

TuBeck: a novel peer-to-peer streaming system with Loopback-MDC for scalable H.264/AVC videos

Chow-Sing Lin · Rong-Hua Chang · Jhe-Wei Lin

Published online: 27 April 2013
© Springer Science+Business Media New York 2013

Abstract With the rapidly development of broadband networks, more and more users nowadays tend to acquire their desired videos from media-on-demand servers. How to efficiently provide multimedia contents for a large number of heterogeneous users on the Internet has become a noticeable issue. In our previous work, we proposed the Loopback-MDC scheme on CDN-P2P network to address such a scalability issue. In this paper, we present the design and implementation of a novel peer-to-peer streaming system with the Loopback-MDC on actual networks, named *TuBeck*. The TuBeck consists of *preprocessor*, *server*, *peer*, and *player* modules with the support of *network infrastructure library*. In TuBeck, multimedia sources are preprocessed to multiple descriptions. Each description is divided as a sequence of H.264/AVC chunks by modified JM encoder for real-time streaming. The network infrastructure library provides the nodes (server and peers) with the fundamental functionalities of network connections and communications. Message exchange and streaming control among nodes follow the pre-defined *LM* protocol. The server, peer, and player modules are functionally multithreaded in order to enhance the system performance. The experimental results show that the Loopback-MDC is practical for realizing a P2P streaming system in terms of the server loading, CPU/Memory usage, and efficiency of failure recovery with respect to various arrival rates, failure rates, and viewing qualities.

Keywords Streaming · Peer-to-peer · Multiple description coding · Loopback-MDC · H.264

C.-S. Lin (✉) · R.-H. Chang · J.-W. Lin
Department of Computer Science and Information Engineering,
National University of Tainan, 33, Sec. 2, Shu-Lin St.,
Tainan 700, Taiwan, Republic of China
e-mail: mikelin@mail.nutn.edu.tw

1 Introduction

The main concern of a media server is to maintain high scalability so that when a great number of requests suddenly arrive at a media server, known as the flash-crowd effect, the system can still serve each request within a reasonable delay without breakdown. It challenges the network bandwidth in a media system to accommodate such bursty requests. To effectively address this issue, multicast techniques, such as *batching* [23] and *patching* [12], have been theoretically considered as effective approaches to scale well with the increase of the number of clients. However, due to lack of the widespread use of the multicast on current IP networks, such schemes are difficult to implement to accommodate large-scale operations.

To practically address the issue of the scalability of media streaming, peer-to-peer (P2P) streaming systems which implements network multicast at application layer have been proposed [5, 11, 22, 24]. In a P2P system peers (clients) collaborate with each other for stream relay in order to reduce server loading, in which a peer not only acts as a client to receive video chunks (blocks) but also as a server to forward received video chunks to other peers. As more and more peers join the network, the aggregate peer resources raise the capacity of a P2P system for servicing more latecomers. In this way the flash crowd problem is effectively addressed. Because in a P2P streaming system peers are free to come and go, the reliability of service provision has become a major issue to be addressed. An alternative to solve the flash crowd problem is the Content Distribution Network (CDN), which is based on placing a number of proxy servers at the edges of the Internet to reduce the access load of the media server. Video contents are first distributed to proxy servers and then delivered to clients in their neighborhoods. It is suggested that the deployment and maintenance of a CDN proxy server [15, 26] is quite expensive. Since a proxy server often has limited resources and may be unable to cache an entire video, a prefix of a video is usually cached [15, 19, 20]. With only caching the partial video in the proxy server, it is crucial to effectively utilize buffer space of clients watching the same vide to help caching the rest of video segments for circulating the entire video stream in order to reduce the server interventions.

Many initiatives have demonstrated the feasibility of combining the CDN and P2P approaches to gain the scalability advantage of P2P as well as the reliability and manageability advantages of CDNs [6, 8, 13, 26]. Such a hybrid CDN-P2P scheme is a cost-effective approach to provide large scale media streaming on the Internet. The *Loopback* proposed in [15] is a hybrid CDN-P2P streaming system which not only aggregates the network bandwidth of peers to reduce the network consumption of a proxy but also effectively utilizes precious buffer space in a proxy by shifting video caching to collaborative peers. In *Loopback*, clients arriving closely form a forwarding *loop*, like *chaining* [12], with the first client receiving video chunks from a proxy and the last client returning chunks to the proxy. A video stream is forwarded from one peer to the next within a loop in the order of their arrival. Even if resources contributed by clients are limited, the *Loopback* scheme has shown the significant reduction in the storage space and bandwidth requirements of a proxy and the loads of central server. Despite its superiority, in the *Loopback* providing only a single version, non-scalable stream is assumed. As a result, it would fail to handle the heterogeneity of peers where peers have various downloading capability and require different viewing qualities.

Recent studies have shown that the multiple description coding (MDC) technique [1–4, 10] can be adopted to solve the issues of *Asymmetric Access Link*, *Heterogeneity*, *Dynamics*, and *Resource Utilization* in P2P systems defined in [18]. MDC is a scalable coding technique and encodes a video into several streams, called *descriptions*. A client can watch a certain quality of a video even though only a subset of descriptions are received. The more descriptions a peer receives, the higher quality it perceives. Different from *Layered Encoding* technique [7, 25], there is no decoding dependency among descriptions. Descriptions can be arbitrarily composed and decoded. From a client's perspective, MDC provides more flexibility in the selection of parent peers and fault tolerance [16]. P2P streaming systems such as *CoopNet* [22], *SplitStream* [5], and *Resilient P2P Streaming* [21] have successfully demonstrated the advantages of MDC.

In our previous work, we applied the MDC technique to the Loopback to solve the issues described above, called Loopback-MDC [17]. In Loopback-MDC we discussed the policies of selecting descriptions to watch for new clients, and proposed the *Open-Loop-First (OLF)* approach to optimize the availability and robustness of a sharing loop for newly arrived peers. In addition to the *intra-description (intra-d) recovery* implied in the Loopback, we also discussed the feasibility of *inter-description (inter-d) recovery* for recovering missing video chunks from the departure of peers. The proposed *sewing* recovery approach for inter-d recovery is to move video chunks of other descriptions to “sew” the disrupt caused by the departure of peers. Simulation results show that the proposed loopback-MDC approach can significantly reduce the amount of uploading bandwidth of a proxy on CDN-P2P video-on-demand systems, and the inter-d recovery, *sewing*, can further effectively recover missing video chunks and restore the continuity of a breaking sharing loop among peers.

In this paper, we present the design and implementation of a novel peer-to-peer streaming system with the Loopback-MDC for H.264/AVC videos on actual networks, named *TuBeck*. To our best knowledge, we believe that this paper is the first one thoroughly presenting the design and implementation of a P2P system for streaming MDC videos in H.264 format. The *TuBeck* consists of *preprocessor*, *server*, *peer*, and *player* modules with the support of *network infrastructure library*. In the *TuBeck*, multimedia sources are preprocessed to multiple descriptions. Each description is divided as a sequence of H.264/AVC chunks by modified JM encoder [14] for real-time streaming. We also developed the network infrastructure library to provide the nodes (server and peers) with the fundamental functionalities of network connections and communications. Message exchange and streaming control among nodes follow the pre-defined *LM* protocol. The server, peer, and player modules are functionally multithreaded in order to enhance the system performance. The experimental results show that the Loopback-MDC is practical for realizing a P2P streaming system in terms of the server loading, CPU/Memory usage, and efficiency of failure recovery with respect to various arrival rates, failure rates, and viewing qualities. It is worth mentioning that we used the encoder and decoder modules in JM reference software written in C++, which are packaged as a dynamic link library (DDL), to transcode video sources between YUV and H.264 formats, and the *TuBeck* was developed in C#.net. In conclusion, the contribution of this paper can be summarized as follows:

1. the design of system architecture of a P2P streaming system with the Loopback-MDC for H.264 videos, named *TuBeck*,

2. the way to preprocess a video into MDC descriptions in H.264 format by the tools of JM reference software,
3. the design and implementation of server module, peer module, and player module,
4. the design of communication protocol for the TuBeck, and
5. the performance study of the TuBeck on real network environment.

The remainder of this paper is organized as follows. In Section 2.2 we summarize the related works of the Loopback-MDC. In Section 3 we describe the overview of system architecture of the TuBeck. The preprocess of multimedia sources, server module, and peer module are introduced in Sections 4, 5, and 6, respectively. In addition, we also present the design of a MDC player for the TuBeck in Section 7. In Section 8, we introduce the LM protocol for communications in the TuBeck. In Section 9, we present the performance study of the TuBeck on emulated network environment. Finally, we conclude this paper in Section 10.

2 Related works

2.1 Loopback

The *Loopback* [15] scheme is a local client cooperation approach for streaming videos on a hybrid CDN-P2P network. Each client has limited buffer space to cache data received from a proxy server for playback. The forwarding bandwidth of a client used for delivering cached video blocks to another client or loopbacking to the proxy server is assumed to be the playback rate. Clients requesting the same video are organized into a collaborative network by their proxy server. Since the proxy storage space is limited, only the prefix of a video is cached in the proxy server.

