Received: 3 March 2016 | Revised: 23 June 2016 | Accepted: 10 August 2016

DOI 10.1002/dac.3193

WILEY

RESEARCH ARTICLE

CDNPatch: a cost-effective failover mechanism for hybrid CDN-P2P live streaming systems

Chih-Chiang Wang[1*] | Ying-Dar Lin[2]

[1]Department of Computer Science, National Kaohsiung University of Applied Sciences, 415 Chien Kung Road Kaohsiung, 807, Taiwan
[2]Department of Computer Science, National Chiao Tung University, No.1001, Ta Hsueh Road Hsinchu, 300, Taiwan

Correspondence
Chih-Chiang Wang, Department of Computer Science, National Kaohsiung University of Applied Sciences, 415 Chien Kung Road, Kaohsiung 807, Taiwan.
Email: ccwang@kuas.edu.tw: steven.cc.wang@gmail.com

Summary

Among the most well-established live media distribution technologies is content delivery network (CDN), which improves user-perceived quality of service by delivering content from proxy servers deployed at the Internet's edge. In recent years, CDN providers started to tap into their subscribers' peer-to-peer (P2P) capacity to alleviate their server costs. Under the inherent peer dynamics, a major challenge of these hybrid CDN-P2P systems is to provide efficient failure recovery with good quality of service guarantees at a reduced server cost. In this work we propose a cost-effective failover solution named *CDNPatch* to address the aforementioned problem. CDNPatch enables peers to periodically precompute a few backup content suppliers by efficient information exchange and maintenance algorithms, and leverages auxiliary CDN servers and an economic server provisioning algorithm to reduce the chance of playback interruption occurring to peers. Our simulation results show that CDNPatch can mask the impact of peer dynamics of 3 real P2P systems, namely, SOPCast, PPStream, and PPTV, with 100% failure recovery success rate and a failure recovery time less than 1 second at a cost of small P2P communication overhead of less than 1 kilobits per second, while using only 10%, 21%, and 51%, respectively, of the pure CDN scheme's server consumption.

KEYWORDS

failure recovery, hybrid CDN-P2P, live streaming, QoS performance

1 | INTRODUCTION

The proliferation of high-speed access networks has made it feasible to stream media content to end users over the Internet. However, it is still technically challenging to distribute live media content to a group of end user subscribers with good quality of service (QoS) performance. Among the most well-established live media distribution technologies is content delivery network (CDN), which improves user-perceived QoS performance by delivering content from *edge servers* (proxy servers) strategically deployed at the Internet's edge. Today, major content providers like National Broadcasting Company, Major League Baseball, and others pay CDN companies to deliver their live media content to their audience of end users. Nevertheless, purely relying on dedicated edge servers, though offering good QoS performance, directly translates into a high operational cost.

In recent years, CDN providers[1,2] started to tap into their subscribers' peer-to-peer (P2P) capacity to alleviate some of their own server costs. In such a hybrid CDN-P2P system design, end user participants, usually called *peers*, typically form 1 or more P2P overlay networks. These peers download most of live media content from the neighboring peers in the overlay, while the CDN edge network is used to prime the P2P networks and supplement the overall system capacity in an on-demand manner. The main challenge of a hybrid CDN-P2P system is to provide good QoS guarantees at a reduced operational cost under the influence of inherent *peer dynamics*—the dynamism of peers joining and subsequently departing from the system. As indicated in literature,[3–5] in any

sort of P2P live streaming network, peer dynamics often cause temporary streaming interruption to the associated downstream peers. The longer the recovery of peers' failed streaming connections, the greater the interruption to their media playback. A natural solution to the aforementioned problem is to equip the hybrid CDN-P2P system with an efficient failure recovery mechanism, whereby any involved peer therein can use the CDN as a failover source for desired content in the event that the given peer cannot locate the content from the neighboring peers.

The goal of this paper is to develop a cost-effective failover solution named *CDNPatch* for hybrid CDN-P2P live streaming systems so that they can easily incorporate our failover solution and hence achieve fast failure recovery at the occurrence of peer dynamics in subject to a service-level agreement. Although similar ideas were sketched in some previous work,[1,2,6–9] to date we are not aware of any existing work that has presented an explicit, complete cost-effective failover mechanism for hybrid CDN-P2P live streaming systems while offering specific QoS guarantees, which motivates our work here. First of all, to speed up the recovery process in hybrid CDN-P2P live streaming, CDNPatch enables peers to periodically precompute a few backup content suppliers by efficient information exchange and maintenance algorithms. CDNPatch reduces the cumbersome task of P2P recovery to only failover to the backups. Furthermore, CDNPatch leverages auxiliary CDN servers and an economic hybrid CDN-P2P service strategy to raise the system's failure recovery success rate and reduce the chance of playback interruption occurring to peers. Finally, we like to emphasize that it is never our intention to invent a whole new hybrid CDN-P2P system from the ground up, nor to renovate, improve any existing CDN or P2P technology. Instead, we limit the scope of this work to merely developing a failover solution upon the basis of today's well-adopted CDN and P2P designs such as[1,10] while minimizing the disruption to their designs.

This work made the following contributions: (1) We have developed a cost-effective failover solution for hybrid CDN-P2P live streaming. Its design includes a server provisioning manager for estimating the amount of CDN server capacity required by the system to meet a specific QoS target, a lightweight protocol for peers to exchange local streaming information, and a neighbor maintenance algorithm to assist peers in precomputing and selecting a sufficient number of backup content suppliers. (2) We have built a system model for our proposal and have analyzed its performance for communication and computation overheads. (3) We have evaluated through simulation the performance of our solution under different system settings.

The rest of this paper is organized as follows: Section 2 surveys the existing work on P2P and CDN live streaming systems and hybrid CDN-P2P designs and relates them to our own. Section 3 introduces the system architecture and model. Section 4 presents the design rationale and the operation of CDNPatch. Section 5 elaborates the implementation of CDNPatch's key mechanisms. Sections 6 and 7 present overhead analysis and simulation-based performance evaluation of CDNPatch under different system settings, respectively. Finally, Section 8 concludes and considers directions for future work.

