行政院國家科學委員會專題研究計畫 成果報告

點對點實況及移時播放之影音服務
研究成果報告(精簡版)

計 畫 類 別 ：個別型
計 畫 編 號 ：NSC 98-2221-E-009-083-
執 行 期 間 ：98 年 08 月 01 日至 99 年 07 月 31 日
執 行 單 位 ：國立交通大學資訊工程學系（所）

計 畫 主 持 人：張明峰

計畫參與人員：碩士班研究生-兼任助理人員：李茂弘
碩士班研究生-兼任助理人員：邱順胤
碩士班研究生-兼任助理人員：戴境余
博士班研究生-兼任助理人員：陳志華

處 理 方 式 ：本計畫可公開查詢

中 華 民 國 99 年 10 月 29 日

行政院國家科學委員會專題研究計畫成果報告

點對點實況及移時播放之影音服務

P2P Live and Time-shifted Streaming Services (I)

計畫編號：NSC 98-2221-E-009-083
執行期限：2009.08.01 至 2010.07.31

主持人：張明峰　交通大學資工系
計畫參與人員：李茂弘、邱順胤、戴境余、陳志華　交通大學資工系

中文摘要

隨著寬頻網路的普及，多媒體串流服務已成為一蓬勃發展的網際網路應用，但此類系統的可延展性是重要的研究個問題。點對點式架構是目前解決可擴展性問題中最效的方法，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。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。在此篇報告中，我們設計實作一點對點實況/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，並在 PlanetLab 平台上進行實驗，分析此一系統的可行性及效能。我們的實驗數據顯示，在規模為 64 節點的實驗中，我們系統中個別節點的啟動延遲平均為 16 秒，訊號源到各節點的端到端平均延遲小於 20 秒，而串流接收的連續係數平均大於 98%。由於使用訊號源備份作為緊急資料源，移時串流節點的連續係數高達 100%，其中 97%來自其他節點，3%來自訊號源。然而，我們設計的隨機延後發佈儲存移時視訊資料於 DHT 的方法有嚴重的同步問題；有相當比例的儲存資料無法成功發佈。未來，我們需要一個有效率的發佈方法，並進行更大規模的測試。

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 to solve. 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 report, we present the design and implementation a P2P live/time-shift streaming system, and a distributed cache management strategies for time-shift video cache files. In addition, we study the system's performance and characteristics on the PlanetLab experiment platform. In our initial experiments with 64 nodes, the live streaming part achieved a startup delay of 15 seconds, an end-to-end delay of 16 seconds and a continuity index over 98%. Moreover, for the time-shift streaming service, our system achieved a continuity index of 100%, with

over 94% of the streaming data were from P2P peers. However, our random back-off publishing strategies for cached video segments have synchronization problems, which lead to incomplete publishing results. New publishing strategy needs to be developed, and large-scale of experiments need to be conducted.

1. Introduction

With the increasing prevalence of broadband Internet access, multimedia streaming service has been a very popular Internet application in recent years, but the system scalability has always been an issue to solve. In the early developments of media streaming applications, client-server architecture suffers from scalability problems; as the number of users increases, the servers are quickly overloaded [1]. Content delivery networks (CDNs) with strategically placed proxies have been developed to balance the loading of the servers, but they are 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 the lack incentives for network operators to carry streaming data traffic [3]. Application-level multicast, by constructing an overlay network with unicast connections between peer nodes in the system, has been proposed to deal with the scalability issue and used in many Internet applications.

Currently, there are mainly two types of streaming services: live streaming and VoD (video on demand) streaming. Live streaming is similar to watching broadcast TV programs; users tune to a selected channel, and the users tuning to the same channel synchronously receive the same content. By contrast, in VoD streaming, a user selects to watch any video clip any time at the user's will. Therefore, contents delivered to VoD users are asynchronous even for the users watching the same video clip.

Another type of streaming service is time-shift streaming service, which provides the ability for a user to watch contents broadcasted in the past; it is like a digital video recorder (DVR) but the user does not need to program the DVR in advance. P2P time-shift streaming can be taken as a special case of P2P VoD streaming. For P2P VoD streaming, video programs are pre-generated and their lengths are known. The video programs are originally stored in dedicated VoD servers. By contrast, in P2P time-shift streaming, the video contents are generated constantly and transmitted to live program viewers. There may not be dedicated servers to store the video contents for time-shift viewers to retrieve them at later time. Therefore, the live program viewers need to coordinately cache the video contents, and thus an efficient peer cache management strategy is needed.

However, there are very few researches on streaming systems that provide both live streaming and time-shifted streaming. In this report, we present the design and implementation of a P2P system providing both live streaming and time-shift streaming functionality. We also propose a distributed cache management strategy to store video contents for time-shift users. In addition, we performed experiments on the PlanetLab platform to study the performance and characteristics of our system. We believe that our work provides valuable knowledge of P2P live/time-shift streaming system for further development on time-shift streaming services.

The remaining of this report is organized as follows. Section 2 describes the current work in P2P streaming studies related to our research. Section 3 presents the design and implementation of our system in details. Section 4 presents the experiment results for system performance. Conclusions and future work are presented in Section 5.

2. Related Work

There have been comprehensive studies on P2P systems. Steaming services based on P2P technologies are also very popular. We will briefly describe the current developments of P2P live streaming systems, P2P VoD streaming systems, and P2P streaming systems with time-shift function.

2.1 P2P live streaming

In P2P live streaming service, peers require the synchronized delivery of streaming media content. One of the important design issue is to form an overlay structure for efficient 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 a stripe on each tree. By constructing a forest of multicast trees, where an interior node in one tree is a leaf node in all the remaining trees, the video forwarding load can be evenly spread across all participating nodes. However, such node-disjointness is a property hard to achieve, especially in heterogeneous environments [8]. In GridMedia [4], the bootstrap procedure uses a rendezvous point to assist peer nodes to join the overlay. A newly joined node first contacts the rendezvous point to obtain a list of nodes that have 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 being selected is in inverse to the end-to-end delay. This allow the node to select nearby peer nodes as partners. 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. For video transmission, 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 video pulling operations accordingly. The scheduling algorithm takes both availability and partners' upload ability into consideration. The block 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 provides users the functionality to watch any programs at any

time. One of the design issues of P2P VoD service is what a peer should cache to share the load of the VoD servers and how to find such cached contents from the peers. In P2Cast [9], peers watching the same video clip within a time interval form a session in a single-tree fashion, each peer caches the beginning part of the video program and a newly joined peer can be patched with the cached beginning part from its parent's buffer. In P2Vod [10], peers form generations, where in each generation, peers have synchronized buffer start. A newly joined peer tries to join a generation, or form a new generation appended to an older generation. Generations are numbered, from G1 as the oldest generation and G*n* 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 created 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 since the system inserts new peers into the system, video disruption is 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 contents, which are too close to the playback deadline but are not arrived yet. Their simulation results indicate that this mechanism reduces 34% of the bandwidth of the server when the users' average outgoing bandwidth is about the same as video bit-rate. However, the required bandwidth from the server still increases linearly as the number of users increases. PONDER [13] also adopts a mesh-based approach similar to BitTorrent, and applies new mechanisms 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 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 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 existing 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 to share with other peers. Their simulation results indicate that P2TSS achieves low server stress by utilizing the peer resource. However, in IPP, the video 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 receives the video contents and the video segment reaches a pre-defined size, the segment

is stored and the peer adds a reference to the segment in a DHT. Although they have presented a demonstration scenario, but 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.

