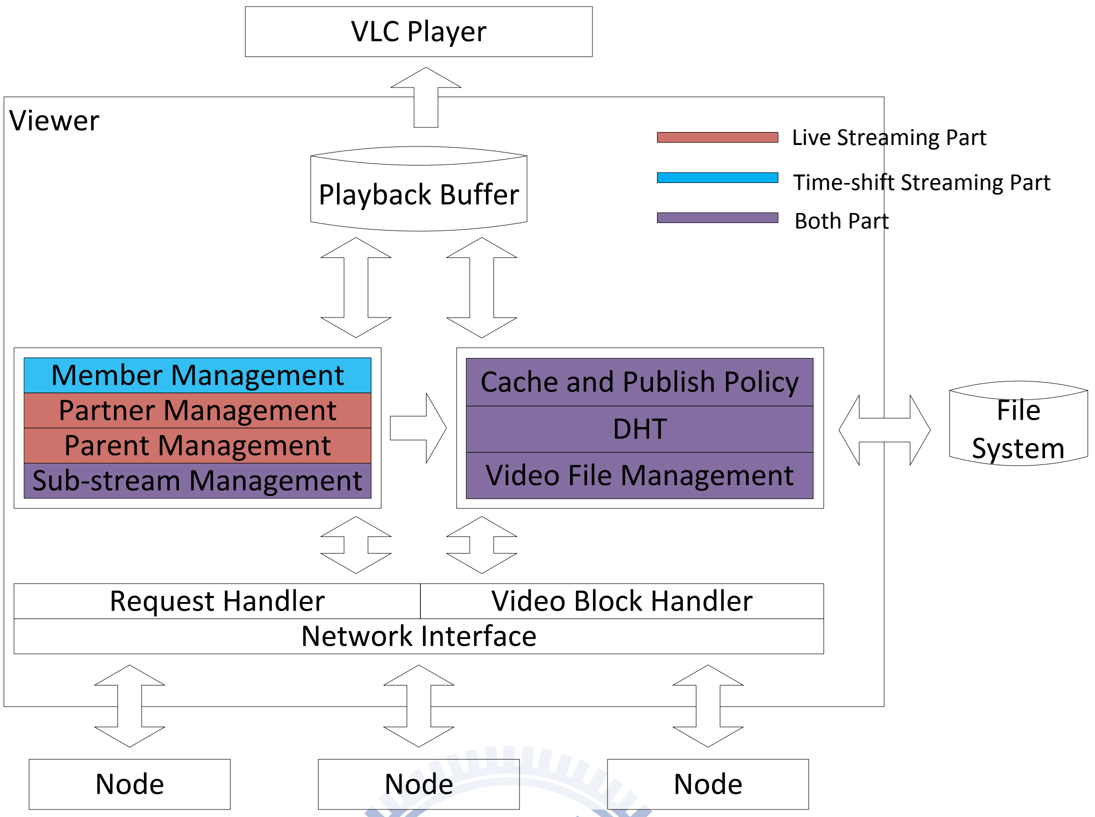


Figure 3-9 The block diagram of a viewer.

Chapter 4
Feasibility analysis of time-shifted streaming

A time-shifted viewer chooses a video position in the past and tries to obtain the time-shifted contents cached by live peers. We intend to determine whether a time-shifted viewer can retrieve his or her desired contents in full. It is clear that the answer depends on whether the desired contents have been cached by live peers and whether the peers with the cached contents are on-line or not.

Assume that the arrivals of live viewers form a Poisson process with arrival rate λ, a live viewer watches the live program for a random duration exponentially distributed with mean $1/\mu$. It is clear that the number of live peers can be modeled as a Markov process depicted in Fig 4-1, i.e., an M/M/∞ queue. If each live peer caches all the contents that the viewer watches, the number of live peers caching a video segment equals to the number of replicas cached for the segment. The steady state probability of state k, denoted by P_k, can be derived as follows,

$$p_k = \frac{(\lambda/u)^k}{k!} e^{-\lambda/\mu} \qquad (1)$$

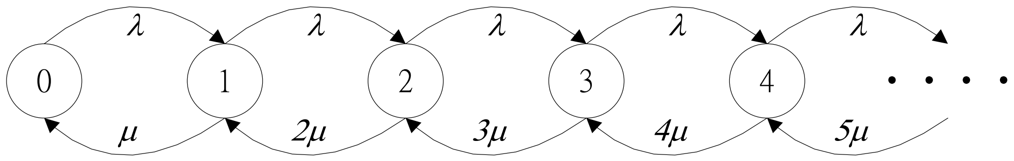


Figure 4-1 Markov process of the number of live viewers

In addition, we need to consider whether the peers who cached the desired time-shifted contents are on-line or not; only the on-line peers with the cached contents can provide.

Assume that the on-line and off-line state of a peer forms an alternating renewal process where the on-line intervals are exponentially distributed with mean $1/\mu_{on}$, and the off-line intervals are also exponentially distributed with mean $1/\mu_{off}$. Let P_{on} denote the probability that a peer is on-line, and P_{off} the probability of being off-line. It is clear that $P_{off} = 1 - P_{on}$, and P_{on} can be obtained as follows.

$$p_{on} = \frac{1/\mu_{on}}{1/\mu_{on} + 1/\mu_{off}} = \frac{\mu_{off}}{\mu_{off} + \mu_{on}} \tag{2}$$

When a time-shifted viewer chooses an initial playback position, the time-shifted peer re-visits the Markov process of the live viewers at the chosen position. The process is recorded in the form of cached contents by the live peers. In addition, the time-shifted peer is a random observer of the on-and-off alternating renewal processes of the live peers at the current time. Counting the number of the on-line live peers observed by the time-shifted peer, is a birth-death process. If the interval between the chosen position and the current time is large enough, the Markov process and the birth-death process are independent. The state initially observed by the time-shifted viewer can be modeled by superimposing the Markov process and the birth-death process, and we obtain a two-dimensional continuous Markov chain as depicted in Fig. 4-2, where State (i, j) represents that $i+j$ peers cached the desired contents, i of them are on-line and j of them off-line. Note that this Markov chain superimposes two different time-spans; the time-span when the chosen playback contents were live, and the time-span of the current time. The former will be referred to as the time-shifted time-span, and the latter as the current time-span. Note that if we merge all states in each column, i.e., states with the same $i+j$ value, the Markov chain in Fig. 4-2 is reduced to that in Fig. 4-1.

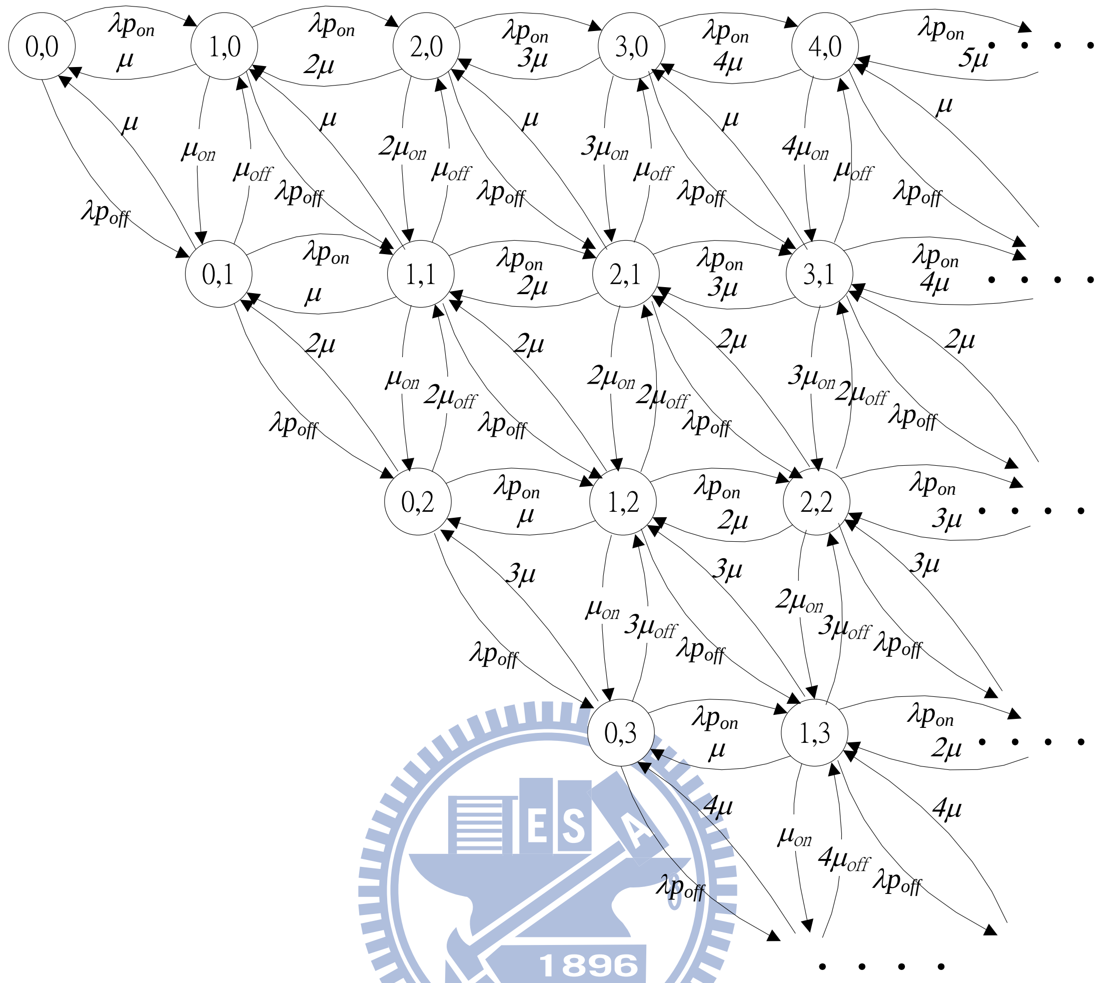


Figure 4-2 The number of available replicas for time-shifted contents

There are at most four types of events that cause out-flow transitions at state (*i*, *j*) in Fig. 4-2. Take state (2, 1) for example, i.e., we have three live peers caching the desired contents in the time-shifted time-span, and two of them are on-line and one of them is off-line in the current time-span. First, in the time-shifted time-span, state (2, 1) transits to state (1, 1) with rate 2μ, indicating one of the on-line live viewers finished watching,, and transits to state (2, 0) with rate μ, indicating the on-line live viewer finished watching. Second, state (2, 1) transits to state (3, 1) with rate λP_{on} and to state (2, 2) with rate λP_{off}, indicating a new live viewer arrived. These transitions are the mixed effect of the time-shifted and current time-spans; the live viewer arrived at the time-shifted time-span, but it on-line/off-line state is determined in the current time-span. Third, in the current time-span, state (2, 1) transits to state (3, 0) with rate μ_{off} indicating the off-line live viewer becomes on-line. Last, state (2, 1) transits to state (1,

2) with rate $2\mu_{off}$ indicating one of the two on-line live viewers becomes off-line.

