QuickSwap: A Lightweight Fast Recovery Protocol for Mesh-based P2P Live Streaming

Chih-Chiang Wang, Bi-Wei Zhuang, Mon-Yen Luo

Department of Computer Science, National Kaohsiung University of Applied Sciences, Taiwan
ccwang@kuas.edu.tw, xnilwind@gmail.com, myluo@kuas.edu.tw

Abstract

In this paper, we propose a failure recovery protocol named *QuickSwap* for mesh-based P2P live streaming systems. Unlike the existing mCache-based failure recovery scheme, QuickSwap enables peers to quickly recover their live streaming quality from neighbor failures by exchanging and pre-processing of local streaming information. Our contributions in this work are two-fold. First, QuickSwap's lightweight streaming-information exchange protocol together with the proposed neighbor maintenance and streaming-quality estimation methods provide a significant performance edge over the conventional mCache-based failure recovery. Second, we have built a system model for our proposal and have evaluated QuickSwap's performance under different system settings through simulation methodology. Our results reveal that at a small, affordable overhead cost, QuickSwap protocol can support fast recovery of live streaming quality with a nearly 100% failure recovery success rate and a fail-over time less than one second for an average of exponentially distributed peer lifetimes as low as 10 minutes.

Keywords: P2P live media streaming, Fault tolerance, Fail-over recovery

1 Introduction

P2P live media streaming is becoming an increasingly popular Internet application, as evidenced by commercial offerings from PPTV [1], PPstream [2], Sopcast [3], among others. Unlike traditional *Content Delivery Networks* which rely on dedicated servers to deliver live media content [4-7], P2P live media streaming utilizes free but unreliable upload capacity of end-user participants (peers), thus achieving benefits of low deployment cost and high scalability. Peers typically form an overlay network, download live media content directly from the media source or from their overlay neighbors at a required streaming rate, and play the received content in approximate synchronicity with the source. However, research studies [8-10] indicate that inherent *peer churn*, or the dynamics of peers joining and leaving the P2P system, can cause peers experiencing frequent neighbor failures and unable to download from the remaining neighbors at the required rate. Therefore, a key design issue in supporting good P2P live streaming quality is how to maintain a sufficient supply of live media stream to each peer under the impact of peer churn.

Of interest in this work is the mesh-based P2P live streaming scheme used to achieve scalable and robust live media delivery in several proposals [11-14]. Modern mesh-based P2P designs reduce the churn's impact by dividing the live media stream into K multiple sub-streams, $K \geq 1$, so each peer can retrieve the sub-streams independently from K different content-supplier neighbors termed parents. To enhance P2P live streaming quality under the churn's impact, our objective is the design of a lightweight fast recovery protocol named *QuickSwap* to periodically maintain K parents in each peer's neighbor set plus a few backup parents in reserve for fast fail-over, so at least K out of all the parents and backup parents are highly likely to survive until the next maintenance operation.

Our proposal is beneficial in two ways. Firstly, maintaining a few backup parents is more cost-effective than maintaining a large set of parents or neighbors. Because each parent failure costs a peer an instantaneous loss of one K-th of its live media supply, the latter approach must use a very large parent/neighbor set to keep this loss fraction negligible, but doing so will incur expensive neighbor-communication overhead. In comparison, our results show that by QuickSwap protocol, maintaining a couple of additional backup parents every 30 seconds is enough to support fast recovery of live streaming quality with a nearly 100% failure recovery success rate and a fail-over time less than one second for an average of exponentially distributed peer lifetimes as low as 10 minutes. Secondly, QuickSwap can efficiently exploit local information to facilitate lightweight fast recovery. A mesh-based P2P system generally adopts a gossip-like protocol to maintain a list of backup neighbors in its membership cache (mCache), but our simulation results reveal that it takes

*Corresponding Author: Chih-Chiang Wang; E-mail: ccwang@kuas.edu.tw
DOI: 10.3966/160792642018031902015

a non-trivial random amount of time for a peer to find itself a qualified parent from the mCache. In comparison, QuickSwap periodically pre-screens local mCache information and streaming-status information exchanged between neighbors to stock up on a few backup parents. By this pre-screening method, QuickSwap reduces the task of failure recovery to only fail-over to the backups. The contributions of this work are two-fold: (1) For achieving fast recovery in mesh-based P2P live streaming, the design of QuickSwap has included a streaming-status protocol for lightweight information exchange, a distributive streaming-quality estimation method, and a neighbor maintenance algorithm to help each peer stock up on a required number of parents and backup parents. (2) We have built a system model for our proposal and have evaluated QuickSwap's performance under different settings through simulation methodology. The balance of this paper is organized as follows. In Section 2, we survey existing work on P2P live streaming systems and fail-over recovery of P2P overlay, and relate them to our own. In Section 3, we model a general mesh-based P2P live streaming system and introduce the terminology and the mathematical symbols used through this paper. In Section 4, we introduce the design rationale, the message format, and the operation of QuickSwap protocol. In Section 5, we conduct simulation experiments to evaluate QuickSwap's performance under different system settings. We finally conclude in Section 6.

2 Related Work

2.1 P2P Live Streaming

Existing P2P live media streaming technologies can be broadly classified into either the tree-based or the mesh-based. The earlier proposals were largely built upon the tree-based overlay where participants are organized into one or more end-system multicast trees. The media source can deliver the live media stream to peers through the tree structure. The tree-based schemes can be further classified into the single-tree [15-17] or the multiple-tree [18-21]. The single-tree has long been criticized for its inefficiency because leaves in a single multi-cast tree can barely serve the live media to other peers. The multiple-tree improves its load balancing by dividing the live media stream into multiple sub-streams and disseminating each sub-stream over an independent multicast tree. Overall speaking, the tree-based schemes suffer performance degradation and expensive maintenance overheads under peer churn.

