國立交通大學

資訊科學與工程研究所

碩 士 論 文

P2P 實況/移時串流系統之實作與分析

Implementation and Analysis of P2P Live/Time-shift Streaming System

研 究 生：李茂弘

指導教授：張明峰　教授

中 華 民 國　九 十 九 年 七 月

P2P 實況/移時串流系統之實作與分析
Implementation and Analysis of P2P Live/Time-shift Streaming System

研 究 生：李茂弘　　　　Student：Mao-Hung Lee

指導教授：張明峰　　　　Advisor：Ming-Feng Chang

國 立 交 通 大 學
資 訊 科 學 與 工 程 研 究 所
碩 士 論 文

A Thesis

Submitted to Institute of Computer Science and Engineering

College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer Science

July 2010

Hsinchu, Taiwan, Republic of China

中華民國九十九年七月

P2P 實況/移時串流系統之實作與分析

學生：李茂弘　　　　　　　　　　　　　　　　　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

隨著寬頻網路的普及，多媒體串流服務近年來已成為一蓬勃發展的網際網路應用，但此類系統的可擴展性一直是個問題。點對點式架構已經是解決可擴展性問題中最有潛力的方法之一，並應用在許多的多媒體實況串流服務以及隨選視訊串流服務之中。

雖然目前已經有相當多的點對點影音串流研究，但是點對點的移時串流服務，也就是提供使用者能夠在沒有預錄的情況下觀看任意過去時間點上的多媒體串流服務，目前只有少數研究。

於此篇論文中，我們將實作一點對點實況/移時串流系統，提出針對移時服務串流資料塊的分散式的快取管理策略，以及在 PlanetLab 平台上進行實驗，了解該類系統的可行性及特性，以及所提出策列之效能。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產這類型的多媒體串流服務。

Implementation and Analysis of P2P Live/Time-shift Streaming System

Student：Mao-Hung Lee Advisor：Dr. Ming-Feng Chang

Institute of Computer Science and Engineering
National Chiao Tung University

ABSTRACT

With the increasing prevalence of broadband Internet access, multimedia streaming services have been among the most popular Internet applications in recent years, but the system scalability has always been an issue. P2P architecture has been one of the most promising solutions addressing the scalability problem, and has been widely applied on live streaming services and video-on-demand (VoD) services.

However, currently there are very few studies in P2P streaming systems that provide time-shift streaming services, where users can watch video streaming programs with an arbitrary offset of time. In this thesis, we design and implement a P2P live/time-shift streaming system and propose two distributed cache management strategies for time-shift video cache files. In addition, we study its performance and characteristics on the PlanetLab experiment platform, Our experiment results show the feasibility of P2P time-shift video streaming systems and the effectiveness of the proposed strategies. We believe our work can provide valuable insightful knowledge of P2P live/time-shift streaming systems for future developments on this kind of streaming services.

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，祐村、岳廷，還有境余以及順胤學弟，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。

李　茂　弘　謹識於
國立交通大學資訊科學與工程研究所碩士班
中華民國九十九年七月

Table of Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Overview

With the increasing prevalence of broadband Internet access, multimedia streaming service has been a rising internet application in recent years, but the system scalability has always been an issue. In the early stage of media streaming, client-server architecture suffers from scalability problem; as the request increases, the system is quickly overloaded [1]. Content delivery networks (CDNs) with strategically placed proxies can balance the load, but it is too costly for general applications [2]. IP multicast being probably the most efficient vehicle, its deployment, however, is very limited due to many practical issues such as the lack of IP multicast supporting infrastructures and incentives for them to carry the data traffic [3]. Application-level multicast, by constructing an overlay network with unicast connections between nodes in the system, has been proposed to deal with the scalability issue and used in many Internet applications.

1.2 Motivation

Currently, there are mainly two types of streaming services: live streaming and VoD (video on demand) streaming. Live streaming is similar to watching TV; users tune to a selected channel, and the users tuning to the same channel synchronously receive the same content. On the other hand, in VoD streaming, a user selects a video clip the user wants to watch, at the time the user wants to start watching, so the contents delivered to the users are asynchronous.

Another type of streaming service is time-shift streaming service, which provides the ability for user to watch contents in the past, like a digital video recorder (DVR) but user does not have to set which program should it record. P2P time-shift streaming can be taken as a special case of P2P VoD streaming. In P2P VoD streaming, videos are pre-generated and their lengths are known, which makes it possible for dedicated servers to hold the whole video, while in P2P time-shift streaming, the contents are generated in real-time with infinite length, thus making peer cache management strategy an interesting problem.

1.3 Objective

To date, there are very few researches on streaming systems supporting both live streaming and time-shifted streaming. In this thesis, we will implement a P2P system supports both live streaming and time-shift streaming functionality and propose a distributed cache management strategy. In addition, we will also perform experiments on the PlanetLab platform to study the performance and characteristics of our system. We believe our work may provide valuable knowledge of P2P live/time-shift streaming system for further development on this kind of streaming services.

1.4 Summary

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 the experiment setup, results, system performance and analysis. Finally, we give our conclusions in Chapter 5.

Chapter 2

Related Work

There have been comprehensive studies on P2P systems. In this chapter, we will first briefly describe the current developments of P2P live streaming systems, P2P VoD streaming systems, and P2P streaming systems with time-shift function, and then we will discuss the overlay topologies used in P2P streaming systems and their data delivery mechanisms.

2.1 P2P streaming overlay

P2P streaming technologies can be broadly divided into two classes: tree-based approaches and mesh-based approaches, following is a brief description.

2.1.1 Tree-based

In tree-based overlay, nodes are connected to form a tree-shaped graph, with the source node as the root and peer nodes as interior nodes or leaf nodes, establishing parent-child relations. Parent nodes are responsible for sending the streaming data to their children. Single-tree structure is the simplest form of this type of structures. The advantage of the tree structure is that the transmission delay is usually shorter because the streaming data is transmitted along the fixed paths. However, there are immediate visible defects. First, when an interior node fails, its offspring nodes are disconnected from the source and cannot receive streaming data immediately. The tree must be rebuilt, causing extra overhead. Second, most nodes in the system are leaf nodes, but since they have no children nodes, they cannot provide its uplink transmission capacity to the system. To solve the mentioned problems, multiple-tree overlay has been proposed. By transmitting part of the streaming content with independent multicast trees, the system distributes the forwarding load on every node and hopefully minimizes the effect of peer churn on the disruption of streaming data..

2.1.2 Mesh-based

In mesh-based overlay, each node is connected to partial nodes in the system forming a mesh distribution graph. Since there is no parent-child relationship between connected nodes, a common strategy is that connected nodes exchange the availability information of the streaming data periodically, and then request their required data from the nodes owning the missing data. Mesh-based systems may have longer setup delay and need extra control messages, such as the data availability information and pull messages for missing streaming data. However, the self-organizing characteristic makes them robust to node failures and peer churn.

2.2 P2P streaming data delivery mechanisms

Three different data delivery mechanisms have been used in P2P streaming systems: push mechanism, pull mechanism, and hybrid push-pull mechanism.

2.2.1 Push mechanism

Using push mechanism, when a node receives data, it pushes the data to other nodes in the network without explicit requests from these nodes. Since this mechanism has no requests for data, it reduces control message overhead and shortens the setup delay, but it is also costly to recover from lost data or lost connection. For example, if the connection between two nodes is broken, the streaming data cannot be transmitted across the broken connection, and the topology must be rebuilt.

2.2.2 Pull mechanism

Using pull mechanism, a node pulls its required data by sending requests to other nodes. With the capability to pull, the system is robust to lost data or lost connection, but the message overhead in requesting every single data block has also make it suffer a longer setup delay, and the pulling operations should be scheduled carefully to avoid redundant data transmission. For example, a request is made with an overloaded node and the requested data are not transmitted in time. The requester may make another request with another node, and consequently receive duplicate data blocks.

2.2.3 Hybrid push-pull mechanism

The hybrid push-pull mechanism extracts the advantages from both the push mechanism and the pull mechanism. This hybrid mechanism is used in GridMedia [4] and the new version of CoolStreaming [5]. In GridMedia, the node first pulls the data it needs. When it detects the pulling procedure is smooth, it then tells the sending peer to push data to it. In the new version of CoolStreaming, when a node pulls data from another node, a subscription-like contract is made, and the following data will be pushed to the subscriber until the contract ends, for instance, if the subscriber un-subscribes the streaming or the parent node has detected the connection with the child node is broken.

2.3 Current development

2.3.1 P2P live streaming

In P2P live streaming service, peers requires the synchronized delivery of streaming media content, so the issue here is to form the overlay structure and adopt a content delivery mechanism. CoopNet [6] adopts a centralized model; the source node is responsible to collect information from the joining nodes and maintain a multi-tree structure. Using a multiple-description-coding (MDC) technique, each tree to transmit different MDC descriptions. However, CoopNet is not a pure P2P system, but a complement to a client-server framework; the multi-tree overlay is only invoked when the server is unable to handle the load imposed by clients.