The Markov process in Fig. 4-2 satisfies Detail Balance Equations, i.e., for each pair of neighboring states, the flows of transitions to each other are the same. For example, consider two neighboring states a and b. Let p_a and p_b denote the state probabilities, respectively, $q_{a,b}$ the transition rate from a to b, and $q_{b,a}$ that of b to a. We have a Detail Balance Equation as

$$p_b q_{b,a} = p_a q_{a,b} \tag{3}$$

Like all Markov processes, the Markov process in Fig. 4-2 also satisfies Global Balance Equations, i.e., the flow of transitions into each state equals to the flow of transitions out of the state. Consider state a, the Global Balance Equation can be express as

$$p_b \sum_{a \neq b} q_{b,a} = \sum_{a \neq b} p_a q_{a,b} \tag{4}$$

Let $P_{i,j}$ denote the stationary state probability of state (i, j) in Fig. 4-2. Since the Markov process of the number of live peers at the time-shifted time-span, and the birth-death process of on-line live-peers at the current time span are independent, $P_{i,j}$ can be obtained as

$$P_{i,j} = \frac{(\lambda\ /\mu\ )^{i+j}}{(i+j)!} e^{-(\lambda\ /\mu\ )} \cdot \frac{\binom{i+j}{i} \mu_{off}{}^{i} \mu_{on}{}^{j}}{(\mu_{off}+\mu_{on})^{i+j}} \tag{5}$$

The intuition behind Eq. (5 is simple; $P_{i,j}$ equals P(there were i + j live peers) times P(i of the live peers are on-line). One can also verify that the $P_{i,j}$ given in Eq. (5 satisfies the Detail Balance Equation in Eq.(3.

The steady state probabilities of the Markov chain in Fig. 4.2 are what a time-shifted viewer would observe when the viewer chooses an initial playback position. Assume that the time-shifted viewer would watch the chosen time-shifted contents for a time interval of length exponentially distributed with mean $1/\nu$. Let P_c denote the probability that the time-shifted viewer can completely watch the desired contents, i.e., there are on-line peers to provide all the desired contents. P_c can be obtained from the Imbedded Markov chain of the Markov chain depicted in Fig. 4-3. Note that the Markov chain is modified from the Markov chain in Fig. 4-2. A new state *c* is added; all states (*i*, *j*), $i \neq 0$, make a transition to state *c* with rate ν, indicating the time-shifted viewer has watched the desired contents in full. In addition, no out-flow transition exists for states (0, *j*), because at those states, it is determined that the time-shifted peer cannot obtain the desired contents, and thus the viewer could not watch the desired contents in full.

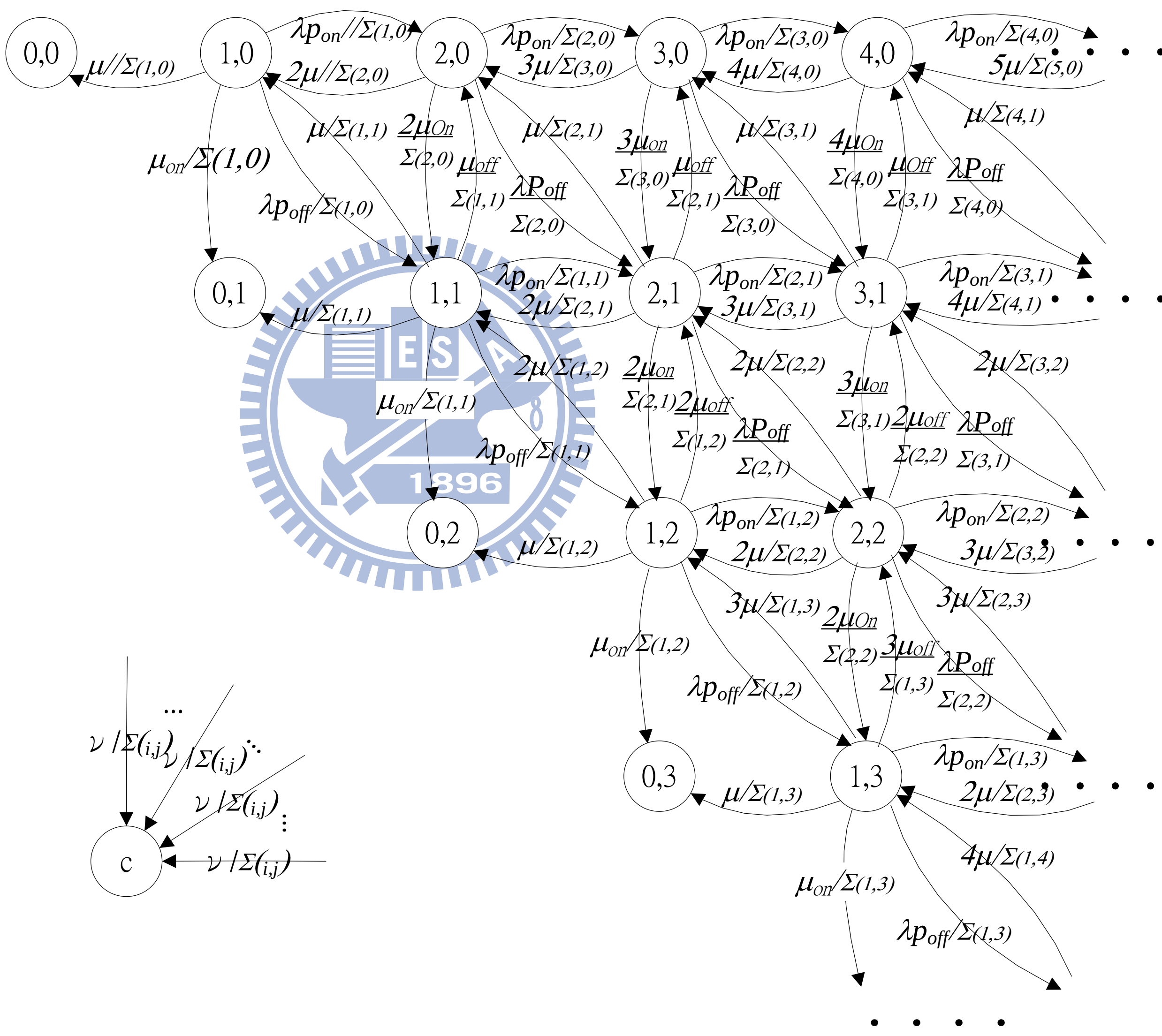


Figure 4-3 A modified Markov process observed by a time-shifted viewer

The steady state probabilities of the Markov chain in Fig. 4-2 were used as the initial state probabilities of the imbedded Markov chain in Fig. 4-3. Note that the initial probability of state *c* is 0. In the imbedded Markov chain, each transition has a transition probability, not a rate. Taking an example of out-flow transitions of state (2,1) in Fig 4-4, the transition probability of a transition equals to its rate divided by the sum of all the transition rates originated from the same state of the transition. Let $\Sigma(i,j)$ denote the sum of state transition rates out of state(i,j), for example, $\Sigma(2,1) = \mu + \mu_{off} + \lambda P_{on} + \lambda P_{off} + 2\mu_{on} + \nu + 2\mu$. A transition probability matrix P for the imbedded Markov chain can be obtained. Let π_i denote the initial probability vector of the imbedded Markov chain, i.e., the steady state probabilities of the Markov chain in Fig. 4-2. Let π denote the limiting probability vector of the imbedded Markov chain. We have

$$\pi = \lim_{m\to\infty} \pi_i P^{(m)}$$

Since state c and states (0, *j*) have no out-going transition, they have non-zero limiting probabilities; other states' limiting probabilities are all zero. P_c, the probability that a TS peer can find all the desired contents from on line peers, equals to the limiting probability of state *c*.

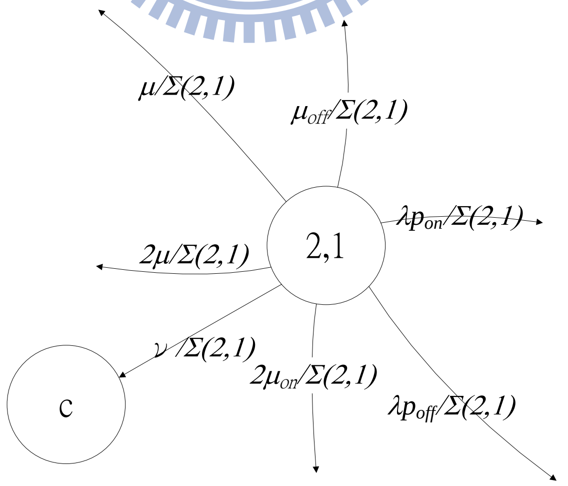


Figure 4-4 Out-flow transitions of a state

Let $P_{i,j}$ denote the stable state probability of state (i,j). Let *ER* denote the expected number of available replicas; $ER = \sum_{\text{all state}} i * P_{i,j}$.

Selecting reasonable values for λ, μ, ν, μ_{on}, μ_{off}, we can obtain P_c and *ER* as indicators of the feasibility of TS streaming service. The average inter-arrival time of LS peers, $1/\lambda$ was assumed to be 4.28 min. This small arrival rate is chosen for being conservative. After arrival, all peers, LS or TS, would watch for 60 min., i.e., $1/\mu = 60$ and $1/\nu = 60$, because TV programs are usually 1 hour long. The expected number of cached replicas for each video block, λ/μ, equals 14. Four combinations of μ_{on} and μ_{off} were tested; $1/\mu_{on}:1/\mu_{off} = 6:18$ (hr), 8:16, 10:14, and 12:12. Fig. 4-5 depicts *ER* for the four combinations. It is clear that *ER* increases linearly as the on-line interval ($1/\mu_{on}$) increases. Fig. 4-6 depicts P_c. The results indicate that P_c drop sharply when the on-line interval is less than 8 hr., i.e, *ER* is less than 5, When the on-line interval is 12 hr., i.e., *ER* equals 7, a TS viewer would view the desired contents in full with probability greater than 99%.