The mesh-based data-driven P2P live streaming was firstly proposed in [11] and later refined in [12-14]. Unlike the tree-based schemes, the mesh-based schemes achieve good fault resilience and load balancing by employing swarm-like content delivery whereby peers form a random mesh overlay and pull live media content from their overlay neighbors. Large-scale P2P streaming experiments [8, 11-12] have validated the feasibility of the mesh-based schemes in real systems over the Internet. For these reasons, our work is built on the basis of the mesh-based P2P live media streaming.

2.2 Fail-over Recovery of P2P Overlay

P2P overlay networks can quickly respond to neighbor failures by fail-over to backup neighbor connections. The history of applying fail-over recovery to P2P overlay maintenance can be traced back to the early era of structured P2P systems. In [22], Zhao et al. use backup routing paths and fault detection techniques to achieve fast failure recovery in a fault-resilient overlay named Tapestry [23]. The work of [24] introduces the property of *K*-consistent neighbor table to improve robustness and failure recovery of structured P2P overlays.

Gossip protocols are commonly used to find good neighbor candidates in the mesh overlay with low management overhead. In mesh-based P2P live streaming [11-12], each peer employs SCAMP-like gossip [25] to maintain its mCache, which records a partial list of currently active peers in the system as backup neighbors. However, the mCache is a stochastic result of the underlying gossip propagation, so there is no guarantee which peers recorded in the mCache are capable of serving the live media stream to the peer host. AnySee [13] has a backup path management design which uses reverse tracing to find better streaming paths in the overlay. In [13], when the media source receives a request for reverse tracing, the source computes the best streaming path and sends the result back to the requester. The work of [26] proposes an ID-based Web Browser with ISP-friendly P2P content delivery property. In [27], the authors leverage under-utilized resources in a community network to form caches of files. The work of [28] proposes an enhanced failover mechanism for P2P streaming over multiple multicast trees, which is considered too rigid to be practical. The work of [29] proposes a ternary cube model named Hyper-Ternary-Cube to support content-based music searches with error tolerance in a peer-to-peer overlay. In the work of [30], the authors propose a new Time-Driven Mesh (TDM) overlay network for peer-to-peer video-on-demand systems. In the work of [31], the authors we propose peer selection with QoS aware for P2P IPTV service by evaluating quality of video content delivered using an end-to-end measuring approach.

Our design of QuickSwap protocol is distinguished from the related work in several aspects. QuickSwap avoids centralized mechanisms which place the computation burden on a single node. Moreover, QuickSwap achieves lightweight fast recovery of P2P live streaming quality by local information exchange

and pre-screening of the received local information.

3 System Model

Our model targets at a general mesh-based P2P system where a live media stream is originated from a media source v_s and is being distributed to the peers in the system throughout a live broadcast session. The live media stream is divided into a sequence of blocks of uniform length without any coding, and the video blocks are grouped into K logical sub-streams such that the i-th sub-stream contains blocks with sequence number $(i+j \cdot K)$, where i is a positive integer from 1 to K and j is a non-negative integer [12]. The required streaming rate for this media stream is r kbps (kilo-bits per second). Suppose at a specific moment of time t, V_t is the set of active peers in the system and E_t is the set of overlay connections among peers in V_t. Then the overlay of this P2P system at time t resembles an undirected graph $G_t = (V_t, E_t)$.

Peers with a direct overlay connection between themselves are termed neighbors. Suppose $v_{x,t}$ is an active peer v_x at time t, then $\Psi_{x,t}$ represents the neighbors of $v_{x,t}$. Each peer records a random partial list of the currently active peers in its mCache and can contact them to establish m neighbors in the overlay. Each peer also stores in buffermaps the availability information of the latest video blocks in its playback buffer [11]. Peers periodically exchange buffermaps with their neighbors in order to determine which neighbor possesses which video block. Based on its received buffermaps, a peer can use a pull-based method to request any of its neighbors to send back specific video blocks. We let peers pull the missing blocks with the closest playback time first as suggested in CoolStreaming [11].

P2P literature [12, 32-33] shows that the hybrid push-pull method offers a good performance tradeoff between streaming throughput and fault resilience. Our model adopts the hybrid push-pull method whereby after a peer subscribes to a sub-stream by connecting to one of its neighbors via a single pull request, the requested neighbor will continue pushing all blocks in need of the sub-stream to the requested peer. When a peer continues pushing video blocks to one of its neighbors, the former and the latter form a parent-child relationship. For ease of explanation, let $Y_{x,t}$ and $\Xi_{x,t}$ represent the parents and the children of $v_{x,t}$ in $\Psi_{x,t}$, respectively; and let $\hat{Y}_{x,t}$ represent $v_{x,t}$'s backup-parent set which is maintained by QuickSwap every τ_p seconds. When a parent in $Y_{x,t}$ fails or is inadequate in satisfying the streaming requirement, $v_{x,t}$ will fail over to a backup parent in $\hat{Y}_{x,t}$, if any exists, or select a new parent from $\Psi_{x,t}$ by picking the neighbor whose latest received video blocks can mostly satisfy $v_{x,t}$'s need. A peer also refines its neighborship by replacing the least active neighbor with a peer in its mCache.

The following assumptions and mathematical symbols are used to establish streaming quality constraints of QuickSwap protocol. To model peer churn, we generate join and failure events by the following procedure. For each join event, a newly created peer contacts the system's tracker server to obtain a list of neighbor candidates, then establishes its overlay neighbors and begins its live streaming session. For each failure event, one or multiple existing peers are randomly chosen to fail. Our model assumes that peers have synchronized clocks by a time synchronization technology like Network Time Protocol, which allows end hosts to be synchronized within a few milliseconds of Coordinated Universal Time [34]. When v_s is about to send out a video block, it will record the current time in the header of the video block as its origin time. Thus, a peer $v_{x,t}$ can evaluate its *PlaybackDelay*, denoted by $\delta_{x,t}$, by subtracting the origin time of its received video block from the time the video block is scheduled to be played. Once a peer starts playing the live media stream, its *PlaybackDelay* approximates a constant value. Because the upload link is usually much slower than the download link, we consider peers' upload capacity be the bottleneck resource in P2P live streaming. We use $\mu_{x,t}$ to denote the free upload capacity of $v_{x,t}$ and use $s_{xy,t}$ and $s_{x,t}$ to denote the rate of the media stream transmitted from $v_{x,t}$ to $v_{y,t}$ and the rate of the media stream received by $v_{x,t}$, respectively.