2 | RELATED WORK

2.1 | P2P live streaming

The early P2P live streaming systems were mostly constructed on a tree-based overlay network in which peers are self-organized into end-system multicast tree(s). At the root of the multicast tree is the media source that distributes a live media stream to peers through the tree structure. The tree-based P2P live streaming systems can be further classified into 2 categories: the single-tree or the multiple-tree.[11] The single P2P multicast tree is considered inefficient because peers on the bottom of the tree can hardly serve other peers. The multiple-tree scheme divides a live media stream into multiple sub-streams, each of which is distributed through an independent multicast tree. Some research effort[8] even proposed an enhanced failover mechanism for P2P streaming over multiple multicast trees to provide backup links during temporary disconnections. Generally speaking, the multiple-tree scheme outperforms the single-tree scheme of load balancing and fault resilience, but both of them suffer performance degradation and expensive maintenance overhead in presence of high peer dynamics.

Different from the tree-based schemes, the so-called mesh-based or data-driven scheme[10] employs swarm-like content delivery, whereby peers randomly connect to one another and pull (fetch) content from any neighboring peers in the overlay, to achieve robust live media delivery over the Internet. The core design that contributes to its robustness is to have peers gossip with one another and locate potential content suppliers based on gossip information stored in their membership cache (mCache). The mesh-based scheme is generally considered more fault-resilient and load-balanced than the tree-based schemes, and its feasibility over the Internet has been validated by several large-scale P2P streaming experiments[10] and real-system implementations.[12,13] For these reasons, this work is built on the design of mesh-based P2P live streaming. In general, a main drawback of pure P2P systems is that their dynamic nature makes it difficult to provide QoS performance guarantees.

2.2 | Content delivery networks

Content delivery networks emerged in the late 1990s when Akamai[14] launched the first large-scale commercial CDN infrastructure to solve the flash crowd problem, in which a sudden surge of user requests invoke large content trans-

missions over long distances and overwhelm the Web site's infrastructure. A CDN is a collection of autonomous computers linked by the Internet and other networks, together with the systems, software, protocols, and mechanisms designed to facilitate delivery of Web content, streaming media, etc. Most of the computers in a CDN are proxy servers, usually called edge servers, placed strategically near end user access networks. Third-party content providers can off-load content delivery to a CDN by aliasing their domains to the domains that are managed by the CDN's authoritative domain name service. The mapping of user requests to edge servers is performed by a simple domain name system (DNS) redirection process. In specifics, Akamai maps each Internet Protocol (IP) block to a certain server region; when a user request triggers a DNS query to the local DNS server, the query is forwarded to Akamai's DNS server, which in turn redirects the user request to some edge server with the requested content in the designated region. The CDN benefits include reduced workload of the origin server and reduced access latency experienced by end users.

With the increasing demands on streaming media in the Internet, now, major CDN providers also incorporate streaming technologies into their server infrastructure. Media content can be streamed either live or on demand—the former distributes media content recorded from live events to end users on the fly, whereas the latter distributes pre-recorded media content from some storage to end users on demand. For on-demand media delivery, a content provider first uploads content into a CDN's storage facility from which the stored content will be replicated to many data centers; an edge server that receives a stream request downloads the content from its optimal data-center location while serving the request. For live media delivery, the entry-point edge server, while receiving a live stream from a content provider, is simultaneously uploading that stream to a set of edge servers that in turn serve the received live stream to end user subscribers. In either case, all the involved edge servers must deliver media content without significant delay or jitter so that end users barely detect interruption in the stream. Media content like a movie requires tens of megabytes to several gigabytes of transmission and storage overheads depending on its quality and duration, so purely relying on dedicated edge servers to serve streaming media results in a high operational cost.

2.3 | Hybrid CDN-P2P service architecture

Several research efforts attempted to develop hybrid CDN-P2P service architecture to support low-cost, QoS-guaranteed live media distribution service. One of the earliest milestones was set by Chen et al.,[15] which proposed a peer-server-peer architecture whereby live media content is distributed through an overlay of dedicated servers to different regions of a P2P network. The work of Xu et al.[16] proposed another hybrid architecture in which the media distribution process evolves from CDN-only at the beginning, to hybrid CDN-P2P, and finally to P2P-only when the content has enough replicas in the P2P swarm. Authors Alasaad, Gopalakrishnan, and Leung,[17] leveraged underutilized resources in a community network to form caches of files that are desired in that community and hence reduced the operational cost of the CDNs. The work of Kao and Lee[18] proposed a set of proxy-assisted transmission schemes for efficiently streaming a set of layered videos from a remote server via the proxy server at the base station to multiple heterogeneous and asynchronous clients in wireless networks. The work of Zhang et al.[7] conducted a comprehensive measurement study on Kankan, a leading video-on-demand (VoD) streaming service provider in China that is based on a hybrid CDN-P2P architecture. Their study showed that Kankan adopts a dual-server mechanism to enhance start-up streaming and provides the CDN acceleration in case of inefficient P2P streaming performance. Authors Esposito and Cerroni[19] proposed a content distribution protocol for hybrid CDN-P2P systems. Their work leveraged Flajolet-Martin sketches to estimate the number of distinct pieces in the CDN and adopted a fractional knapsack-problem approach to use upload capacity of peers. The work of P2PWeb[20] proposed a hybrid protocol associating the client/server and P2P architectures for IPTV delivery and optimized the provision of the rarest chunk to improve the QoS. Authors Li et al.[21] examined the relationship between the Internet infrastructure and the video delivery throughput of a large-scale VoD system. They found that crossing the Internet service provider (ISP) or regional network border yields 15% to 20% speed loss, and based on this observation, they evaluated the potentials of edge caching and hybrid CDN-P2P in the improvement of VoD streaming performance.