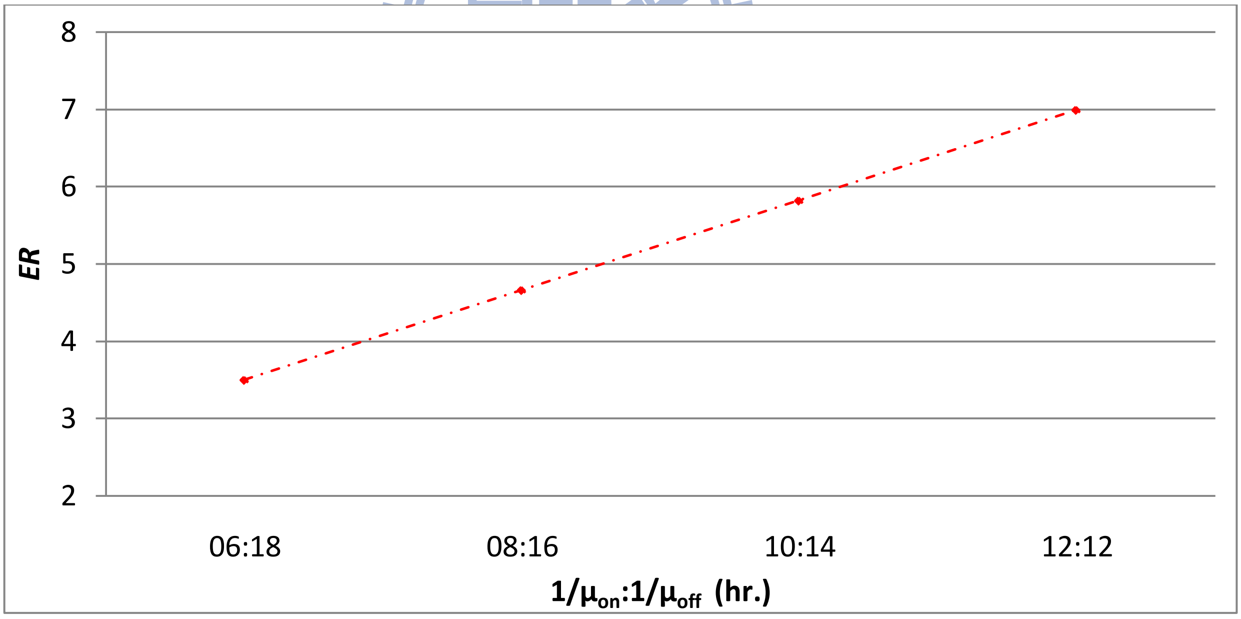


Figure 4-5 The expected number of replicas with $1/\lambda = 4.28$, $1/\mu = 60$, $1/\nu = 60$ (min.)

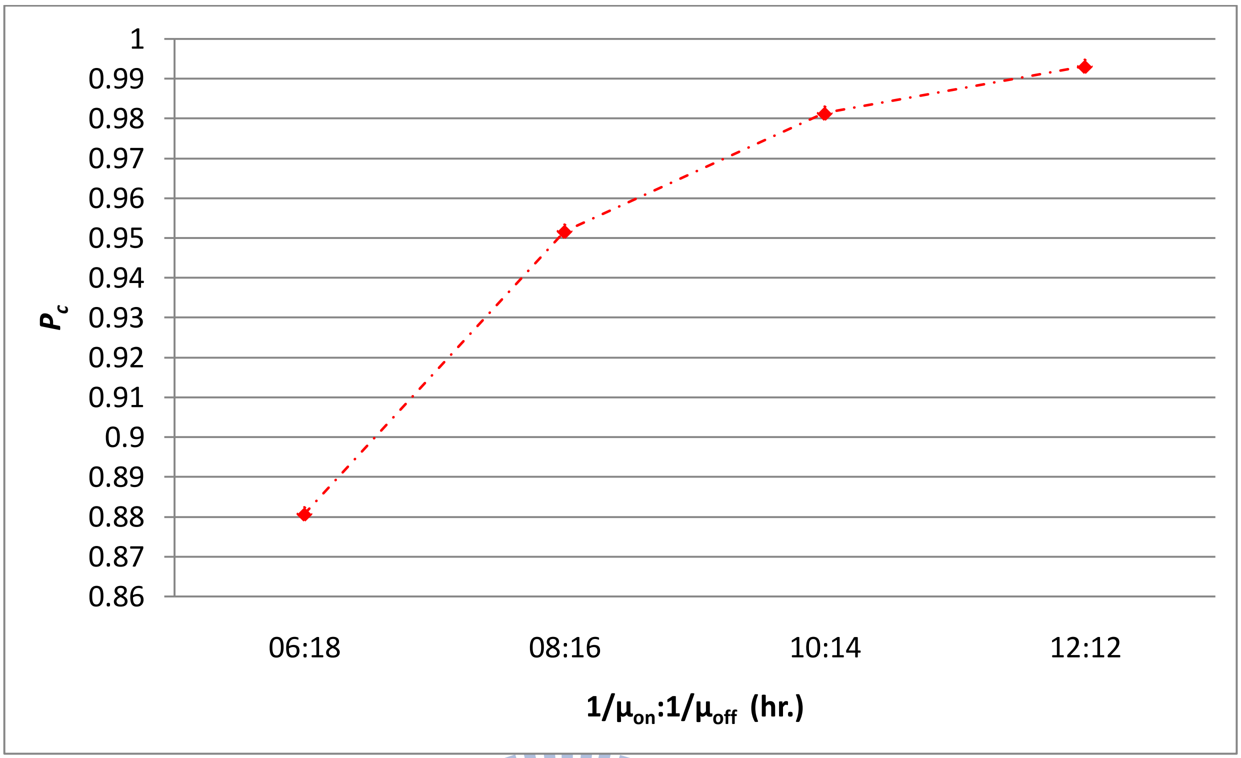


Figure 4-6 Pc with $1/\lambda = 4.28$, $1/\mu = 60$, $1/\nu = 60$ (min.)

Computer simulations have been performed to validate our analytic model. Computer simulation consists of two stages. At stage 1, the arrival and departure records of live viewer were recorded. As depicted in Fig. 4-7, the arrivals of live viewers form a Poisson process with arrival rate λ and a live viewer watches the live program for an exponentially distributed duration mean $1/\mu$. For each live peer, we record its arrival and departure times. Stage2 simulates the on-and-off alternating renewal process of the live peers, and time-shifted viewers retrieving the desired contents. As depicted in Fig. 4-8, the on-line or off-line status of each live peer is determined independently with probability P_{on} being on-line. For each peer, its alternating on-line and off-line periods were generated with mean $1/\mu_{on}$ and $1/\mu_{off}$, respectively. For each peer, we record the on-line and off-line events. A time-shifted viewer randomly chooses an initial video position, and a viewing duration exponentially distributed with mean $1/\nu$. The peers' arrival and departure records at the chosen position and the peer's on-line and off-line records are compared to verify if there are on-line peers to provide the contents to the time-shifted viewers for the entire duration.

We randomly generate and record 1000 sets of LS peers' events. Each set contains LS peers' arrival and departure events in a three-day period. For each event set, 10000 TS viewers view the three-day video content starting from independently random positions. The simulation and analytic results are plotted in Figs. 4-9 and 4-10. We can see that the analytic and simulation results match. This validate our analytic model.

