On the Frame Forwarding in Peer-to-Peer Multimedia Streaming

Chun-Chao Yeh
National Taiwan Ocean University
Department of Computer Science
2, Pei-Ning Road, Keelung 202, Taiwan
ccyeh@mail.ntou.edu.tw

Lin Siong Pui
National Taiwan Ocean University
Department of Computer Science
2, Pei-Ning Road, Keelung 202, Taiwan
m92570011@mail.ntou.edu.tw

ABSTRACT
One of the unique features of P2P streaming is that each peer plays both client and server roles at the same time. Distribution of program streams is achieved by peer forwarding instead of providing from a set of centralized servers. Consequently, loss of a frame not only causes the peer to miss the frame, but also the descendant peers feel the effect as well. In this paper, we investigate the design issues on frame forwarding in peer-to-peer multimedia streaming. To provide smooth streaming services, we argue that the selection of a new parent peer for a child peer should consider not only network quality (e.g. delay and bandwidth), but also the frame-buffer status between the parent-child peers. When there is a large mismatch on the frame-buffer status between parent-child peers, the child peer suffers interruption during playback. We discuss the effects caused by both frame caching and peer selection. With regard to frame caching, we discuss issues of frame buffering, frame forwarding rate, frame synchronization, and old frame management schemes. With peer selection, we discuss issues of selection criteria and their impact. Two selection criteria, ALF and AUF, are discussed regarding critical conditions encountered. Based on the above different design considerations, we propose ten forwarding mechanisms for discussion and comparison. Through simulation, we show how the proposed forwarding mechanisms based on frame synchronization and aggressive old frame management schemes can effectively reduce frame loss.

Categories and Subject Descriptors
C.2 [Computer-Communication Networks]: Distributed Systems; C.4 [Performance of Systems]: Design studies

General Terms
Algorithm, Performance, Design

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
P2PMMS’05, November 11, 2005, Singapore.
Copyright 2005 ACM 1-59593-248-6/05/0011 ...$5.00.

Keywords
Frame Forwarding, P2P multimedia streaming, Frame-buffer mismatch

1. INTRODUCTION
Peer-to-peer (P2P) technologies have drawn much attention from researchers and Internet users in recent years. One popular peer-to-peer application is P2P file sharing (swapping), especially for music file exchanging. Popular P2P file sharing systems/protocols include Napster, KaZaa, FastTrack, eDonkey, and eMute, to name a few. The great success of P2P file sharing services encourages researchers to investigate the feasibility of providing multimedia streaming services over P2P networks. In contrast to the conventional centralized client-server service model, P2P streaming services need not rely on a powerful media server to provide streaming services to all clients. Instead, distribution of requested streams is achieved by cooperation among all participant peers. Each of the participants is responsible to forward the streams to some of others. Different from P2P file sharing, P2P streaming is intrinsically imposed with more stringent timing constraints. While it is possible to download entire media files (with P2P file sharing approaches) before playback, it demands enough storage space to store the whole media file. In addition, extra waiting time for file download is required before playback. Moreover, for live-broadcasting programs, streaming is a more natural mechanism to view the program in real time.

In P2P streaming, all participant peers form one or multiple multicasting trees to forward the multicasting stream from source to all participant peers. Subject to the dynamics of network traffic conditions and peer behaviors (join and leave (or failure) randomly), providing smooth streaming services to each participant peer becomes a challenge, especially when peer population becomes large. Various design strategies have been proposed to construct and maintain effective multicast trees. Also, some research efforts reduced the perceived impacts of frame loss during frame forwarding. Despite recent research results on P2P streaming systems, some design issues have not been well addressed, especially when real deployment is considered. In this paper we investigate frame forwarding issues to provide better streaming quality for playback. We point out why frame forwarding is a critical issue to provide quality P2P streaming services in real deployment.

In our previous work [23], we have discussed the problems caused by frame-buffer mismatch. To provide smooth
streaming services. To reduce perceived impact, we argue that selection of a new parent peer should consider not only network quality (e.g. delay and bandwidth), but also frame buffer status between parent and child peers. In this paper, we look at details in performance impact, based on intensive simulation. We consider different frame forwarding strategies. Different design considerations such as frame buffering, management of old-frames, forward forwarding rates, frame synchronization mechanisms, and peer selection strategies are discussed. Intensive performance evaluations were made for these different design strategies. We show from our simulation results that the proposed frame-forwarding scheme reduces frame loss effectively.