In recent years, CDN providers started to tap into their subscribers' P2P capacity to alleviate some of their own server costs. For instance, Akamai acquired RedSwoosh P2P distribution technology in year 2007 and later in year 2012 obtained a US patent on its hybrid CDN-P2P architecture design.[1] In such a hybrid CDN-P2P system, end user participants typically form 1 or more P2P overlay networks and download most of live media content from the neighboring peers in the overlay, while the CDN edge network is used to prime the P2P networks and serve as a failover media source in the event the end users cannot locate the content from the neighboring peers. Nevertheless, the work of Michael, Thomson, and Jay[1] merely sketched a general idea about its invention and did not present an explicit failover mechanism to cope with peer dynamics. Another industrial effort was published in,[2] presenting the design and deployment experiences with LiveSky, a commercially deployed hybrid CDN-P2P live streaming system, but regarding the issue of system instability under peer dynamics, it gave only 1 general statement that peers who suffer performance degradation because of peer dynamics can retrieve data directly from the edge server until the peers can find suitable content suppliers from the P2P network.

The work of Lu et al.[6] proposed a live streaming service scheme through coordinating CDN and P2P, which allows some CDN servers, termed *CDN strong nodes*, to decide whether to serve content requests directly or to redirect the requests to other peers known as *P2P strong nodes*. However, it is unclear how the design of Lu et al.[6] precomputes and allocates a sufficient amount of strong-node capacity in advance, be it derived from the CDN or from the P2P network, to ensuring that in presence of the peer dynamics the hybrid CDN-P2P system can meet specific QoS guarantees at the minimum server cost.

Some research efforts were dedicated to improving hybrid CDN-P2P streaming by coding techniques or by energy-efficient mechanisms. Authors Lin and Lee[22] discussed the adaptation of multiple description coding to reduce the amount of bandwidth consumption of proxy servers in CDN-P2P VoD systems. The work of Zhou, Xu, and Zhang[23] modeled and evaluated the strength and weakness of a family of coding strategies for CDN-based VoD streaming systems. It showed that the system designers can take advantage of the proposed model to balance the trade-off between coding gain and coding overheads. Authors Milani and Calvagno[24] presented a distributed packet classification strategy for multiple description coded video streams that is based on a non-cooperative game. The proposed classification strategy was shown to be an efficient solution to support a diversity of loss patterns of P2P or CDN multimedia streaming. Authors Mandal et al.[25] analyzed the energy consumption of a hybrid CDN-P2P system in an IP-over-WDM network. They also proposed energy-aware and energy-unaware content source selection mechanisms to exploit P2P capacity and reduce the load and/or the energy consumption of the CDN while meeting a same timing constraint. The work of Lawey, El-Gorashi, and Elmirghani[26] investigated different aspects related to BitTorrent's energy efficiency in IP-over-Wavelength Division Multiplexing (WDM) networks, validating the power savings obtained by modeling and simulation through experimental results.

Some recent research efforts aimed to improve bandwidth allocation, scheduling, and other aspects of hybrid CDN-P2P streaming systems. Authors Wu et al.[9] studied systematic strategies for allocating CDN bandwidth under the influence of P2P factors and a fixed budget constraint. They showed that a more optimal strategy would bias toward allocating proportionally more CDN bandwidth to smaller ISP networks. The work of Meskovic, Kos, and Meskovic[27] proposed a chunk scheduling algorithm and a knapsack-problem model to address the performance optimization problem of hybrid CDN-P2P live streaming. The proposed scheduling algorithm uses a Taboo search method to support prioritized chunk delivery while optimizing bandwidth utilization by identifying and disposing of "useless available chunks" based on chunks' playback deadline. The work of Jeon, Jung, and Chun[28] proposed an ID-based Web Browser with ISP-friendly P2P content delivery property. Authors Nam et al.[29] considered a server farm CDN that uses P2P networking to connect its edge servers and studied how the delay-cost trade-off in the CDN's server deployment can be improved by the parameter tuning of P2P networking. Authors Sales et al.[30] showed that Global Media Transmission Protocol can reduce the start-up streaming delay in mobile devices and scale the number of end users in a streaming system without increasing the bandwidth consumption. They also discussed an incentive technique for encouraging users to share their network resources.

As aforementioned, in hybrid CDN-P2P live streaming systems, peer dynamics can lead to temporary streaming interruption to the associated downstream peers and degrade the QoS performance of these systems. Although the previous research efforts addressed a broad range of issues related to the hybrid CDN-P2P design, none of them presented a specific failover mechanism with relevant server provisioning and maintenance algorithms to quickly and efficiently cope with the transient effect of peer dynamics while offering QoS guarantees, which motivates our work here.

3 | SYSTEM ARCHITECTURE AND MODEL

In this section we introduce the system architecture and model of our proposal. Table 1 lists the key notations used through this paper. Our hybrid CDN-P2P system model is built upon the basis of 2 well-adopted designs: Akamai's CDN design[1] and mesh-based P2P live streaming,[10] and its architecture is shown in Figure 1. In Figure 1, the label "Edge Server (Source)" denotes an edge server that serves as a media source to prime a regional P2P network, while the label "Edge Server (Auxiliary)" denotes an edge server that serves as an auxiliary failover source to a P2P network. As illustrated in the figure, the default entry-point edge server, while receiving a live media stream from the content provider, is simultaneously uploading that stream via the CDN edge network to a set of edge servers, and the edge servers in turn serve the received stream to end user subscribers. In a conventional CDN model, an edge server acts only as a proxy server for legacy clients.