3. System Design & Implementation

3.1 System overview

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

1. Live streaming framework

A live streaming framework provides a base for our system, because it supports live streaming service and the live streaming nodes would store the received contents for the future use of time-shift streaming viewers. Since many live streaming frameworks have been 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 and cache replacement policy

Live streaming nodes need to cache the contents they have 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, have been considered. It is clear that having all live streaming nodes caching all the contents they have watched provides the most data redundancy, but the shortest service span as an overall system, because each node only has a limited storage space. On the other hand, having only one replica cached in the system provides a storage space equal to the sum of all nodes' storage space, but this provides poor data redundancy. The departure or failure of a node means the loss of its cached data. Therefore, a mechanism that keeps a balance between the two factors is needed, 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 stored in 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 depicts the architecture of our system. The system consists of three types of nodes:

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 of live streaming data, and it registers its channel information with the bootstrap server. When a viewer joins the system, it first obtains the information of available channels and participating nodes from the bootstrap server, and then retrieves the desired video contents for live or time-shift streaming.

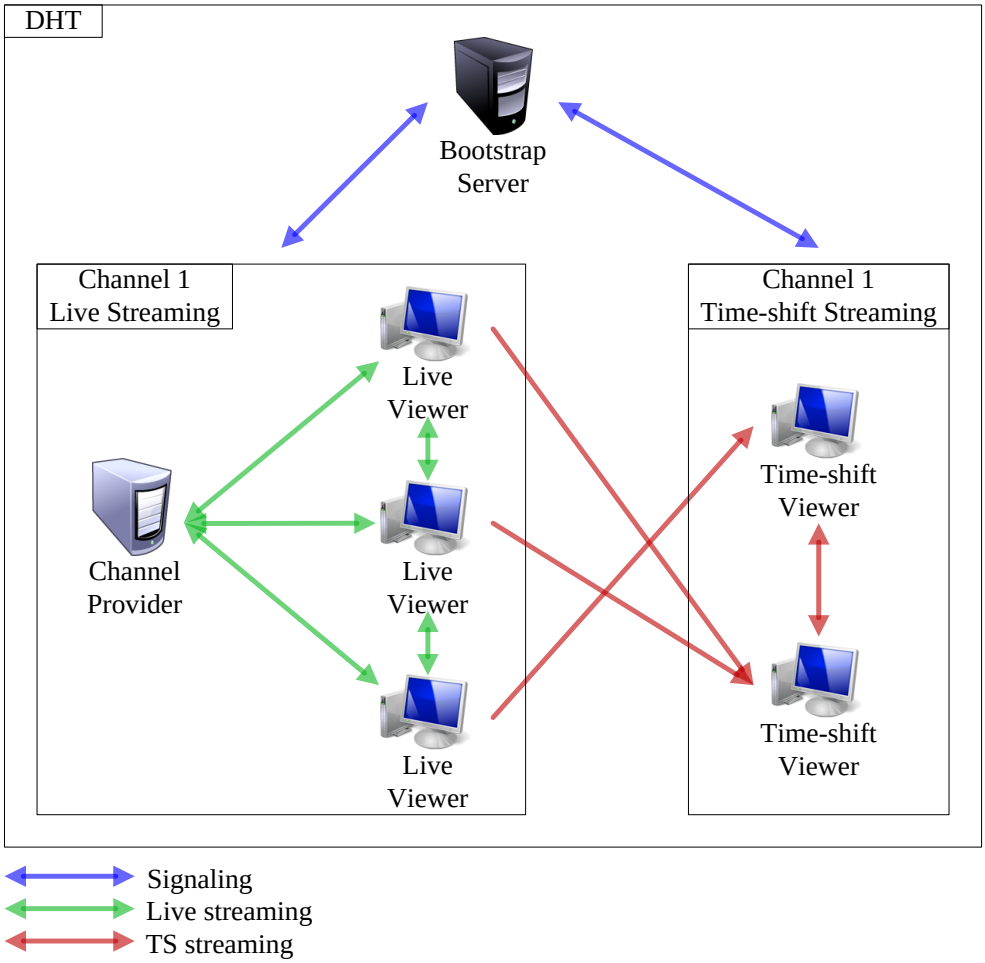

Figure 3-1 The system architecture.

Figure 3-2 depicts the block diagram of a node; a node can be a channel provider or a viewer. The player-buffer relationship depends on the type of the node. For a provider, the player encodes the original video stream into packet stream, and the stream data is put in the buffer for data transmission and genertaing its buffering status. For a viewer, the video content is also put in the buffer for data transmission, generating buffering status and playback. To share video content among peers, the buffered content can be transmitted to other peers for live streaming or time-shift streaming. 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 over TCP connections to avoid data losses in the network layer. Kademlia DHT is used to publish the cahed content, and its messages are transmitted over UDP packets.