The remaining paper is organized as follows. A short survey on P2P streaming systems is presented in Section 2. The problems discussed are described in Section 3. System models and assumptions are depicted in Section 4. Detailed discussions of design strategies and the proposed approaches are presented in Section 5. Section 6 reports the experiment results. Conclusions are made in Section 7.

2. SURVEY: P2P STREAMING SYSTEMS

Various P2P streaming systems have been proposed. In this section, we give a short survey below.

2.1 SpreadIt

SpreadIt architecture was proposed by H. Deshpande, M. Bawa and H. Garcia-Molina of Stanford Peers Group [13, 8]. It constructs an application level multicast tree over a set of peer nodes for streaming live media. A simple topology maintenance scheme is employed to maintain the multicasting tree, based on request redirection. SpreadIt protocol runs in fully distributed fashion. Each peer node in the network maintains limited information about others, typically including the root node, its parent node, and its immediate children nodes. When a new peer wants to join the multicasting tree, it sends a request to any peer node, said X, already in the network. If X is unsaturated, then X accepts the request and setups a channel to forward the media stream to the new join peer. However, if X is saturated, X forwards the request to other peers, either its parent peer or its immediate child peers.

2.2 Narada

Narada protocol was designed for end system multicast by Yang-hua Chu, Sanjay G. Rao, Srinivasan Seshan, and Hui Zhang of Carnegie Mellon University [11, 10]. Narada was designed for a wide range of group communication applications such as multimedia streaming and network games. Each peer node in Narada maintains a partial list of other peers in the multicast group. To be able to adapt to network dynamics and to be resilient to inaccurate measurement of network parameters, intensive message exchanges between peers should be made. Narada constructs multicast trees in a two-step process. First it constructs a richer connected graph, called mesh. The mesh is constructed by requiring that each peer periodically generates a refresh message to its neighbors. And, a distance vector routing algorithm is run on top of the mesh for exchanging update of routing messages between neighbors. In the second step, Narada constructs spanning trees of the mesh to generate each multicast tree rooted at corresponding source nodes, using well known routing algorithms. Due to the needs of intensive interactions between peer nodes, Narada is targeted towards medium sized groups, not suitable for large scale peer groups.

2.3 Nice

Nice protocol was designed for application-layer multicast by Suman Banerjee, Bobby Bhattacharjee, and Christopher Kommarreddy of University of Maryland [6, 7]. Different from SpreadIt and Narada, Nice multicast tree is constructed in a more compact way. Hosts in Nice are partitioned into a set of clusters. Each cluster is of size between k to 3k-1, where k is a constant number and is given as a system parameter. Each cluster has a cluster leader which is the one has minimal maximum-distance to all other hosts in the cluster. These clusters form layer-0 in Nice-tree hierarchy. Leaders of each cluster in layer-0 form the nodes for layer-1 in the same Nice multicasting-tree, and they are constructed as a set of clusters with the same strategy as layer-0 (maintaining the size between k to 3k-1). Higher layers of the Nice-tree hierarchy, if available, are constructed recursively. It is not hard to find out that the depth of a Nice tree is bounded by O(log N), where N is the total number of nodes joining the multicast tree. Consequently, Nice protocol reduces worst-case of state and control overhead from any member to O(log N). Since join and leave of any host would change group size, dynamic cluster maintenance is needed.

2.4 Zigzag

Zigzag protocol was designed by Duc A. Tran, Kien A. Hua, and Tai Do of University of Central Florida [19, 20, 18]. Zigzag constructs a multicast tree with multi-layer hierarchical clustering concepts, similar to the approach adopted by Nice. The size of Zigzag clusters is kept between k to 3k-1, same as that in Nice. However, unlike Nice, Zigzag separates administrative organizations with data forwarding organizations. Nice always uses the head of a cluster to forward both the data content and the control message to the other members in the cluster, whereas Zigzag uses head node to maintain control message exchanges only and additional node, called foreign head, to forward data content. With such a different control and data forwarding strategies, Zigzag confines node degree to O(k^2)(the value is further improved to 6k-3 with a slightly different forwarding police [20]) and the tree high to O(log N). By contrast, the node degree in Nice could increase as the group size increase.