In SplitStream [7], the streaming content is split into multiple stripes and independent multicast trees are constructed for delivering each stripe. By constructing a forest of multicast trees such that an interior node in one tree is a leaf node in all the remaining trees, the forwarding load can be evenly spread across all participating nodes, but such node-disjointness is a property hard to achieve, especially in heterogeneous environments [8]. In GridMedia, the bootstrap procedure uses a rendezvous point to assist the bootstrap of the overlay. A newly joined node first contacts the rendezvous point to obtain a list of nodes that already joined the overlay. Then, it measures the end-to-end delay to each node in the list and selects a number of node as partners, with the probability of a node is selected is in inverse to the end-to-end delay, thus making nodes nearby more likely to be selected. In DONet/CoolStreaming [3], a newly joined node first contacts an origin node and the origin node randomly selects a deputy and redirects the new node to the deputy. The new node can obtain a list of partner candidates from the deputy and establish partnership with these candidates. In the system, the video stream is divided into segments of uniform length, and the availability of segments in the buffer of a node is represented as a bitmap called Buffer Map (BM). Each node continuously exchanges its BM with its partners and then schedules the pulling operation accordingly. The scheduling algorithm took both availability and partners' upload ability into consideration; the block with 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.

2.3.2 P2P VoD streaming

Video-on-Demand (VoD) service provides users the functionality to watch whatever and whenever they want. Here, the issue is what should a peer caches to support the system, and how to find such cached content in the system. In P2Cast [9], peers watching the same video clip within a time threshold form a session in single-tree fashion, each peer caches the beginning part of the video and a newly joined peer can be patched with the cached beginning part and its parent's buffer contents. In P2Vod [10], peers form generations, where in each generation, peers have synchronized buffer start. A newly joined peer will try to join a generation, or form a new generation appended to the older generation. Generations are numbered, from G1 as the oldest generation and Gn as the youngest generation. Nodes in these generations excluding the server form a video session. In a session, if there is no client that still has the first block of the video, the session will be closed, and a new video session is needed for newly joined clients. Both P2Cast and P2Vod only support start-from-beginning VoD viewing. oStream [11] provides peers the ability to watch from arbitrary positions, but because the system inserts new peers into the system, video disruption will be noticeable on the child nodes of the new peers.

BASS [12] applied BitTorrent protocol to download video content, with the VoD server to support emergency content, which is too close to the playback deadline but is not arrived yet. The simulation result shows the mechanism helps reducing 34% of the bandwidth of the serverwhen users' average outgoing bandwidth is about the same as video bit-rate. However, the required bandwidth from the server still still increases lineraly as the number of users increases. PONDER [13] also applied mesh-based approach similar to BitTorrent, and adopts new mechanisms to accommodate VoD service. While BitTorrent treats all data unit, 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 download priority so that the urgent data can be downloaded first. PONDER also gives up the tit-for-tat incentives; peers are served based only on their needs without considering their contributions. This maximizes the amount of data that 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 for users' average outgoing bandwidth being 112% of the video bit-rate.

2.3.3 P2P live streaming with time-shift streaming support

To the best of our knowledge, P2TSS [14], LiveShift [15] and an IPTV variation [16] are the few researches on providing both live streaming and time-shift streaming. P2TSS presents two distributed cache algorithms: Initial Play-out Position Caching (IPP) and Live Stream Position Caching (LSP). It allows peers to decide which video block to be cached locally and shared with other peers. Their simulation results indicate that P2TSS achieves low server stress by utilizing the peer resource.However, in IPP, the availability is not uniform for each video block, while in LSP, though the availability is uniform for each video block, it requires extra bandwidth and more connections for each peer to fill its distributed streaming cache.

LiveShift is a software prototype. It is a live streaming system based on a multiple tree overlay. As a peer watches the video and the video data reaches a predefined size, the data is stored and the peer adds a reference to the segment in a DHT. Although they have presented a demonstration scenario, there is no detailed analytic results of the system.

IPTV is an integrated media delivery architecture that provides four basic functionalities of video delivery: linear TV, video on demand (VoD), time-shifted TV (tsTV) and network personal video recorder (nPVR). The system adopts native IP multicast for linear TV, and distributed caching and P2P mechanism for VoD, tsTV and nPVR services.

2.4 Live streaming framework based on DONet/Coolstreaming

Since live streaming frameworks have been comprehensively studied, in the system, we would not create a new one; instead, we adopted the new implementation of DONet/Coolstreaming as the live streaming framework to deliver live contents. In the following, we'll introduce the characteristics of the new DONet/Coolstreaming.

1. Node hierarchy

For each node in the system, it maintains three levels of nodes: members, partners and parents. Members give a partial view of currently active nodes in the system, and no connection is established between the node and its known members. Connections established between partners to exchange block availability information. Parent-child relations are formed when connections are established for actual block transmission.Apparently a node's parents and children are a subset of its partners set.

2. Multiple Sub-Streams

The video stream is encoded and packed into continuous blocks and can be 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 nodes independently, which means a node can retrieve data from up to S nodes. Figure 2-1 shows a video stream divided into four sub-streams with S=4.

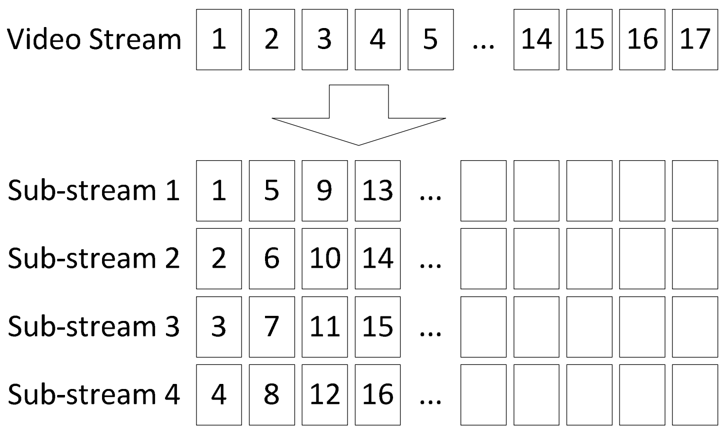


Figure 2-1 Sub-streams dividing

3. Joining procedure

A newly joining node first contacts the bootstrap server and retrieves a list of available channels. After selecting a channel, the node retrieves a partial list of the currently active nodes in the channel, and put the nodes in the list to its membership cache. Then the node randomly selects some nodes, with whom connections are established so that they will (1) exchange their membership cache knowledge and (2) exchange block availability information periodically. The exchanged information helps the node to decide where it should start requesting data. Then again, the node randomly selects some partners to establish a parent-child relationship, where actual data transmission takes place. A parent can be subscribed with multiple sub-streams.

4. Hybrid push-pull mechanism

To form a parent-child relationship, the node subscribes a sub-stream with another node. When a node receives a subscription message with a designated starting timestamp, the node becomes the parent node of the subscriber node and stores the subscriber's information, including its IP, communication port number and data port number in a sub-stream subscriber list. The parent node starts sending to the subscriber all blocks in the subscribed sub-stream starting from the timestamp given. The parent can be either the source or another node. In the source case, it pushes a block to the subscribers whenever it finishes packing a new block, and in the another node case, it pushes a block to subscribers whenever it receives a new block. The subscription contract is ended when the subscriber sends an unsubscribing message, or when the parent node is unable to push blocks to the subscriber because of underlying network problems.

5. Parent re-selection

As the subscription increases, a node may be overloaded and starts to lag pushing blocks to its subscribers. A node can detect such lagging by (1) comparing sub-stream receiving status between parents, or (2) comparing sub-stream receiving status between its own buffer and its partners' buffer. As shown in the upper part of Figure 2-2, the node compares the receiving status in its buffer, and can discover that sub-stream 2 is lagging behind sub-stream 1 by three blocks. As shown in the lower part of figure 2-2, the node compares the receiving status in its buffer with a partner's buffer, and can discover that its sub-stream 2 is lagging behind the partner's sub-stream 2 by three blocks. If the lagging range is larger than a certain threshold, which can potentially indicates the node is overloaded, the parent re-selection is triggered, and a new parent node will be selected to support the lagging sub-stream and the original subscription is cancelled. The new parent node can be selected from the current partners if there's any, or from current parents with better buffering status, if there's no available partners.

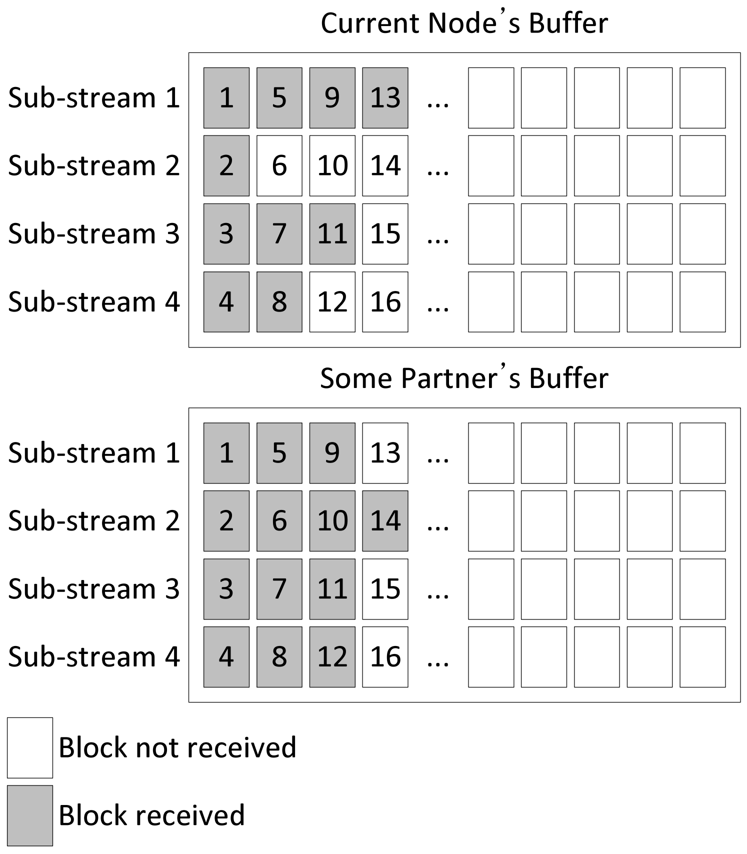


Figure 2-2 Comparing sub-stream status in parent re-selection

2.5 Kademlia DHT

To store the information of the cached contents distributedly, a distributed hash table (DHT) is used. We adopted Kademlia for the purpose. 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, which we use SHA-1 hash function on the name of wanted file to generate. Given two identifiers x and y, the distance between them is the bitwise XOR 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 given id, i.e., PUT<key, value> had been performed.