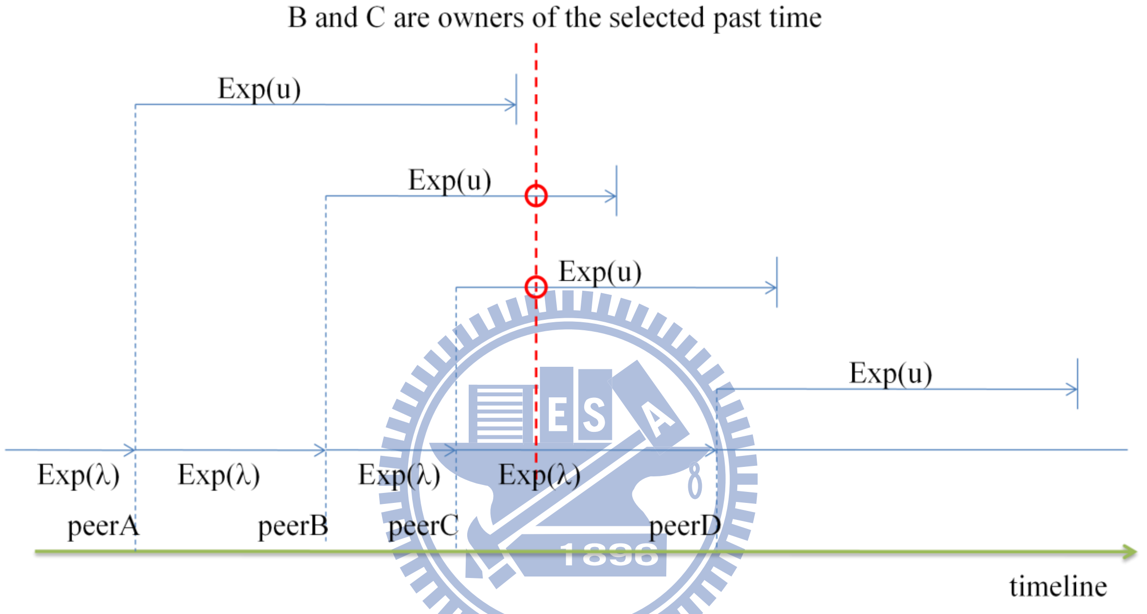


Figure 4-7 The arrivals and departures of LS viewers

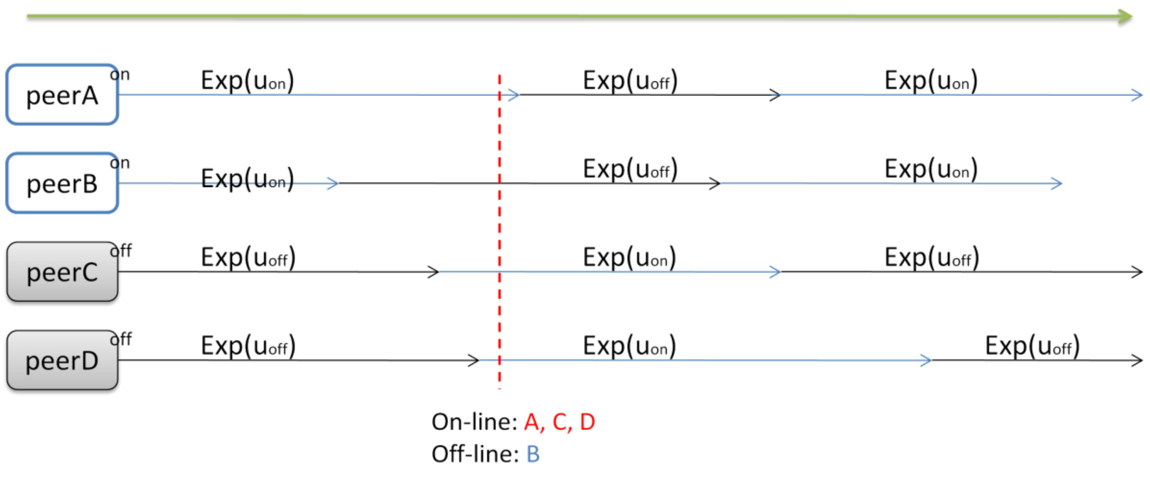


Figure 4-8 Simulation with LS peers getting online and offline

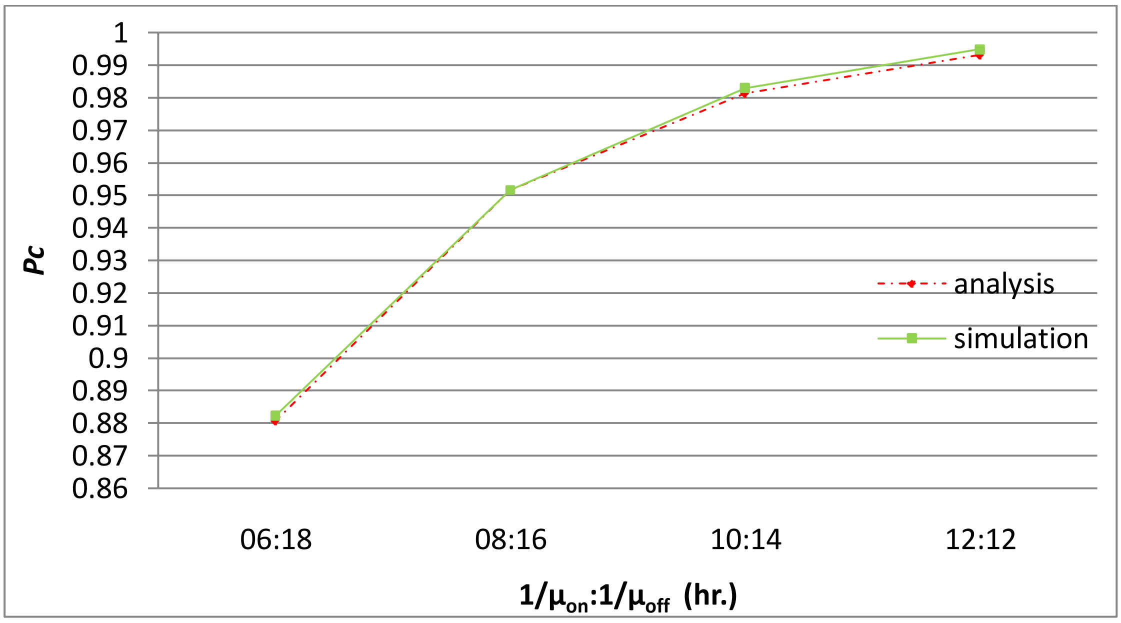


Figure 4-9 Pc with 1/λ = 4.28, 1/μ = 60, 1/ν = 60 (min.)

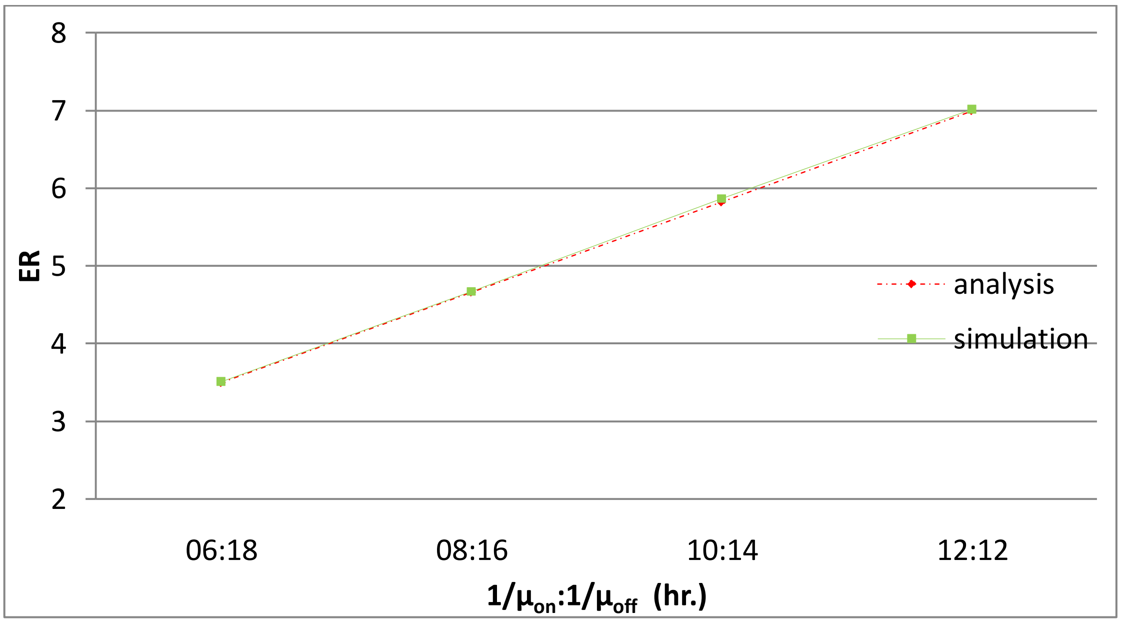


Figure 4-10 The expected number of replicas with 1/λ = 4.28, 1/μ = 60, 1/ν = 60 (min.)

Chapter 5
Performance Measurements

5.1 Experiment Environment

To evaluate the system performance, we performed experiments on PlanetLab, an open global research network [20]. A channel provider was located in the Internet Communication Laboratory, NCTU. The number of PlanetLab nodes we use is not always the same in every experiment because of the availability of the nodes. The total number of nodes was targeted at 120; half of them were live peers, and the other half were time-shifted peers. The peers were located in the United States, European and East Asia. The bit-rate of the video stream is 400 kbps, the number of sub-streams is 8, and each peer can connect to up to 24 peers as partners. The buffer size of each peer is 120 video blocks, i.e., 120 seconds. The system intends to keep 10 replicas for each video block. Time-shifted peers cache each received block with probability 0.5. Table 4-1 lists the system parameters used in our system.

Table 5-1 Experiment Parameters

System parameters	Value
Video streaming bit-rate	400 kbps
The number of sub-streams	8
The maximum number of partners	24
The number of replicas to keep	10
Buffer size	120 blocks

In the experiment, we first started the bootstrap server and the streaming provider, and then peers joined the system as in a Poisson process, with an expected inter-arrival time of 60 seconds. Whether a PlanetLab node is chosen to be a live or a time-shifted viewer is determined in advance. For each time-shifted peer, it randomly selects streaming playback position between the beginning of the streaming and the current streaming. The experiment lasts 2 hours, and no peer churns.

5.2 System performance and Analysis

5.2.1 The live streaming

First, we examine three commonly used criterions in evaluating a live streaming service: startup delay, end-to-end delay and playback continuity. The startup delay is the time after a user tunes to a channel, and before the video content can be played out. End-to-end delay, also referred to as playback delay, is the average transmission delay of video packets between the source and the viewer peers. Continuity index is the number of blocks that arrive before the playback deadlines over the total number of blocks that a peer should receive.