2.5 P2Cast

P2Cast protocol was designed by Yang Guo, Kyoungwon Suh, Jim Kurose, and Don Towsley of University of Massachusetts at Amherst [14]. P2Cast uses P2P approach and patching technique [15] to enable VoD services to a set of peer nodes. Especially, P2Cast addresses two key technical issues: (1) constructing an application overlay for streaming; and (2) providing continuous stream playback (VoD services) concerning the dynamic of peer behavior. Different from real-time broadcasting stream, VoD services should let later joining viewer to view the program stream from beginning instead of from the segment broadcasted currently. Consequently, P2Cast partitions the peers into a set of clusters, called sessions, according to their arrival time. A constant value T, standing for the threshold of time skew of a session, is associated with P2Cast. The peers arriving within the threshold constitute a session. A new session will be cre-
ated for a peer arriving late behind the threshold of the previous session. The session head, the one arrives the session first, receives stream from server node directly; new peers joining the session later receive stream from peers already in the session. For each session, each peer except the head could have two kinds of stream received from other peers: base stream and patch stream. The base stream is the program stream forwarded from server node to the session head. A new joining peer except session heads will miss a portion of initial part of the program forwarded by the session head. As a result, additional stream, called patch stream, carrying the missed initial part of the program stream is needed for the new joining peer. To fulfill such a patching approach, peers in P2Cast should reserve a buffer to store initial portion of program stream and an addition buffer to store the pre-cached data of base stream.

2.6 CoopNet

CoopNet protocol was designed by Venkata N. Padmanabhan, Helen J. Wang, Philip A. Chou of Microsoft Research and Kunwadee Sripanidkulchai of Carnegie Mellon University [16]. Different from others, CoopNet uses MDC (multiple description coding) and P2P approach to deliver live streaming or on-demand streaming to clients. CoopNet constructs multiple distribution trees for peers, each for one description coding stream. The approach, applying multiple description coding streams over multiple distribution trees, strengthens fault-resilient capability. Whenever a peer fails, the descendents of the fault peer will be blocked only from receiving the coding streams forwarded by the fault peer. They still can receive all other coding streams normally. Consequently, descendents of a fault peer will experience a graceful degradation of stream quality, but not be totally blocked off. Management of the distribution trees is based on a centralized scheme imposed on a set of sever nodes (or the source node). One of criticism of the CoopNet approach is the centralized control strategy and the control overhead required for management of multiple distribution trees.

2.7 Others

Besides CoopNet, researchers investigated potential benefits to partition a media stream into multiple sub-streams with or without redundant visual information embedded in the sub-streams. PALS [17] partitions a program stream into multiple layered streams, which enables PALS to adapt heterogeneity of client bandwidth and to be resilient to transient (or permanent) link failure. Similar strategy based on layered streaming was adopted by [12] as well. Xu, et al. [21] proposed an optimal scheduling algorithm to partition a real-time stream into multiple sub-streams forwarded by a set of peers. Advantages of such a bandwidth aggregation strategy is to well utilize heterogeneity of peer bandwidth. The authors assumed that each streaming session may involve multiple supplying peers, where each of the supplying peers is responsible to forward a portion of the streaming data. The work is based on well-defined mathematic models.

On the issues related to efficient overlay network construction, the authors of Borg [24] used a hybrid approach to build a P2P overlay networks. Borg exploits the asymmetry in P2P routing and leverages the reverse-path multicast scheme to reduce link stress on the physical network. Borg has been implemented on top of Pastry. RITA [22] is based on a combination of landmark clustering and RTT measurements to construct an overlay network for P2P multicasting. Tree reconfiguration in RITA is initiated just-in-time by the clients at receivers upon the media quality falling below a specific threshold. As a result, dynamic tree reconfiguration with low switching delay can be achieved. BroadcastFederation [9] adopts an approach to combine different broadcast-capable networks to provide application-layer broadcast services. The authors of BroadcastFederation believed that none of the existing multicast or broadcast protocols will become the sole dominant Internet broadcasting technology in the near future. Under such network architecture, the broadcast gateway becomes the point of peering between a broadcast network supported by a single broadcast or multicast protocol natively. A broadcast tree is built to span all the broadcast gateways. The tree is constructed by a P2P approach. Andreev, et al. [5] proposed a polynomial time approximation algorithm on designing a multicast overlay network for streaming. The authors constructed the overlay network with a 3-level hierarchy. In the network the top level is for sources nodes; the middle level is for a set of reflectors to forward the stream from source nodes; the bottom level is for a set of sinks node which receive streams from reflectors and forward the streams to clients. Given a set of system parameter (e.g., bandwidth constraints and a set of source nodes, reflector nodes and sink nodes) and cost functions, the authors modeled the resource allocation problem as an integer programming problem (IP), and proposed an approximation algorithm to solve the problem.

Some P2P streaming systems have been deployed for real applications. For example, research group on Narada has deployed their p2p streaming system to distribute selected live programs in SIGCOMM Conference to researchers unable to attend the conference. We also found some companies start to deliver P2P streaming services to internet users. For example, CoolStreaming [3] provides selected TV programs (most of them are hot Chinese TV programs), which are distributed by P2P streaming.