Chapter 3
System Design & Implementation

P2P time-shift streaming is similar to P2P VoD services, except that the size and duration of a VoD program can be pre-calculated, while those of time-shift video streaming can't be done in advance. Therefore, caching mechanism is the major issue in the system.

3.1 System overview

By studying related research on P2P live streaming and P2P VoD streaming systems, we conclude that our system needs to cope with following issues: live streaming, content caching, publishing, searching and fetching, which we sorted to three major topics:

1. Live streaming framework

Live streaming framework provides a basis for this system, as time-shift streaming contents are provided by the contents that live streaming viewers had watched and cached in their local storage. The details of the applied live streaming framework based on DONet/Coolstreaming had been presented in Section 2.4.

2. Caching strategy and cache replacement policy

Live streaming nodes cache the contents they had watched to support time-shift streaming nodes, and thus the caching strategy is an important issue of the system design. Two factors, cached data redundancy and time-shift service span, were considered. It is clear that having all live streaming nodes caching all the contents they had watched provides the most data redundancy, but the shortest service span, because each node only has a limited storage space, and the time-shift service span hat a single node can provide is equal to the node's storage space. On the other hand, having only one replica in the system provides a storage space equal to the sum of all nodes' storage space, but this provides poor data redundancy since the departure or failure of a node means the lose of data. Therefore, a mechanism keeping a balance between them is important, and thus we propose a probability algorithm to keep a desired number of replicas in the system.

3. Time-shift content search/fetching mechanism

The cached content must be located before it can be retrieved, we adopted Kademlia [17-18] distributed hash table (DHT) for content publishing and content search. With the published knowledge collected from the DHT, time-shift contents can be fetched from multiple sources in an efficient and load-balancing manner.

3.2 System architecture

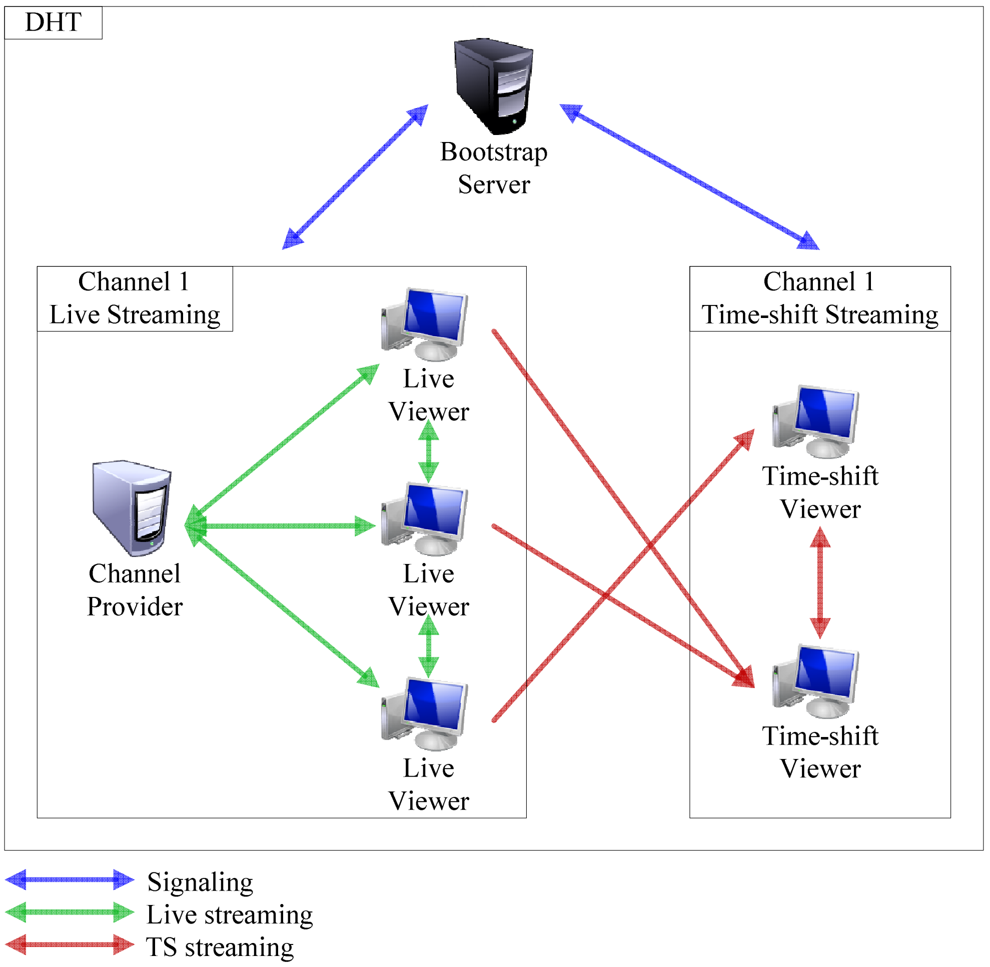


Figure 3-1 System Architecture

Figure 3-1 depicts the architecture of our system. The system contains three types of components: bootstrap server, provider and viewer. The bootstrap server maintains a list of available channels and a list of participating nodes of each channel, in order to bootstrap the newly joined nodes. A provider is also a source node in the live streaming network and it registers its providing channel's information with the bootstrap server. Viewers first join the system with the help of the bootstrap server, and then retrieves the desired video contents for live streaming playback or time-shift playback.

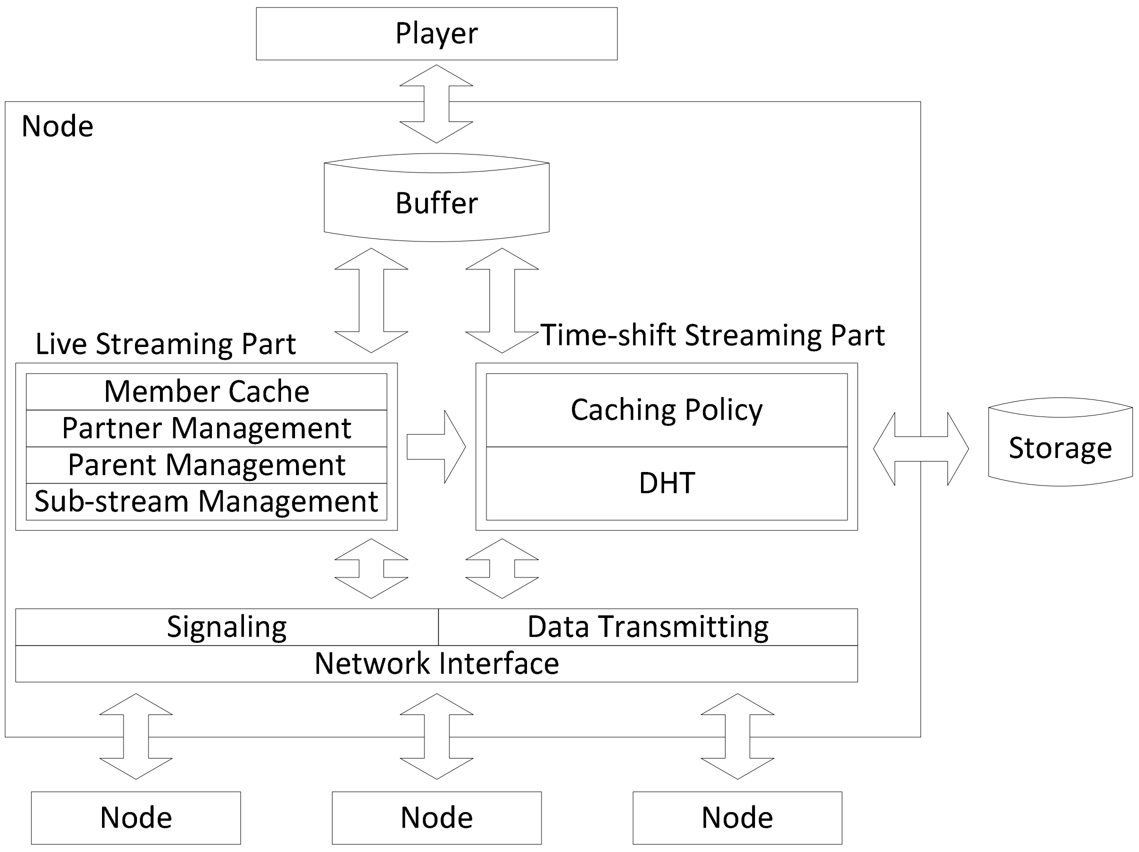


Figure 3-2 System diagram of a node

Figure 3-2 depicts the system diagram of a node; the node can be a channel provider or viewer. The player-buffer relationship depends on the type of the node. For a provider, the player encodes the original video stream in to packet stream, and the stream data is put in its buffer for data transmission and genertaing its buffering status. For a viewer, the video content is also put in its buffer for data transmission, generating buffering status and playback. To share video content among peers, the buffered content can be transmitted through either live streaming mechanism or time-shift streaming mechanism. The live streaming part handles the content transmission for live streaming, and cooperates with the time-shift streaming part to cache and publish the contents. Transmissions are carried out on TCP connections to avoid network layer data losses. Kademlia DHT is used to published the cahed content, and its messagesare transmitted over UDP packets.