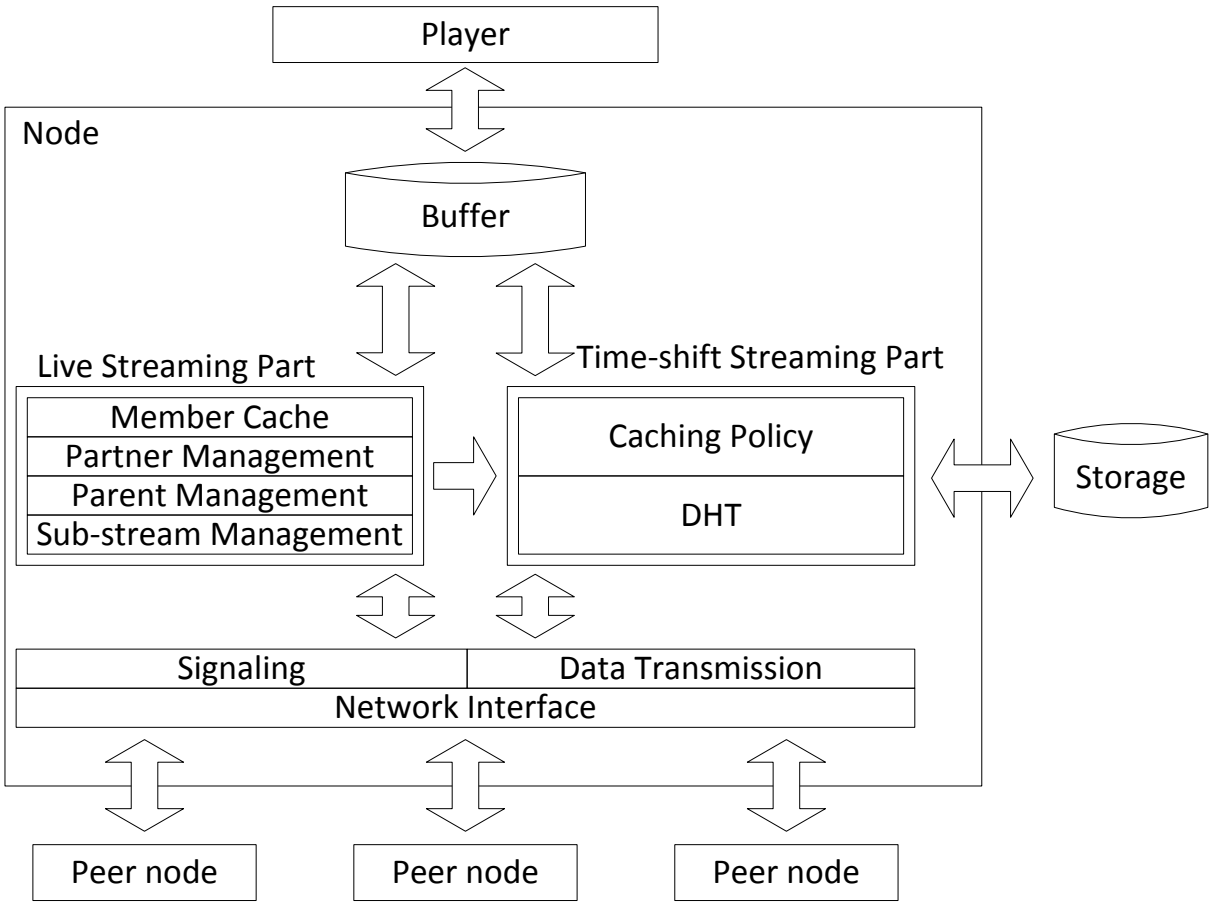

Figure 3-2 The block diagram of a node.

3.3 The streaming transmission unit

The streaming transmission flow of our system is depicted in Figure 3-3. The video stream is generated from a video source. A video provider encodes the video contents into continuous packets and transmits the video packets 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, the packet receiving times at the provider needs be recorded, so that the video packets can be played back synchronously. Therefore, we record the duration of each video packet.

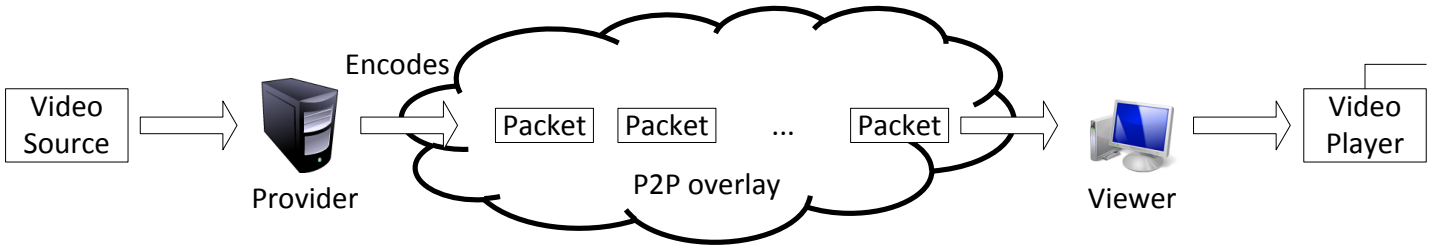

Figure 3-3 The streaming packet transmission flow.

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 provider, via local loopback interface. The video provider measures each packet's duration. Since it is inefficient to track each packet individually, continuous packets received in a second are packed into a video block. Furthermore, in order to support time-shift streaming, 10 consecutive blocks, with the starting block's timestamp aligned to 10's multiples, are packed into a video file for local storage purpose. The file is named after the information given by the channel provider, along with a timestamp. For example, a video file with name "ProviderName_Channel1_20100620182520" stands for 10 blocks of Channel 1 provided by ProviderName, with timestamp 2010/06/20 18:25:20. Figure 3-4 shows the structures of a video block and a video file.

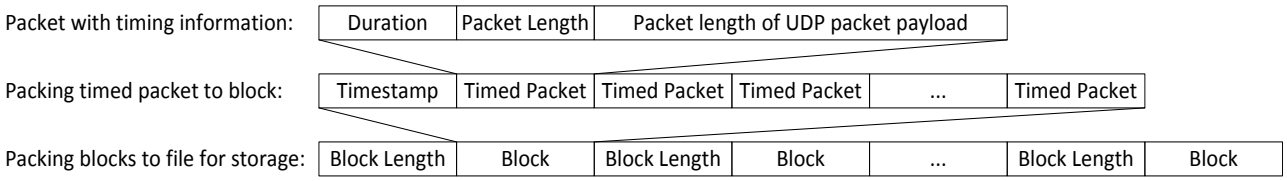

Figure 3-4 The structures of a video block and a video file.

2.2 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. In the following, we present the characteristics of the latest DONet/Coolstreaming, and our modifications as well.

1. Node hierarchy

Each live steaming node maintains three levels of nodes known to it: members, partners and parents. Members give the node a partial view of currently active nodes in the system, but no connection is established between the node and the known members. Connections are established between partners to exchange block availability information. Parent-child relations are formed when connections are established for video 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, each of which is one-second long and time-stamped. The blocks are 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 3-5 shows a video stream divided into four sub-streams with *S*=4. In our design, the video stream is divided into 8 substreams.