3. PROBLEM DESCRIPTIONS

In this section, we present the problem. In our previous work [23], we have pointed out the frame-buffer mismatching problem. Here, we give a short description on the problem to provide necessary background information. Besides, we present our observation results on the phase-skew in conventional streaming services provided by centralized servers, which provides a rough estimation on one-hop skew.

3.1 Frame-Buffer Mismatching

Maintaining the frame buffer at the application level is a common approach to smooth out playback interruptions due to transient network faults. Limited memory space is allocated to media clients to cache preloaded frames from upstream. Figure 1 shows a typical buffering mechanism for a pure media client (that is, no need for the client to forward the received frames to other peers). The most recently received frame from upstream is appended to the tail of the frame-buffer, while the least recently received frame is retrieved from the head of the buffer for playback. During playback time, the frame segment valid in the frame-buffer, referred to as frame-window from this point on, is updated to accommodate those new frames received for playback. The frame-window proceeds while the media client continues to
3.2 Phase-Skew

It is quite natural for viewers to think that the frame presented is the same (or nearly the same) frame that others see at the same time when they are watching a broadcasting program. It is more likely true for conventional TV broadcasting systems. However, people might perceive some degree of progress lags between two program streams when they view the program over the Internet. To evaluate this effect, we refer to peer phase from this point on, as the time lag between the playback time of a frame on the source node and the playback time of the same frame on a peer. The lags could be larger and more diverse when the program is broadcasted with P2P streaming systems.

Figure 3 presents the results of our observations on the program progress skews of two websites (BBC News [2] and ABC News [1]). The experiments were made on three successive days (Dec.20-24/2004). Six experiments were made each day with an interval of half to one hour between two successive experiments. We used WMP (windows media player) in each experiment to receive a live broadcast stream (news TV broadcast) provided by the specified website, from a PC located on our campus. Three minutes later, we opened another WMP stream for the same broadcast stream as shown in Figure 1(b). Then, we checked the two WMP streams to measure the progress skew. The frame rates of the two web sites are 310 kbps (for the ABC News streams) and 300 kbps (for the BBC News streams). The results (Figure 3) show that the skew can be as large as five seconds. We also made similar observations on two domestic websites (also providing on-line news services) at the same time, and received similar results (with skew up to six seconds). Many factors contribute to the skew. We do not intend to point out skew quantity from our experiments, but rather skew existence.

The skew presented in Figure 3 is measured directly between the client and the server. That is, it is one hop distance from the viewpoint of overlay network. In P2P streaming, a long forwarding path exists between two peers in the multicasting tree when peer population is large. Each hop could contribute some skew. As a result, accumulative large skew between the peers in the top of the multicasting tree and those in the bottom of the tree, due to the cascading skew effect.

4. SYSTEM MODEL AND ASSUMPTION

One of the unique features of P2P streaming is that each peer plays both client and server roles at the same time.

Figure 1: A typical structure of frame-buffer for a pure media client. (a) frames in /out the frame-buffer; (b) the frame-window proceeds with time.

Figure 2: An example of peer reconnection before and after the peer b fails. (a) frame-window status of the four peers before peer b fails; (b) frame-window status of the nodes after peer b fails.

Figure 3: Program progress skew on two web streams (ABC News and BBC News). Measurements were made on Dec.20-24, 2004, six observations per day. Each point on the plot corresponds to the skew length (in second) of the two streams viewing the same on-line broadcast program at the same time.
Distribution of program streams is achieved by peer forwarding instead of providing from a power server (or a set of servers). Consequently, loss of a frame not only causes the peer to miss the frame, but all descendant peers take the effect.

Different situations result in frame loss, for example buffer overflow and frame-window mismatching during peer reconnection. Also, another common reason is due to errors propagating from ascendant peers. In this paper, our major concern is frame loss during frame forwarding. To keep our discussion focused, we do not take into account the faults due to network transmission error. That is, we assume no package error during transmission between two peers. Also, we assume no packet re-transmission mechanism is imposed on the system. While these two factors (packet loss during network transmission and packet retransmission capability) could be present in real systems, they make the system too complicated to analyze the impacts of individual system factors we considered. Their effect is worthy of future study.