TABLE 1 Key mathematical notations

Symbol	Meaning
m	number of allowed neighbors per peer
K	number of the live media sub-streams
r	media streaming rate
$v_{x,t}$	an active peer v_x at time t
$\mu_{x,t}$	$v_{x,t}$'s free upload capacity
$\mathcal{N}_{x,t}$	neighbors of $v_{x,t}$
$\mathcal{P}_{x,t}$, $\mathcal{C}_{x,t}$	parents/children of $v_{x,t}$, respectively
$\hat{\mathcal{P}}_{x,t}$	backup parents of $v_{x,t}$
$d_{x,t}$	the delay from the time the media was sent by the edge server to the time it is played by $v_{x,t}$
$r_{xy,t}$	rate of the media stream from $v_{x,t}$ to $v_{y,t}$
$r_{x,t}$	rate of the media stream received by $v_{x,t}$
τ_m	length of CDNPatch's maintenance period

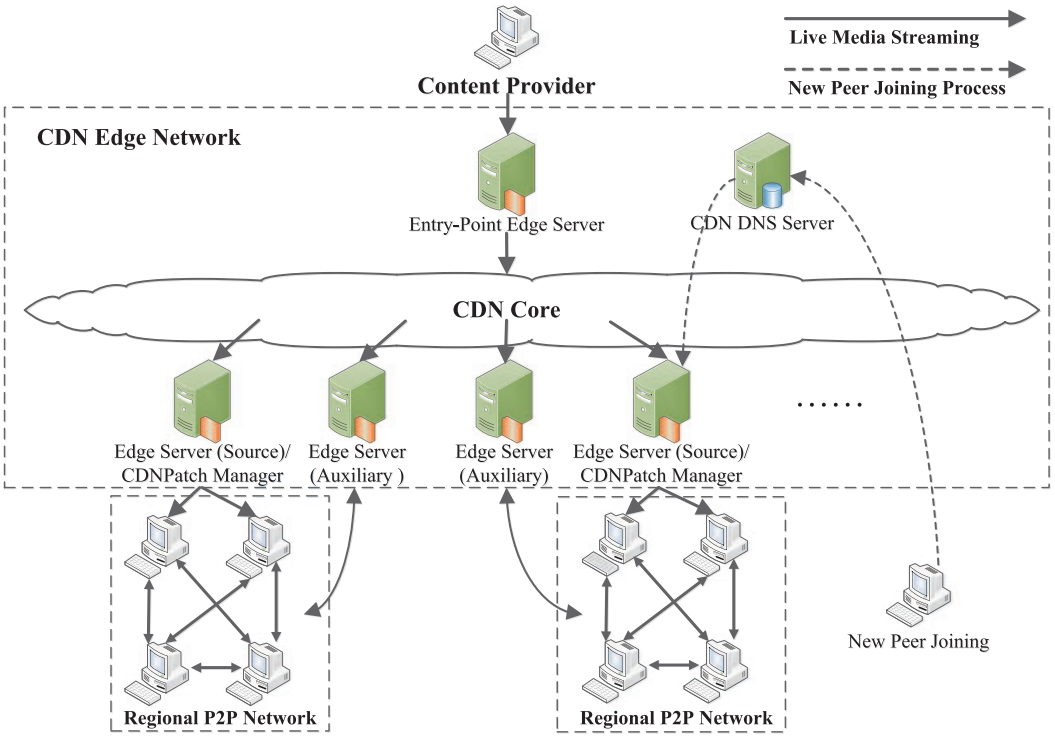

FIGURE 1 System architecture. CDN, content delivery network; P2P, peer-to-peer

However, in our hybrid CDN-P2P model, an edge server can serve as a media source to prime a regional P2P network or as an auxiliary failover source to be used by the P2P network in the event that the peers therein cannot locate the content from the neighboring peers. The former is required to host a CDNPatch tracker/manager for (1) assisting new peers in the joining process and (2) provisioning the minimally required amount of auxiliary CDN server capacity for its regional P2P network with respect to a service-level agreement. Note that if a CDNPatch manager alone fails, its host will simply restart the CDNPatch manager and everything will be back to normal because of its soft-state maintenance mechanism. However, if the host of the CDNPatch manager also fails, all the involved peers will contact the CDN's DNS server and then be redirected to another nearest edge server that is arranged to server as their new CDNPatch manager.

A newly created peer joins the hybrid CDN-P2P live streaming of our system by first contacting the CDN's DNS server, which in turn redirects the user request to some nearest edge server. The edge server then decides whether to serve the new peer directly or redirect the new peer to other active peers in its regional P2P network. In the latter scenario, the new peer obtains from the edge server a list of active regional peers as neighbor candidates and uses the peer list to establish *m neighbors* in the P2P overlay. We use the term *neighbors* to refer to peers with a direct overlay connection between themselves. Note that neighbors of a peer are only potential media content suppliers because some neighbors just do not have the requested media content or do not have enough upload bandwidth to serve the requested media content. Peers also maintain in their mCache a subset of active regional peers by an underlying gossip protocol and contact them to maintain m neighbors in the P2P overlay when needed. In case that a peer and its neighbor barely exchange content or simply that its neighbor has failed, the peer will replace this inactive or failed neighbor with someone recorded in its mCache.

In the the remainder of the paper, we simply focus on the system model of a single regional P2P network and its associated CDN edge servers because all the regional P2P networks in the system operate in the same manner independently from one another. For a specific regional P2P network in the system, there must be 1 associated edge server acting as the media source distributing the live media stream through this P2P network. The media source divides the live media stream into a sequence of video blocks of uniform length and organizes them into K sub-streams so that the i-th sub-stream contains blocks with sequence number $(i + j{\cdot}K)$, $i \in \{1,2,\ldots,K\}$, and $j \in \mathbb{Z}_{\geq 0}$. The required streaming rate is r kilobits per second (kbps). This regional P2P network at a specific time t resembles an undirected graph $G_t = (V_t,E_t)$, where V_t is the set of active peers in the system at time t and E_t is the set of overlay connections among peers in V_t.