Figure 1 shows the illustration of the Loopback scheme, and each square box represents a client and indicates its buffer size. The *first* client requesting a video receives the video prefix from the designated proxy server and the remaining portion of a video from the central server. The received video chunks are played and cached in sequence in the client's buffer. If the next client requesting the same video arrives before its buffer is filled up, i.e., the initial chunk is still kept in the buffer, the video

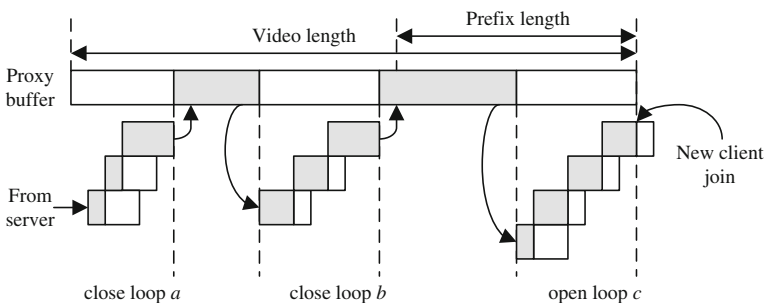


Fig. 1 Illustration of the Loopback scheme

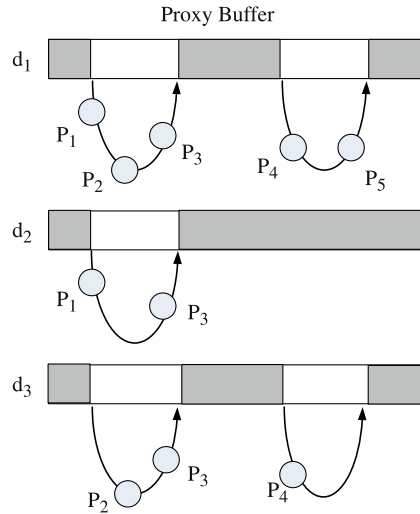
chunks will be delivered to the newcomer. By this cache-and-relay approach, clients requesting the same video can form a loop to share video chunks. As long as the last client of the loop still keeps the initial chunk of the video in the buffer, a new client requesting the same video can still join the loop. This kind of loop is called, *open loop*, as the open loop *c* illustrated in Fig. 1. On the contrary, if the next request for the video does not arrive in time and the buffer of the last client in the loop becomes full, the chunks of the last client are passed back to the proxy server in order. Since the initial chunk of the video is no longer available in the loop, no more new clients can join and the loop is *closed*. This kind of loop is called *close loop*, as close loops *a* and *b* illustrated in Fig. 1. In general, the bandwidth consumption of a proxy server for streaming a video to clients is proportional to the number of loops, instead of the number of clients. The duration of an open-loop depends on the buffer length and the arrival rate of peers.

2.2 Loopback-MDC

In Loopback-MDC a video is further encoded into n descriptions by the MDC technique. The buffer space allocated to a video in the proxy server is equally divided into n sub-buffers. Each sub-buffer is used to cache a description. The loopback scheme is applied on each sub-buffer for sharing description chunks among peers. When a new peer joins, it first decides to receive which descriptions and then searches for an existing open loop to join for each requesting description. If there is none for the requested description, then it opens a new loop for receiving. It is noted that a peer can serve exactly one peer for each requested description in the loopback-MDC scheme. When a new peer joins an open loop, it starts to receive the initial and subsequent chunks from the last peer in the loop and resets the expiration time of the open loop based on its buffer capacity. It now becomes the last peer of the loop. Before its buffer space is filled up, the loop is still open and can serve other new peers. If there are no new peers joining in, this open loop will be eventually closed and cannot serve other new peers. Figure 2 shows the loopback-MDC scheme. In this example, the video is encoded into three descriptions, d_1 , d_2 and d_3 . Peer P_1 requests the view quality of two descriptions and receives d_1 and d_2 descriptions from the proxy server. Two open loops are generated, one for d_1 and the other for d_2 . Next, peer P_2 requests the viewing quality of two descriptions and receives d_1 from P_1 and d_3 from the proxy server, creating another new open loop for d_3 . Then, peer P_3 arrives and joins the three open loops for receiving the descriptions d_1 and d_3 from P_2 and the description d_2 from P_1 . Later, peer P_4 arrives and there are no open loops available. P_4 then receives d_1 and d_3 descriptions from the proxy server, forming two new open loops. Lastly, peer P_5 requesting the viewing quality of one description arrives and selects the open loop of d_1 to join.

Since the consumption of the proxy bandwidth is proportional to the number of loops, it is crucial to extend the availability of open loops as long as possible to reduce the probability of opening new loops when a peer joins. For extending the availability of open loops, The *Open-Loop-First* (OLF) policy for description selection was proposed in Loopback-MDC. With the OLF the selection of descriptions for a new peer is based on the existence of open loops. If there are multiple open loops to be selected, the one with the shortest expiration time is selected first. Note that the

Fig. 2 Illustration of the Loopback-MDC scheme

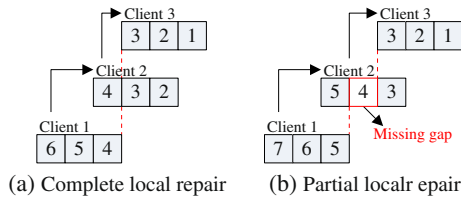


expiration time of an open loop can be determined by the time that the buffer space of last peer in the open loop is filled up. Figure 3 shows an example of the OLF for description selection with a video encoded as 4 descriptions and each peer equipped with the buffer of 3 time units. The first peer A arrives at time 1 and requests two descriptions. Descriptions d_1 and d_2 are allocated to peer A to receive, and each creates a new open loop. In the figure $X - n$ represents the n -th description selection of peer X . For example, $A - 1$ and $A - 2$ denote the first and the second description selection of peer A. Peer B arrives at time 2 and requests three descriptions. Since there are only two open loops, peer B joins d_1 and d_2 open loops, and randomly selects another description from d_3 and d_4 , say d_3 , and then creates a new open loop of d_3 for receiving. Peer C arrives at time 3 and requests 4 descriptions of full viewing quality. Similarly, three open-loop descriptions d_1 , d_2 , d_3 are first selected, and a new open loop is created for receiving description d_4 . Peer D arrives at time 4 and only requests two descriptions. Since the expiration times of the four open loops are the same, two of the four open-loop descriptions are randomly selected, say d_1 and d_2 . Next, peer E arrives at time 5 and requests three descriptions $\{d_3, d_4\}$ which can be selected from the four open-loop descriptions. Descriptions $\{d_3, d_4\}$ with the same shortest expiration time are selected first. The

Fig. 3 An example of the OLF for description selection with a video encoded as 4 descriptions and each peer equipped with the buffer of 3 time units

| Des. \ Time | 1 | 2 | 3 | 4 | 5 |
|-------------|-------|-------|-------|-----|-----|
| d_1 | (A-1) | B-1 | C-1 | D-1 | E-3 |
| d_2 | (A-2) | B-2 | C-2 | D-2 | |
| d_3 | | (B-3) | C-3 | | E-1 |
| d_4 | | | (C-4) | | E-2 |

Fig. 4 Two scenarios of local repairs in Loopback



remaining one is randomly selected from the d_1 and d_2 open-loop descriptions, say d_1 .

2.3 Peer recovery process

In P2P network peers are free to come and go. When a collaborative client fails, other clients in the loop may experience service disruption and the server has to transmit extra data to recover such failures. In Loopback, such a failure can be completely or partially repaired by using *caching duplication*. Figure 4 shows the two scenarios of local repairs. In Fig. 4a peer 1 can still smoothly relay video blocks to peer 2 without missing any blocks when peer 2 leaves. The failure of peer 2 can be completely locally repaired. On the other hand, in Fig. 4b there is a missing gap of block 4 between peer 1 and peer 3, and the continuity is broken. In this case, the failure of peer 2 is only partially local repaired. As a result, the server needs to consume one extra streaming bandwidth to serve peer 3.

In Loopback-MDC scheme, since a video stream is encoded into multiple descriptions, missing blocks of the broken loop may be repaired by *intra-d* or *inter-d* recovery. When a peer fails, the absent blocks may be completely locally repaired by intra-d recovery as the loopback scheme does. It is noted that the intra-d recovery is similar to failure recovery of P2P streaming found in previous studies, such as [9, 15, 23]. With the Loopback-MDC scheme, missing description blocks happening in the case of partial local repair may be repaired by blocks from other descriptions, which are called inter-d recovery. When a peer fails, first we apply intra-d recovery to repair missing blocks locally, and then apply inter-d recovery if necessary. Finally, the server is responsible for repairing any remaining missing blocks. The key concept

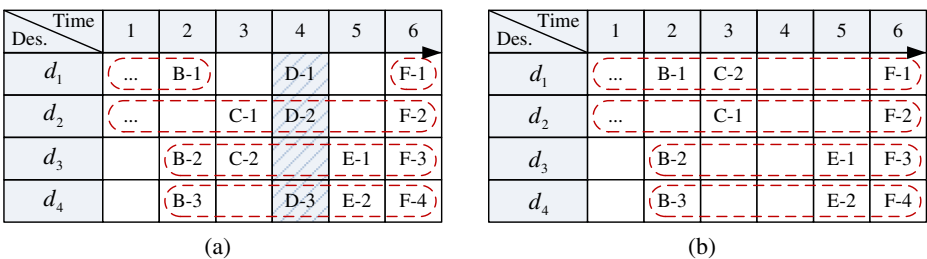


Fig. 5 An example of inter-d recovery (sewing) with a video encoded as 4 descriptions and each peer equipped with the buffer of 3 time units

of inter-d recovery for a broken loop is to move a nearby peer of the failed peer to join the broken loop and “sew” up the gap. This approach is also called *sewing* in the Loopback-MDC scheme. Figure 5 shows an example of sewing with a video encoded as 4 descriptions and each peer equipped with the buffer of 3 time units. When peer D fails shown in Fig. 5a, there are three broken loops of d_1 , d_2 , and d_4 . Loops of d_2 and d_4 are repaired by intra-d recovery. Since peer B and peer F are three blocks apart, the gap in loop d_1 cannot be locally repaired. By applying the inter-d recovery, as shown in Fig. 5b, peer C now starts to receive description d_1 instead of d_3 and joins the loop of d_1 to narrow the gap. After the repairing, in loop d_1 there is no gap between peer B and peer C and the gap between peer C and peer F is two blocks apart without causing loop broken. Note that in this case peer C still receives two descriptions without degrading viewing quality.