Meanwhile, we assume a multicasting tree is formed and maintained during stream multicasting. The peer providing the program stream source is referred as source peer. Each participant peer received the program stream directly from the source peer or indirectly forwarded from other peers. Connecting information of the source peer (or other boosting peers) is known by all participant peers in order that they can join the multicasting tree. Moreover, we assume the system is homogeneous. Each peer can support an equal number of maximal child peers.

The proposed frame-forwarding scheme is particularly important when peers reserve only limited memory space as frame buffer or when system size becomes large. Due to cascading skew effect, the skew of cached frames between two peers could be enlarged when the size of the multicast tree becomes large. Allocating more memory space for frame buffer in each peer can tolerate more frame-buffer skew, but in reality peer users might be reluctant to do that. We suspect users would feel comfortable if the required frame buffer in P2P streaming is several times the size of that in conventional centralized streaming. Consequently, in this research study we use a medium size of frame buffer (size to accommodate 10 seconds of frames) for the simulation.

Meanwhile, we assume the frame buffer is allocated from memory only. We do not consider using other storage devices such as hard-drives to cache frames. Due to lengthy delay and high overhead, hard-drives are not feasible for short-term frame cache (that is for those frames needed in a short time). On the other hand, from overall system performance viewpoint, hard-drives could be useful for long-term frame cache (that is for those frames either needed after a period of time, or having been playback but being needed later by other peers). However, in reality from user point of view, they might have less incentive to use hard-drives for long-term frame cache (if they have no intention to save a copy of the multicasting stream). Using hard-drives to accommodate temporary frame data imposes large volume of disk access operations intensively on the hard-drive, which is thought harmful to the hard-drive. According to Cool-Streaming [3] (a popular web site providing p2p streaming services in China), they claim the system do not use hard-drive as frame buffer to prevent possible damage to the hard-drive. Similarly, we found PPStream [4] (another web site providing p2p streaming services) has similar claims as well.

5. DESIGN STRATEGY

Here, we discuss some fundamental design strategies on frame forwarding. Two subsystems are closely related to frame forwarding: one is frame caching; the other is peer selection.

5.1 Frame Caching

We consider following four design issues regarding frame caching: frame buffering, forwarding rate, frame synchronization, and old frame management.

5.1.1 Frame buffering

A simple design strategy is with a non-buffer mechanism, in which a peer once receiving a frame from its parent peer forwards the frame to all its child peers without caching. As discussed previously, such a system would be subject to frame loss due to different kinds of system/network errors. To provide a smooth playback, a fixed size of memory space is reserved to accommodate preload frames forwarded by parent peers.

5.1.2 Forwarding rate

Usually, for a constant-rate-encoding stream, a simple approach to feed the streaming data to clients is based on the same rate of client playback. Clients can preload a portion of streaming data before they start to playback to absorb occasional network delay jitter. This strategy works in client-server streaming systems, but it receives more challenges in P2P streaming due to the high dynamics of peer behaviors. In principle, we would like to fully utilize the frame buffer space. Consequently, we propose a strategy of dual forwarding mode: normal (forwarding) mode and fast (forwarding) mode. During the peer join and reconnection, the child peer enters fast mode to have its parent peer forward the frames at a faster rate. Whenever the child peer cannot accommodate more frames to its frame buffer, it returns to normal mode, in which the parent peer forwards the frame at normal speed (same as playback rate).

5.1.3 Frame synchronization

Upon peer join, the parent peer of the new join peer can forward the program stream starting from the least recent frame available in the frame buffer. This strategy is simple and can provide the most available frames to the new peer at that moment. However, in our previous work [23], we showed that this simple strategy leads to a phase-skew problem, and we proposed a frame-synchronization scheme to alleviate this problem. To synchronize the phase of a new join peer with its parent peer, we should guarantee that the time of the first frame playback at the new join peer is equal to (or nearly equal to) the time of the same frame playback at the parent peer. Since the frame buffer is organized as a circular queue, the first frame into the frame-buffer of the new peer is the frame same as the starting frame (the first frame the parent peer sends to the child peer) selected by the parent peer. Consequently, proper choice of the starting frame for a new join peer is the key in reducing phase-skew between the two peers.

Lemma 1 ([23]). In the proposed P2P system model, assume a new join peer i successfully finds a peer j as its parent peer and the parent peer j gets ready to send the starting frame to peer i. The time $t_{phase}$, defined as the elapsed
time between the time the starting frame sends out from peer j and the time the frame would be retrieved from the frame-buffer of peer j, can be calculated by

$$t_{\text{phase}} = d_{i,j} + T_{\text{init}},$$

where $d_{i,j}$ is the network delay between peer i and peer j, and $T_{\text{init}}$ is the elapsed time, representing the time difference between the time the first frame in the frame-buffer is retrieved for playback and the time the first frame enters the frame-buffer of the new join peer.