Video blocks received by a peer are first stored into a synchronization buffer and then combined into 1 continuous media stream. The resultant media stream is then fed into a playout buffer from which the peer plays the media stream. Peers store in *buffermaps* the availability information of the latest video blocks in their buffers and exchange buffermaps with their neighbors every second. Based on the received buffermaps, peers subscribe to a sub-stream by sending a

push-pull request to 1 of their neighbors and, if necessary, pull from their neighbors the missing blocks with the closest playback time first. If the push-pull request has been granted, the requested neighbor and the requesting peer will form a *parent-child* relationship so that the parent will continue pushing all blocks in need of that sub-stream to its child[‡]. The parent re-selection strategy of Li et al.[10] is adopted such that each peer compares the buffermap status of its current parents with that of its neighbors and replaces the parent whose buffermap status is lagging behind that of the neighbor or is insufficient for timely delivery of the associated sub-stream. At each peer, CDNPatch maintenance process is regularly invoked every τ_m seconds to maintain a sufficient number of parents and backup parents and to send its demand for auxiliary server capacity to the CDNPatch manager. When a peer's parent has failed or is inadequate in serving the requested sub-stream, the peer will try to fail over to a backup. Its last resort is to fetch the sub-stream directly from its designated auxiliary CDN server. For ease of explanation, we use $\mathcal{N}_{x,t}$, $\mathcal{P}_{x,t}$, $\mathcal{C}_{x,t}$, and $\hat{\mathcal{P}}_{x,t}$ to denote the neighbors, the parents, the children, and the backup parents of a peer $v_{x,t}$, $v_{x,t} \in V_t$. Moreover, we use $\mu_{x,t}$ to denote $v_{x,t}$'s free upload capacity and use $r_{xy,t}$ and $r_{x,t}$ 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.

There already exists some work[33] on measuring the available bandwidth of residential Internet access and some[34,35] on provisioning dedicated server capacity to assist the P2P system in achieving a streaming rate higher than the average uplink capacity of peers. To avoid repeating the work of others, we make the following simplifications in our model. First, we assume that r is less than the average peer uplink capacity. Second, we assume that a peer can estimate its own free upload bandwidth by using available bandwidth estimation tools such as Spruce.[36] Finally, to speed up the search for backup parents in the mCache, our design requires the media source and peers to maintain synchronized clocks by some technology like Network Time Protocol[37]; and when the media source is about to send out a video block, it will record the current time in the block's header as its *origin time*. Thus, any peer $v_{x,t}$ can evaluate its *PlaybackDelay*, denoted by $d_{x,t}$, by subtracting the origin time of its received block from the time the block is scheduled to be played at $v_{x,t}$. Note that after a peer starts playing the live media stream, its PlaybackDelay approximates a constant value.

4 | DESIGN OF CDNPATCH

This section elaborates the design of CDNPatch, including its server provisioning, protocol, and maintenance operation.

[‡]Peer-to-peer (P2P) literature[10,31,32] shows that this hybrid push-pull method offers a good performance trade-off between streaming throughput and fault resilience.

4.1 | Design rationale

CDNPatch implements a simple but effective concept—achieve fast recovery of system performance by failover to the backups. Modern mesh-based P2P design, as described in Section 3, divides a live media stream into K sub-streams and allows peers to retrieve the sub-streams from different parents by the push-pull method. The objective of CDNPatch is to let every peer run its protocol and maintenance operation to periodically maintain K parents in the neighbor set plus a few backup parents for fast failover, so the peer's streaming rate is highly likely to be maintained at r kbps until the next maintenance operation. The auxiliary CDN server is used as the last recovery resort to meet the CDNPatch's service-level agreement.

4.1.1 | Service-level agreement

The core of CDNPatch maintenance operation is to select *qualified* backup parents for peers. The first qualification for backup-parent selection is that if a peer $v_{y,t}$ is qualified to serve as a parent of $v_{x,t}$, then $v_{y,t}$ must have adequate free upload capacity to push a sub-stream to $v_{x,t}$. The second qualification is that $v_{x,t}$ must be able to form a parent-child relationship with $v_{y,t}$ via a push-pull request as described in Section 3, which yields an approximate timing constraint between their PlaybackDelay $d_{x,t}$ and $d_{y,t}$ as

$$d_{x,t} - B \le d_{y,t} + 3\delta_{\mathrm{xy},t} + \frac{|BM| + |block|}{\mu_y} \le d_{x,t} \ \forall v_{y,t} \in \hat{\mathcal{P}}_{x,t},$$

where B is the playback buffer size in unit of seconds, $\delta_{xy,t}$ is the end-to-end delay between $v_{x,t}$ and $v_{y,t}$ in unit of seconds, μ_y 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 kilobits, respectively. Because the end-to-end delay is between peers within a region of nearby end user access networks, the transmission latency of a buffermap or of a video block is mostly bounded by 0.2 seconds[10]; the second qualification can be approximated, for use in practice, as

$$d_{x,t} - B - 1 \le d_{y,t} \le d_{x,t} - 1, \ \forall v_{y,t} \in \hat{\mathcal{P}}_{x,t}.$$

The following gives a formal definition on CDNPatch's service-level agreement based on the aforementioned design rationale and backup-selection constraints:

Given a mesh-based P2P system $G_t = (V_t, E_t)$, if CDNPatch maintenance operation completes at some peer $v_{x,t}$, $v_{x,t} \in V_t$, then (1) $v_{x,t}$ should have K parents to sustain the required streaming rate r, be them ordinary peers or virtual parents acted by some auxiliary CDN server, and (2) with a high probability greater than $(1 - \epsilon)$, where the value of ϵ represents the *playback discontinuity probability*, $v_{x,t}$ should have sufficient backup parents or CDN server capacity in reserve for fast failover. This service-level agreement can be expressed in 4 inequalities as

$$\mu_{y,t} \geq \frac{r}{K},\ \forall v_{y,t} \in \hat{\mathcal{P}}_{x,t}, \tag{1}$$

$$d_{x,t} - B - 1 \leq d_{y,t} \leq d_{x,t} - 1,\ \forall v_{y,t} \in \hat{\mathcal{P}}_{x,t}, \tag{2}$$

$$\Pr(|\mathcal{P}_{x,t+\Delta t}| + |\hat{\mathcal{P}}_{x,t+\Delta t}| < K \mid \hat{\mathcal{P}}_{x,t}) \leq \epsilon,\ \forall \Delta t \in [0, \tau_m], \tag{3}$$

$$r_{x,t} \geq r,\ |\mathcal{P}_{x,t}| = K. \tag{4}$$

4.1.2 | Operation of CDNPatch

According to its service-level agreement, CDNPatch maintenance process should be regularly invoked at every peer to perform 4 tasks as illustrated in Figure 2: (1) *information dissemination*: periodically disseminate streaming-status information to neighbors and a subset of active peers in the system by CDNPatch protocol and the underlying gossip protocol; (2) *backup maintenance*: from the received streaming-status information, a peer, say $v_{x,t}$, should pick a minimum number of qualified backup parents in subject to constraint (1), (2), and (3); (3) *CDN server provisioning*: if the number of available backup parents falls short of what is needed, $v_{x,t}$ should send a backup-demand message to the CDNPatch manager to be able to obtain some auxiliary CDN server v_a, while the CDNPatch manager adjusts the CDN server provisioning according to the received backup-demand information; (4) *failover recovery*: $v_{x,t}$ rapidly fails over to ordinary backup parents if violation of constraint (4) occurs, and if necessary, using v_a as the last recovery resort. The remainder of this section will elaborate how CDNPatch is designed to perform these tasks.