3 System architecture of TuBeck

Based on the data dependency, the system architecture of TuBeck, showed in Fig. 6, is functionally divided into modules which are *Preprocessor*, *Server*, *Peer*, *Player*, and *Network Infrastructure Library*. The *Preprocessor* module is responsible for acquiring video in YUV format, disassembling multiple descriptions, and encoding descriptions as H.264 chunks by JM [14]. In this paper video sources are acquired in YUV format and they are disassembled into multiple descriptions by *Sub-sampling Multiple Description Coding (SMDC)* [4]. These descriptions are then converted into serial video chunks in H.264 format for efficient network delivery by *JM H.264 encoder*. Finally, the descriptions in H.264 are stored in disk arrays for vide access. The server module consists of four components, stream transmitter, stream control management, P2P topology, and resource management. The *stream transmitter* is responsible for establishing socket connection between the server and a peer and delivering required video chunks to a peer under a specified rate control through networks. The *stream control management* is responsible for managing control messages of session initiations and interactions between a server/peer and a peer. The peers’ status and the network structure of P2P topology are maintained in *P2P topology*. The *resource management* is responsible for providing access functionality for retrieving and storing video chunks of a specific video resided in disk arrays

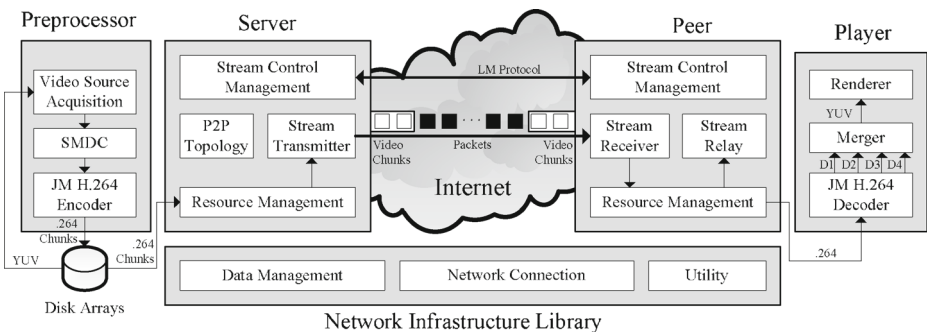


Fig. 6 System architecture of TuBeck

of a server. The peer module includes four components. In addition to stream control management and resource management whose functionalities are similar to corresponding components in server module, the *stream receiver* sequentially receives video chunks from server/peer and then pass them to the resource management for video caching. The *stream relay* is responsible for relaying cached video chunks to subsequent peers based on the instructions sent by the stream control management. Note that the communication between the stream control management in server module and peer module is based on our pre-defined LM protocol. The *JM H.264 decoder* in the player module regularly fetches the buffered video chunks and decodes them into various descriptions in YUV format. These descriptions are combined in the *merger* as one YUV video stream, and finally it is sent to the *renderer* for display. Lastly, the *Network Infrastructure Library* is responsible for providing the server and peer modules with common functions including data management, network connection, and other utilities.

4 Preprocess of multimedia sources

Figure 7 shows the data flow of preprocessing video sources. In TuBeck, the video sources were acquired in planar YUV format with chroma subsampling 4:2:2. Video sources were pre-processed into four multiple descriptions by SMDC and each was then sent to a separate H.264/AVC encoder to output compressed H.264 videos for efficient video repository and network delivery. We used the JM encoder module, *lencod.exe*, developed by Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG [14]. Because the restriction of JM encoding process, the width and the height of a video must be a multiple of 16 and therefore we only adopted CIF (352 × 288) and QCIF (176 × 144) resolution videos. In TuBeck, each description was partitioned as a series of *chunks* which contains 10 frames, one group of pictures

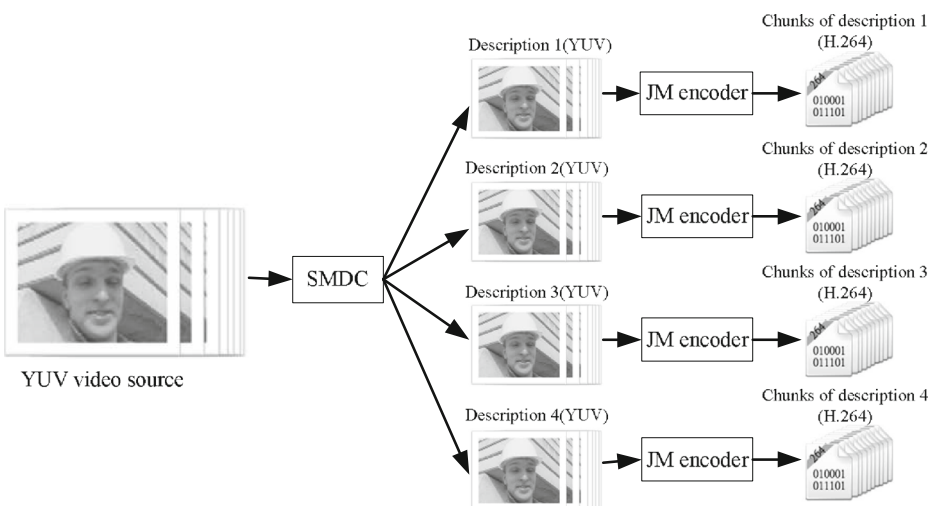
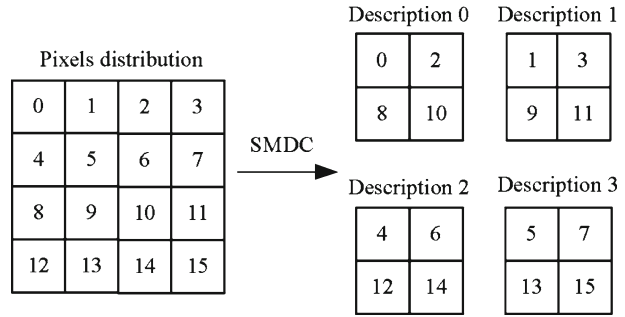


Fig. 7 Data flow of preprocessing video sources

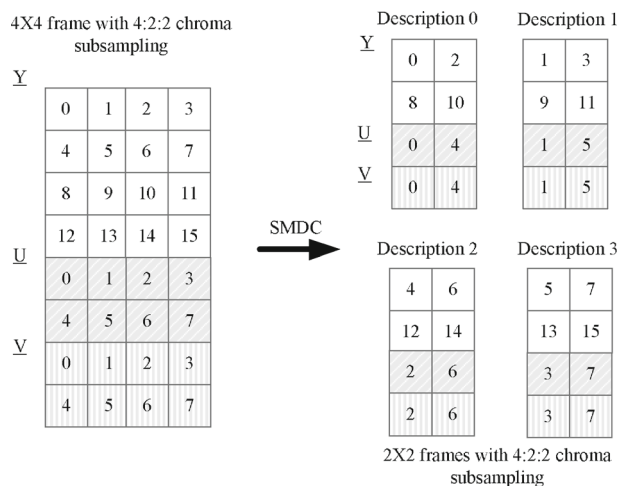
Fig. 8 An example of coding a 4×4 luma samples into 4 descriptions by SMDC



(GOP). The frame rate is set to be 30 fps. These important factors were fed into the JM encoder along with the default coding parameters to transcode a description in YUV format into a series of video chunks in H.264 format.

In the SMDC, each input frame is sampled along its rows and columns with a sampling factor of 2. Let $x(i, j)$ be the luminance sample of the current frame at position (i, j) , then the four sub-sequences as descriptions are formed with pixels $x(2i, 2j)$, $x(2i + 1, 2j)$, $x(2i, 2j + 1)$, and $x(2i + 1, 2j + 1)$, respectively. In this way, four sub-sequences with halved resolution on both spatial directions, and a quarter of the original size, correspond to each input sequence [4]. Figure 8 shows an example of example of coding a 4×4 luma samples into four 2×2 descriptions by SMDC. Since the video source was acquired in planar YUV format of with chroma subsampling 4:2:2, each U and V component whose amount is only half of the amount of Y component needed to be evenly distributed to descriptions by SMDC as well. Figure 9 shows an example of SMDC for evenly coding a 4×4 frame in YUV format with 4:2:2 chroma subsampling into 4 descriptions. As shown in the figure, each description is encoded into 2×2 frame with luminance samples of four Y, chrominance samples of two U and two V.

Fig. 9 An example of SMDC for coding a 4×4 frame in YUV format with 4:2:2 chroma subsampling into 4 descriptions



5 Server module

In TuBeck, the server module provides functionalities of stream management, network delivery, P2P topology, and resource management. Figure 10 shows the function blocks of the server module which are *Stream Control Management (SCM)*, *Resource Management*, *P2P Topology*, and *Stream Transmitter*. In SCM the service acceptor is responsible for accepting a new connection and spawning a peer handler (PH) to provide consequent services, such as available source peers, based on LM protocol. Those PHs are managed under the peer handler collection (PHC). In case a peer is left expectedly or unexpectedly, its corresponding PH is removed from PHC. The recovery processor is responsible for processing intra-d and inter-d recoveries with the aid of P2P topology for reconstructing peer connectivity. In P2P topology, loops in the system are managed as a loop list by the loop provider. Each loop is represented as an allotment list managed by allotment provider. An allotment is the abstract entity of a peer which contains the information of peer id, requested description id and video id, and the first and the last received chunk ids in buffer.