**Lemma 2** ([23]). With same assumptions as Lemma 1, the initial frame s for the new peer i can be obtained by

$$s = f(t_0 + t_{\text{phase}},)$$

where function $f(x)$ indicates the frame expected to be playback at time $x$ (at the child peer). The term $t_0$ is the time the initial frame is sent (at the parent peer).

Lemma 1 and Lemma 2 provide a formula for a parent peer to calculate the initial frame on its frame-buffer for a new peer to join. If the initial frame is available on the parent peer, then the parent peer proceeds to forward the consequence of streaming data starting from the initial frame. If it is not, the lag for the initial frame to arrive can be estimated easily. It would be reasonable to have the new join peer wait if the lag is not large.

Calculation of the initial frame needs to be conducted during peer-join process only. In the case that a peer switches to a new parent peer (for example, due to leave or failure of its parent peer) after the peer has successfully joined the multicast tree, the new parent peer needs to reevaluate the initial frame for the peer. Information about the first frame the child peer should have is provided by the parent peer along with the peer reconnection requests. The first time is corresponding to the next frame the child peer expects to have, which is corresponding to the next to the last frame the child peer received last time.

### 5.1.4 Old frame management

For a peer, those frames having been playback are referred as old frames in this paper. From the peer point of view, old frames are useless and can be dropped. However, from a system’s point of view, these old frames could be useful for other peers. For example, assume a peer for some reasons is lagging behind the others in the progress of playback (e.g. at some time instance, all other peer are to playback the program segment of about the 30th second, while the peer is to playback the program segment of the 10th second.). If old frames are not kept in the frame buffer of peers, a lag peer will very likely find it hard to find a proper parent peer to smoothly provide the next frames it needs. Consequently, one possible strategy is to keep those most recently old frames once there is free space in the frame buffer. Keeping old frames would provide benefits to other peers but it will consume buffer space for a peer to preload more new frames. As a result, another alternative is to drop all the old frames and reserve more space for new frames. This alternative strategy is good for the peer itself but it might be harmful for those lag peers.

### 5.2 Peer Selection

Peer selection involves in the procedures of peer join and peer reconnection. Since we ignore the network effects, we do not take into account the variance of network delay between different peers. Instead, we consider availability of network bandwidth to support more new child peers, and frame buffer status. On peer join, since new peers contain no frame in their frame buffer, they can accept any frame available in parent peers as their starting frames. As a result, our criteria for peer join is simply to find an existing peer which can accept the new peer as its child peer with enough frames to satisfy the new peer. On the other hand, the case for peer reconnection is more complicated. Figure 5 shows all possible cases of buffer status between the child peer (n) and the parent peer (x). Ideally, we would like to find a parent peer with the buffer status as Figure 5(a), so that the next frames (that is, the frames starting from n.lastF +1) the child peer needed can be provided by the parent peer immediately.

#### 5.2.1 Dynamic peer selection criteria

Since only a limited number of peers are contacted by the child seeking for a new parent peer on peer reconnection, it is possible that all the candidates cannot meet the feasible condition as the one shown in Figure 5(a). Although it is possible to do the reconnection process again by contacting other possible candidates, finding a feasible peer in the next retry cannot be guaranteed. Consequently, we propose a dynamic decision scheme based upon the emergency of child peer status. The decision algorithm (named Reconnect_accept) is shown in Figure 4.

First, if the buffer status of the parent peer candidate (peer x) perfectly matches that of the reconnecting peer (peer n), then the system accepts the peer (x) as the new parent peer of the reconnecting peer (n) (Line 3). Otherwise, we check to see if the reconnecting peer contains enough preloaded frames to tolerate spending more time to try again. If the available frames are under some threshold (the predefined high-water-mark (HWM)) (Line 4), then we consider relaxing the constraints. If the available frames cannot af-
5.3 Frame Forwarding Mechanisms

Based on the above different design strategies, we designed a set of different forwarding mechanisms, summarized in Table 1. All mechanisms except the mechanism M_NoneBuf utilize frame buffer. The none-buffer mechanism is provided as a baseline for comparison. The mechanism with simple buffering strategy (M_Buf) is also provided as a baseline for comparison with other sophisticated mechanisms. The M_FM mechanism supports the fast forwarding mode we discussed before. The M_FM_FS mechanism supports the proposed frame synchronization scheme, which is expected to play an important role in providing better performance of frame forwarding. The first four mechanisms (M_NoneBuf, M_Buf, M_FM, and M_FM_FS) provide four different strategies regarding frame caching issues. As for old frame management, we use +nof notation to indicate the mechanism without keeping old frames. Alternatively, for the mechanisms to keep old frames, we remove the +nof notation. Consequently, the first four mechanisms (all without the +nof notation) would keep old frames during frame forwarding. Also, as for peer selection on peer reconnection, two alternative strategies, ALF and AUF, can be applied under critical conditions. By default, we use the ALF strategy. For those mechanisms that apply to the AUF strategy, we explicitly add the notation +auf to the corresponding mechanism names.