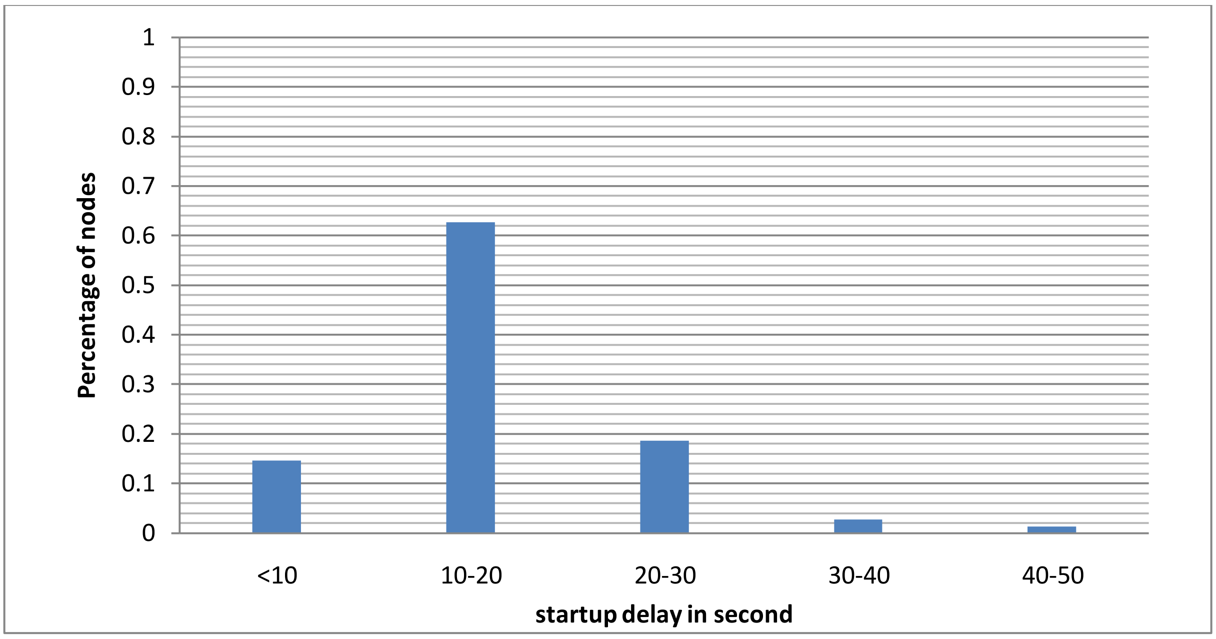


Figure 5-1 The startup delay distribution

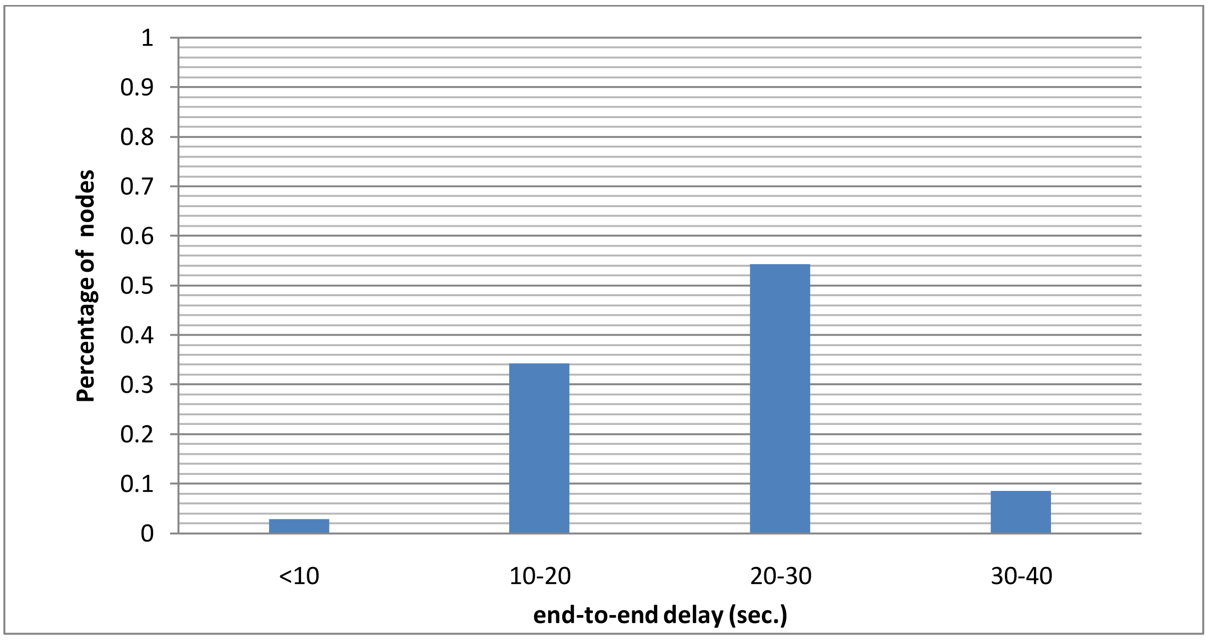


Figure 5-2 The end-to-end delay distribution

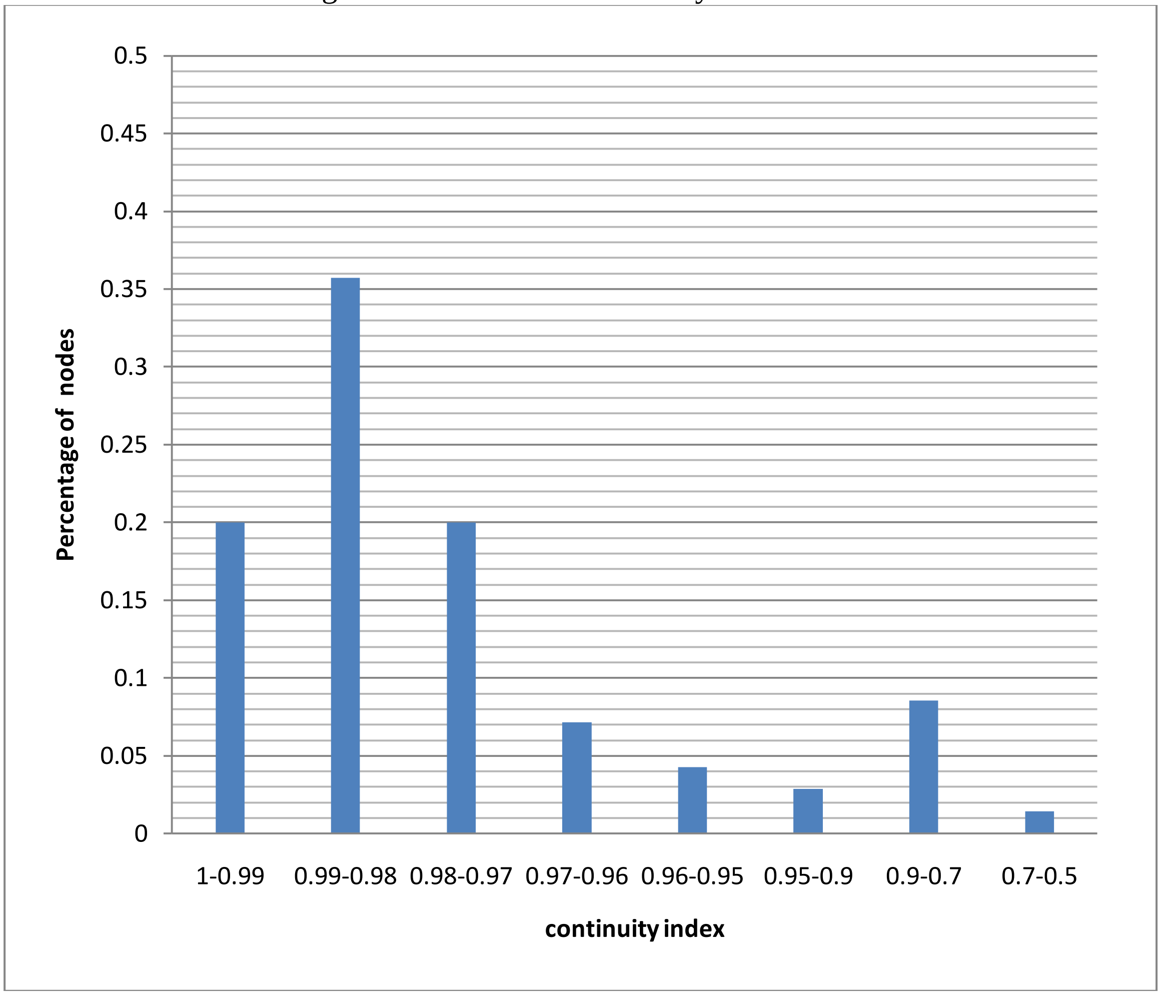


Figure 5-3 The continuity index distribution

Figure 5-1 depicts the distribution of startup delay of the live streaming peers in the experiment. Most of the peers experience a startup delay less than 20 sec.; the average startup delay is 16 sec., which is a satisfactory result for P2P live streaming service. Figure 5-2 depicts the distribution of the end-to-end delay of the live streaming nodes. Most of the nodes experience an end-to-end delay less than 30 sec., and the average end-to-end delay is 22 sec. Note that the total number of peers in Fig. 5-2 is different from that of Fig. 5-1. It is because peers with negative end-to-end delay, which is impossible, is eliminated from Fig. 5-2. The negative end-to-end delays were the results of un-calibrated NTP time on the PlanetLab nodes, but we do not have the authority to adjust the NTP time. Fig. 5-3 depicts the distribution of continuity index of live streaming nodes. 75% of them achieve over 97% continuity index; the average continuity index is 96%.

5.2.2 The time-shifted streaming

To serve time-shifted peers, live streaming video files collected by the live peers need to be published on the DHT properly. First, we examine the probability that a file is successfully published. The probability can be obtained by dividing the total size of the owner lists of all video files on the DHT by the total number of video files cached by live peers. Note that all the owner lists are retrieved from the DHT at the end of the experiment. In the experiment on PlanetLab described above, the probability of success publishing is 97.39% , i.e., only less than 3% of the cached video files are not successfully published. Fig. 5-4 depicts the number of failures on published files. The number of failures of each video file is the difference between the number of replicas cached by peers and the size of the owner list on the DHT. 74.2% of the files experience no publishing failure and 25.8% experience one publishing failure. Only three of them experience more than one publishing failures, 3, 4, and 5 failures respectively. The 3 is an unknown Kademlia failure on provider. The 4 is a failed retrieval of the result data, and the publishing of this file is in fact successful by checking the logs of viewers. The 5 is happened when a provider helps peers who are unable to access the DHT publish a file but races the file entry on DHT with the collector.