In resource management video files are managed and indexed hierarchically. As mentioned above, a video is divided into multiple descriptions, and each description is then further divided into a series of chunks. Available videos, descriptions, and chunks in the system are indexed in video table, description table, and chunk table, respectively. The resource I/O accessor is responsible for retrieving physical chunks from video repository. The stream transmitter is responsible for accepting streaming requests from peers. Once accepted, each requested description would be handled by a stream handler (SH) which is responsible for retrieving chunks from the resource management and then streaming to the requested peer. In addition to managing

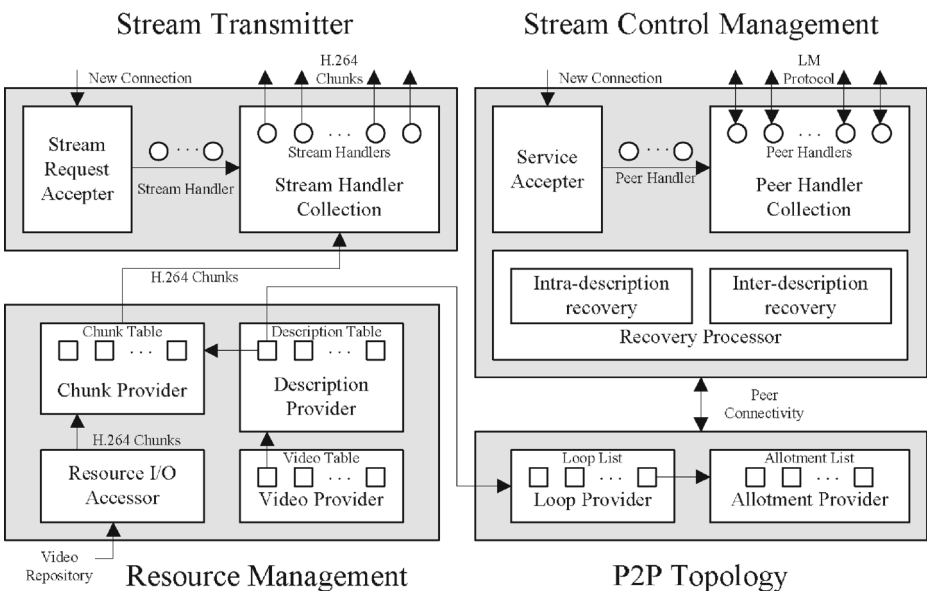
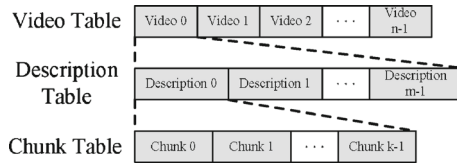


Fig. 10 The function blocks of server module

Fig. 11 The hierarchy of video indexing in TuBeck



SHs, like PHC, the stream handler collection (SHC) is responsible for dealing with expected and unexpected broken streams as well. Figure 11 shows the hierarchy of video indexing in TuBeck.

6 Peer module

The peer module consists of four function blocks which are *Stream Control Management (SCM)*, *Resource Management*, *Stream Receiver*, and *Stream Relay*. Figure 12 shows the function blocks of peer module. The stream control management is responsible for delivering stream control messages and processing failure recovery. Control messages based on *LM protocol* such as peer register, video request, and intra/inter-d recovery are issued by the *service connector* in SCM. Once the source peer or the child peer of a peer is failed, the intra-d or inter-d recovery process in recovery requester is activated to fix the broken stream. The received descriptions and their chunks are managed by description provider and chunk provider in resource management. In TuBeck, the video index is kept in a peer for the purpose of the initial video selections. A complete video description can be acquired through service connector in SCM by issuing a query to the server. Once the source peer of

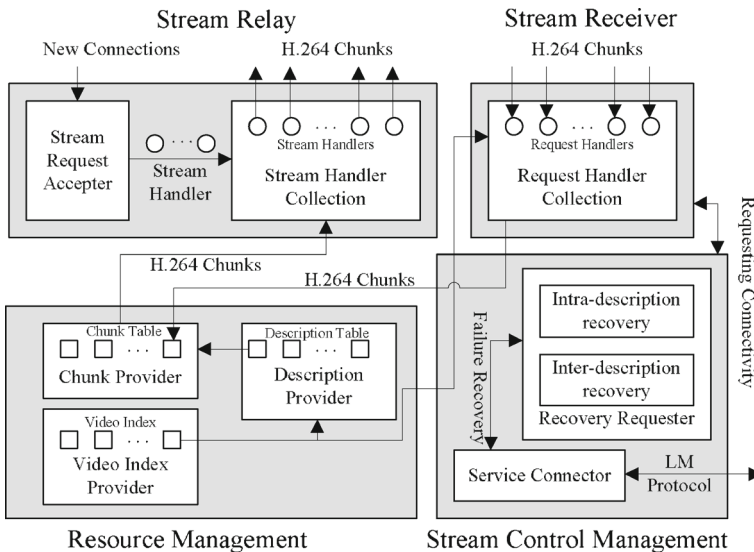


Fig. 12 The function blocks of peer module

a desired description is determined, the service connector in SCM spawns a request handler thread managed by request handler collection in stream receiver to start receiving video chunks. Received chunks are then stored in peer’s buffer and their related information is recorded in the chunk table. The stream relay is responsible for relaying buffer chunks to other peers based on requests. When a relay service for a description is granted in the stream request acceptor, a stream handler thread is spawned to handle the later streaming service. These stream handlers are managed by the stream handler collection.

7 Player module

When received chunks are stored in a peer’s buffer, they can be sent to the player module for rendering. The player module provides functionalities of chunk decoding, merging, rendering, and merging control. Figure 13 shows the function blocks of the player module which are *JM H.264 Decoder*, *SMDC Merger*, *Render*, and *Merging Controller*. In addition, there are two buffer spaces which are *Merged Chunks Buffer* for network transmission buffering and *Decoded Chunks Buffer* for playback buffering. The merged chunks buffer is implemented as a circular queue to efficiently utilize memory to store merged chunks. The decoded chunks buffer is a hash table to provide fast access of merging chunks. The JM H.264 decoder, modified from the reference code of JM 18.0 [14], is responsible for decoding the incoming undecoded chunks from H.264 format to YUV format. Those decoded chunks are

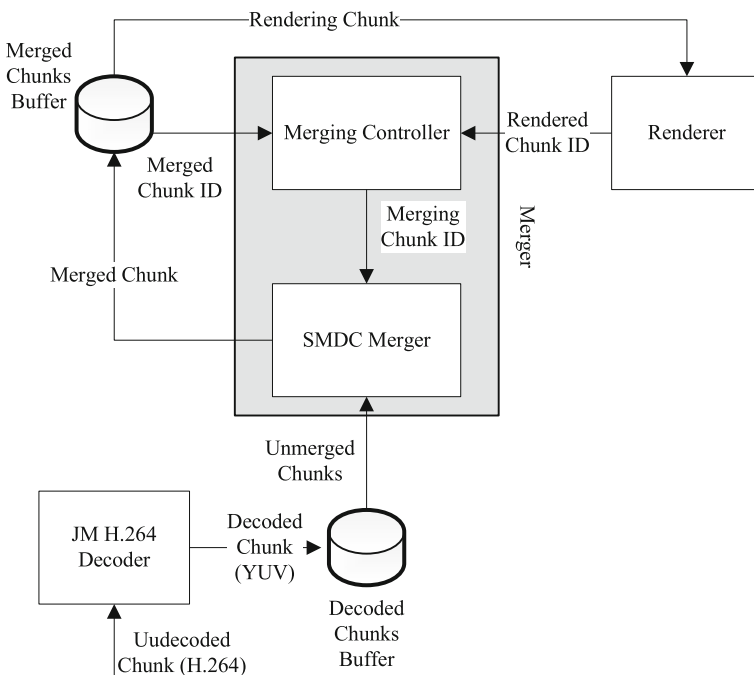


Fig. 13 The function blocks of player module

then saved into the decoded chunks buffer. The SMDC merger is responsible for periodically merging chunks of the same description id and put them into the merged chunks buffer waiting for being retrieved by the render for playback. The render is responsible for regularly getting chunks from the merged chunks buffer to display based on the playback rate. It is obvious that how often the SMDC merger is executed to merge chunks of the same description id depends on the speed of rendering, which may cause the overflow and underflow of the merged chunks buffer if the merging is triggered too fast and too slow. With the rendered chunk id from the renderer and the merged chunk id from the merged chunks buffer, the merging controller can precisely issue the command of merging with a specific chunk id to the SMDC merger to prevent the overflow and underflow of the merged chunks buffer.

8 LM protocol

Figure 14 shows the workflow of streaming service for the TuBeck. When a peer enters the system, it first registers itself to the server with the information of communication ports, available downloading and forwarding bandwidth, and etc. The server then sends a unique peer id and the information of available video sequences back to the peer. A peer selects a desired video and viewing quality in terms of the number of descriptions to watch from the list and sends the video request to the server. The server then decides which descriptions the peer is going to receive based on the OLF description selection policy and sends these description ids back to the peer. Next, the peer sends a request to the server for joining the service loops of these receiving descriptions. After granted, the peer starts the provider identification process. First it asks the server to provide the source peers of the receiving descriptions. The peer then connects to those source peers to start the streaming of requested descriptions. While streaming these descriptions, failure caused by expected or unexpected peer leave would result in breaking service loops. In those cases, we first apply intra-d recovery and then inter-d recovery if necessary to ensure seamless streaming services. It is no doubt that the communication protocol of those operations needs to be clearly pre-defined in order to provide seamless streaming services. The proposed *LM* protocol is served for this purpose. In addition, we define the general packet

Fig. 14 The workflow of streaming service for TuBeck

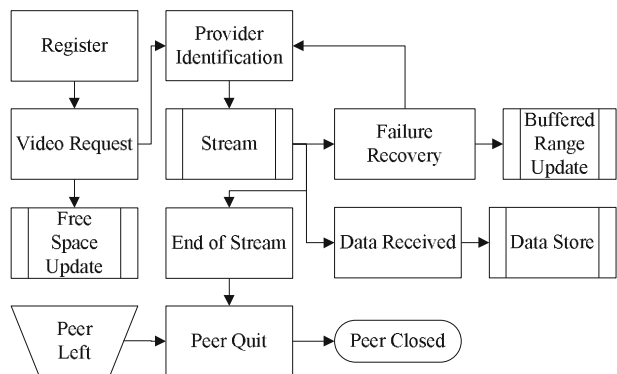




Fig. 15 The general packet format of commands used in LM protocol

format of commands used in LM protocol shown in Fig. 15. These command packets are transmitted in TCP protocol.