4 QuickSwap Protocol Design

This section describes the design of QuickSwap protocol. In Subsection 4.1, we explain the design rationale of QuickSwap protocol. We elaborate the message format and the operation of QuickSwap protocol in Subsection 4.2 and 4.3, respectively. In Subsection 4.4, we present the overhead analysis.

4.1 Design Rationale

In a generic mesh-based P2P system $G_t = (V_t, E_t)$ as described in Section 3, our objective is to let each peer $v_{x,t}$ in V_t employ QuickSwap protocol to periodically maintain K parents in $\Psi_{x,t}$ plus a few qualified backup parents in $\hat{Y}_{x,t}$ for fast fail-over, so at least K out of all the parents and backup parents are highly likely to survive and supply the sufficient sub-streams to $v_{x,t}$ until the next maintenance operation. For convenience of reference, we term this desired property K-survival parentship. Before we proceed any further, we first clarify how QuickSwap picks qualified (backup) parents for $v_{x,t}$ by two constraints. By the first constraint, if a peer $v_{y,t}$ is qualified to serve as a parent of $v_{x,t}$, $v_{y,t}$ must have adequate free upload capacity to push a sub-stream to $v_{x,t}$, which yields:

$$\mu_{y,t} \geq \frac{r}{K}, \forall_{y,t} \in \hat{r}_{x,t} \quad \textbf{(1)}$$

Secondly, $v_{x,t}$ must successfully connect to $v_{y,t}$ via a single pull request before they can form a parent-child relationship. Note that this constraint can be tested directly by $v_{x,t}$ sending a pull request to $v_{y,t}$ and waiting for the reply. But if, say, a peer wants to quickly determine if some peer recorded in its mCache could serve as its parent, then the following approximation method can be used.

By the pull method, v_y periodically sends buffermap messages to its neighbors. Once v_x receives a buffermap message from v_y, v_x sends a block request back to v_y. When receiving v_x's block request, v_y sends the requested video block back to v_x. We derive an approximate timing constraint from the above process:

$$\delta_{x,t} - B \leq \delta_{y,t} + 3d_{xy,t} + \frac{|BM| + |block|}{\mu_y} \leq \delta_{x,t}$$

where B is the playback buffer size in unit of seconds, $d_{xy,t}$ is the end-to-end delay between v_x and v_y in unit of seconds, $\mu_{y,t}$ is v_y's upload capacity in unit of kbps, and $|BM|$ and $|block|$ are the sizes of a buffermap message and a video block in unit of kilo-bits, respectively. Because a slow 500-kbps upload link can transmit a large 100 kilo-bit video block/buffermap message [12] in 0.2 seconds and the end-to-end delay is mostly bounded by 0.2 seconds [35], we approximate the second constraint as follows for use in practice:

$$\delta_{x,t} - B \leq \delta_{y,t} \leq \delta_{x,t} - 1, \forall_{y,t} \in \hat{r}_{x,t} \quad \textbf{(2)}$$

Next, we derive two additional maintenance constraints from the properties of K-survival parentship. The first one states an objective that when a maintenance operation completes at $v_{x,t}$, $v_{x,t}$ shall have K parents to sustain the required streaming rate. The second one states that when a maintenance operation completes at $v_{x,t}$, the peer shall have at least K parents or backup parents survive until time $(t + \tau_p)$ with a high probability greater than $(1 - \in)$. We express these two maintenance constraints as follows:

$$s_{x,t} \geq r, |r_{x,t}| = K \quad \textbf{(3)}$$

$$P_r(|r_{x,t+\Delta t}| + |\hat{r}_{x,t+\Delta t}| < K \,|\, \hat{r}_{x,t}) \leq \in, \forall \Delta t \in [0, \tau_p] \quad \textbf{(4)}$$

The next two sub-sections will elaborate how peers maintain K-survival parentship in a distributive manner through QuickSwap protocol.

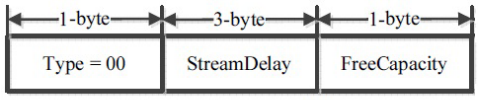
4.2 QuickSwap Messages

We define three types of QuickSwap messages: *neighbor-status*, *backup-status*, and *backup-request*, for peers to exchange streaming-status information with one another. Neighbor-status messages are sent from any arbitrary peer $v_{x,t}$ to its neighbors $\Psi_{x,t}$. A neighbor-status message contains the 2-tuple information: *PlaybackDelay* and *FreeCapacity*. Backup-request messages are used by some peer $v_{x,t}$ to request another peer $v_{y,t}$ to send its backup-status message to $v_{x,t}$. When $v_{x,t}$ receives a backup-request message from $v_{y,t}$, $v_{x,t}$ will include $v_{y,t}$ in a set $\hat{\Xi}_{x,t}$. Finally, backup-status messages are sent from $v_{x,t}$ to $v_{y,t}$ if $v_{y,t}$ is in $\hat{\Xi}_{x,t}$; they are also piggybacked on the membership messages for gossip propagation. A backup-status message has three fields: *PlaybackDelay*, *FreeCapacity*, and *StreamRate*. Figure 1 illustrates the structures of QuickSwap messages, and their fields are explained below:

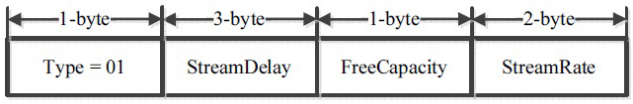
(1) *PlaybackDelay* of $v_{x,t}$ contains a 3-byte value which records $\delta_{x,t}$ in unit of milliseconds.