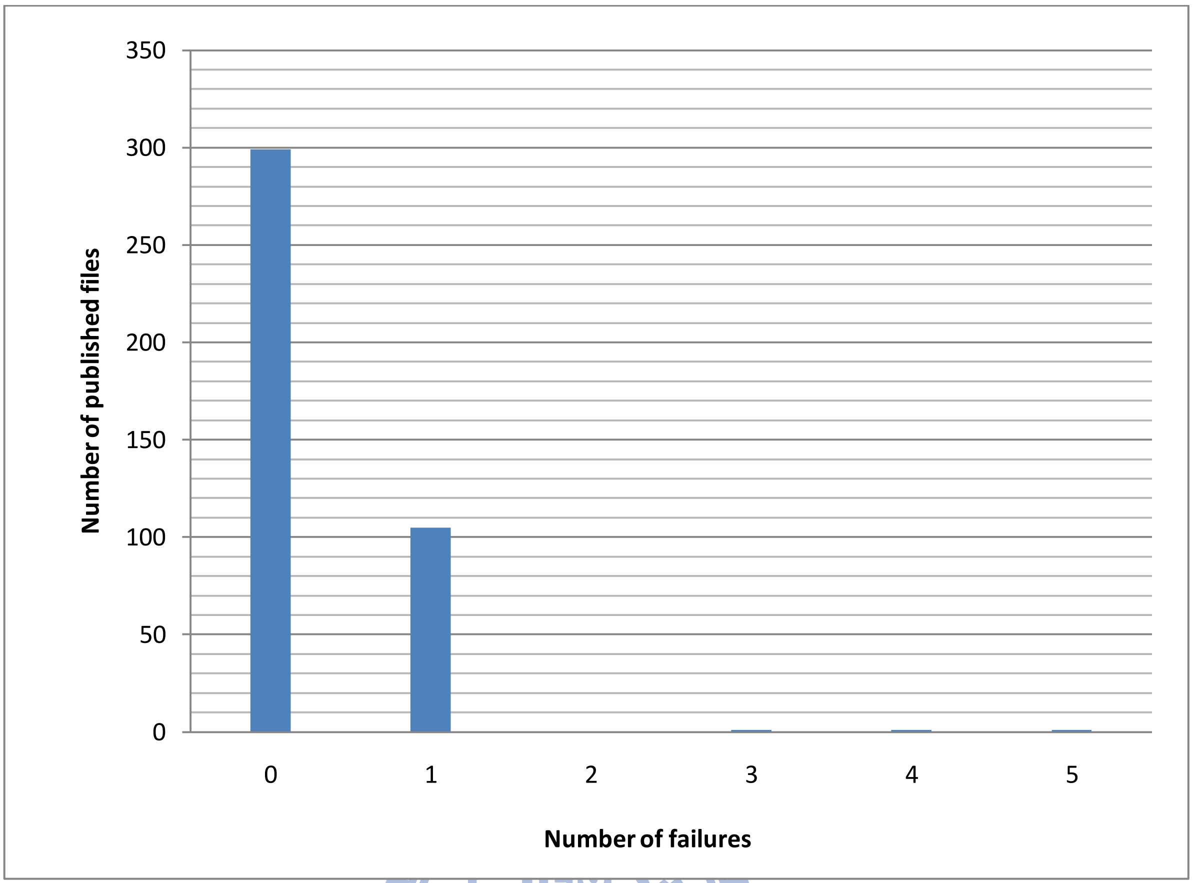


Figure 5-4 The publishing failure distribution of video file

The performance measurements that we are interested for time-shifted streaming include startup delay, the missed ratio of time-shifted blocks, and the provider stress. In our experiment, there were 51 time-shifted peers and 60 live peers. Fig. 5-5 depicts the distribution of startup delay. The average startup delay of TS peers is 16.5 sec. The peak in the interval 26-30 seconds is the consequence of a 30-second startup delay bound in forced starting for TS streaming.

Fig. 5-6 depicts the distribution of the block missed ratios. The total number of video blocks received by all the time-shifted viewers was 176,802. The block missed ratio was only 0.42%, which is very satisfactory. Half of the time-shifted peers experience no missed block. 85% of them achieve a low missed rate less than 1%. Only one peer encounters an unacceptable missed rate of 9.38%. The reason may be the temporary overloading with other people's application on that node, since all nodes on PlanetLab are shared.

The provider stress can be measured by the ratio of blocks that are obtained from the provider. These blocks can furthermore be divided into three categories: emergency pulling, failure to obtain from the owners, no owner found on the DHT. The experiment results in Table 5-1 indicate that only 2.73% of the video blocks were obtained from the provider, i.e., more than 97% of the blocks were served by peers. Among the 2.73% of the blocks obtained from the server, 1.11% were for emergency pulling, 0.28% were due to failures to obtain from the owners, and 1.34% were because of no owner found. The stress on server is quite small, and it results mostly from emergency handling.

Table 5-2 The TS streaming load on the channel provider

From provider	Emergency pulling	2.73%	1.11%
	Failing to obtain from the owners		0.28%
	No owner found		1.35%

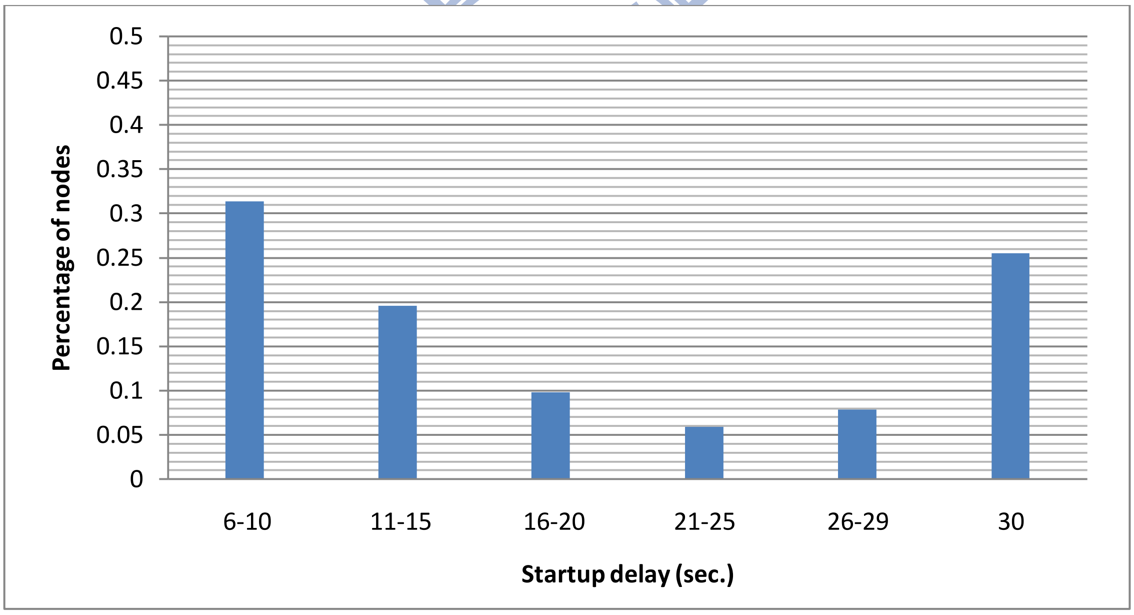


Figure 5-5 The startup delay distribution

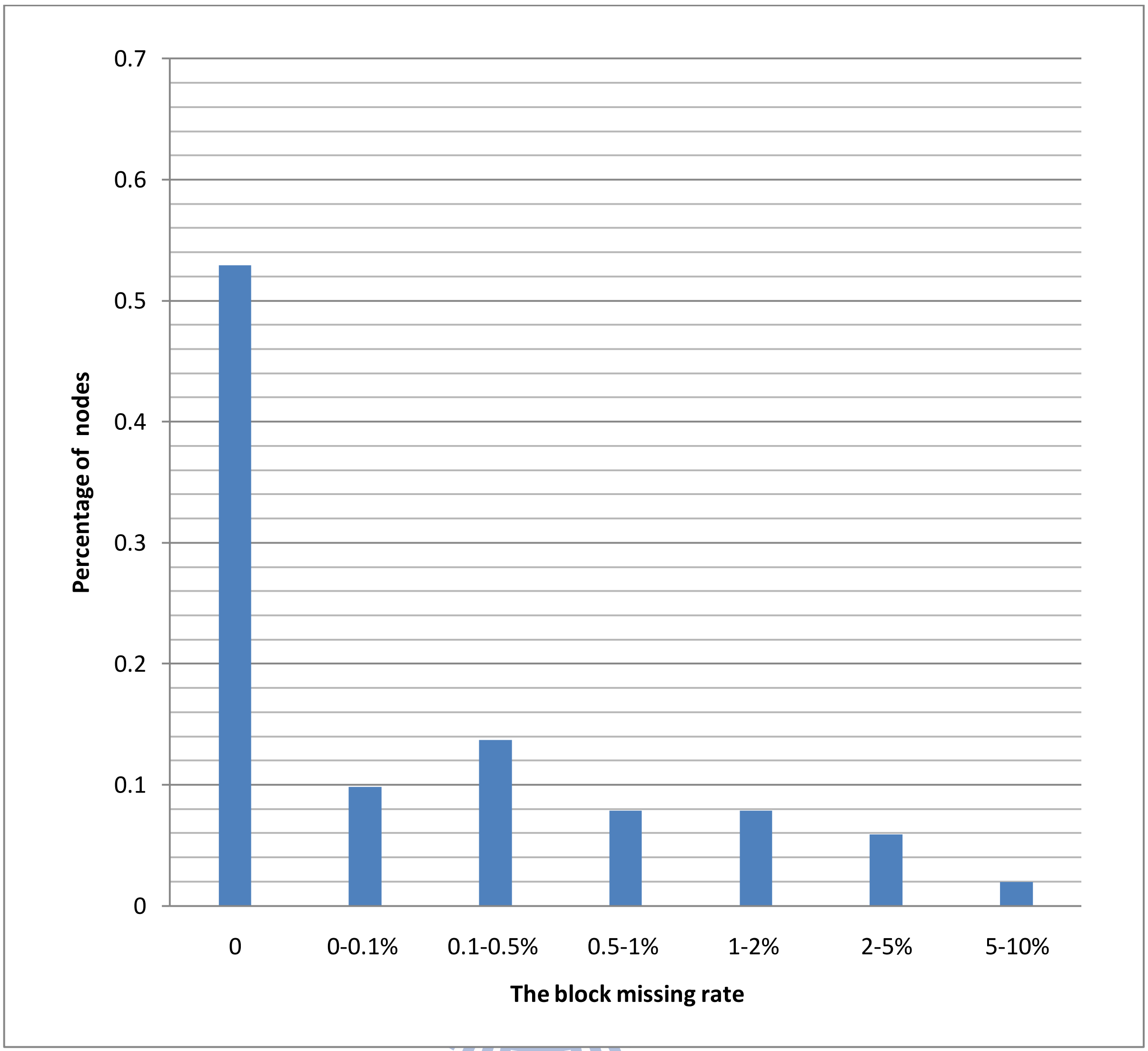


Figure 5-6 The distribution of block missing rates

We also performed another experiment in which time-shifted viewers may change their playback positions. In this experiment, TS viewers watch for a random interval from 15 to 25 minutes, and then change the playback position randomly chosen between the system startup time and the current time. The number of total received blocks was 195632, and the number of participating TS peers in this experiment was 56. Fig 5-7 depicts the distribution of startup delay. The delay in playing out after a TS viewer changes his/her playback position is also considered as startup delay. The average startup delay is 15.0 sec., which is 1.5 sec. shorter than that of the previous experiment. This is because some startup procedures, such as initializing network connections, are only necessary in the first startup.