8.1 Video request

The purpose of video request as the command “VIDEO” issued by a peer is to get the detailed information of a desired video, such as the segmentation method of MDC, the number of coded descriptions, the number of frames per chunk, the number of chunks for each description, the number of frames in the last chunk, the number of frames per second, display size, and YUV format (4:2:2), shown in Fig. 16. These video information is used to initialize the video player. Next, a peer issues the desired quality of the video in terms of the number of description to the server. Once the server receives the request, it issues the free buffer space update to the last peers of the open loops of the desired video to estimate the expiration times of the open loops. Description selection is based on the OLF policy. The ids of selected descriptions are then sent back to the peer. Figure 17 shows the communication protocol of video request.

8.2 Provider identification

When a peer requests streaming service for a specific description, e.g., peer join or failure recovery, a provider identification as the command “PROVIDER” issued to the server to request the information of source peers, such as IP address and communication ports. Figure 18 shows the payload format of PROVIDER command. If no such a source peer exists, the server or the proxy is returned as the provider. The requesting peer then sends out a message to the source peer to confirm its service availability, such as the available forwarding bandwidth and the existence of requested description. If the provision of the requested is confirmed, the requesting

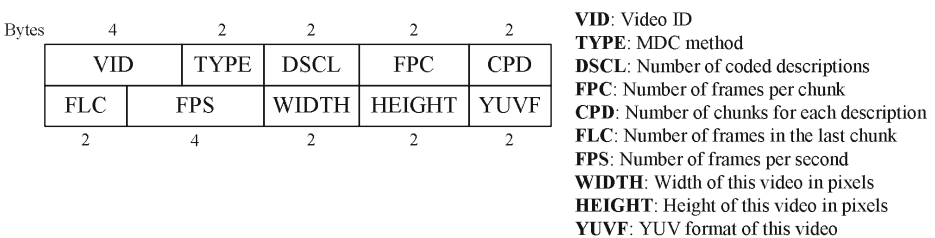
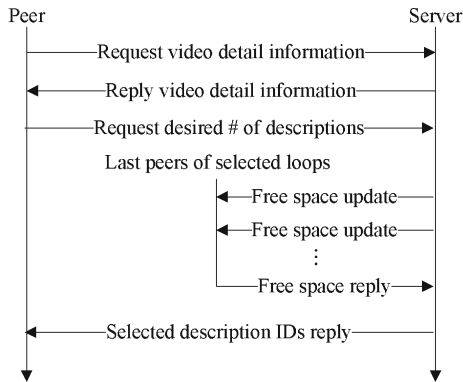


Fig. 16 The payload format of VIDEO command

Fig. 17 The communication protocol of video request



peer next sends a streaming start message along with a chunk range $[s, e]$. The s denotes the starting chunk index of streaming, especially the $s = 0$ representing streaming from the beginning of the video. The e denotes the last chunk index of streaming and the $e = -1$ represents streaming to the end of the video. Such a message of chunk range is necessary to cope with all cases of streaming services, such as new peer join (e.g., $[0, -1]$), intra-d recovery (e.g., $[134, -1]$), and inter-d recovery (e.g., $[146, 150]$). Finally, a commencing message is sent to the requesting peer to notify the start of streaming. Each chunk is transmitted as the command “CHUNK” with the payload format shown in Fig. 19. Note that such a streaming for a specific chunk range may fail if there are no such chunks available. In this case, a streaming failed message would send back to the requesting peer. Figure 20 shows the communication protocol of provider identification.

8.3 Failure recovery

When a peer failed, the server and its child peer would immediately sense its absence by picking up offline signals. The server removes the allotment of the failed peer in the service loop and re-organizes the P2P topology by replacing the parent peer of recovering peer with the parent peer of failed peer. Meanwhile, the recovering peer would issue the provider identification to the server for acquiring a new source peer by intra-d recovery. If failed, the recovering peer then issues inter-d recovery as the command “INTER-RECOVERY” to the server. The payload format of this command is shown in Fig. 21. Next, the server needs to update the buffer range of peers involved in the recovering process by the command “BUFFER-RANGE” whose payload format is shown in Fig. 22, in order to correctly select the new provider (rescue peer) for sewing recovery. Such peers can be classified into three

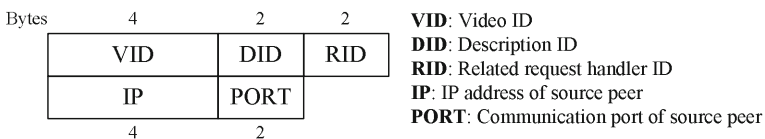


Fig. 18 The payload format of PROVIDER command

Fig. 19 The payload format of CHUNK command

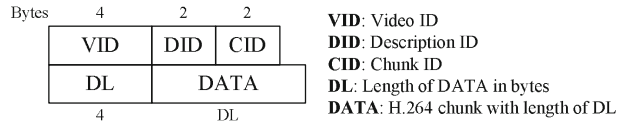
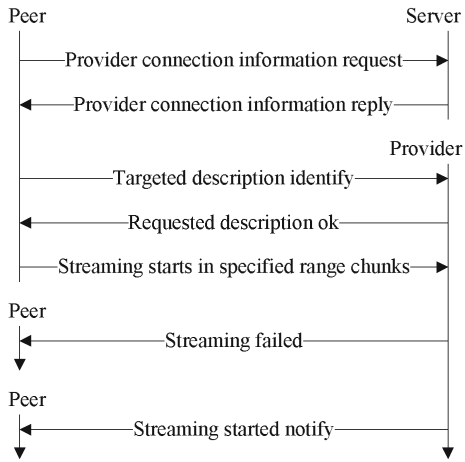
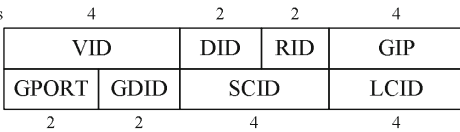


Fig. 20 The communication protocol of provider identification



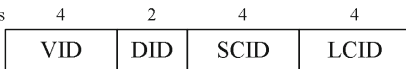
Bytes



VID: Video ID
DID: Description ID
RID: Related request handler ID
GIP: IP address of rescue peer
GPORT: Communication port of rescue peer
GDID: Source description ID of rescue peer
SCID: Starting chunk ID of the source description
LCID: Last chunk ID of the source description

Fig. 21 The payload format of INTER-RECOVERY command

Bytes



VID: Video ID
DID: Description ID
SCID: Starting chunk ID in buffer
LCID: Last chunk ID in buffer

Fig. 22 The payload format of BUFFER-RANGE command

Fig. 23 The payload format of CHANGE command

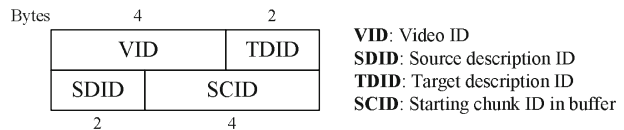
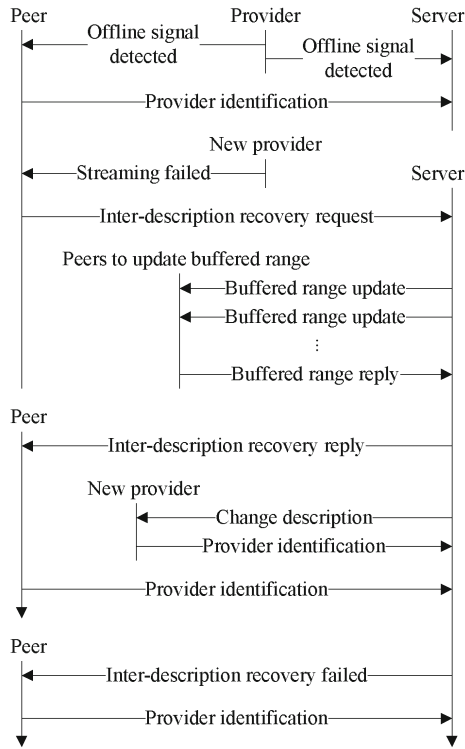


Fig. 24 The communication protocol of failure recovery



types, 1) the parent peer and recovering peer in the recovering descriptions; 2) the grandparent peer, parent peer, and recovering peer in other allocated descriptions; 3) all peers in unallocated descriptions. To minimize the impact on the robustness of the loop of the rescue peer, the head peer or the tail peer of a loop is preferentially selected as a rescue peer. Once the rescue peer is determined, the server notifies the recovering peer of the successful recovery and the rescue peer of changing description as the command “CHANGE” whose payload format is shown in Fig. 23. Next, the recovering peer and the rescue peer both issue the provider identification to the server for stream connection. Note that the child peer of the rescue peer simply executes the intra-d recovery to remedy the broken loop due to the leaving of the rescue peer. Finally, if inter-d recovery again fails, the recovering peer issues the provider identification to the server for the server recovery. Figure 24 shows the communication protocol of failure recovery.