3.3 Transmission unit in streaming

The basic streaming flow of our system is depicted in Figure 3-3. The video stream is generated from the video source. A video server encodes the video stream into continuous packets and transmits to the viewer nodes. Each viewer node receives the video packets, and the video player decodes the received packets back to video. It is intuitive to replay the packets using a buffer-then-play scheme for both live and time-shift P2P streaming. However, since packet encoding is synchronized with the video, each generated packet will have its corresponding position on the timeline. The packet receiving pattern also needs be recorded for packet replay to rebuild the original video content, and thus each packet requires extra timing information. In our system, we record the duration of each packet, so the relative receiving pattern can be rebuilt.

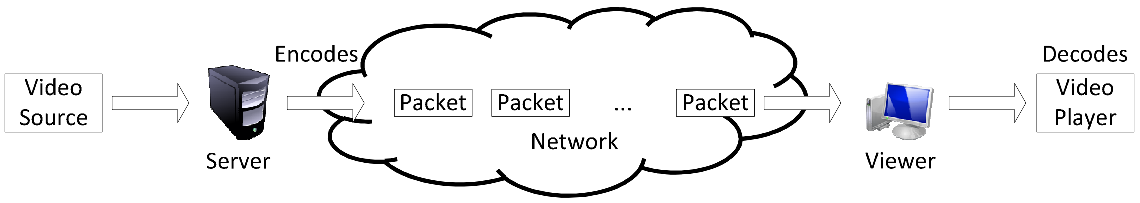


Figure 3-3 Basic streaming

At the video source, the video is encoded into UDP packets by a VLC media player [19]. The UDP packets are then sent to the video server, which is a Provider node, via local loopback interface. The video server measures each packet's duration. Since it is inefficient to track each packet individually, continuous packets received in a second are packed into a block used in the system. Furthermore, in order to support time-shift streaming, 10 consecutive blocks, with the starting block's timestamp is aligned to 10's multiples, are packed into a file for local storage purpose. The file is named after the information given by the channel provider, with a readable format of timestamp; for example, file with name “ProviderName_Channel1_20100620182520” stands for 10 blocks of Channel 1 provided by ProviderName, with timestamp from 2010/06/20 18:25:20 to 2010/06/20 18:25:29. Figure 3-4 shows the structures of a block and a file.

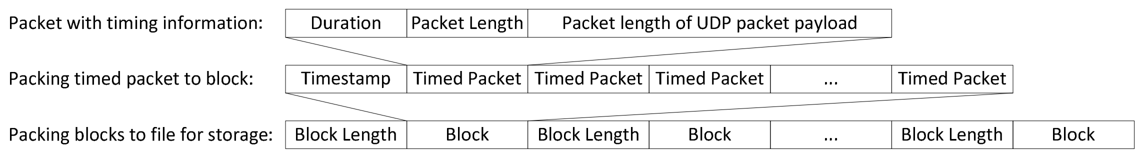


Figure 3-4 The structures of a block and a file

3.4 Distributed cache management strategy

The goal of our distributed cache management strategy is to effectively keep a desired number of replicas for the cached contents. The strategy is composed of two parts: publishing/re-publishing policy and content caching based on probability.

1. Content publishing/re-publishing policy

After a video file is collected, the node will publish the cached content on the DHT. However, the provider node works a little differently; it caches all contents but never publishes the ownership information. The purpose is to use the provider node as a backup node, and can only be accessed at emergency. For example, when a block is 5 seconds to the playback deadline but had not been received, or when there is no owner of the wanted content. Since the system will keep multiple replicas for each video file, the published record put into the DHT is a list of <IP, Port, Last_Update_Time> triples. Fig.3-5 depicts the relation between a file name and its owner list found on the DHT with the structure of the list.

When a node wants to update a list, it first tries to get the list from the DHT. If the list does not exist, it creates a new list. Then the node removes the record of two types: (1) the record put by itself in the past that the node will update later, and (2) the out--of-date records that can be determined by comparing the records' Last_Update_Time with the current time. In our system, we consider a record out-dated if the record is last updated more than thirty minutes ago. This thirty-minute interval could give the node enough time to do multiple updates, which we will mention later in this section. After removing the record , the node checks the size of the list, if the size has reach the desired number of replicas the system, the node deletes its cached file; otherwise, it add its record to the list, and put the list back to the DHT. However, the accesses of the DHT from the peers are not coordinated, which means a published record may be overwritten by another node. Consider the following scenario. Node A and node B both wants to update the published list for file F. A gets the list, updates the list, and just before A put the list back to the DHT, B also gets the list, and updates the list. After that, A puts the list back to the DHT, but the list will be overwritten when B puts the list back to the DHT. As a result, A's record is not stored in the list.

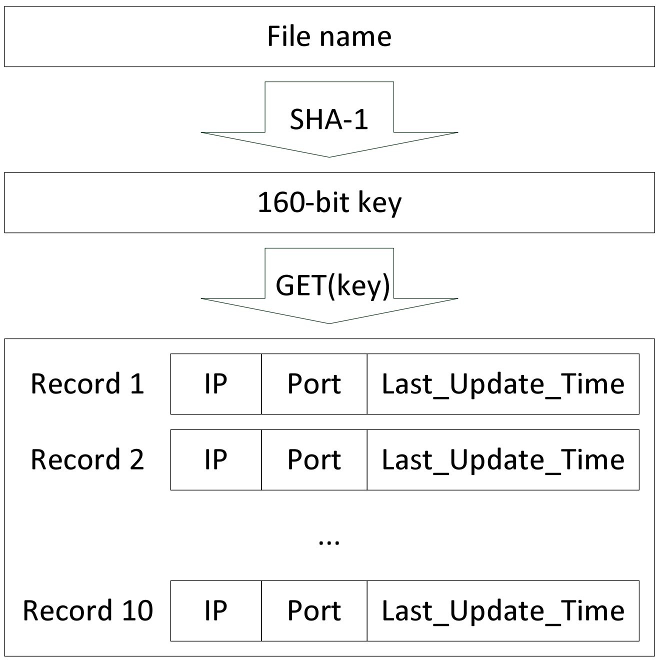


Figure 3-5 Getting file owner list from DHT and list structure

To deal the synchronization issue, each node will back-off for a random interval before its publish operation to reduce such collisions, for the first time a node update a list, it will have a random back-off time uniformly distributed in [0, 50), at a 5-second stepping. A node also republishes its cached files. The republish operation is similar to the initial publish operation, but is done periodically in order to keep the lists up to date and to alleviate the effect of missed publishing. A node will periodically do the republish operation with a random back-off time uniformly distributed in [600, 1200), also at a 5-second stepping. As mentioned above, the records on DHT have a thirty minute out-date threshold, so for our settings for the random back-off for publish and republish operations, each file a node had cached will perform at least one republish operation before the record is out-dated.

Algorithm for random caching and random back-off for publishing is listed as following:

```
while(waitngForBlocks)
    block = node.receiveBlock();
    buffer.put(block);
    if(block and nearby blocks can be dumped)
        viewerKnowledge = MAX(parent.size()+partner.size(), viewerCount);
        rand = a random integer generated between (0, viewerKnowledge]
        if(rand < replicasRequired)
            dump blocks to local storage;
            random back-off for DHT publish;
            fileOwnerList = DHT.get(filename);
            remove out-dated entry and this node’s entry in fileOwnerList
            if(DHT.get(filename).size() < replicasRequired)
                FileOwnerList.add(this node);
                DHT.put(filename, fIleOwnerList);
            else
                delete dumped file
            end if
        end if
```

```
19          end if
20      end while
```

Algorithm 3-1 Random Caching and Random Back-off

2. 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 decide whether a file should be cached or not. Assume that the system wants to keep R replicas, and the system has N viewers. It is clear that each node should cache the received content with a probability of R/N. Since R is a constant, 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 node keeps connections with its partners and parents, these nodes must be active nodes in the system. Therefore, the node has the first parameter as the value of the number of partners plus the number of parents. In addition, the number of the current active viewers can be obtained by a modified node-startup procedure. When a node joins the system, heartbeat messages are periodically sent to the bootstrap server to update the membership cache, and the number of currently active viewers is piggybacked to the node in the replying messages. With the two values, N is selected as the larger one of the two. The local knowledge helps the node to react fast to the change of active nodes, especially when the size of viewer is small, since they would form an almost fully-connected mesh structure; and the number of the current active viewers helps the node to make better decisions when the size of viewers becomes larger.

Note that the content publishing/re-publishing policy can be done without the probability caching mechanism. In this case, each node has the responsibility of replica control totally based on the content publishing/re-publishing mechanism.