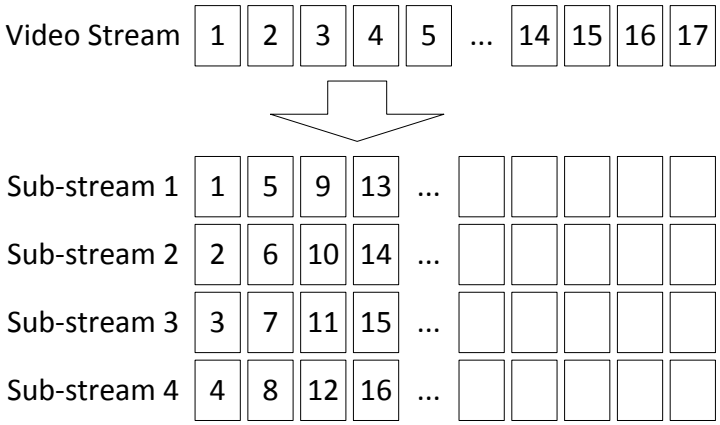

Figure 3-5 Sub-stream decomposition.

3. Joining procedure

When a node joins, it 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 on a list as its membership cache. Then the node randomly selects some of the nodes as its partners. Partners exchange their membership cache and the video block availability information periodically. In our design, a newly joined node first obtains the video block availability from its partners. Since each partner node may receive the video sub-streams at different paces, for each sub-stream, the node determine the fastest pace (i.e., the newest block) among its partners. The node would set its playback time point to be the slowest of all the fastest sub-streams, so that all the blocks the node would request have been received by some of its partners. After that, for each sub-stream, the node would subscribe the sub-stream from a partner whose pace is the closest to the initial playback time point. This would allow the node to receive all substreams at about the same pace.

4. Hybrid push-pull mechanism

To form a parent-child relationship, the node subscribes a sub-stream with a partner. When a partner node receives a subscription message with a designated starting timestamp, the partner 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 requested. The parent can be either the provider or another peer node. If it is the provider, it pushes a block to the subscribers whenever it finishes packing a new block. If it is another node, it pushes a block to the subscribers whenever it receives a new block. The subscription contract ends 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 lags in pushing blocks to its subscribers. A node can detect such lagging by comparing sub-stream receiving status of its parents, or comparing sub-stream receiving status between itself and its partners. The node compares the pace of each substream with the average of all sub-streams. If the largest difference among all sub-streams

is over a threshold, the node replaces the sub-stream that has the largest difference by subscribing to the partner whose timestamp of the substream is the nearest to the average. As shown in the lower part of Figure 3-6, 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 may indicate the parent node is overloaded, the parent re-selection procedure 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.

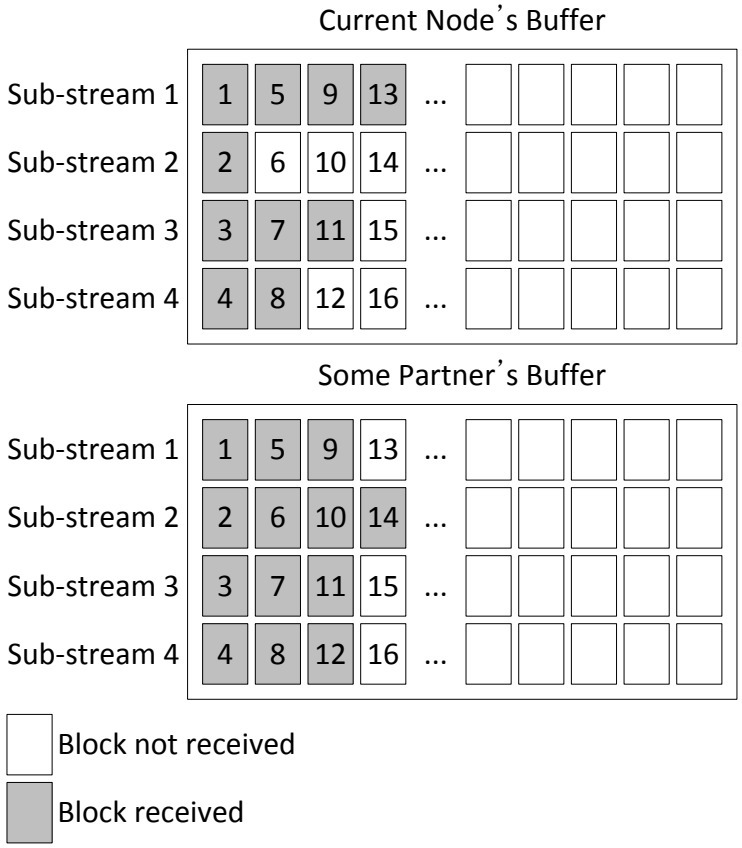

Figure 3-6 Comparing sub-stream status in parent re-selection.

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 for future time-shift viewers, the node publishes the ownership information on the DHT. However, the provider node caches all video contents but never publishes the ownership information. The provider node would act as a backup node; its cached contents can only be accessed at emergency. For example, when a block is 5 seconds to the time-shift playback

deadline but had not been received, or when there is no owner of a desired video file 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, Last_Update_Time> triples. Fig.3-5 depicts the relations between a file name and its owner list found on the DHT, and the structure of the owner list.

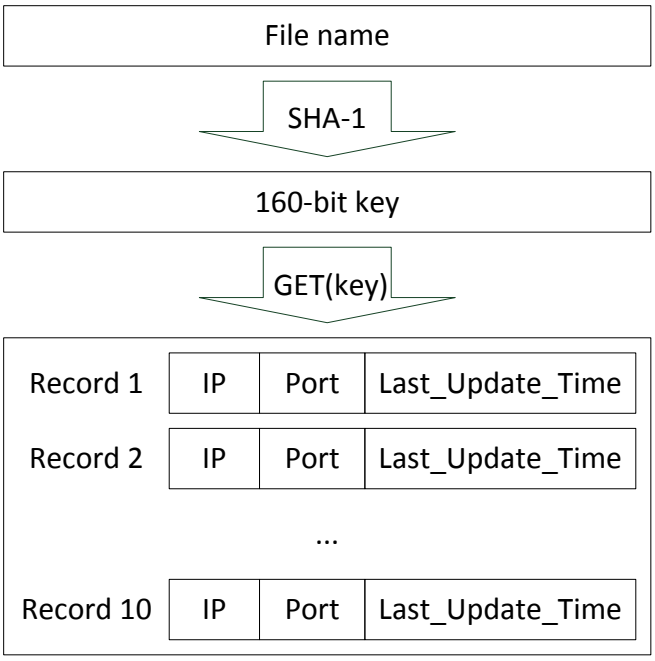

Figure 3-7 Getting the owner list from the DHT.