4.2 | CDNPatch messages

We define 5 types of CDNPatch messages: *neighbor-status*, *backup-status*, *backup-request*, *backup-demand*, and *backup-reply*, for peers to report their streaming-status information to the regional CDNPatch manager or to other peers, and vise versa. Figure 3 illustrates the structures of CDNPatch messages. Neighboring peers are required to exchange neighbor-status messages, each of which contains the 3-tuple information: *PlaybackDelay*, *FreeCapacity*, and *Substitute*. Peer $v_{x,t}$ can request a backup-status message from another peer $v_{y,t}$ by sending $v_{y,t}$ a backup-request message. Backup-status messages are also piggybacked on the membership messages for gossip propagation. A backup-status message has 3 fields: *PlaybackDelay*, *FreeCapacity*, and *StreamRate*. Finally, a backup-demand message is used by any arbitrary peer $v_{x,t}$ to inform the CDNPatch manager of the number of backup parents that are needed by $v_{x,t}$, whereas the CDNPatch manager replies with a backup-reply message that contains a list of auxiliary CDN edge servers to be used as backups. The fields of CDNPatch messages are explained below:

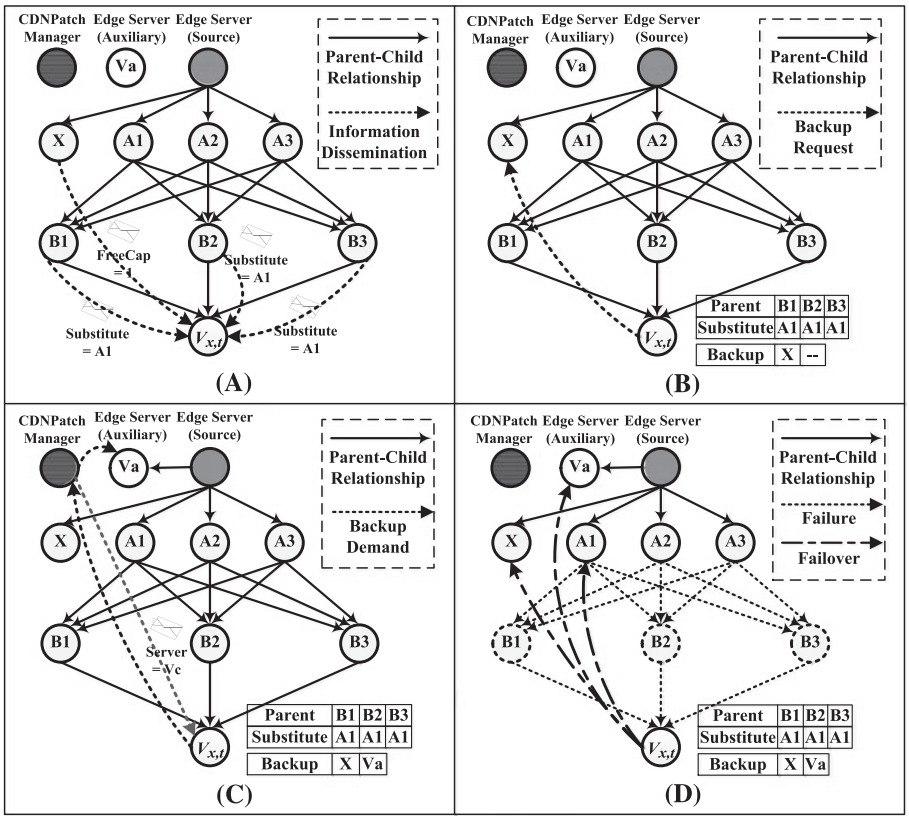

FIGURE 2 Illustration of the overall operation of CDNPatch: (A) information dissemination; (B) backup maintenance; (C) CDN server provisioning; and (D) failover recovery. See Section 4.2 for the definition of *Substitute* and the difference between a *Substitute* and a backup parent. CDN, content delivery network