3.5 Time-shift streaming

In time-shift streaming, we applied per-block pulling mechanism for content retrieval. After a node decides which channel it wants to watch, and where it wants to start playback, the name of the file containing the required content is known. By querying with the file name on the DHT, the node obtains the list of file owners. Then, a timer is started and for each interval of 1 second, the node will try to pull up to 4 blocks, each from a randomly selected owner in the list. The reason why there's a limit on the number pulling blocks in each interval is that the available content cached may be much larger the buffer's capacity, so that it is required to keep the pulling timestamp stay in a distance with the playback timestamp. For emergency handling, contents close to playback deadline but not received will be pulled directly from the provider.

3.6 System Implementation

We implemented the system in Java 1.6, based on request-reply model: node communicates with each other with request messages, and the recipient will reply with corresponding reply messages.

3.6.1 System Components

1. Bootstrap server

   The bootstrap server creates a ServerSocket for incoming messaging connections, Thread's are created for each incoming connection and received messages are handled and replied to the connecting node.

2. Provider/Viewer

   Provider/Viewer node creates two ServerSocket's, one for incoming messaging connections and the other for block transmission connections, Thread's are created for each incoming connections. For incoming messaging connections, received messages are handled and replied to the connecting node. And for incoming block transmission connections, received blocks are then transferred to this node's buffer, where the block can be played or cached.

3.6.2 Message Format

Message contains its type and required options of that type of message. Fig 3-6 depicts basic message format. After a message has been generate, it is sent through TCP with Java Socket.

Type	Option 1	Option 2	Option 3	...

Figure 3-6 Message Format

3.6.3 Message Types

(1) Channel Registration

Channel providers registers its information with the bootstrap server, options including this node's messaging port number, channel provider's name and channel description. Bootstrap server replies with whether the registration is ok.

(2) Channel List

Viewer requests for available channels registered at bootstrap server, options including this node's messaging port number, channel provider's name and channel description. Bootstrap server replies with a list of available channels' provider name and channel description.

(3) Channel Join

In live streaming, this message is used for channel joining procedure, which we had mentioned the joining procedure in 3.4.1, options including this node's control port number, channel provider's name and channel description. Bootstrap server replies with a list of currently active nodes in the channel. And in time-shift streaming, the message is used for DHT joining procedure, where bootstrap server replies a DHT bootstrap node for the DHT bootstrap procedure.

(4) Buffermap Exchange

The message is used for buffer map information exchanges between nodes, options including this node's control port number and buffer map. The recipient replies with its buffer map.

(5) Sub-stream Subscription

This message is used for sub-stream subscription, the options including this node's messaging port number, block transmission port number, subscribing timestamp and its buffer map of subscribing sub-stream. The recipient replies with the subscription result.

(6) Sub-stream Un-subscription

This message is used for sub-stream un-subscription, the options including this node's messaging port number and the index of the un-subscribing sub-stream. The recipient replies with the un-subscription result.

(7) Time-shift Block Request

This message is for time-shift streaming viewer nodes to request a block from other nodes, the options including this node's messaging port number, block transmission port number and its requesting timestamp. The recipient replies with the requested result and (1) if it has the block, the requested block is sent to the requesting node, or (2) if it does not has the block, it tells the node to ask the server.

Chapter 4
Performance & Analysis

To evaluate the system performance, we performed experiments on PlanetLab, an open global research network [19].

4.1 Experiment Environment

The streaming server is located in the Internet Communication Laboratory, NCTU. 48 PlanetLab nodes were used as live streaming viewers, and 16 PlanetLab nodes were used as time-shift streaming viewers; most of them are located in the United States. The video is streamed at bit rate of 400 kbps, the number of sub-streams was set to be 8, and each node can connect to up to 24 other nodes as partners. The buffer size of each node is 120 blocks The random back-off time of first time publishing was uniformly distributed in [0, 50), at a 5-second stepping. The random back-off for republishing was uniformly distributed in [600, 1200), also at a 5-second stepping, 10 replicas would be kept in the system. Time-shift nodes cache each received block with a probability of 0.5. Table 4-1 lists the system parameters used in our system.

Table 4-1 System Parameters

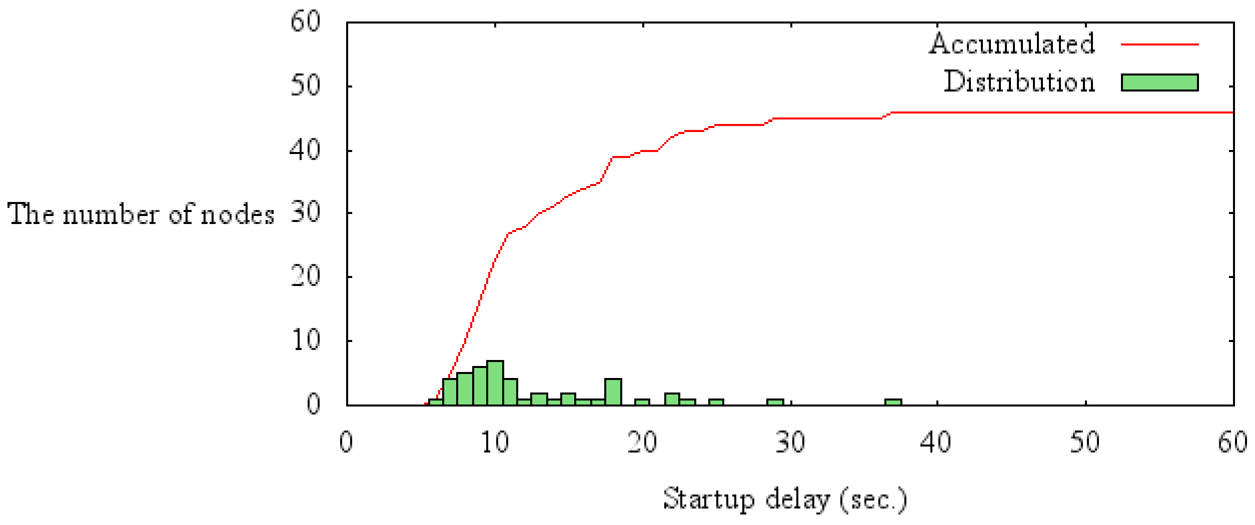
System Parameter	Value
Video streaming bit-rate	400 kbps
The number of sub-streams	8
The maximum number of partners	24
The number of replicas to keep	10
Buffer size	120 blocks
Random back-off for the first time publishing	0~50 second, stepping 5 seconds
The random back-off for re-publishing	10~20 minutes, stepping 5 seconds

In both trials, we first started the bootstrap server and streaming provider, and then all 64 nodes joined the system as a Poisson process, with the inter-arrival time set to be 60 seconds. For live streaming nodes, after the exchange of block availability information, our heuristic is to set the node’s starting timestamp for playback to be the smallest timestamp in the received availability information plus the number of sub-streams. For each time-shift node, it randomly selected a time between the time when the streaming had started and the time it joined the system to start to playback. The experiment lasted 2 hours, and we assumed no peer churn. We will examine the performance of both proposed methods: without the information of the number of currently active nodes in the system and with the information of the number of currently active nodes in the system. The results will be placed on each figure’s upper side and lower side, respectively.

4.2 System Performance and Analysis

4.2.1 The live streaming

First, we examine three commonly used criterions in evaluating a streaming service: startup delay, end-to-end delay and playback continuity. The startup delay is the time between when a user tunes to a channel, and when the video content is visible. End-to-end delay, also called playback delay, is the delay of the video content between the viewer and the source. Continuity index is the number of segments that arrive before or on playback deadlines over the total number segments a node should have received.