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 one. Then the node removes the record of two types: (1) the record put by itself in the past, and (2) the out-of-date records, which 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 would give the node enough time to do multiple updates by the republishing policy described later. After that, the node checks the number of replica. If the number has reached the desired number of replicas, the node deletes its cached file; otherwise, it adds 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. 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 node will back-off for a random interval before publishing to reduce such collisions. For the first time when a node updates a list, it will have a random back-off time uniformly distributed in (0, 50) sec. A node also republishes its cached files. The republishing operation is similar to the initial publishing operation, but is done periodically in order to keep the lists up to date and to alleviate the effects of over-written publishing. A node will periodically do the republishing operation with a random back-off time uniformly distributed in [600, 1200) sec. As we described above, the records on the DHT have a thirty-minute out-of-date threshold.

This means that each node can perform at least one republishing operation before the record is out-dated. The algorithm for random caching and random back-off for publishing is listed as follows.

```
01  while(waitngForBlocks)
02      block = node.receiveBlock();
03      buffer.put(block);
04      if(block and nearby blocks can be dumped)
05          viewerKnowledge = MAX(parent.size()+partner.size(), viewerCount);
06          rand = a random integer generated between (0, viewerKnowledge]
07          if(rand < replicasRequired)
08              dump blocks to local storage;
09              random back-off for DHT publishing;
10              fileOwnerList = DHT.get(filename);
11              remove out-dated entry and this node's entry in fileOwnerList
12              if(DHT.get(filename).size() < replicasRequired)
13                  FileOwnerList.add(this node);
14                  DHT.put(filename, fIleOwnerList);
15              else
16                  delete dumped file
17              end if
18          end if
19      end if
20  end while
```

Algorithm 3-1 Random caching and random back-off for publishing.

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 determine 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 active viewers is piggybacked to the node in the reply 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.

3.5 Time-shift streaming

For time-shift streaming, we adopt per-block pulling mechanism for content retrieval. After a node decides the channel and the starting time to watch, the name of the video file containing the required content is known. By querying with the file name on the DHT, the node obtains a list of the file owners. Then, the node would try to pull up to 4 blocks every sec., each from a randomly selected owner in the owner list. The reason why there's a limit on the number of pulling blocks in each sec. is that the available cached content may be much larger the buffer's capacity, so that it is necessary to keep the pulling timestamp staying 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. To ease the load of live streaming peers, we need to enhance the cooperation between time-shift peers. It means time-shift peers can share video content with time-shift peers that their watching points are near.

To share contents with near time-shift peers, time-shift peers form groups. A group consists of peers watching close-by video contents, i.e., their viewing time points are within a threshold. The center of a group is defined to be the median of the newest and the oldest viewing time points of the peers in the group. The radius of a group is the larger value of ten minutes or the distance from the center to the oldest viewing time points. A peer joins a group if its viewing time point is in the group's radius. If there is no such a group, the peer creates a group and makes itself as the center of the group. When a peer joins a group, the group adjusts its center.

The peers in a group exchange buffer map information with each other. The peer watches the older video contents can fetch contents from the peer watching newer contents. If their buffer maps are overlapped, peers watching older contents can fetch video blocks into their own buffer. If not overlapped, peers watching older contents can also prefetch video files beyond its buffer capacities and store them into local disks in the same way as live streaming peers store video files for time-shift viewers. The stored video files are likely to be used for time-shift playback by the peer itself and by other peers.

3.6 System Implementation

We have implemented the system in Java 1.6 using a request-reply model. The nodes communicate with each other with request messages, and the recipients repond with corresponding reply messages. The bootstrap server creates a ServerSocket for incoming messaging connections, Threads are created for each incoming connection and received messages are handled and replied to the connecting node. On the other hand, a provider/viewer node creates two ServerSockets, one for incoming messaging connections and the other for block transmission connections.

A message consists of the message type and the required options of the message. Messages are transmitted over TCP with Java Socket. The messages used in our system are listed below.

(1) Channel Registration

A channel provider registers its information with the bootstrap server. The options include

this node's messaging port number, channel provider's name and channel description. The bootstrap server replies with whether the registration is successful or not.

(2) Channel List

A viewer requests for the available channels registered at the bootstrap server. The options include the node's messaging port number, channel provider's name and channel description. The bootstrap server replies with a list of available channels' provider names and channel descriptions.

(3) Channel Join

For live streaming, this message is used for channel joining procedure; the options include this node's control port number, channel provider's name and channel description. The bootstrap server replies with a list of currently active nodes in the channel. For time-shift streaming, the message is used for DHT joining procedure, where the 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. The options includes 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 include 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 include 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 used for time-shift streaming viewer nodes to request a block from other nodes. The options include the 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 informs the node to request from the source.

4. Performance Measurements

4.1 Experiment Environment

To evaluate the system performance, we performed experiments on PlanetLab, an open global research network [20]. The streaming provider was 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 were located in the United States.

The video stream bit rate is 400 kbps, the number of sub-streams is 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 is uniformly distributed in (0, 50) sec. The random back-off for republishing is uniformly distributed in (10, 20) min. The system intends to keep 10 replicas for each block. Time-shift nodes cache each received block with probability 0.5. Table 4-1 lists the system parameters used in our system.

In the experiment, we first started the bootstrap server and streaming provider, and then all 64 nodes joined the system as a Poisson process, with an expected inter-arrival time of 60 seconds. For each time-shift node, it randomly selected a time between the time when the streaming started and the time it joined the system as the playback point. The experiment lasted 2 hours, and we assumed no peer churn.

Table 4-1 Experiment 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
The random back-off for publishing	(0, 50) seconds
The random back-off for re-publishing	(10, 20) minutes

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 can be played. End-to-end delay, also called playback delay, is the delay of the video content between the source and the viewer. Continuity index is the number of segments that arrive before or on the playback deadlines over the total number of segments a node should receive.

Figure 4-1 depicts the distribution of startup delay of the live streaming nodes in the experiment. Most of the nodes experience a startup delay less than 20 sec.; the average startup delay is 15 sec., which is a satisfactory result for P2P live streaming service. Figure 4-2 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 15sec., and the average end-to-end delay is 16 sec. Figure 4-3 depicts the statistics of the blocks received by each live streaming node. The red bars indicate the number of blocks received before the playback deadlines, the green bars indicate blocks received after the playback time, and the blue bars indicate blocks not received. The results indicate that most of the nodes have 100% continuity index,

i.e., received all blocks that they need. The average continuity index is 98.5%. However, Nodes 19, 26, 29, and 40 experience large numbers of lost blocks; this is due to the underlying TCP errors. Note that Node 19 experiences a large number of delayed blocks. The reason is still unclear to us.