(2) *StreamRate* of $v_{x,t}$ contains a 2-byte value in unit of kbps. It is computed by dividing the total size of the video blocks $v_{x,t}$ has played during the last maintenance period by τ_p.

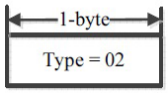
(3) *FreeCapacity* of $v_{x,t}$ contains a 1-byte value which records the maximum number of children that $v_{x,t}$ currently can support.



(a) a neighbor-status message



(b) a backup-status message



(c) a backup-request message

Figure 1. QuickSwap messages

4.3 QuickSwap Protocol

The operation of QuickSwap protocol is implemented in two processes: *information propagation* and *neighbor maintenance*. Peers can execute these two processes in parallel with small computation and communication overheads. By the process of *information propagation*, peers periodically send their streaming-status information in QuickSwap messages to other peers. The *neighbor* maintenance process tries to maintain K parents and a few backup parents by periodically pre-screening the received neighbor information and QuickSwap messages. If violation of Constraint (3) is detected, this process

willreplace failed or unqualified parents with backup parents until the violation is resolved or no more backup parents can be found.

The pseudo code of *information propagation* is exhibited in Figure 2 and explained below. When this process begins at a peer $v_{x,t}$, it firstly schedules the next round of *information propagation* to begin at time ($t + \tau_p$) (Line 1). Then it prepares neighbor-status and backup-status messages for transmission (Line 2 and 3) and sends these messages to the corresponding recipients, $\Psi_{x,t}$ and $\hat{\Xi}_{x,t}$, respectively (Line 4 to 6). The membership of $\hat{\Xi}_{x,t}$ is determined according to the backup-request messages that $v_{x,t}$ has received since the last round of *information propagation* (Line 5). The backup-status messages are also piggybacked on the gossip membership messages for gossip propagation (Line 7). If any change in $\Psi_{x,t}$ alters the content of QuickSwap messages, $v_{x,t}$ will send new QuickSwap messages to the corresponding recipients.

Info_Propagation(peer $v_{x,t}$):

1. *SetTimer*($t + \tau_p$, *Info_Propagation*($v_{x,t}$))
2. N_MSG ← *MakeNeighborMSG*($v_{x,t}$)
3. B_MSG ← *MakeBackupMSG*($v_{x,t}$)
4. *Send*(N_MSG, $\Psi_{x,t}$)
5. $\hat{\Xi}_{x,t}$ ← received BACKUP_REQUEST
6. *Send*(B_MSG, $\hat{\Xi}_{x,t}$)
7. *UpdateGossipMSG*(GOSSIP_MSG+B_MSG)

Figure 2. Pseudo code of information propagation

The *neighbor maintenance* process consists of two procedures named *backup selection* and *parent selection*. Figure 3 shows its pseudo code. *Backup selection* firstly pre-screens the received backup-status messages and adds those which meet Constraint (1) and (2) into the backup-parent set $\hat{Y}_{x,t}$ until Constraint (4) is met (Line 2 to 7). The rest messages are stored in the mCache (Line 8). This procedure optimizes the structure of the neighbor set by storing those in ($\Psi_{x,t} \backslash Y_{x,t}$) which meet Constraint (1) and (2) in the front of the neighbor set and in the descending order of their *StreamRate* (Line 9). The execution time of Line 9 is trivial because the neighbor set to be sorted contains 100 elements at most.

The *parent selection* procedure tries to fill the parent set with the pre-screened backup candidates until Constraint (3) is met or no more backup candidates can be found. This procedure is invoked after the execution of *backup selection* or at the buffermap timeout every second. In specifics, this procedure detects violation of Constraint (3) by checking the counts of how many video blocks were received from each parent during the last second (Line 1 to 2). If Constraint (3) is not met, the *Replace* or *ReplaceQualified* function will replace a failed or unqualified parent with a backup parent (Line 3), or a neighbor (Line 4 to 5), or a mCache peer (Line 6 to 7) that is qualified to serve the sub-stream. At the end, if the neighbor set is still not full, parent selection will fill the neighbor set with peers randomly selected from the mCache to retain the random-mesh property (Line 9). The computation speed of this procedure is fast due to the pre-processing of received QuickSwap messages and the optimization performed by backup selection (for detailed analysis, see Subsection 4.4).

Backup_Selection(peer $v_{x,t}$):

1. *SetTimer*($t + \tau_p$, *Backup_Selection*($v_{x,t}$))
2. While (Constraint(4) fails and moreB_MSG()) do
3. v_i = getB_MSG()
4. If (v_i meets Constraint(1) and (2)) and ($v_i \notin (\Psi_{x,t} \cup \hat{Y}_{x,t})$)
5. *Add*($\hat{Y}_{x,t}$, v_i)
6. *BackupRequest*(v_i)
7. End-While
8. MCACHE ← the rest B_MSG
9. *SortRate*($\Psi_{x,t} \setminus Y_{x,t}$)
10. Call *Parent_Selection*($v_{x,t}$)

Parent_Selection(peer $v_{x,t}$):

1. For each $Parent_i \in Y_{x,t}$ do
2. If ($Parent_i$ fails or $s_{ix,t} < \frac{r}{K}$)
3. If ($|\hat{Y}_{x,t}|$) *Replace*($Parent_i$, $\hat{Y}_{x,t}$)
4. Else if (*MoreQualified*($\Psi_{x,t} \setminus Y_{x,t}$))
5. *ReplaceQualified*($Parent_i$, $\Psi_{x,t} \setminus Y_{x,t}$)
6. Else if (*MoreQualified*(MCACHE))
7. *ReplaceQualified*($Parent_i$, MCACHE)
8. Else Break
9. If ($|\Psi_{x,t}| < m$) *FillNeighbor*($\Psi_{x,t}$, MCACHE)

Figure 3. Pseudo code of neighbor maintenance

4.4 Overhead Analysis

QuickSwap's communication overhead is mainly determined by the sizes of the neighbor set and the backup-parent set. By QuickSwap, neighbors exchange their neighbor-status messages every τ_p seconds, so the communication overhead of this exchange is bounded by $5m/\tau_p = \Theta(m/\tau_p)$ bytes per second per peer. In regard to the communication overhead associated with the backup-parent set, let us consider the worst-case scenario in which the QuickSwap maintenance operation needs to prepare K backup parents. Then the worst-case communication overhead associated with the backup-status messages is $O(m/\tau_p)$ bytes per second per peer. Summing up, the overall communication overhead associated with QuickSwap protocol is bounded by $\Theta(m/\tau_p)$ bytes per second per peer.