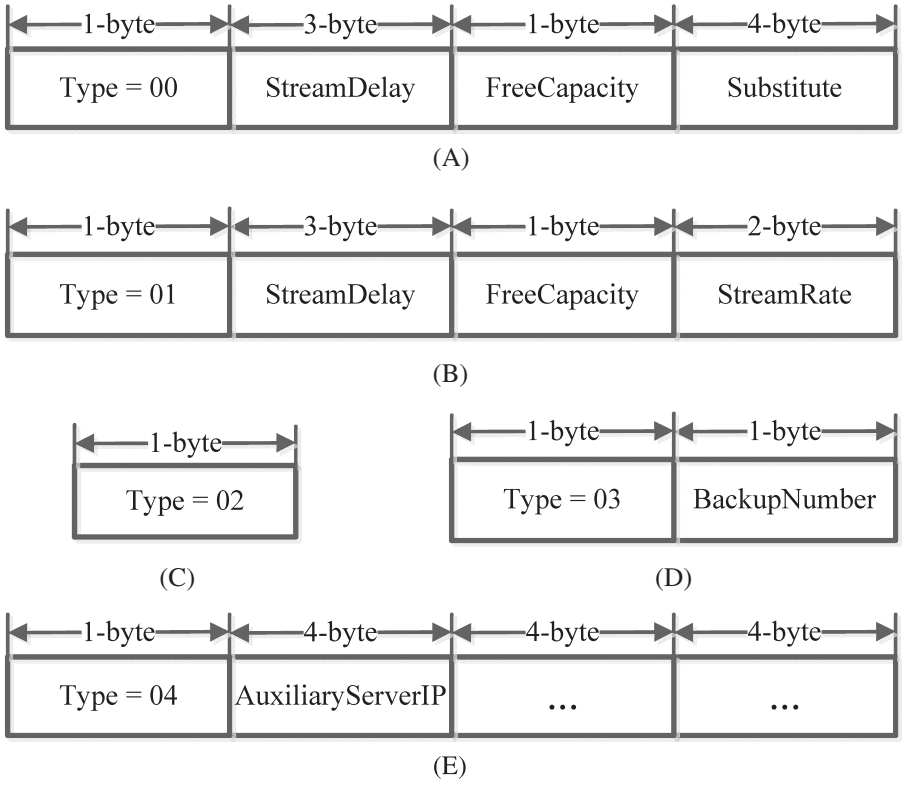

FIGURE 3 CDNPatch messages: structure of (A) a neighbor-status message, (B) a backup-status message, (C) a backup-request message, (D) a backup-demand message, and (E) a backup-reply message

1. *PlaybackDelay* of $v_{x,t}$ records $d_{x,t}$ in milliseconds.
2. *StreamRate* of $v_{x,t}$ is computed as the total size of the video blocks $v_{x,t}$ has played during the last maintenance period divided by τ_m.
3. *FreeCapacity* of $v_{x,t}$ records the maximum number of children that $v_{x,t}$ currently can support.
4. *Substitute* sent from $v_{x,t}$ to its child $v_{y,t}$ records a non-child neighbor of $v_{x,t}$ that is used as a recommended substitute of $v_{x,t}$ for $v_{y,t}$ in case that $v_{x,t}$ suddenly fails. Note that there is a significant difference between a *Substitute* and a backup parent—a backup parent can be used to replace any failed parent, but a *Substitute* can only be used to replace its recommender when the recommender fails. The assignment of *Substitute* will be addressed in Section 5.
5. *BackupNumber* of $v_{x,t}$ records the number of backup parents that are needed by $v_{x,t}$.
6. *AuxiliaryServerIP* records the IP address of a recommended auxiliary CDN server to be used as a backup parent.

4.3 | CDNPatch manager

A CDNPatch manager is responsible for provisioning of auxiliary CDN servers in subject to the service-level agreement. However, such resource provisioning must be built upon some knowledge about the peer dynamics. We simply assume that by employing some measurement mechanism such as[38–40§], a CDNPatch manager can estimate the basic *average peer lifetime* information of its regional P2P network, which is the average time interval from the moment a peer joins the P2P network to the moment it subsequently crashes or departs from the P2P network.

Given that peer lifetimes follow a general cumulative distribution function (CDF) $L(u)$ with an average value of L, then a CDNPatch manager can estimate the value of K', the largest number of the survivals out of K parents after a period of τ_m seconds on condition that the associated survival probability must exceed $(1-\epsilon)$. In other words, K' is the largest integer such that $F(K'-1;K,S(\tau_m)) \leq \epsilon$, where $F(k;n,p) = \sum_{i=0}^{k} \binom{n}{i} p^i(1-p)^i$ denotes a general binomial CDF and $S(u) = 1-L(u)$ is the minimum survival probability of a parent after a period of u seconds. As a result, the CDNPatch manager needs to provision at most a fraction $(1-\frac{K'}{K})$ of r-kbps auxiliary CDN server capacity for each active peer in the system.

The server provisioning method and the associated backup-demand handler method used by the CDNPatch manager are shown in Figure 4. When a CDNPatch manager receives a backup-demand message (D_MSG), it will process this message with the backup-demand handler method. If the received message is not a duplicate (lines 1 to 2),

§Here, we do not discuss the actual techniques to monitor and measure the peer-churn statistics because these issues are out of the scope of this paper.

Demand_Handler(D_MSG, τ_m, K, ϵ):
1. v_i = *GetSource*(D_MSG)
2. If $v_i \notin V_t$ do
3. BN = *GetBackupNumber*(D_MSG)
4. *Add*(V_t, v_i)
5. *Add*($BNList$, (v_i, BN))
6. $K' = F^{-1}(\epsilon; K, S(\tau_m))$
7. $CAP = CAP + \min(\frac{BN}{K}, 1 - \frac{K'}{K}) \cdot r$
8. End-If

Resource_Provision(τ_m):
1. $ServerList$ = *ProvisionCDNServer*(τ_m, CAP)
2. For each $v_i \in V_t$ do
3. BN = *GetBackupNumber*($BNList$)
4. *UDPSendDemandReply*(v_i, BN, $ServerList$)
5. End-For
6. *Reset*(V_t, CAP)

FIGURE 4 Pseudo code of the CDNPatch manager's server provisioning method

the CDNPatch manager will retrieve from the message the *BackupNumber* (*BN*) information, which is the number of backup parents demanded by the sender (line 3). An amount of $(\min(\frac{BN}{K}, 1 - \frac{K'}{K}) \cdot r)$-kbps auxiliary CDN capacity should be reserved for this peer and hence is added to the count variable *CAP* (line 7). The server provisioning method is invoked regularly by a CDNPatch manager every τ_m seconds. This method begins by provisioning a list of auxiliary CDN servers (*ServerList*) according to the computation result stored in *CAP* (line 1). To avoid disrupting the original CDN operation, we assume that the CDNPatch manager can accomplish the provisioning of auxiliary CDN servers via a function call ProvisionCDNServer supported by the underlying CDN infrastructure. For each received backup-demand message with an associated *BN* value, the CDNPatch manager replies with a backup-reply message that contains a list of *BN* auxiliary CDN servers uniformly selected from *ServerList* with respect to their capacity (lines 2 to 5). Finally, the values of *CAP* and V_t are reset (line 6).