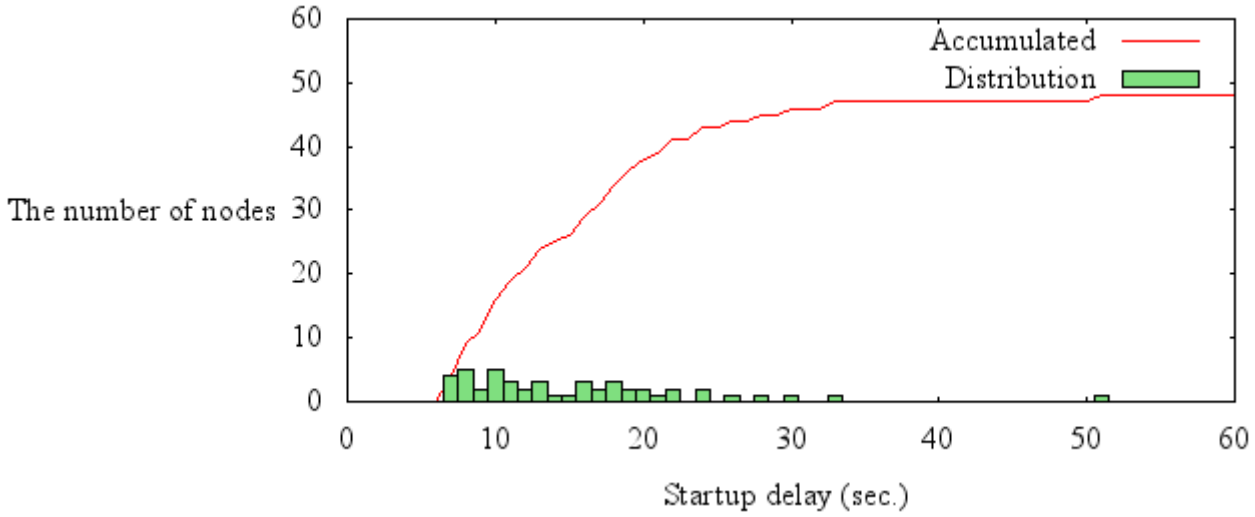

Figure 4-1 The distribution of startup delay of live streaming nodes.

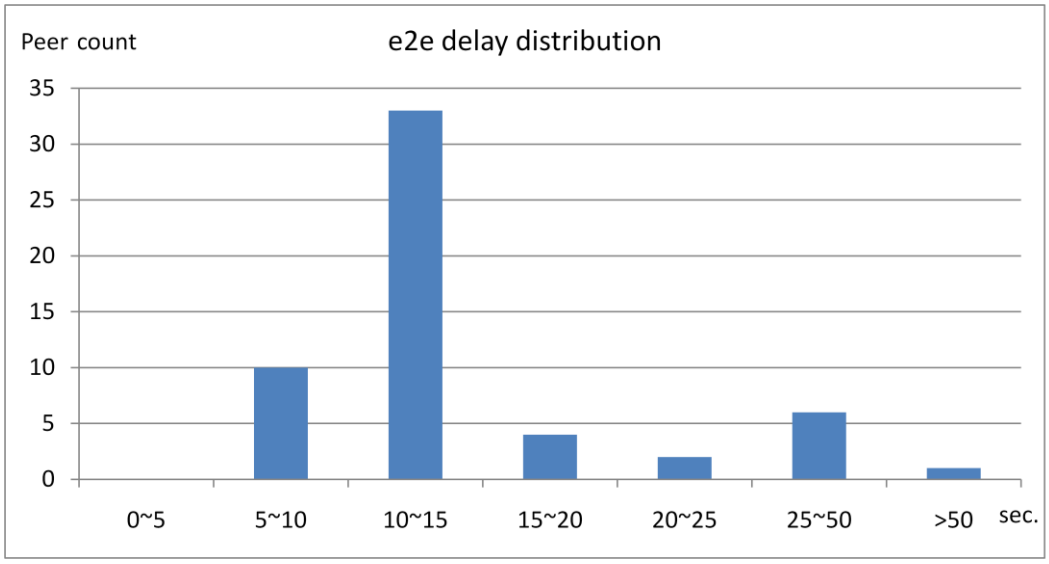

Figure 4-2 The distribution of end-to-end delay of live streaming nodes.

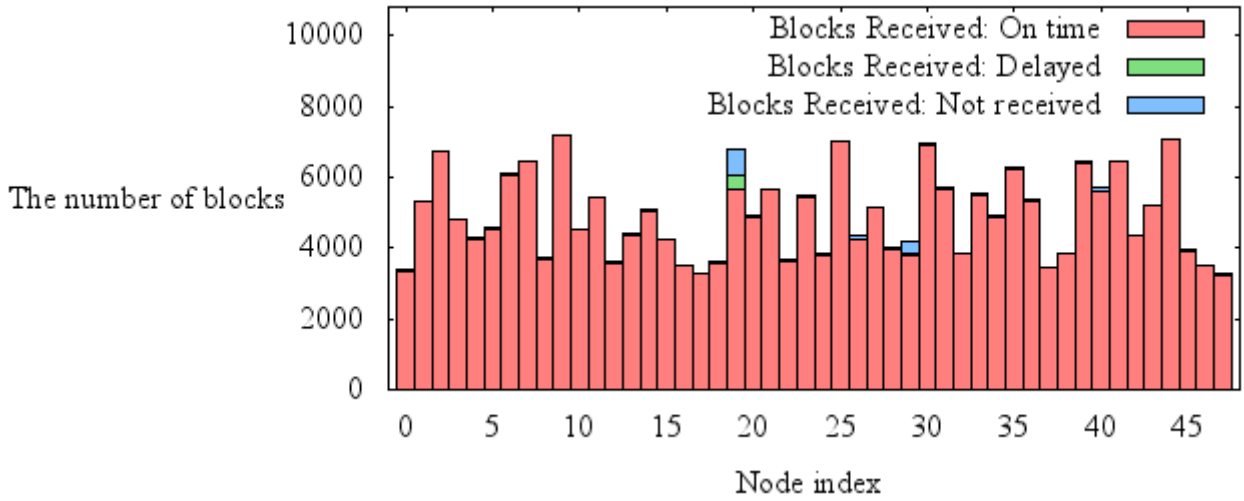

Figure 4-3 The block reception distributions of live streaming nodes.

4.2.2 The Time-shift streaming

To alleviate the effects of the initial stage when there are very few nodes and many unfinished publishing/re-publishing procedures, we only examine the vedio blocks generated between 30 to 90 minutes in the trial, which represents the stable system state. Fig. 4-4 shows the number of video files cached at each node; nodes with index in (0,48) are the live streaming nodes, and nodes with index in (54, 69) are the time-shift streaming nodes. The red bars indicate the number of file replicas cache on each node, and the green bars indicate the number of file replicas successfully published on the DHT from each node. Note that there are few nodes suffering from DHT failures, which make them unable to publish their file availability on the DHT. Each node caches 61.87 files in average, with standard deviation of 35.54. The results indicate that with the information of currently active nodes of the system, the caching responsibility obtained a satisfactory balancing among nodes in the system.