Parent selection is the only QuickSwap procedure which handles the recovery of P2P live streaming quality on the fly. Parent selection aims to quickly replace failed or unqualified parents with the pre-selected backup parents if violation of Constraint (3) has been detected. In the worst-case scenario, parent selection has to establish K new parent connections.

Therefore, the worst-case time complexity of parent selection approximates *O(K)*.

Table 1 summarizes the complexity of QuickSwap protocol in terms of communication and computation overheads. Note that even if the conventional mCache-based failure recovery could recover a failed parent in less than one second, each peer needs to maintain at least $(1/\in)$ parents to maintain the QoS described in Constraint (4). For $\in$=1%, it means that each peer needs at least 100 parents. In comparison, to achieve the same fail-over probability $(1-\in)$, QuickSwap requires each peer to maintain only 20 neighbors plus a couple of backups.

Table 1. Summary of complexity of QuickSwap protocol

QuickSwap Overhead	Asymptotic Bound
Communication Overhead	$\Theta(m/\tau_p)$
- *neighbor-status* message	$\Theta(m/\tau_p)$
- *backup-status* message	$O(m/\tau_p)$
Computation Overhead (on-the-fly)	O(K)
- *parent selection*	O(K)

5 Performance Evaluation

In this section, we conduct simulation experiments to demonstrate the effectiveness and the efficiency of QuickSwap failure recovery protocol. The experimental results are primarily used to: (1) compare the P2P system's failure recovery performance with and without using QuickSwap protocol; and (2) study the impact of QuickSwap's parameters on its failure recovery performance. We first explain the performance metrics and the setup of our simulation experiments and then present some representative results and discuss their implications.

5.1 Performance Metrics

We use the following metrics to evaluate the failure recovery performance of a mesh-based P2P live streaming system.

Playback continuity index (PCI). To evaluate video playback quality perceived by a peer, we define this metric as the ratio of the number of video blocks that have been played by the peer during the last second over the total number of video blocks that should have been played every second. (PCI ≡ 1) means perfect video playback without any interruption, and (PCI ≡ 0) means that the playback buffer has run out of data.

Fail-over time. This metric measures the time elapsed from the occurrence of a failure event to the moment when the affected peer has fully recovered its PCI.

Recovery success rate. This is the fraction of the peers which were affected by the failure event but subsequently have fully recovered their PCI within a limited measurement period.

Communication overhead. This measures the amount of upload bandwidth per peer consumed by exchange of buffermap or QuickSwap messages.

5.2 Simulation Setup

We use an OMNeT++-based open-source simulation framework, Oversim [36], to build a packet-level simulation which captures the functionality of a generic mesh-based P2P live streaming system and QuickSwap protocol. The simulation of QuickSwap in this section only serves as a proof of concept, so we make a few simplifications in the simulation: (1) we dismiss the effect of the physical network topology and consider P2P upload capacity at the network edge be the only bottleneck to live media streaming; (2) we simulate the end-to-end propagation delay by a random value from 10 to 150 milliseconds, which is appropriate in general cases according to real-world measurement results [12, 37]; (3) we model the mCache as a stochastic process which uniformly samples the active peers in the system every τ_p seconds for the purpose of comparison with QuickSwap's performance.

The following are the default simulation settings, similar to those used in [12]. There is one media source and one tracker server in the system initially. The media source has an upload capacity of 10 Mbps and can handle up to 20 children. At the beginning of the simulation, 1000 peers contact the tracker server and join the system. The tracker informs peers that the expected peer lifetime is 10 minutes. Each peer can maintain at most m = 20 neighbors and has an average upload capacity of roughly 500 kbps. To simulate the heterogeneity in peers' upload capacity, we adopt the two-class model from [38]; a small fraction, say 1%, of the peers are classified as super peers, each with an upload capacity of 1 Mbps, and the remaining peers are classified as ordinary peers, each with an upload capacity of 500 kbps. The live media is streamed at r = 450 kbps with K = 10 sub-streams and with a uniform block size about 10 kilo-bits, which leaves only 50-kbps free upload capacity per peer to be used for exchange of buffermap and QuickSwap messages. Each peer can buffer up to 10 seconds' content and start playing the video when the buffer is loaded. The mCache can store a list of 100 peers. The buffermap transmission timeout is 1 second, whereas the QuickSwap maintenance timer is set to τ_p = 30 seconds. The streaming-quality parameter $\in$ is 1%.

5.3 Simulation Results

For a clean comparison of how well peers recover from failures with and without using QuickSwap protocol, in each of these two experiments we sample 30 active peers throughout the simulation run-time, force a fraction α of their parents to simultaneously depart from the system, and then measure their averaged playback continuity index, fail-over time, recovery success rate, and communication overhead

during the subsequent 300-second measurement period. For an average peer lifetime of 10 minutes, the chance that a peer loses more than half of its parents in 30 seconds is slim (less than 0.000001), so we use a set of failure rates, $\alpha \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$, to test the P2P system's failure recovery performance with and without using QuickSwap protocol. To prevent the system from dying out, we let the failed peers re-join immediately as newly initialized peers. In Figure 4, we plot the resultant averaged fail-over time and recovery success rate as to reflect the effectiveness of QuickSwap protocol. Figure 4 reveals the following conclusions: (1) The pure mCache mechanism alone cannot ensure timely failure recovery. Even in presence of a single parent failure (α = 0.1), the pure mCache mechanism takes tens of seconds to find a qualified parent candidate from the mCache. As the failure rate increases from 0.1 to 0.5, the averaged fail-over time of the pure mCache mechanism quickly rises and exceeds 100 seconds while the averaged recovery success rate quickly drops from 90% to below 20%. (2) On the other hand, by pre-screening the mCache and local streaming information, QuickSwap reduces the task of failure recovery to merely fail-over to the backup parents. As a result, for $\alpha \leq$ 0:4, QuickSwap delivers a remarkable failure-recovery performance with a 100% recovery success rate and a fail-over time less than one second. Even for the experiment with (α = 0.5), QuickSwap still achieves a 90% recovery success rate and a fail-over time less than one second. Table 2 makes a comparison between the averaged communication overheads due to periodic exchange of QuickSwap and buffermap messages.