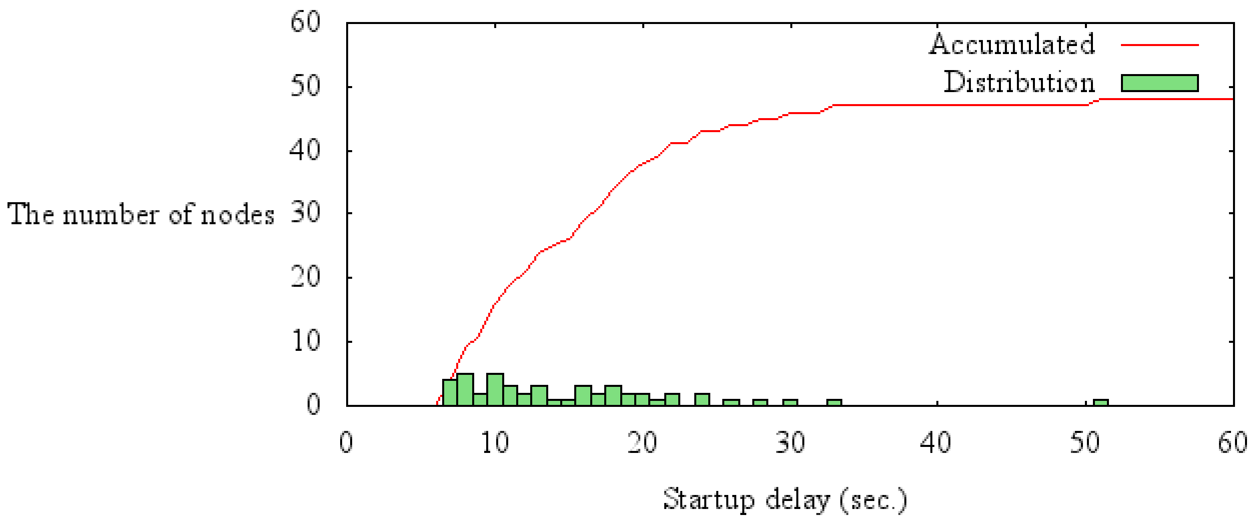
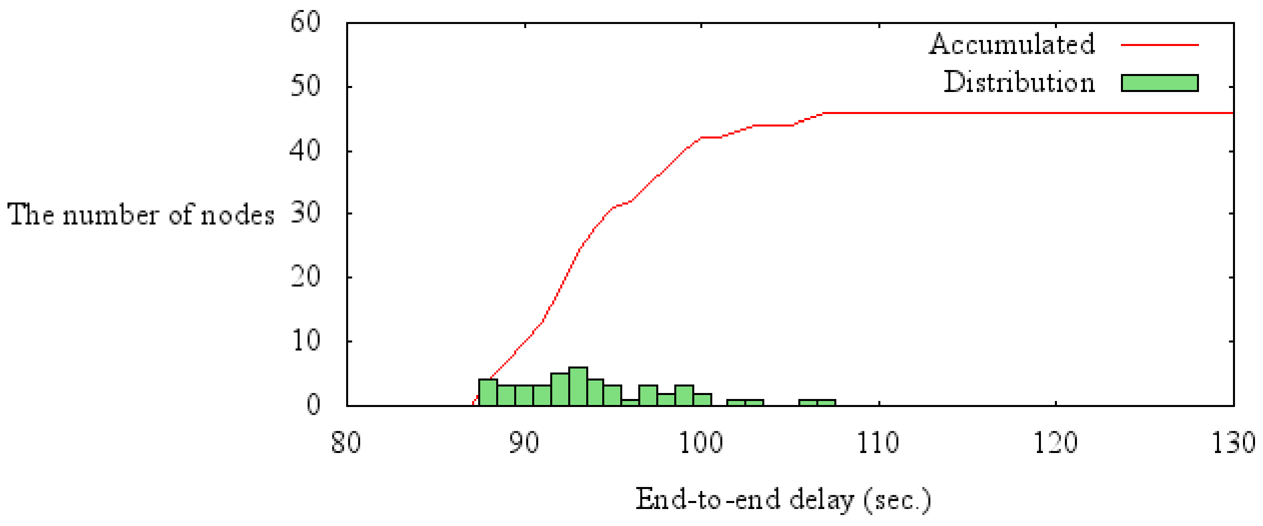


Figure 4-1 The Distribution of Startup Delay



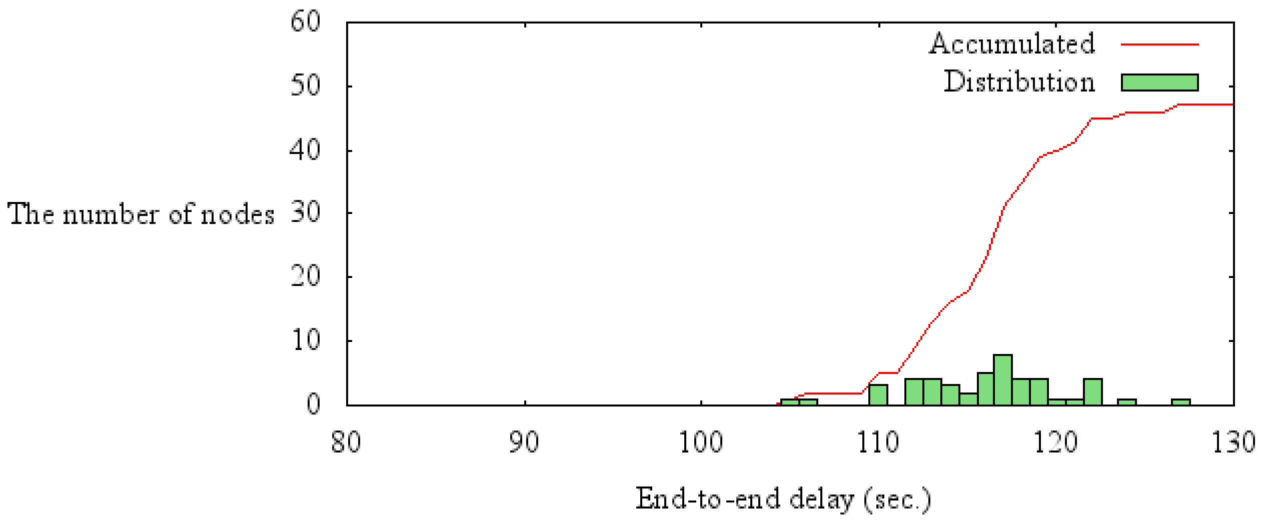
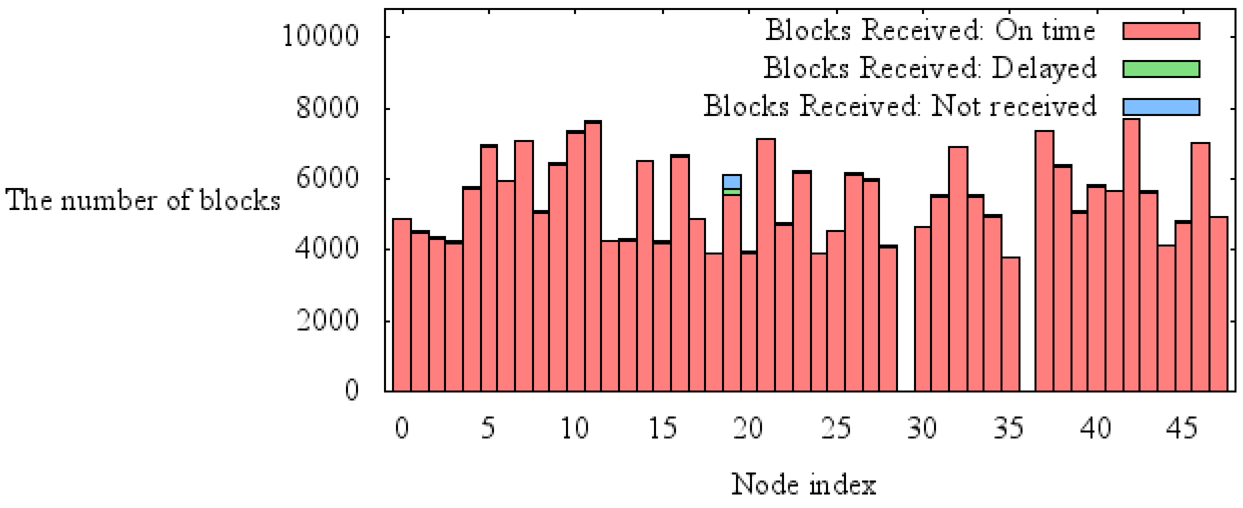


Figure 4-2 The Distribution of End-to-End Delay



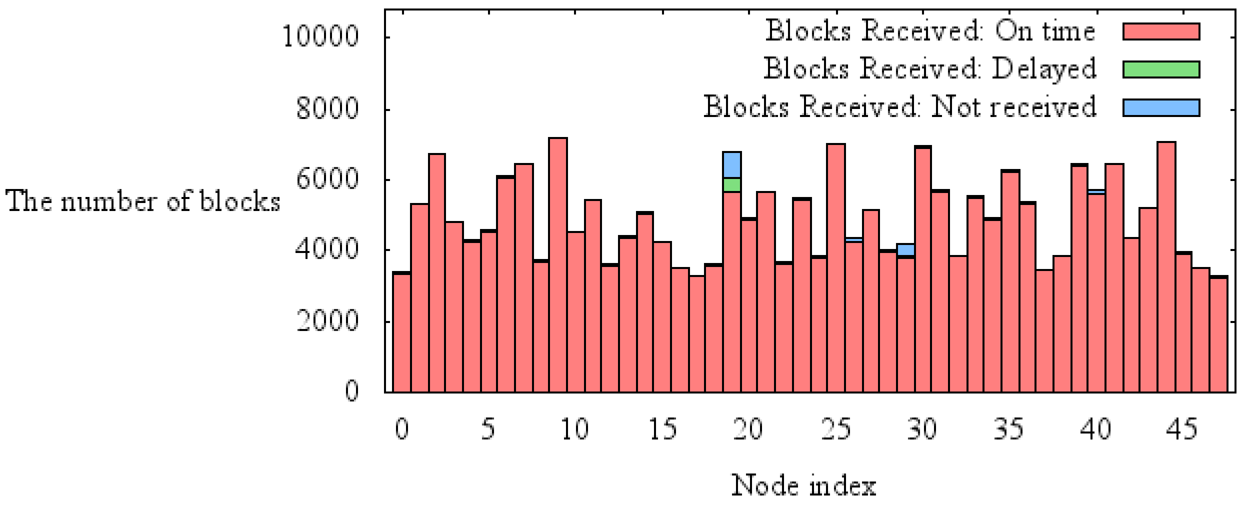
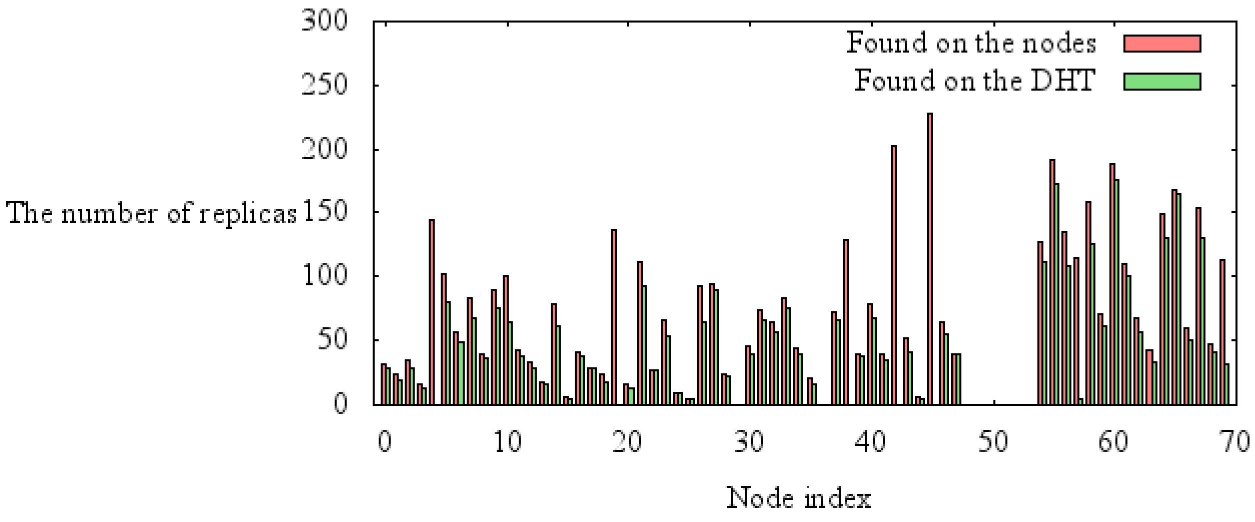