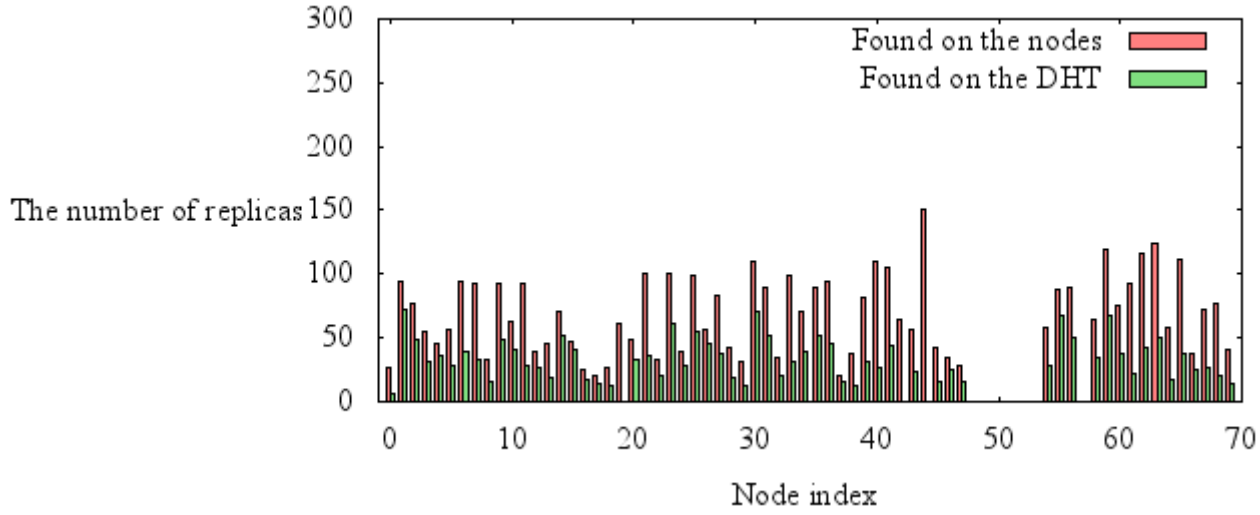

Figure 4-4 The number of cached files in each node.

Fig. 4-5 depicts the distribution of the number of replicas of each cached files on all the live streaming nodes and the time-shift streaming nodes. The red bars indicate the number of files that have the corresponding number of replicas cached by all the nodes. The green bars indicate the number of files that have the corresponding number of replicas successfully published on the DHT. The results indicate that our publishing and re-publishing algorithms need to be improved. While most video files have more than 10 replicas cached in all the nodes, only a small percentage of the files are successfully published 10 times on the DHT. This indicates that re-publishing processes may need a longer time to stabilize than we have expected.

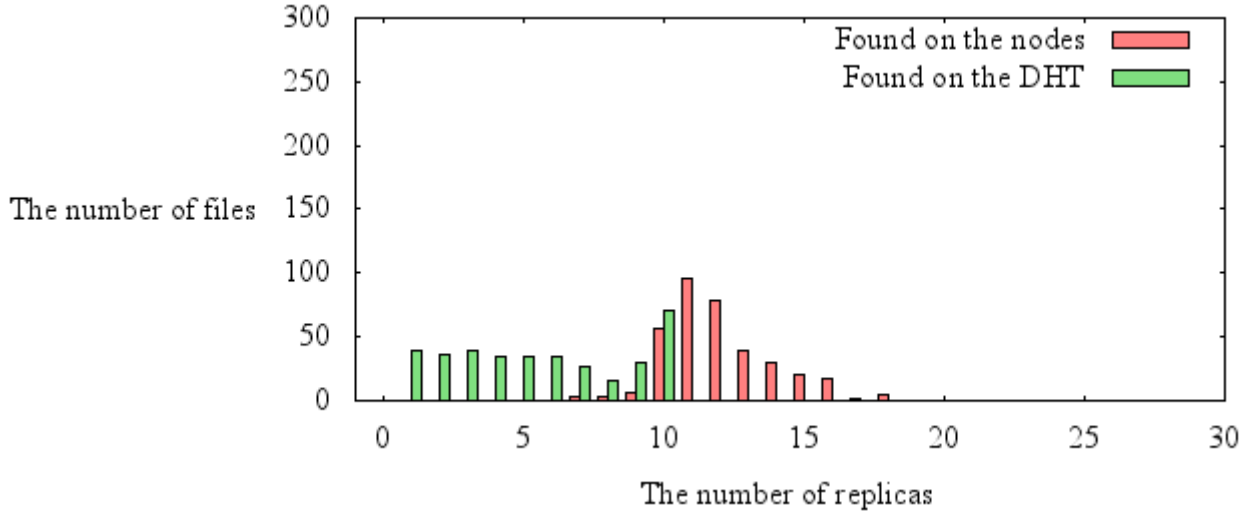

Figure 4-5 The distribution of replica counts on all the nodes and on the DHT.

Table 4-2 The sources of time-shift streaming blocks.

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 peer nodes	3227	2577	3789	0	4893	4830	4178	2742	6014	5356	2904	5330	2516	3524	3667	1796
From the provider	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 peer owner	0	9	9	0	0	9	0	16	0	0	0	0	0	80	0	0

Table 4-2 lists the number of blocks received from different sources for the time-shift streaming nodes in the experiment. All time-shift streaming nodes (TS 01-TS 16) have received a total number of 59033 blocks. 57343 (97.14%) of the received blocks were served by peer nodes. The provider served another 1690 blocks, 1044 of which were emergency handling, 522 were unable to obtain from peers, and 123 were not cached by peers. The results indicate that P2P time-shift streaming service is feasible since 97% of the video blocks are provided by the peers.