To examine the stability of QuickSwap protocol, for the experiment with (α = 0.5), we plot the averaged playback continuity index of the peers which were affected by the parent failures in Figure 5. The sub-figure on the left shows that the failure recovery performance of the pure mCache mechanism is poor. At the end of the 300-second observation period, only 20 percent of the affected peers have been fully recovered (PCI=1.0), half of the remaining peers have their playback rate drop to a value between 300 and 400 kbps (PCI = [0:7; 0:9]), and the rest peers have their playback rate drop to below 300 kbps (PCI = [0:5; 0:7]). The sub-figure on the right shows that our proposed pre-processing of local streaming information. From our simulation experiments, QuickSwap was found to be effective, efficient, and stable up to a 50% parent-failure QuickSwap protocol delivers a very stable and timely failure recovery performance - only a slight "glitch" appears in the curves within the first 10 seconds after the occurrence of the failure event, then the curves corresponding to PCI= 1:0 and PCI=[0:9; 1:0] quickly stabilize at 90% and 10%, respectively. In other words, when using QuickSwap protocol, 90% of the affected peers can fully recover from the failure's impact in one second, and the remaining 10% can restore their playback rate to about 400 kbps in one second.

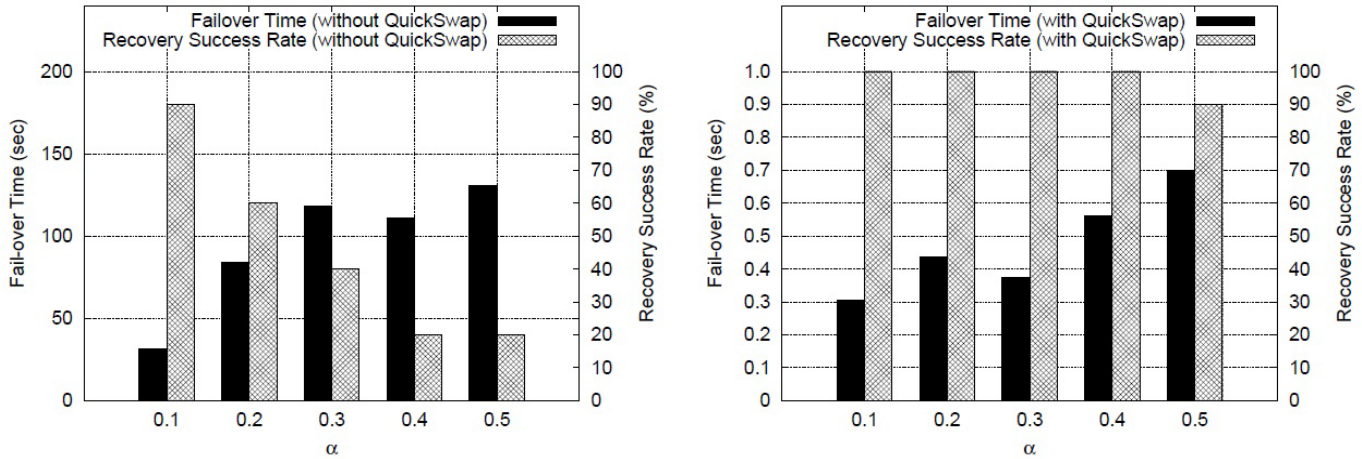


Figure 4. Fail-over time and recovery success rate against failure rate α: with (right) and without (left) using QuickSwap protocol while τ_p is set to 30 seconds

Table 2. Summary of QuickSwap's communication overheads

Message Type	Overhead (τ_p = 30 s)	Overhead (τ_p = 10 s)
QuickSwap Message	1.59 kbps	1.96 kbps
- *neighbor-status* message	1.41 kbps	1.41 kbps
- *backup-status* message	0.18 kbps	0.55 kbps
Buffermap Message	43.5 kbps	43.5 kbps

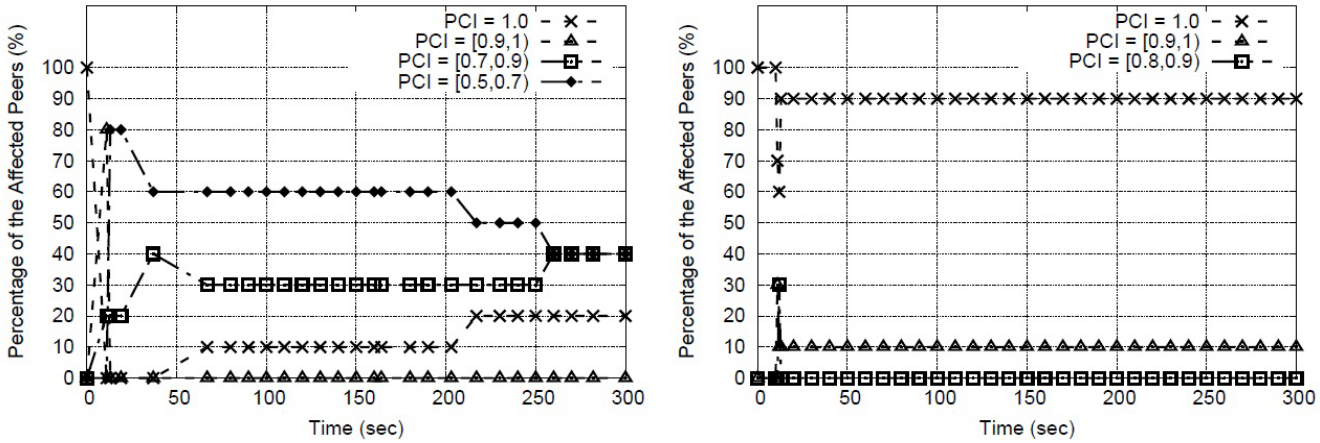


Figure 5. Playback continuity of the peers affected by failure event against time: under the system settings of (α, τ_p) = (0.5, 30s) with (right) and without (left) using QuickSwap protocol