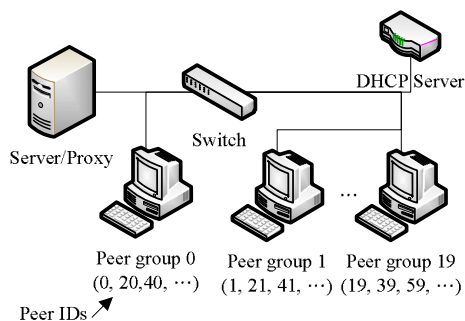
9 Performance study

We deployed experiments to measure the performance of the TuBeck in terms of CPU/memory usage, extensibility of open loops, server/proxy loading, and efficiency of failure recovery on actual network environment. Those emulated experiments would help us to discover the feasibility of the design and implementation of the Loopback-MDC on P2P streaming networks. The 2000-frame video sequence,

Highway (352×288) with 30 fps, was encoded as four descriptions by SMDC and then duplicated into 16 dummy descriptions. Each run last 10 min with loop-playing the video sequence. The arrival rate is 8 peers/min in average with poisson distribution. The number of requested descriptions for a peer is normally distributed with 8 descriptions in average. The failure rate is equal to the number of failed peers divided by the number of total peers. During the experiments, failed peers were equally distributed into the 10-min run. Twenty one computers are used to emulate the experiment, one for the server and twenty for hosting peers. These computers are interconnected in a local-area network by 100BaseT Ethernet switch, and each equips with Intel Core 2 Duo E7500 @ 2.93 Ghz with 2 GB RAM. Emulated peers are distributed into these twenty computers in a round-robin fashion. These peers resided in the same computer form a group. Each peer has 30-second buffer space in order to increase the possibility of forming open loops at low arrival rate, such as 2 peers/min. Since decoding video chunks would consume a great amount of CPU power which might affect the experimental results, in our experiments received chunks are buffered without actually rendering on screen. The default arrival rate, failure rate, and viewing quality is 8, 0.5, and 8, respectively. Note that the experimental results were obtained by the data collected and averaged from running the 10-min emulation ten times. Figure 25 shows the emulation environment for the performance study of the TuBeck.

Figure 26 shows the CPU/Memory usage of the experiments for the TuBeck. The CPU usage on the server is increased along with the increase of the arrival rates shown in Fig. 26a. Even at the arrival rate of 14 peers/min, the CPU usage is only about 20 %. In Figure 26b, the CPU usage is increased from 7.8 % to 13.6 % along with the increase of failure rate from 0.1 to 0.4. When the failure rate increases, the chance of peers in the system failing to recover lost descriptions and then asking the server to process the missing chunks also increases. It is interesting to see that the CPU usage is decreased beyond the failure rate of 0.4. Since the total number of active peers in the system is relatively lesser when the failure rate is 0.5 and beyond, the CPU usage for system processing in the server starts to decrease accordingly. When the average viewing quality increases, the CPU usage for data processing described in Section 5 server module also increases, as shown in Fig. 26c. In our experiments, since the received chunks are buffered without actually rendering on screen, the CPU usages of a peer shown in Fig. 26a, b, and c are all less than 1 %. Figure 26d, e, and f show the memory usage of the server and a peer with respect to various arrival rates, failure rates, and viewing qualities. In general, the server and a

Fig. 25 The emulation environment for the performance study of TuBeck



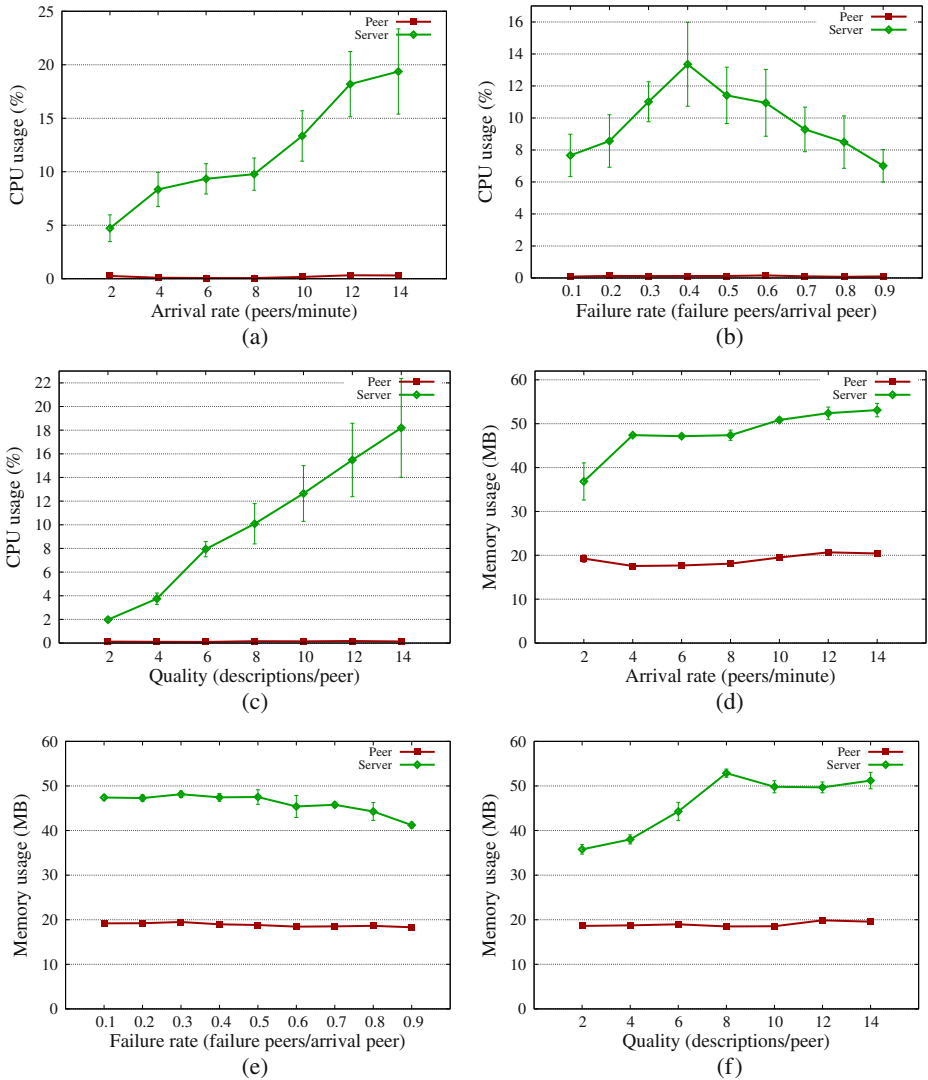


Fig. 26 The CPU/Memory usage of the experiments for TuBeck

peer consumes about 50MB and 20MB memory space in our experiments, which are only 2.5 % and 1 % of 2GB onboard memory.

Figure 27 shows the accumulated duration of open loops with respect to various arrival rates, failure rates, and viewing qualities. As shown in Fig. 27a, the accumulated duration of open loops increases as the arrival rate increases. As expected, the more peers joins the system, the longer an open loop lasts. As shown in Fig. 27b, the accumulated duration of open loops is not decreased significantly along with the increase of failure rate. It is 117 min at 0.1 failure rate and is gradually reduced to 80 min at 0.9 failure rate. As we know that the optimal accumulated duration of open loops in a 10-min run is $16(\text{descriptions}) \times 10(\text{min}) = 160$ min, the TuBeck still

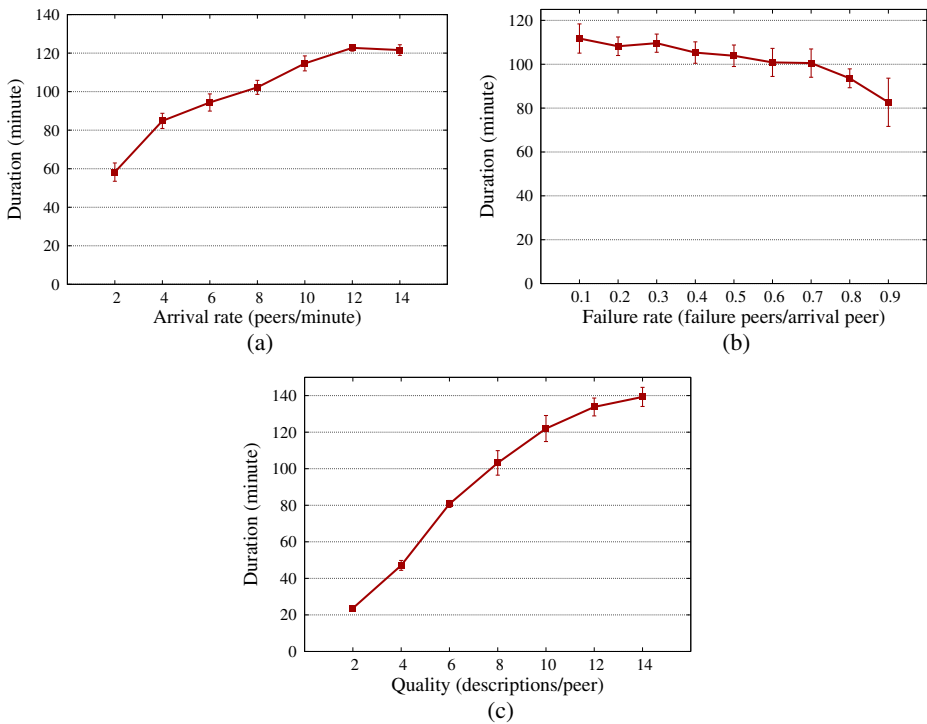


Fig. 27 The accumulated duration of open loops with respect to arrival rate, failure rate, and viewing quality

possesses the average accumulated duration of open loops for 101.8 min, which is 63.6 % of the optimum. In other words, in average an open loop of a description would last for 6.36 min in a 10-min run under various failure rates. This result shows the superior failure recovery of the Loopback-MDC. Figure 27c shows the increase of the accumulated duration of open loops along with the increase of the viewing quality. As more descriptions are cached among peers, more buffer space is contributed to the system to prolong the availability of an open loop.