6. EXPERIMENTS

In this section, we report experimental results of evaluation on the proposed mechanisms. We did the experiments with computer simulation. According the system model described in Section 4, we assume peers are homogeneous. Each peer is with same network bandwidth to accept a limited number of child peers. The network delay between two peers is ignored. Meanwhile, we assume the program length of the multicasting stream is two-hour (7200 seconds). The total number of participant peers is up to 1000, with a constant join rate of three seconds per peer join. Besides, different input traffic scenarios are considered. Different rates of peer leave and failure are specified, as shown in Table 2 (the “—” in table stands for same setting as S_Normal). Also, different delay cost settings are modeled as those shown in Table 3. Descriptions of the terms and other related system parameters are summarized in Table 4.

6.1 Results under S_Normal Traffic Scenario

First we present the results under the S_Normal traffic scenario. We used different performance measurements in our experiments. Due to space limitations, here we only

![Figure 5: Possible buffer status relations between the new peer n and the parent peer x on peer reconnection. (a) the parent peer can provide the next frame immediately; (b) some waiting time is needed for the child peer to have the next frame from the parent peer; (c) the next frame is no longer available in the parent peer.]

Table 1: Frame forwarding mechanisms

<table>
<thead>
<tr>
<th>Mechanism</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>M_NoneBuf</td>
<td>No frame buffer.</td>
</tr>
<tr>
<td>M_Buf</td>
<td>Simple frame buffer strategy.</td>
</tr>
<tr>
<td>M_FM</td>
<td>M_Buf + fast forward mode (FM).</td>
</tr>
<tr>
<td>M_FM_FS</td>
<td>M_FM + frame sync. strategy (FS).</td>
</tr>
<tr>
<td>(M_Buf, M_FM, M_FM_FS) +nof</td>
<td>Same as xxx (xxx = M_Buf, M_FM, or M_FM_FS) but without keeping old frames.</td>
</tr>
<tr>
<td>(M_Buf, M_FM, M_FM_FS) +auf</td>
<td>Same as xxx+nof (xxx = M_Buf, M_FM, or M_FM_FS) but with the AUF strategy instead of ALF.</td>
</tr>
</tbody>
</table>

Table 2: Traffic scenario

<table>
<thead>
<tr>
<th>Parameter</th>
<th>S_Normal</th>
<th>S_Large Join</th>
<th>S_Large Leave</th>
<th>S_Large Leave Failure</th>
</tr>
</thead>
<tbody>
<tr>
<td>n Leave Events</td>
<td>100</td>
<td>400</td>
<td>—</td>
<td>400</td>
</tr>
<tr>
<td>Leave Rate (sec/peer)</td>
<td>72</td>
<td>18</td>
<td>—</td>
<td>18</td>
</tr>
<tr>
<td>n Failure Events</td>
<td>50</td>
<td>—</td>
<td>200</td>
<td>200</td>
</tr>
<tr>
<td>Failure Rate (sec/peer)</td>
<td>144</td>
<td>—</td>
<td>36</td>
<td>36</td>
</tr>
</tbody>
</table>

Table 3: Different setting of delay values

<table>
<thead>
<tr>
<th>Parameter</th>
<th>P_Normal</th>
<th>P_Large Join</th>
<th>P_Large Leave</th>
<th>P_Large Failure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Join delay</td>
<td>0.2 sec</td>
<td>0.5 sec</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>Leave delay</td>
<td>0.5 sec</td>
<td>1.0 sec</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>Failure delay</td>
<td>2.0 sec</td>
<td>—</td>
<td>4.0 sec</td>
<td>—</td>
</tr>
<tr>
<td>Reconnect delay</td>
<td>0.2 sec</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>
discuss the measurement on frame loss. We calculated the accumulative number of frame losses of the system along the program delivery time (7200 seconds).