6 Conclusions

We have shown in this paper that the conventional mCache mechanism alone cannot ensure timely failure recovery in an unreliable mesh-based P2P live streaming system. To enhance P2P live streaming quality under the churn's impact, we designed a lightweight failure recovery protocol named QuickSwap, which enables peers to quickly recover from parent failures by exchanging and pre-processing of local streaming information. From our simulation experiments, QuickSwap was found to be effective, efficient, and stable up to a 50% parent-failure rate for a 1000-node P2P network. Our results reveal that by QuickSwap protocol, maintaining a couple of additional backup parents every 30 seconds per peer is enough to support fast recovery of live streaming quality for an average of exponentially distributed peer lifetimes as low as 10 minutes. For a 40% parent-failure rate or less, QuickSwap delivers a remarkable failure-recovery performance with a 100% recovery success rate and a fail-over time less than one second. QuickSwap's lightweight streaming-information exchange protocol together with the proposed neighbor maintenance and streaming-quality estimation methods provide a significant performance edge over the conventional mCache-based failure recovery at a cost of small communication overhead of less than 2 kbps. However, when there are fewer high-bandwidth super peers in the system, finding qualified backup parents often takes longer. We believe that an excellent approach to solve this issue is the combination of QuickSwap protocol and the addition of auxiliary cloud servers. Thus, exploiting the on-demand resource provisioning of cloud computing to further enhance QuickSwap's failure recovery performance will be part of our future work.

References

[1] PPTV, http://www.pptv.com

[2] PPstream, http://www.ppstream.com

[3] Sopcast, http://www.sopcast.org

[4] C. D. Cranor, M. Green, C. Kalmanek, D. Shur, S. Sibal, J. E. Merwe, C. J. Sreenan, Enhanced Streaming Services in a Content Distribution Network, *IEEE Internet Computing*, Vol. 5, No. 4, pp. 66-75, July, 2001.

[5] J. Dilley, B. Maggs, J. Parikh, H. Prokop, R. Sitaraman, B. Weihl, Globally Distributed Content Delivery, *IEEE Internet Computing*, Vol. 6, No. 5, pp. 50-58, September, 2002.

[6] L. Kontothanassis, R. Sitaraman, J. Wein, D. Hong, R. Kleinberg, B. Mancuso, D. Shaw, D. Stodolsky, A Transport Layer for Live Streaming in a Content Delivery Network, *Proceedings of IEEE*, Vol. 92, No. 9, pp. 1408-1419, September, 2004.

[7] J. Ni, D. H. K. Tsang, Large-scale Cooperative Caching and Application-level Multicast in Multimedia Content Delivery Networks, *IEEE Communications Magazine*, Vol. 43, No. 5, pp. 98-105, May, 2005.

[8] N. Magharei, R. Rejaie, Understanding Mesh-based Peer-to-peer Streaming, *International Workshop on Network and Operating Systems Support for Digital Audio and Video*, Newport, Rhode Island, 2006, pp. 56-61.

[9] N. Magharei, R. Rejaie, Y. Guo, Mesh or Multiple-tree: A Comparative Study of Live P2P Streaming Approaches, *26th IEEE International Conference on Computer Communications*, Barcelona, Spain, 2007, pp. 1424-1432.

[10] C. Vassilakis, I. Stavrakakis, Minimizing Node Churn in Peer-to-peer Streaming, *Elsevier Computer Communications*, Vol. 33, No. 14, pp. 1598-1614, September, 2010.

[11] X. Zhang, J. Liu, B. Li, T. P. Yum, Coolstreaming/donet: A Data-driven Overlay Network for Peer-to-peer Live Media Streaming, *IEEE International Conference on Computer Communications Societies*, Miami, FL, 2005, pp. 2102-2111.

[12] B. Li, S. Xie, Y. Qu, G. Y. Keung, C. Lin, J. Liu, X. Zhang, Inside the New Coolstreaming: Principles, Measurements and Performance Implications, *IEEE International Conference on*

Computer Communications, Phoenix, AZ, 2008, pp. 1705-1713.

[13] X. Liao, H. Jin, Y. Liu, L. M. Ni, D. Deng, Anysee: Peer-to-peer Live Streaming, *IEEE International Conference on Computer Communications*, Barcelona, Spain, 2006, pp. 1-10.

[14] N. Magharei, R. Rejaie, Prime: Peer-to-peer Receiver-driven Mesh-based Streaming, *IEEE International Conference on Computer Communications*, Barcelona, Spain, 2007, pp. 1415-1423.

[15] Y.-H. Chu, S. G. Rao, S. Seshan, H. Zhang, A Case for End System Multicast, *IEEE Journal on Selected Areas in Communications*, Vol. 20, No. 8, pp. 1456-1472, October, 2002.

[16] S. Banerjee, B. Bhattacharjee, C. Kommareddy, Scalable Application Layer Multicast, *Annual Conference of the ACM Special Interest Group on Data Communication*, Pittsburgh, PA, 2002, pp. 205-217.

[17] D. A. Tran, K. A. Hua, T. Do, Zigzag: An Efficient Peer-to-peer Scheme for Media Streaming, *IEEE International Conference on Computer Communications*, San Francisco, CA, 2003, pp. 1283-1292.