Figure 4-3 Live Streaming - Continuity Index Diagram

Figure 4-1 depicts the startup delay in our system. The average delay has an average of 13.19 seconds in the first trial that is without the information of the number of currently active nodes, and 15.75 in the secondtrial that uses the information of the number of currently active nodes. The end-to-end delay, as depicted in Figure 4-2, has an average of 94.33 seconds in the first trial, and 116.46 in the second trial. The continuity index is 99.00% and 98.46%, as shown in fig. 4-3. However, there are two nodes failed to join the system, because they were unable to contact with the bootstrap server and thus received no block, we can also see 3 nodes perform poorly in the trials, which was caused by the underlying TCP errors.

4.2.2 The Time-shift streaming

For alleviate the effect of the initial node-joining procedures and unfinished publishing and re-publishing procedures, we only examine the blocks generated between 30 to 90 minutes of each trial. Fig. 4-4 shows the cache results at each node; node index below 50 are results from the live streaming nodes, and node index above 50 are results from the time-shift streaming nodes. Note that there are nodes suffering from DHT failures, which makes them unable to publish their file availability on the DHT. In the trial without the information of the number of currently active nodes, each node caches 67.86 files in average, with standard deviation of 56.39. and in the trial with the information of the number of currently active nodes, each node caches 61.87 files in average, with standard deviation of 35.54. We can also see from the figure that with the help of the information of currently active nodes of the system, the caching responsibility is more evenly distributed among nodes in the system.



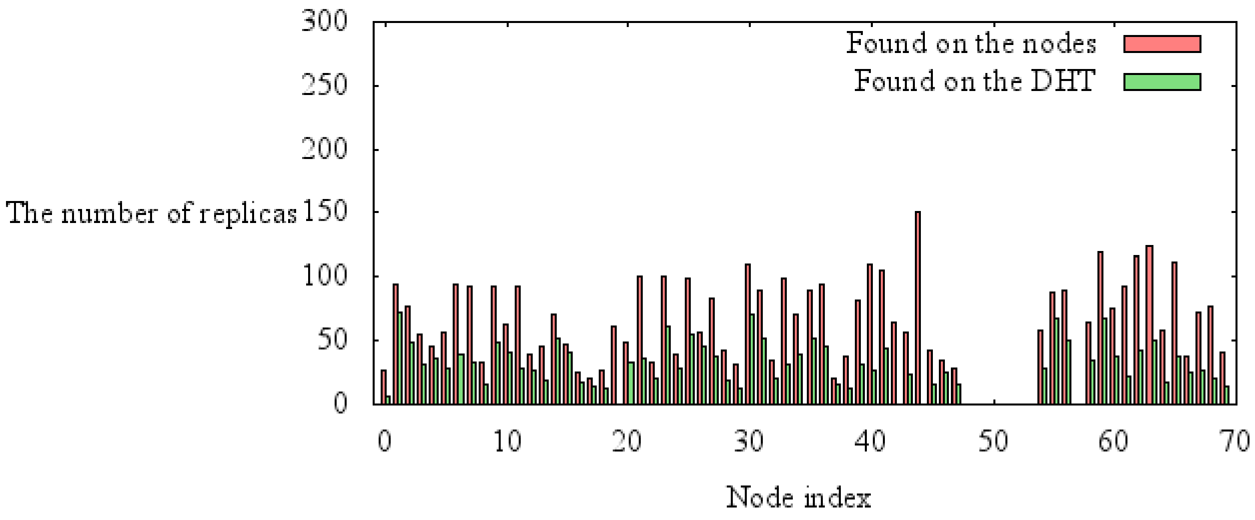


Figure 4-4 Cache Results

Fig. 4-5 depicts the distribution of the number of replicas of each cached files among all live streaming nodes and time-shift streaming nodes. The method without the information of currently active nodes provides much more files that have ten replicas on the DHT, but in this method, more files are first cached and then deleted with the publish/republish mechanism. Note that files with more replicas would have more records on the DHT. On the other hand, although the method with the information of currently active nodes also provides ten replicas for most files on the DHT, but since there are more files that have less owner information found on the DHT comparing to the method without the information of the number of currently active nodes in the system. The republishing processes may need a longer time to stabilize, in order to provide the information of cached file owner status on the DHT.

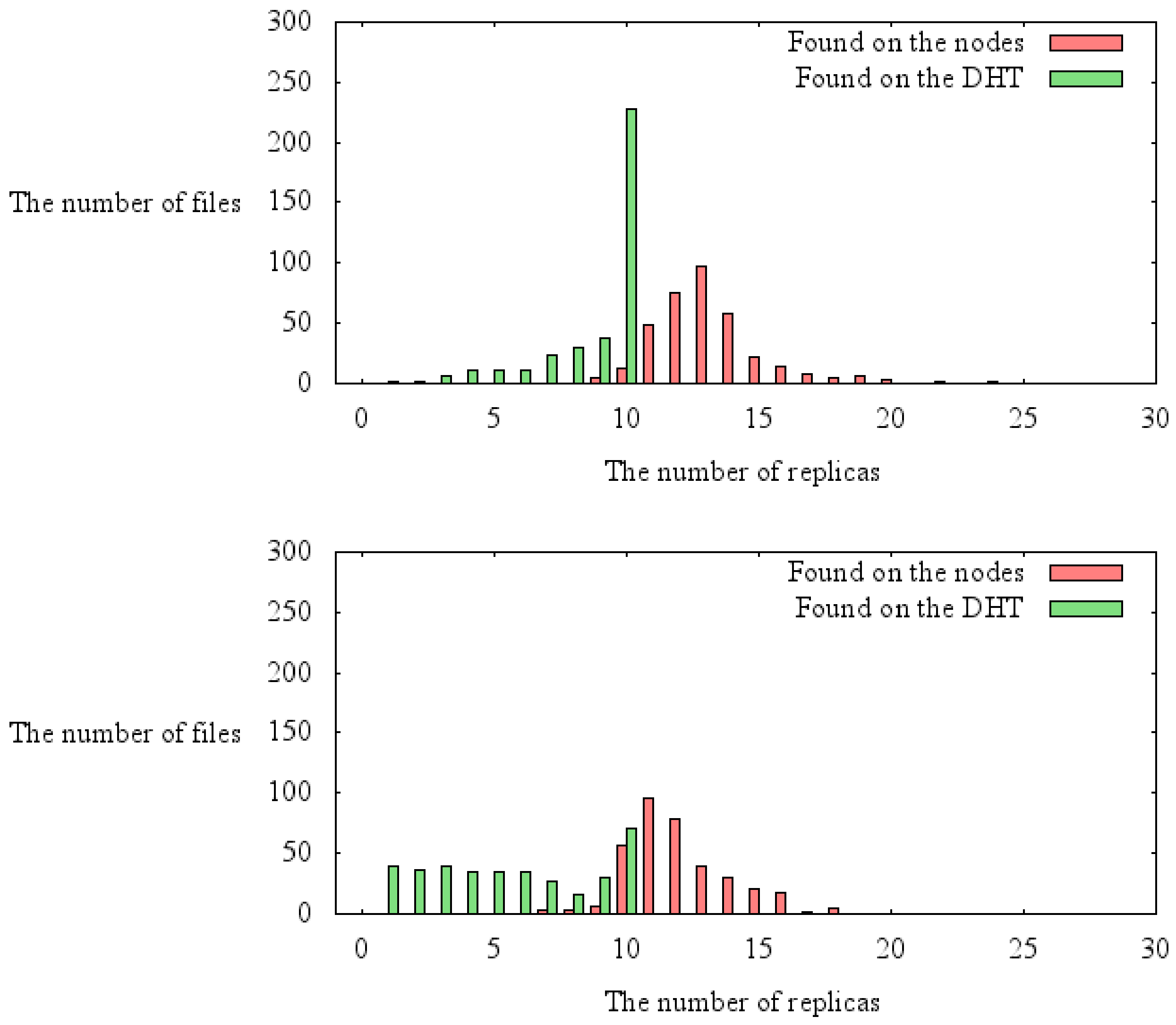


Figure 4-5 Distribution of Replica Count

Table 4-2 The sources of time-shift streaming blocks in the first trial

Node Index	TS 01	TS 02	TS 03	TS 04	TS 05	TS 06	TS 07	TS 08	TS 09	TS 10	TS 11	TS 12	TS 13	TS 14	TS 15	TS 16
From other nodes	3657	3484	4122	1187	3441	5903	4035	5816	5539	5788	3652	4999	1729	2614	2995	2940
From Server	72	116	122	1413	100	189	96	276	310	307	107	64	6	38	89	489
Failed to Get	72	114	90	10	100	186	91	97	220	193	86	61	3	38	89	70
Emergency	0	2	26	1381	0	3	5	19	90	114	21	3	3	0	0	279
No Owner	0	0	6	22	0	0	0	160	0	0	0	0	0	0	0	140