To observe the efficiency of failure recovery in the Loopback-MDC, we counted how many descriptions recovered by the intra-d/inter-d recovery and by the server at various arrival rates, failure rates, and viewing qualities. Figure 28a shows that both the numbers of descriptions recovered by peers and the server increase along with the increase of arrival rate. The number of descriptions recovered by peers is larger than that recovered by the server. The difference between these two numbers is getting larger as the arrival rate increases, and at arrival rate equal to 14 peers/min the number of descriptions recovered by peers is 177 % more than that recovered by the server. In this case, most broken descriptions in recovering peers were repaired by collaborated peers without much intervention of the server. Therefore, the large number of peers in the system favors the efficiency of failure recovery. Similarly, a higher requested viewing quality results in more descriptions buffered in the system, which also favors the efficiency of failure recovery, as shown in Fig. 28c. With the average 14-description viewing quality per peer, the number of descriptions

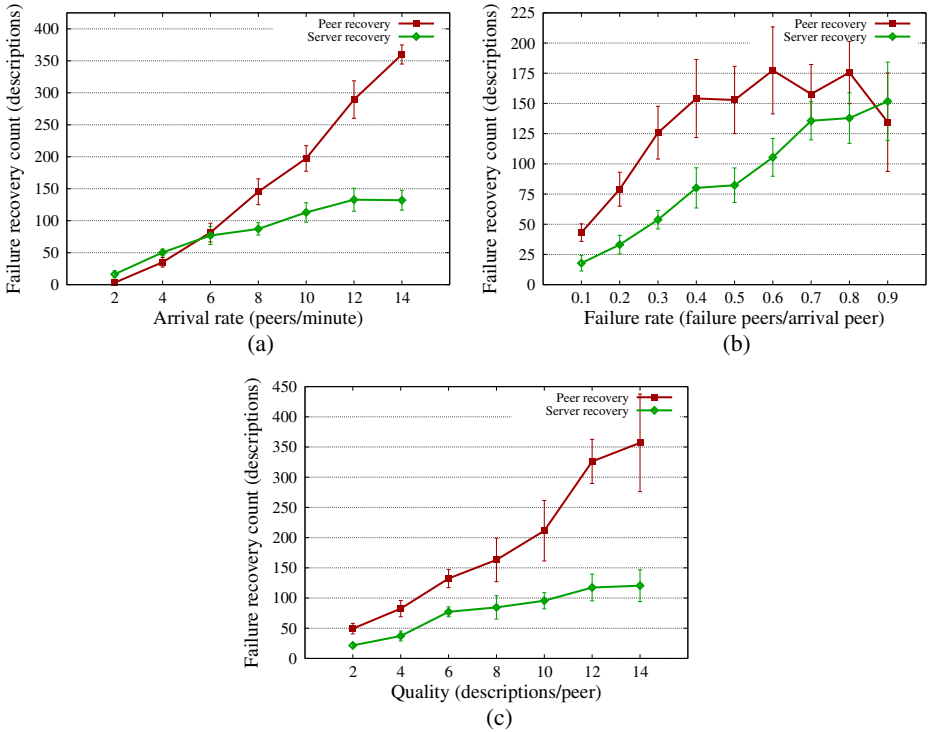


Fig. 28 The accumulated number of recovered descriptions with respect to arrival rate, failure rate, and viewing quality

recovered by peers is 180 % more than that recovered by the server. Figure 28b shows the numbers of descriptions recovered by peers and the server with respect to failure rate. As expected, peers recovered more lost descriptions than the server at most arrival rates. It is noted that the number of descriptions recovered by peers is 87.5 % more than that recovered by the server at failure rate of 0.5, which is the case of the largest difference. When the failure rate is beyond 0.7, peers and the server almost equally share the recovery loading since there are fewer peers left for sharing descriptions.

Figure 29 shows the average durations (delay) of intra-d, inter-d, and server recovery for recovering one lost description with respect to various failure rates. Since applying intra-d recovery only involves in asking the server for a new source peer of the lost description and establishing a new streaming connection, the duration of recovering one lost description is not varied much along with the increase of failure rate as shown in Fig. 29a. The delay for recovering one lost descriptions roughly ranges from 0.23 to 0.27 seconds. Figure 29b shows that the duration of applying inter-d recovery decreases with the increase of failure rate. As stated in Section 8.3, once the server receives a request of inter-d recovery, it needs to update the buffer range of peers involved in the recovering process in order to correctly select a rescue peer for sewing recovery. The complexity of such a process is proportional to the number of peers in the system. Therefore the duration of

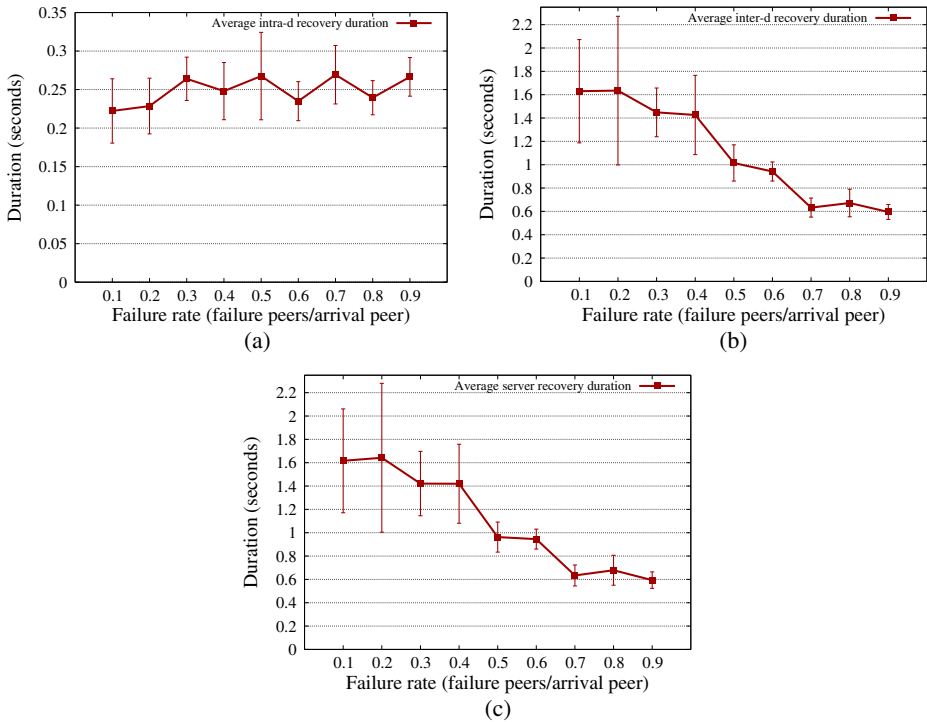


Fig. 29 Average duration for intra-d, inter-d, and server recovery for one lost description

inter-d recovery is inverse proportional to failure rate, which ranges roughly from 1.6 to 0.6 seconds. Finally, if intra-d and inter-d recovery all fail to recover the lost description, the server will directly establish a connection with the recovering peer to provide the missing video chunks. Only a slight computation and communication overhead involves in executing server recovery. As shown in Figure 29c, the time to recover a lost description by the server is almost identical to the time spent in inter-d recovery. Based on our experiment results, the average percentages of recovering lost descriptions by intra-d, inter-d, and server recoveries are 85.47 %, 0.69 %, and 13.84 %, respectively. With the average buffer amount of 300 frames (=10 seconds) for a peer, in our experiments the average delay to recover a lost description is 0.38 second. Compared to the length of a peer’s buffer space, this short recovering delay is not going to affect the quality of services.

Let s and p denote the number of descriptions served by the server and peers, then the server loading can be computed by $s/(s + p) \times 100\%$. Figure 30 shows the server loading with/without intra/inter-d recovery for the Loopback-MDC. As shown in Fig. 30a, the server loading decreases along with the increase of the arrival rate for cases of with and without intra/inter-d recovery. The server loading with intra/inter-d recovery is always lower than that without intra/inter-d recovery. The difference of these two server loadings becomes larger when the arrival rate gets higher. At the arrival rate equal to 14 peers/min, it is interesting to see that the server loading

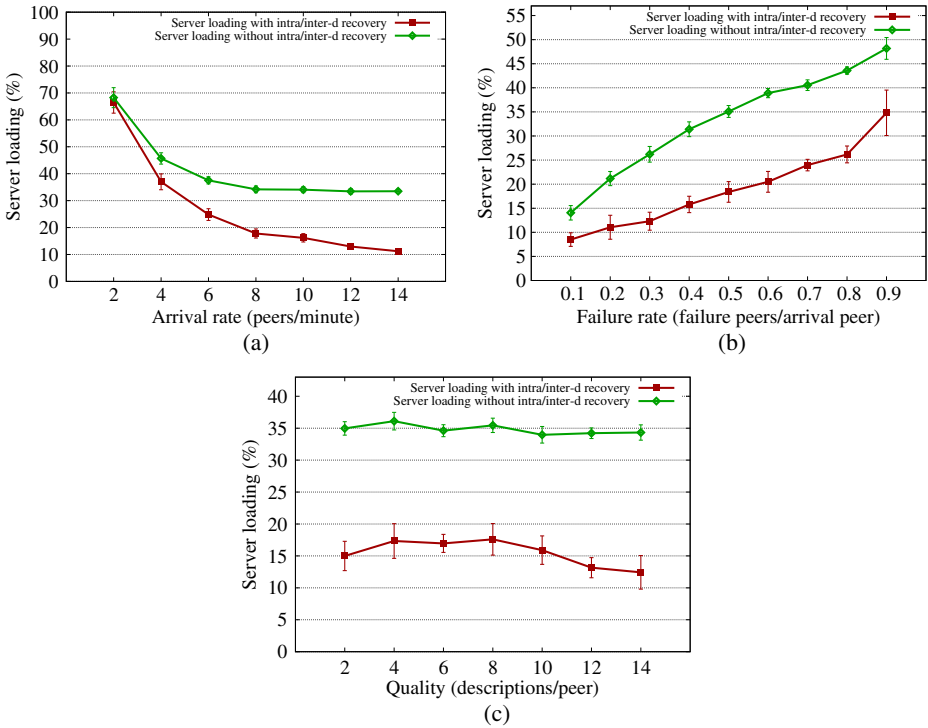


Fig. 30 The server loading with/without intra-d and inter-d recovery for the Loopback-MDC

without intra/inter-d recovery is 22 % more than that with intra/inter-d recovery in which the server loading is as low as 11 %.