6.1.1 Comparison between different frame cache strategies

First, we wanted to know the effects of utilizing different strategies on frame cache. We compared the four mechanisms: M_NoneBuf, M_Buf, M_FM, and M_FM_FS. The result (for the case of P_normal setting) is shown in Figure 6. The x-axis is the simulation time (in seconds); the y-axis is the accumulative number of frame loss (in log scale). The results show that without buffering (the case of M_NoneBuf) the number of frame loss soars, which indicates the essential need of frame buffering in the P2P streaming. Meanwhile, we also observed that frame loss is even worse with simple buffering strategy (M_Buf) than that with no buffering (M_NoneBuf). This is due to the combining effects of old frames management and peer selection. To fully utilize free space in the frame buffer, with M_Buf mechanism the peer will spend most of the buffer space for old frames while at worse only one frame in the buffer is new for the peer, which makes the valid frame size (the number of new frames) always below the low water mark threshold and causes the peer to accept a parent peer unconditionally. Although a similar situation is applied to the other two mechanisms (M_FM and M_FM_FS), both of the mechanisms utilize fast forward mode to cache more new frames in the buffer. Consequently, they suffer less. Meanwhile, it is not surprising that the one with frame synchronization strategy (M_FM_FS) behaves best among the four mechanisms. With the frame synchronization strategy, the problem of buffer mismatching is alleviated, which effectively reduces the number of frame losses.

6.1.2 Comparison between different old frame management strategies

Regarding old frame management, we compared those buffering mechanisms with retaining old frames (M_Buf, M_FM, and M_FM_FS) and discarding old frames (M_NoneBuf, M_FM+nof, M_FM_FS+nof). The result is shown in Figure 7. Again, we compared the number of accumulative frame losses (y-axis, in log scale) over the 2hr (7200 sec) of program stream delivery (x-axis). In general, we found that the forwarding mechanisms that discarded old frames achieve better performance in terms of frame loss. As discussed previously, there is a dilemma in retaining old frames. Old frames would be useful for lag peers, while at the same time they consume spare buffer space. Consequently, when there is large phase-skew among peers, retaining old frames is useful for some lag peers to find a proper parent peer, providing those relatively old frames are needed during re-connection. However, in a system providing consistence of buffer-status among most of the peers, the old frames for one peer are very likely the old frames for the other peers. Consequently, under such circumstances, there is no need to retain old frames in the frame buffer. Instead, the spare buffer space should be reserved to cache more new frames for a peer. This explains why the frame synchronization strategy that discards old frames (the M_FM_FS+nof) behaves best in this comparison.

6.1.3 Comparison between ALF and AUF policies

We also evaluated the difference in applying the two peer selection criteria (ALF v.s. AUF) when the number of avail-
In this paper, we investigate related design issues on frame forwarding and their impacts on playback quality in P2P streaming. One of the unique features of P2P streaming, compared with conventional client-server paradigms, is that the distribution of program streams is achieved by frame forwarding. The results of other input traffic scenarios with larger number of leave, failure, and both large leave and failure events are shown in Figures 9 to 11 (the actual parameters corresponding to each input scenario are shown in Table 2). Due to space limitations, we only show the results under the P_Normal setting. Similar to the results under the S_Normal scenario, the M_FM_FS+nof mechanism behaves best. Also, the ALF and AUF polices do not make much difference under both of the M_FM+nof and the M_FM_FS+nof mechanisms. Under the scenarios with a larger number of leave events (S_LargeLeave and S_LargerLeaveFailure), the merits of the mechanism based on frame synchronization strategy that discards old frames (M_FM FS+nof) become more significant compared with other mechanisms. This demonstrates the power of the proposed frame synchronization strategy.

7. CONCLUSIONS

The results on the large join delay case (in the last column of Table 5) could be somewhat misleading. The amount of frame loss we measured is based on a system-wide measurement, contributed by all peers. When join delay is large, the number of active peers in the system decreases. As a result, the total number of frame losses might decrease.
forwarding between peers instead of providing from central servers. We point out the importance of the frame buffer, and the problem of frame buffer mismatching between peers. From our simulation results, we show that by combining efficient strategies on both frame caching and peer selection, an effective P2P frame forwarding mechanism can be achieved. The proposed mechanism, frame synchronization with discarding old frames, reduces frame loss significantly.

8. ACKNOWLEDGMENTS

We thank Mr. Ji-Zhon Shieh for his help in building the prototype system. Also, we thank anonymous reviewers for their valuable comments. This work is supported in part by the National Science Council, Taiwan under grant number NSC93-2213-E-019-013-, and by National Taiwan Ocean University under grant number NTOU92-008.

9. REFERENCES