Table 4-3 The sources of time-shift streaming blocks in the second trial

Node Index	TS 01	TS 02	TS 03	TS 04	TS 05	TS 06	TS 07	TS 08	TS 09	TS 10	TS 11	TS 12	TS 13	TS 14	TS 15	TS 16
From other nodes	3227	2577	3789	0	4893	4830	4178	2742	6014	5356	2904	5330	2516	3524	3667	1796
From Server	68	40	48	0	203	276	119	61	167	113	25	417	14	106	15	18
Failed to Get	54	31	32	0	55	23	66	37	75	43	22	20	13	25	12	14
Emergency	14	0	6	0	148	244	53	8	92	70	3	397	1	1	3	4
No Owner	0	9	9	0	0	9	0	16	0	0	0	0	0	80	0	0

Table 4-1 and Table 4-2 lists the block counts from different sources for the time-shift streaming nodes in each trial, respectively. With the help of the server as an emergency handling node, both trials achieves 100% continuity index. In the first trial, the TS nodes had received a total of 65695 blocks, and 61901 (94.22%) of them were served by peer nodes. 3794 blocks were supported by the providing server, where 1946 of them were emergency handling, 1520 of them were unable to get from peers, and 328 of them have no file owner.Most of the requests to the providing server were by node TS 04, that suffered from a temporary DHT failure, and thus during that failure period, all blocks were supported by the server. In the second trial, the TS nodes had received a total of 59033 blocks, and 57343 (97.14%) of them were served by peer nodes, or. 1690 blocks were supported by the providing server, where 1044 of them were emergency handling, 522 of them were unable to get from peers, and 123 of them have no file owner. The reason of failing to get from peers was because just after the time-shift node acquires the owner list of its wanted file from the DHT, one of the owners in the list detects there is more than 10 replicas of that file in the system and deletes the file it cached, and thus the requests to the node that no longer has the file will all fail. Fig 4-8 depicts the distribution of each node's contribution to the time-shift streaming nodes, node index below 50 are live streaming nodes, and node index above 50 are time-shift streaming nodes, and we can see the load is distributed through the nodes by the random algorithm, and each node’s distribution is basically follows the number of its cached files, as shown in Fig. 4-4.

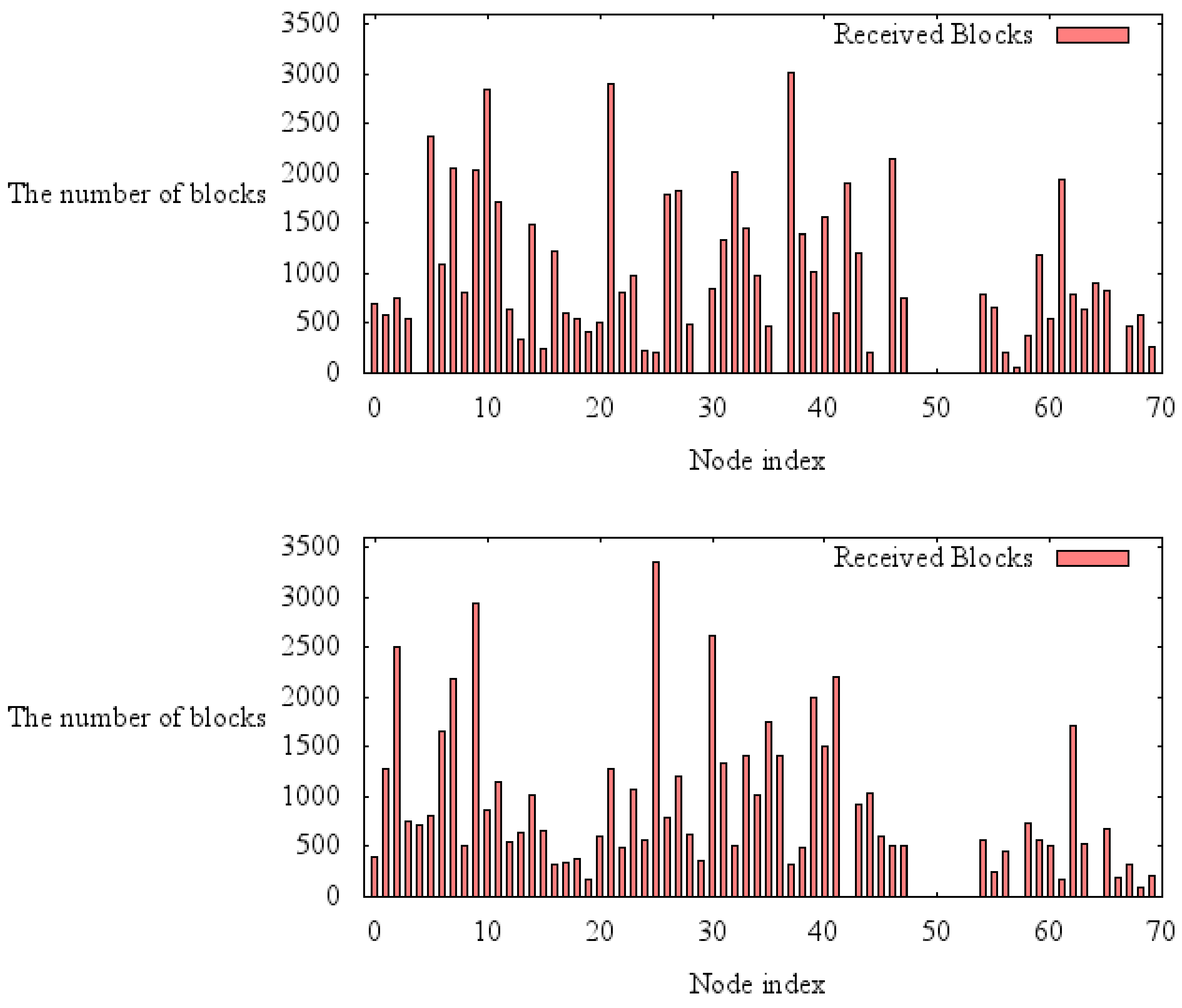


Figure 4-6 The Distribution of Nodes' Contribution to Time-shift Streaming Nodes

Chapter 5
Conclusion and Future Work

In this thesis, we had implemented a P2P live/time-shift streaming system and presented two distributed cache management strategies for time-shift video segments to cache a desired number of replicas based on the DHT knowledge and publishing/republishing mechanisms. We also studied the performance of the system on PlanetLab. Our experiment results show the feasibility of live/time-shift systems. In our experiments, the live streaming part achieved a startup delay of 16 seconds, an end-to-end delay of 120 seconds and a continuity index over 98%. Moreover, for the time-shift part, with the streaming server as an emergency handler, it achieved a continuity index of 100%, with over 94% of the streaming data were from P2P peers. Our proposed caching strategies effectively distribute the load of storing the time-shift contents and provide ten replicas for most files.. The information of the number of currently active nodes in the system also helps in distributing the load for storing the time-shift contents more evenly among the nodes, as the standard deviation of the number of files cached on the nodes was reduced. .

However, in this implementation, publishing on the DHT has synchronization issues. Although random back-off publishing/republishing may alleviate this problem, collisions still occur, which lead to the difference between the owner lists on the DHT and the caching status in reality, and thus lowers the effectiveness of the cached files for time-shift viewers. More investigation is need on this DHT issue and larger experiments of the system would provide more insightful knowledge on P2P time-shift 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] Vakali, A.; Pallis, G., “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] Venkataraman, V. ; Yoshida, K. ; Francis, P., “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] Do, T.T. ; Hua, K.A. ; Tantaoui, M.A., “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 ; Nahrstedt, K., “oStream: asynchronous streaming multicast

in application-layer overlay networks" Selected Areas in Communications, IEEE Journal on,
vol. 6, no. 1, Jan. 2004

[12] Dana, C. 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] Deshpande, S. ; Noh, J.,"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

[15] Hecht, F.V. 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

[16] Gallo, D. et al. "A Multimedia Delivery Architecture for IPTV with P2P-Based Time-Shift Support" Consumer Communications and Networking Conference, 2009. CCNC 2009. 6th IEEE, pp. 1-2. Jan. 2009

[17] 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

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

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

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