As a result, the average startup delay decreases.

Table 5-3 The TS streaming load on the channel provider with peers changing playback positions

From provider	Emergency pulling	3.01%	0.75%
	Failing to obtain from the owners		1.01%
	No owner found		1.12%

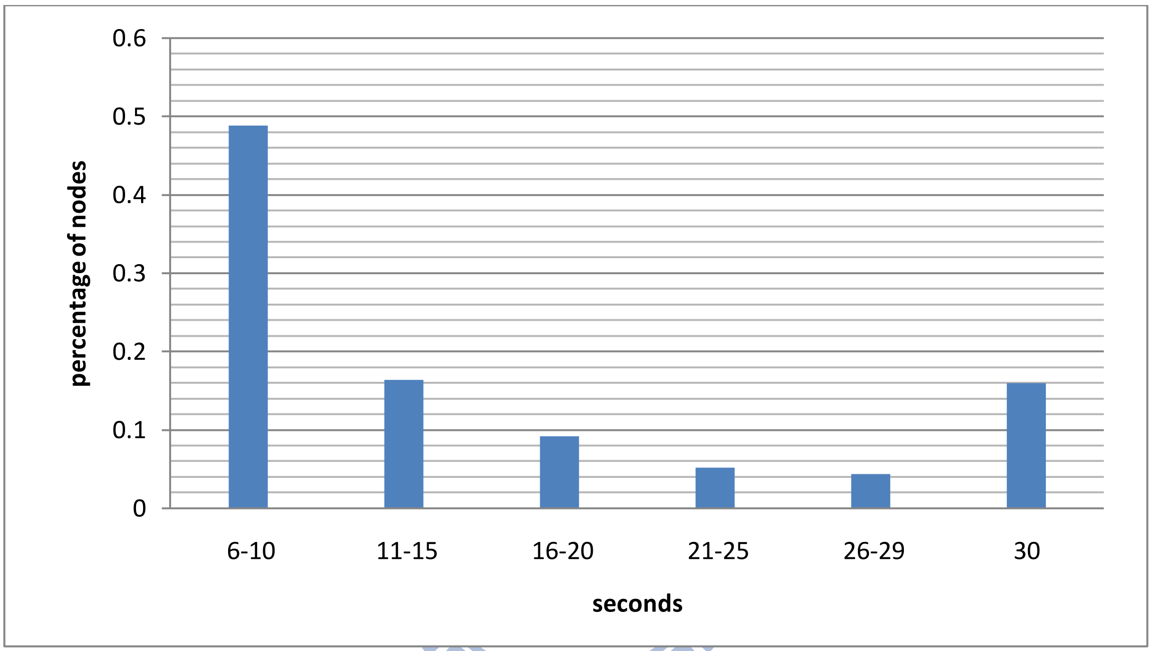


Figure 5-7 The startup delay distribution with changing playback positions

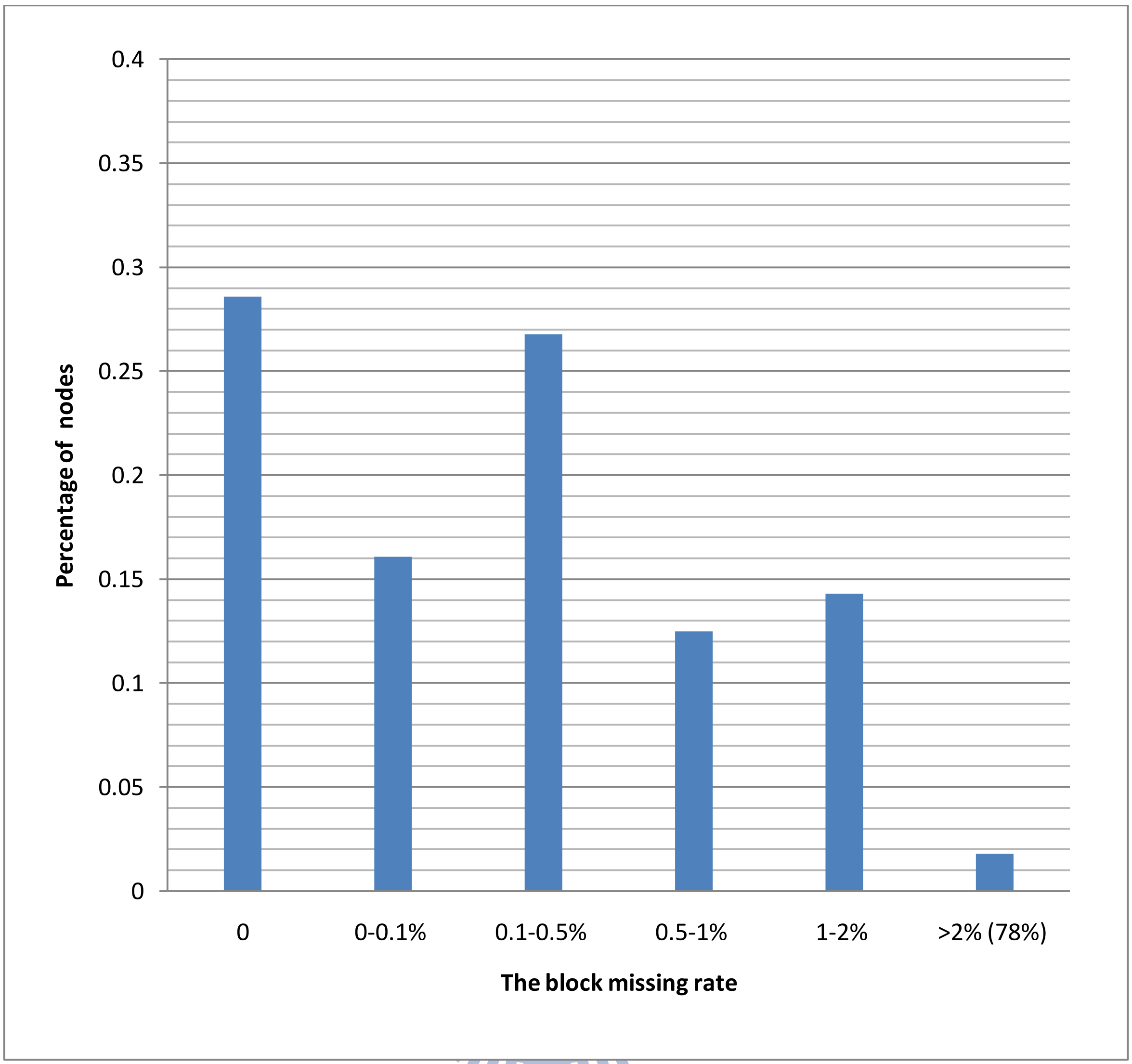


Figure 5-8 The distribution of block missing rate with peers changing playback positions

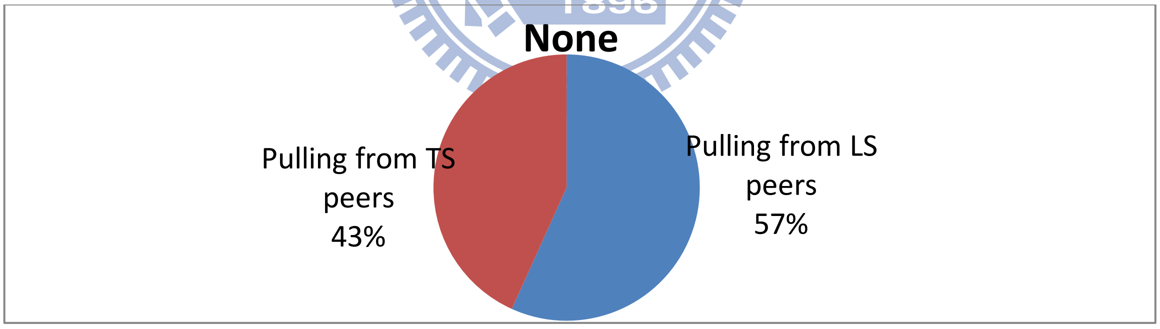
Fig. 5-8 depicts the distribution of the block missing ratios. The average block missed ratio was 0.65%. 84% of all TS peers achieve a low missing rate less than 1%. Only one peer experience an ultra high missed ratio of 78%, which may be due to a temporary overloading on that node.

Table 5-2 lists the percentage of video blocks that is obtained from the provider by TS peers. The results indicate that although time-shifted viewers change their playback positions, only 3% of video blocks were obtained from the provider. The provider stress only increases slightly by 0.3%. It may imply that peers suffer from the more shortage of receiving blocks in the early stage.

The subscription rate is the proportion of blocks received by subscription to the total received blocks. For how efficient the time-shifted stream subscription enhances the cooperation between time-shifted viewers and the block transmission load on LS peers, we performed further experiments to analyze the time-shifted streaming sources. We categorize the experiments into four cases, as indicated in Table 5.4, depending on whether time-shifted stream subscription and/or changing playback positions are allowed or not. The way that TS peers change playback positions is the same as the previous experiment, and the number of time-shifted peers are also around 60.

Table 5.4 The categories of experiments to analyze the time-shifted streaming sources

	With no peer changing playback positions	**With** peer changing playback positions
With no TS subscription	None	CP
With TS subscription	Sub	CP+Sub



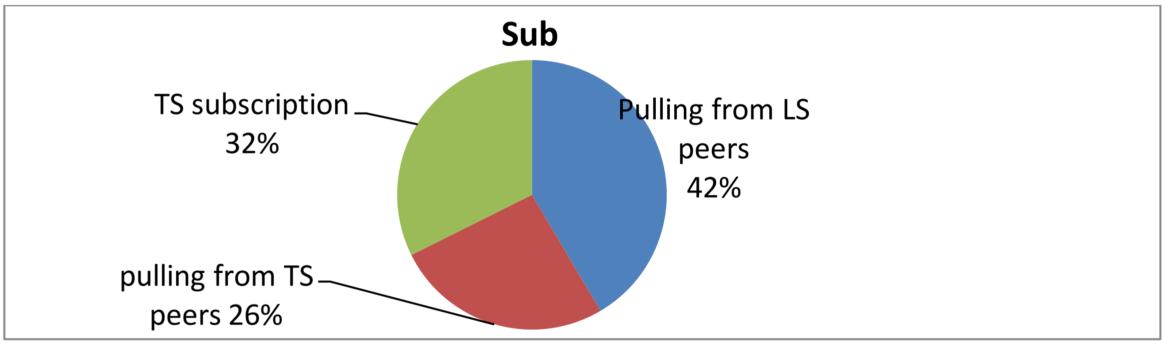


Figure 5-9 The effects of TS subscription with no peer changing playback positions

Fig. depicts the percentage of video blocks from different sources for case (1) and case (2).