As the failure rate increases, the loadings of both schemes increases accordingly as shown in Fig. 30b. The Loopback-MDC with intra/inter-d recovery would have lower server loading than that without intra/inter-d recovery at various failure rates. The largest difference of these two schemes, 18.8 %, happened at failure rate equal to 0.6 peers/min, getting smaller when failure rate is beyond 0.7 peers/min. Figure 30c shows the server loading of the Loopback-MDC with intra/inter-d recovery decreases along with the increase of viewing quality. The one without intra/inter-d recovery appears to consistently react to the increase of viewing quality, where the server loading is about 35 %. In general, the server loading without intra/inter-d recovery has average 18.2 % more than that of with the intra/inter-d recovery with various viewing qualities, and the largest difference of server loadings for these two schemes reaches 22.3 % at the viewing quality equal to 14 descriptions/peer.

In Figs. 27a, 28a, and 30a we show the duration of open loops, failure recovery count, and server loading with respect to various arrival rates ranged from 2 to 14 peers/min. The maximum number of peers in our experiments is 140. To enlarge the population of peers in the experiments during the 10-min run, we can simply increase the arrival rate. However, when the arrival rate is greater than 12 peers/min, the duration of open loops shown in Fig. 27a and the server loading shown in Fig. 30a are not significantly varied. Furthermore, the failure recovery count shown

in Fig. 28a possesses the tendency of rapid increase when the arrival rate is greater than 8 peers/min. Since we can clearly predict the characteristic of the TuBeck based on those figures, we believe that collecting data with arrival rate greater than 14 peers/min for larger peer population is not necessary in our experiments.

10 Conclusion

In this paper, we introduce the design and implementation of a P2P streaming system with the Loopback-MDC for H.264/AVC videos, named *TuBeck*. The system architecture, preprocess of media sources, server module, peer module, and player module in details. We also elaborate the LM protocol which is used to facilitate the message exchange and streaming control among nodes. In addition, we deployed the experiments to observe the characteristics of the TuBeck on real network infrastructure. The experimental results show that the Loopback-MDC is practical for realizing a P2P streaming system in terms of the server loading, CPU/Memory usage, and efficiency of failure recovery with respect to various arrival rates, failure rates, and viewing qualities.

Acknowledgement This work was partially supported by National Science Council under contracts NSC 97-2221-E-024-014-MY3.

References

1. Akyol E, Tekalp A, Civanlar M (2006) Adaptive peer-to-peer video streaming with optimized flexible multiple description coding, pp 725–728
2. Akyol E, Tekalp AM, Civanlar MR (2007) A flexible multiple description coding framework for adaptive peer-to-peer video streaming. *IEEE J Sel Topics Signal Process* 1:231–245
3. Bai HH, Wang AH, Zhao Y, Pan JS, Abraham A (2011) Distributed multiple description coding: principles, algorithms and systems. Springer-Verlag New York Inc
4. Campana O, Cattani A, Giusti AD, Milani S, Zandona N, Calvagno G (2006) Multiple description coding schemes for the h.264/avc coder. In: Proceedings of the international conference on wireless recognizable terminals and protocols, pp 217–221
5. Castro M, Druschel P, Kermarrec AM, Nandi A, Rowstron A, Singh A (2003) Splitstream: high-bandwidth multicast in cooperative environments. In: Proc. of ACM symposium on operating system principles, pp 298–313
6. Chen Z, Yin H, Lin C, Liu X, Chen Y (2007) Towards a trustworthy and controllable peer-server-peer media streaming: an analytical study and an industrial perspective. In: Proc. of global telecommunications conference (Globecom '07). IEEE, pp 2086–2090
7. Dai L, Cui Y, Xue Y (2007) Maximizing throughput in layered peer-to-peer streaming. In: IEEE international conference on communications, 2007 (ICC '07), pp 1734–1739
8. Dong Y, Kusmierek E, Duan Z, Du D (2004) A hybrid client-assistant streaming architecture: modeling and analysis. In: Proc. of 8th internet and multimedia systems and applications (IMSA)
9. Fouliras P, Xanthos S, Tsantalis N, Manitsaris A (2004) Lemp: lightweight efficient multicast protocol for video on demand. In: SAC '04: proceedings of the 2004 ACM symposium on applied computing (2004), pp 1226–1231. doi:10.1145/967900.968150
10. Goyal VK (2001) Multiple description coding: compression meets the network. *IEEE Signal Process Mag* 18:74–93
11. Guo Y, Suh K, Kurose J, Towsley D (2003) P2cast: peer-to-peer patching scheme for vod service. In: Proceedings of the twelfth international world wide web conference
12. Hua KA, Cai Y, Sheu S (1998) Patching: a multicast technique for true video-on-demand services. In: Proceedings of the sixth ACM international conference on multimedia, pp 191–200
13. Hunag C, Wang A, Li J, Ross KW (2008) Understanding hybrid cdn-p2p: why limitlight needs its own red swoosh. In: Proc. of Nossdav'08. ACM, pp 75–80

14. Joint Video Team (JVT) of ISO/IEC MPEG & ITU-T VCEG. Joint model (jm) - h.264/avc reference software. <http://iphome.hhi.de/suehring/tml/>. Ver. 18.0. Accessed 8 June 2012
15. Kusmirek E, Dong Y, Du DHC (2006) Loopback: exploiting collaborative caches for large-scale streaming. *IEEE Trans Multimedia* 8(2):233–242
16. Lin CS (2011) Enhancing p2p live streaming performance by balancing description distribution and available forwarding bandwidth. *Int J Commun Syst* 24(5):568–585
17. Lin CS, Lee IT (2010) Applying multiple description coding to enhance the streaming scalability on cdn-p2p network. *Int J Commun Syst* 23(5):553–568
18. Lin CS, Syu WT, Lee IT (2008) Improving the scalability of p2p streaming based on fine-grained balancing scheme. In: Proc. of the IEEE 22nd international conference on advanced information networking and applications (AINA2008), pp 795–802
19. Ma W, Du DHC (2002) Reducing bandwidth requirement for delivering video over wide area networks with proxy server. *IEEE Trans Multimedia* 4(4):539–550
20. Ma WH, Du DHC (2004) Design a progressive video caching policy for video proxy servers. *IEEE Trans Multimedia* 6(4):599–610
21. Padmanabhan V, Wang H, Chou P (2003) Resilient peer-to-peer streaming. In: Proc. of IEEE network protocols, pp 16–27
22. Padmanabhan VN, Wang HJ, Chou PA, Sripanidkulchai K (2002) Distributing streaming media content using cooperative networking. In: Proc. of ACM NOSSDAV, pp 177–186
23. Sheu S, Hua KA, Tavanapong W (1997) Chaining: a generalized batching technique for video-on-demand. In: Proceedings of IEEE international conference on multimedia computing and systems '97, pp 110–117
24. Tran DA, Hua KA, Do T (2003) Zigzag: an efficient peer-to-peer scheme for media streaming. In: Twenty-second annual joint conference of the IEEE computer and communications
25. Xiao X, Shi Y, Gao Y (2008) On optimal scheduling for layered video streaming in heterogeneous peer-to-peer networks. In: MM '08: proceeding of the 16th ACM international conference on multimedia. ACM, New York, NY, USA, pp 785–788
26. Xu D, Kulkarni SS, Rosenberg C, Chai HK (2004) A cdn-p2p hybrid architecture for cost-effective streaming media distribution. *Comput Netw* 44(3):353–382



Chow-Sing Lin received the Ph.D. degree in Computer Engineering from the University of Central Florida, Florida, USA, in 2000. He is currently a Professor in the Department of Computer Science and Information Engineering, National University of Tainan, Taiwan. His research interests include wired/wireless media delivery, peer-to-peer media streaming, distributed multimedia systems, mobile computing, and sensor networks.



Rong-Hua Chang received his M.S. and B.S.Eng. degree in Computer Science and Information Engineering from National University of Tainan, Taiwan, in 2012 and 2010. His research interests include peer-to-peer media streaming systems, multiple description coding techniques, and mobile computing.



Jhe-Wei Lin received his B.S.Eng. degree in Computer Science and Information Engineering from National University of Tainan, Taiwan in 2011. He is currently a M.S student in the department of Computer Science and Information Engineering at National University of Tainan, Taiwan. His research focuses on bitTorrent with real-time video service, peer-to-peer media streaming, and multiple description coding on P2P networks.