[18] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, A. Singh, Splitstream: High-bandwidth Multicast in Cooperative Environments, *ACM Symposium on Operating Systems principles*, Bolton Landing, NY, 2003, pp. 298-313.

[19] V. N. Padmanabhan, H. J. Wang, P. A. Chou, K. Sripanidkulchai, Distributing Streaming Media Content Using Cooperative Networking, *International Workshop on Network and Operating Systems Support for Digital Audio and Video*, Miami, FL, 2002, pp. 177-186.

[20] V. Venkataraman, K. Yoshida, P. Francis, Chunkyspread: Heterogeneous Unstructured Tree-based Peer-to-peer Multicast, *IEEE International Conference on Network Protocols*, Santa Barbara, CA, 2006, pp. 2-11.

[21] J. Mol, D. Epema, H. Sips, The Orchard Algorithm: Building Multicast Trees for P2P Video Multicasting without Free-riding, *IEEE Transactions on Multimedia*, Vol. 9, No. 8, pp. 1593-1604, December, 2007.

[22] B. Y. Zhao, L. Huang, J. Stribling, A. D. Joseph, J. D. Kubiatowicz, Exploiting Routing Redundancy via Structured Peer-to-peer Overlays, *IEEE International Conference on Network Protocols*, Atlanta, GA, 2003, pp. 246-257.

[23] B. Y. Zhao, L. Huang, J. Stribling, S. Rhea, A. D. Joseph, J. D. Kubiatowicz, Tapestry: A Resilient Global-scale Overlay for Service Deployment, *IEEE Journal on Selected Areas in Communications*, Vol. 22, No. 1, pp. 41-53, January, 2004.

[24] S. S. Lam, H. Liu, Failure Recovery for Structured P2P Networks: Protocol Design and Performance under Churn, *Elsevier Computer Networks*, Vol. 50, No. 16, pp. 3083-3104, November, 2006.

[25] A. J. Ganesh, A. M. Kermarrec, L. Massoulie, Peer-to-peer Membership Management for Gossip-based Protocols, *IEEE Transactions on Computers*, Vol. 52, No. 2, pp. 139-149, February, 2003.

[26] H. S. Jeon, H. Jung, W. Chun, ID Based Web Browser with P2P Property, *International Conference on Future Generation Communication and Networking*, Jeju Island, South Korea, 2015, pp. 41-44.

[27] A. Alasaad, S. Gopalakrishnan, V. C. M. Leung, A Hybrid Approach for Cost-effective Media streaming Based on Prediction of Demand in Community Networks, *Telecommunication Systems*, Vol. 59, No. 3, pp. 329-343, July, 2015.

[28] K. Birkos, F. Andriopoulou, C. A. Papageorgiou, T. Dagiuklas, S. Kotsopoulos, Enhanced Failover Mechanisms for Tree-based Peer-to-Peer Streaming, *IEEE International Conference on Communications*, London, UK, 2015, pp. 7024-7029.

[29] I.-J. Wang, G. Lee, S.-L. Peng, Y.-C. Chen, Supporting Content-Based Music Retrieval in Structured Peer-to-Peer Overlays, *Journal of Internet Technology*, Vol. 17, No. 3, pp. 401-407, May, 2016.

[30] Y. Q. Gui, H. K. Choi, Time-Driven Mesh Overlay Network for Fully Distributed Peer-to-Peer Video-on-Demand Services, *Journal of Internet Technology*, Vol. 16, No. 5, pp. 789-800, September, 2015.

[31] S. Kamolphiwong, S. Chanpong, T. Kamolphiwong, QoS Aware for Peer Selection on P2P Streaming Services, *Journal of Internet Technology*, Vol. 15, No. 6, pp. 881-891, November, 2014.

[32] M. Zhang, J.-G. Luo, L. Zhao, S.-Q. Yang, A Peer-to-peer Network for Live Media Streaming Using a Push-pull Approach, *ACM International Conference on Multimedia*, Singapore, 2005, pp. 287-290.

[33] A. Ghanbari, H. R. Rabiee, M. Khansari, M. Salehi, Ppm- A Hybrid Push-pull Mesh-based Peer-to-peer Live Video Streaming Protocol, *International Conference on Computer Communications and Networks*, Munich, Germany, 2012, pp. 1-8.

[34] Network Time Protocol, http://www.ntp.org

[35] C. J. Bovy, H. T. Mertodimedjo, G. Hooghiemstra, H. Uijterwaal, P. V. Mieghem, Analysis of End-to-end Delay Measurements in Internet, *Passive and Active Measurement Workshop*, Fort Collins, CO, 2002, pp. 26-33.

[36] Oversim, http://www.oversim.org.

[37] SIGCOMM 2005 Meridian Internet Latency Data Set, http://www.cs.cornell.edu/people/egs/meridian/data.php.

[38] R. Kumar, Y. Liu, K. Ross, Stochastic Fluid Theory for P2P Streaming Systems, *IEEE International Conference on Computer Communications*, Barcelona, Spain, 2007, pp. 919-927.

Biographies

Chih-Chiang Wang Chih-Chiang Wang obtained his Ph.D. degree in Computer Science from North Carolina State University, USA, in 2007. He is currently a faculty member in the Department of Computer Science and Information Engineering at National Kaohsiung University of Applied Sciences in

Taiwan. His research interests are in the general areas of SDN/NFV systems, overlay networks, and routing protocols.

Bi-Wei Zhuang Bi-Wei Zhuang is a master's student in Department of Computer Science and Information Engineering at National Kaohsiung University of Applied Sciences in Taiwan. His research interests include network protocols, peer-to-peer systems, and distributed systems.

Mon-Yen Luo received his Ph.D. degree in Computer Science from the National Sun Yat-Sen University, Taiwan. He is currently an Associate Professor at the Department of Computer Science and Information Engineering in National Kaohsiung University of Applied Sciences, Taiwan. His research interests are in the areas of cloud computing, Internet technology, network/system management, and education technology