32% of video blocks were provided from the suppliers by time-shifted stream subscription. The amount of reducing blocks is 57% (percentage of total blocks pulled from live in case (1)) – 42%(percentage of total blocks pulled from live in case (2)) = 15% of total, which is approximately a half of the subscription rate 32%. It could be thought of as a half from live viewers a half from time-shifted viewers by instinct in the simple case

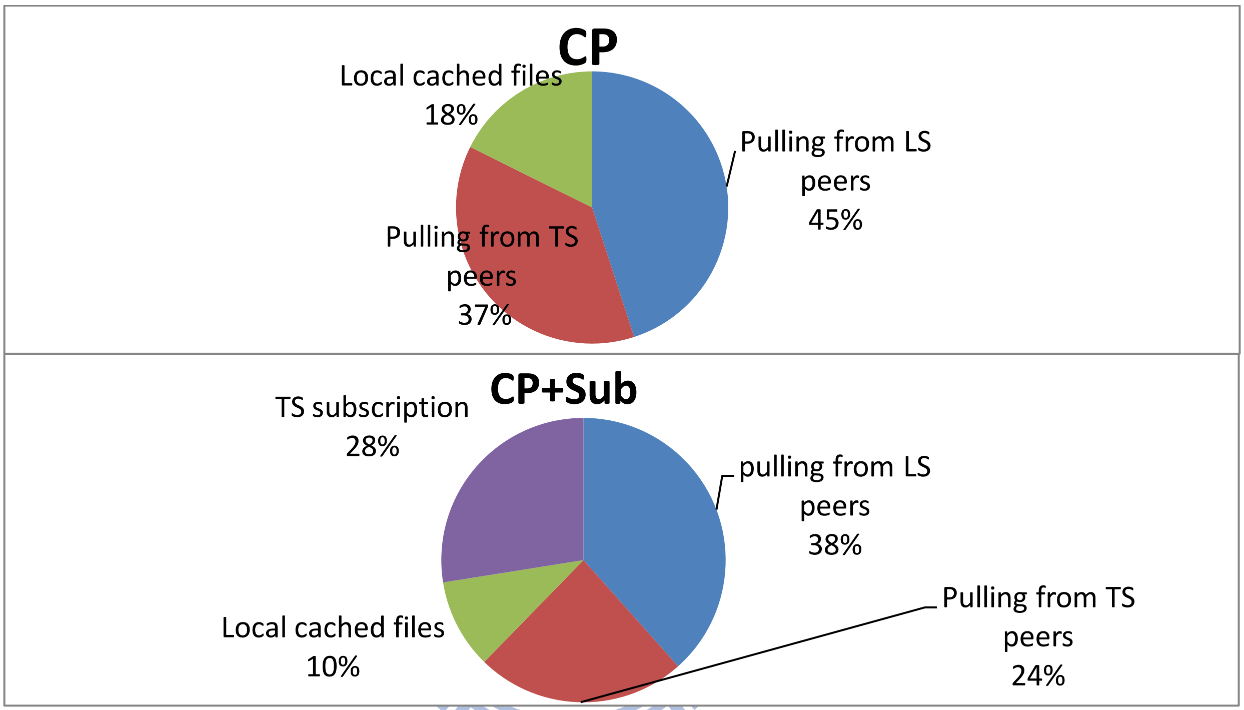


Figure 5-10 The effects of TS subscription with peer changing playback positions

In the case of peer changing playback positions, peers have a chance to rewind to the previously watched video segment, and some of the video contents may be cached on local storage. Peers just need to load them from local file system. Though it seems in the case peer changing playback positions reduce the influence on time-shifted subscription, it could be thought of as the local cached files take over the responsibility for reducing peers pulling from live viewers. It is a consequential result of the setting of changing playback positions in such the range.

Chapter 6

Conclusions

In this thesis, we have developed an analytic model to estimate the number of video replicas required to support TS streaming. Our analytic results indicate seven on-line replicas are sufficient to serve 99.5% of TS viewers obtaining all the needed video. The desired number of replicas is 10 in our PlanetLab experiment. We had implemented a P2P live/time-shifted streaming system and presented a distributed cache management strategy to cache a desired number of time-shifted streaming replicas for time-shifted streaming and an indirect publishing mechanism to improve the successful publishing rate. The time-shifted stream subscription further enhances the cooperation between time-shifted peers. We also studied the performance of the system on PlanetLab. Our experiment results show the feasibility of live/time-shifted P2P streaming systems. In our experiments, the live streaming achieved a startup delay of 16 seconds, an end-to-end delay of 22 seconds, a continuity index over 96% and a successful publishing rate over 97%.. For the time-shifted streaming, with the video provider as an emergency handler, we achieved a block missing rate of 0.42%, with less than 3% of the streaming data from the provider. The time-shifted stream subscription mechanism reduces 15% load on live peers when TS peers do not change playback positions, and 8% when they change playback positions.

However, in this implementation, there are further issues on the time-shifted stream subscription mechanism to discuss. First, the size of playback buffer is intuitively related to the subscription rate. As the larger the buffer size is, the more likely the time-shifted peers' buffers overlap with each other. Second, the behaviors of time-shifted peers would affect the performance of the time-shifted streaming system. How far away and how frequently a viewer changes the playback position is still unknown to us, but this indeed would affect the effectiveness of TS subscription. Third, whether all the peers join the DHT or only peers with long on-line intervals join is also of merits to study. More investigation and experiments of the system would provide more insightful knowledge on P2P time-shifted streaming services.

References

[1] F. Douglis and M.F. Kaashoek, "Scalable Internet Services," IEEE Internet Computing, vol. 5, no. 4, 2001, pp. 36–37.

[2] A. Vakali, and G. Pallis, "Content delivery networks: status and trends" IEEE Internet Computing, vol. 7, no. 6, 2003, pp. 68-74.

[3] Xinyan Zhang, et al., "CoolStreaming/DONet: a data-driven overlay network for peer-to-peer live media streaming" INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE, vol. 3, pp. 2102-2111. Mar. 2005

[4] Li Zhao, et al., "Gridmedia: A Practical Peer-to-Peer Based Live Video Streaming System" Multimedia Signal Processing, 2005 IEEE 7th Workshop on, pp. 1-4. Nov. 2005

[5] Bo Li, et al., "Inside the New Coolstreaming: Principles, Measurements and Performance Implications" INFOCOM 2008. The 27th Conference on Computer Communications. IEEE, pp. 1031-1039, Apr. 2008

[6] V. N. Padmanabhan, et al., "Distributing streaming media content using cooperative networking," in Proc. 12th international workshop on Network and operating systems support for digital audio and video, pp. 177-186. Apr. 2002.

[7] M. Castro, et al., "Splitstream: High-bandwidth content distribution in a cooperative environment," in Proc. nineteenth ACM symposium on Operating systems principles, pp. 292-303. Oct. 2003.

[8] V. Venkataraman. K. Yoshida, and P. Francis, "Chunkyspread: Heterogeneous Unstructured Tree-Based Peer-to-Peer Multicast" Network Protocols, 2006. ICNP '06. Proceedings of the 2006 14th IEEE International Conference on, pp 2-11. Nov. 2006

[9] Yang Guo, et al., "P2Cast: Peer-to-peer Patching Scheme for VoD Service" Multimedia Tools and Applications, vol. 33, pp. 109-129, 2007

[10] T.T. Do, K.A. Hua, and M.A. Tantaoui, "P2VoD: providing fault tolerant video-on-demand streaming in peer-to-peer environment" Communications, 2004 IEEE International Conference on, vol. 3, pp. 1467-1472, Jun. 2004

[11] Yi Cui, Baochun Li, and K. Nahrstedt, "oStream: asynchronous streaming multicast in application-layer overlay networks" Selected Areas in Communications, IEEE Journal on, vol. 6, no. 1, Jan. 2004

[12] C. Dana et al., "BASS: BitTorrent Assisted Streaming System for Video-on-Demand" Multimedia Signal Processing, 2005 IEEE 7th Workshop on, pp. 1-4. Nov.2005

[13] Yang Guo et al., "PONDER: Performance Aware P2P Video-on-Demand Service" Global Telecommunications Conference, 2007. GLOBECOM '07. IEEE, pp. 225-230, Nov. 2007

[14] F.V. Hecht et al., "LiveShift: Peer-to-Peer Live Streaming with Distributed Time-Shifting" Peer-to-Peer Computing , 2008. P2P '08. Eighth International Conference on, pp. 187-188, Sept. 2008

[15] S. Deshpande, and J. Noh, "P2TSS: Time-shifted and live streaming of video in peer-to-peer systems" Multimedia and Expo, 2008 IEEE International Conference on, pp.649-652. Jun. 2008

[16] Hou Xiuhong, Ding Ruipeng, "Research of providing live and time-shifting function for

structured P2P streaming system” Machine Vision and Human-Machine Interface (MVHI), 2010 International Conference on, pp. 239 - 242. April. 2010

[17] S. Deshpande, and J. Noh, “Pseudo-DHT: Distributed Search Algorithm For P2P Video Streaming” Multimedia, 2008. ISM 2008. Tenth IEEE International Symposium on, pp. 348 - 355. Dec. 2008

[18] Weihui, Junpeng Xu, Cong Qing, “Dynamic Multicast Tree to Support time-shifting in Real-Time Streaming” Intelligent Computing and Cognitive Informatics (ICICCI), 2010 International Conference on, pp. 251 - 254. June 2010

[19] P. Maymounkov and D. Mazi`eres, “Kademlia: A peerto- peer information system based on the XOR metric.” Electronic Proceedings for the 1st International Workshop on Peer-to-Peer Systems, Mar. 2002

[20] Plan-x, http://www.thomas.ambus.dk/plan-x/routing/

[21] VideoLAN – VLC Media Player, http://www.videolan.org/vlc/

[22] PlanetLab, http://www.planetlab.org