5. Conclusion and Future Work

In this report, we had implemented a P2P live/time-shift streaming system and presented a

distributed cache management strategies to cache a desired number of video file replicas by a random back-off publishing/republishing mehod. We also studied the performance of the system on PlanetLab. Our experiment results show the feasibility of live/time-shift systems. In our small scale experiments with 64 nodes, the live streaming part achieved a startup delay of 15 seconds, an end-to-end delay of 16 seconds and an average continuity index over 98%. Moreover, for the time-shift streaming service, using the streaming provider as an emergency handler, it achieved a continuity index of 100%, with over 97% of the streaming data were from P2P peers. Our proposed caching strategies effectively distribute the load of storing the time-shift contents and store at least ten replicas for most files. However, in this design, publishing cached video files on the DHT has synchronization problems. Although the problems are alleviated by the random back-off publishing/republishing, publishing collisions still occur. Some of the cached video files cannot be successfully published on the DHT. In the future, more investigation is needed on this publishing issue. In addition, 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] 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] 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

[15] 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

[16] D. Gallo 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/

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

國科會補助計畫衍生研發成果推廣資料表

日期 2010年10月29日

國科會補助計畫	計畫名稱: 點對點實況及移時播放之影音服務 計畫主持人: 張明峰 計畫編號: 98 -2221-E -009 -083 - 學門領域: 計算機網路與網際網路		
研發成果名稱	(中文) 點對點實況及移時播放之影音服務		
	(英文) P2P Live and Time-shifted Streaming Services		
成果歸屬機構	國立交通大學	發明人 (創作人)	張明峰, 李茂弘, 邱順胤
技術說明	(中文) 我們設計實作一點對點實況/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，並在PlanetLab平台上進行實驗，分析此一系統的可行性及效能。我們的實驗數據顯示，在規模為64節點的實驗中，我們系統中個別節點的啟動延遲平均為16秒，訊號源到各節點的端到端平均延遲小於20秒，而串流接收的連續係數平均大於98%。由於使用訊號源備份作為緊急資料源，移時串流節點的連續係數高達100%，其中97%來自其他節點，3%來自訊號源。		
	(英文) We have designed and implementated a P2P live/time-shift streaming system, and a distributed cache management strategies for time-shift video cache files. In addition, we study the system's performance and characteristics on the PlanetLab experiment platform. In our initial experiments with 64 nodes, the live streaming part achieved a startup delay of 15 seconds, an end-to-end delay of 16 seconds and a continuity index over 98%. Moreover, for the time-shift streaming service, our system achieved a continuity index of 100%, with over 94% of the streaming data were from P2P peers.		
產業別	資訊服務業		
技術/產品應用範圍	網路視訊服務		
技術移轉可行性及預期效益	P2P即時視訊傳送，以及節點儲存視訊資料支援移時視訊服務之軟體可供技轉。此一軟體為P2P視訊服務提供新功能 — 移時視訊服務，能為業者吸引新型態的用戶。		

註：本項研發成果若尚未申請專利，請勿揭露可申請專利之主要內容。

98 年度專題研究計畫研究成果彙整表

計畫主持人：張明峰	計畫編號：98-2221-E-009-083-
計畫名稱：點對點實況及移時播放之影音服務	

成果項目			量化			單位	備註（質化說明：如數個計畫共同成果、成果列為該期刊之封面故事...等）
			實際已達成數（被接受或已發表）	預期總達成數(含實際已達成數)	本計畫實際貢獻百分比		
國內	論文著作	期刊論文	0	0	100%	篇	
		研究報告/技術報告	1	1	100%		
		研討會論文	0	0	100%		
		專書	0	0	100%		
	專利	申請中件數	0	0	100%	件	
		已獲得件數	0	0	100%		
	技術移轉	件數	0	0	100%	件	
		權利金	0	0	100%	千元	
	參與計畫人力（本國籍）	碩士生	3	3	100%	人次	
		博士生	1	1	100%		
		博士後研究員	0	0	100%		
		專任助理	0	0	100%		
國外	論文著作	期刊論文	0	1	0%	篇	
		研究報告/技術報告	1	1	100%		
		研討會論文	0	0	100%		
		專書	0	0	100%	章/本	
	專利	申請中件數	0	0	100%	件	
		已獲得件數	0	0	100%		
	技術移轉	件數	0	0	100%	件	
		權利金	0	0	100%	千元	
	參與計畫人力（外國籍）	碩士生	0	0	100%	人次	
		博士生	0	0	100%		
		博士後研究員	0	0	100%		
		專任助理	0	0	100%		

其他成果 (無法以量化表達之成果如辦理學術活動、獲得獎項、重要國際合作、研究成果國際影響力及其他協助產業技術發展之具體效益事項等，請以文字敘述填列。)	我們設計實作一點對點實況/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，並在 PlanetLab 平台上進行實驗。此一成果將撰寫論文投稿國際期刊，並可技轉給業界。

	成果項目	量化	名稱或內容性質簡述
科教處計畫加填項目	測驗工具(含質性與量性)	0	
	課程/模組	0	
	電腦及網路系統或工具	0	
	教材	0	
	舉辦之活動/競賽	0	
	研討會/工作坊	0	
	電子報、網站	0	
	計畫成果推廣之參與（閱聽）人數	0	

國科會補助專題研究計畫成果報告自評表

請就研究內容與原計畫相符程度、達成預期目標情況、研究成果之學術或應用價值（簡要敘述成果所代表之意義、價值、影響或進一步發展之可能性）、是否適合在學術期刊發表或申請專利、主要發現或其他有關價值等，作一綜合評估。

1. 請就研究內容與原計畫相符程度、達成預期目標情況作一綜合評估

■達成目標

□未達成目標（請說明，以 100 字為限）

□實驗失敗

□因故實驗中斷

□其他原因

說明：

2. 研究成果在學術期刊發表或申請專利等情形：

論文：□已發表 ■未發表之文稿 □撰寫中 □無

專利：□已獲得 □申請中 ■無

技轉：□已技轉 □洽談中 ■無

其他：（以 100 字為限）

我們設計並實作一點對點實況/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，並在 PlanetLab 平台上進行實驗，分析此一系統的可行性及效能。目前已完成第一版文稿準備投稿。

3. 請依學術成就、技術創新、社會影響等方面，評估研究成果之學術或應用價值（簡要敘述成果所代表之意義、價值、影響或進一步發展之可能性）（以 500 字為限）

目前 P2P 串流系統極少同時支援即時和移時服務的系統。我們設計並實作一點對點實況/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，並在 PlanetLab 平台上進行實驗，分析此一系統的可行性及效能。我們的實驗數據顯示，在規模為 64 節點的實驗中，我們系統中個別節點的啟動延遲平均為 16 秒，訊號源到各節點的端到端平均延遲小於 20 秒，而串流接收的連續係數平均大於 98%。由於使用訊號源備份作為緊急資料源，移時串流節點的連續係數高達 100%，其中 97%來自其他節點，3%來自訊號源。我們初步的成果顯示 P2P 串流系統的可行性，但我們設計之節點之儲存視訊發佈方式尚待提昇效率。此一研究成果有觀念和技術的創新，應用於目前普遍之 P2P 串流服務之功能提升。