4.4 | CDNPatch maintenance operation

Peers perform CDNPatch maintenance operation by executing 2 parallel processes, namely, *information dissemination* and *CDNPatch maintenance*, every τ_m seconds. The *information dissemination* process periodically sends streaming-status information in CDNPatch messages to other peers, whereas the *CDNPatch maintenance* process periodically screens the received CDNPatch messages and maintains a sufficient number of parents and backup parents. When violation of constraint (4) occurs, *CDNPatch maintenance* will replace failed or unqualified parents with backup parents or auxiliary CDN servers until the violation is resolved.

Figure 5 shows the pseudo code of *information dissemination*. This process first sends neighbor-status and backup-status messages to the corresponding recipients, $\mathcal{N}_{x,t}$ and $\hat{\mathcal{C}}_{x,t}$, respectively (lines 1 to 5). The set $\hat{\mathcal{C}}_{x,t}$ records the senders of the received backup-request messages (line 4).

Info_Propagation(peer $v_{x,t}$):
1. NS_MSG ← *CreateNeighborMSG*($v_{x,t}$)
2. BS_MSG ← *CreateBackupMSG*($v_{x,t}$)
3. *UDPSend*(NS_MSG, $\mathcal{N}_{x,t}$)
4. $\hat{\mathcal{C}}_{x,t}$ ← received BACKUP_REQUEST
5. *UDPSend*(BS_MSG, $\hat{\mathcal{C}}_{x,t}$)
6. *Gossip*(GOSSIP_MSG+BS_MSG)

FIGURE 5 Pseudo code of *information dissemination* process

Backup_Maintenance(peer $v_{x,t}$):
1. For each *Substitute*$_i$ ∈ received NS_MSG do
2. $\mathcal{P}_{x,t} \leftarrow$ *Substitute*$_i$
3. End-For
4. While (Constraint(3) fails) do
5. If (noBS_MSG()) do
6. D_MSG ← *CreateBackupDemandMSG*($v_{x,t}$)
7. *UDPSend*(D_MSG, CDNPatchManager)
8. Break
9. End-If
10. v_i = getBS_MSG()
11. If (v_i meets Constraint(1) and (2))
and ($v_i \notin (\mathcal{N}_{x,t} \cup \hat{\mathcal{P}}_{x,t} \cup$ *Substitute*))
12. If (any *Parent*$_i$ is an auxiliary CDN server)
13. *ReplaceParent*(*Parent*$_i$, v_i)
14. Else
15. *Add*($\hat{\mathcal{P}}_{x,t}$, v_i)
16. *BackupRequest*(v_i)
17. End-While
18. MCACHE ← the rest BS_MSG
19. *SortRate*($\mathcal{N}_{x,t} \setminus \mathcal{P}_{x,t}$)

Failover_Recovery(peer $v_{x,t}$):
1. For each *Parent*$_i \in \mathcal{P}_{x,t}$ do
2. If (*Parent*$_i$ fails or $r_{ix,t} < \frac{r}{K}$)
3. If (*Parent*$_i$ fails and *Substitute*$_i$ exists)
4. *ReplaceParent*(*Parent*$_i$, *Substitute*$_i$)
5. Else if ($|\hat{\mathcal{P}}_{x,t}| > 0$) *ReplaceParent*(*Parent*$_i$, $\hat{\mathcal{P}}_{x,t}$)
6. Else if (*MoreQualified*($\mathcal{N}_{x,t} \setminus \mathcal{P}_{x,t}$))
7. *ReplaceParent*(*Parent*$_i$, $\mathcal{N}_{x,t} \setminus \mathcal{P}_{x,t}$)
8. Else if (*MoreQualified*(MCACHE))
9. *ReplaceParent*(*Parent*$_i$, MCACHE)
10. Else
11. v_c = pickServerBR_MSG()
12. If (v_c) *ReplaceParent*(*Parent*$_i$, v_a)
13. End-While
14. If ($|\mathcal{N}_{x,t}| < m$) *FillNeighbor*($\mathcal{N}_{x,t}$, MCACHE)

FIGURE 6 Pseudo code of *CDNPatch maintenance* process

This process also piggybacks the backup-status messages on the gossip messages (GOSSIP_MSG) that will be disseminated to other peers by the underlying gossip protocol (line 6). If anything alters the content of CDNPatch messages, new CDNPatch messages will be created and sent to the corresponding recipients.

The *CDNPatch maintenance* process has 2 methods: *backup maintenance* and *failover recovery*, and the pseudo code is shown in Figure 6. When *backup maintenance* begins at peer $v_{x,t}$, it first stores all the received *Substitute* information (lines 1 to 3), then screens the received backup-status messages and inserts those that meet constraints (1) and (2) into the backup-parent set $\hat{\mathcal{P}}_{x,t}$ until constraint (3) is satisfied (lines 4 to 17). If the number of available backup parents falls short of what is needed, a backup-demand message will be created and sent to the CDNPatch manager (lines 5 to 9). In case that some *Parent*$_i \in \mathcal{P}_{x,t}$ is a virtual parent acted by some auxil-

iary CDN server, this method will try to replace $Parent_i$ with a qualified backup parent v_i that has just been found (lines 12 to 13). The rest backup-status messages are stored in the mCache (line 18). To speed up the search in the neighbor set, those in $(\mathcal{N}_{x,t} \setminus \mathcal{P}_{x,t})$ that meet constraints (1) and (2) are sorted in the descending order of their *StreamRate* (line 19).

The *failover recovery* method is executed every second at the buffermap transmission timeout to check if parents are still alive and to count the number of video blocks that were received from each parent during the last second (lines 1 to 2). If constraint (4) is not met, this method will call the *Replace-Parent* function to replace a failed or unqualified parent with the corresponding *Substitute* (lines 3 to 4), or a backup parent (line 5), or a neighbor (lines 6 to 7), or a mCache peer (lines 8 to 9) that is qualified to serve the sub-stream. The last resort is to replace a failed or unqualified parent with some auxiliary CDN server v_a obtained from the backup-reply message (lines 10 to 12). At last, this method will fill the neighbor set with peers randomly selected from the mCache to retain the random-mesh property (line 14). The computation speed of this method is fast because of the preprocessing of received CDNPatch messages. See Section 6 for detailed analysis.

5 | IMPLEMENTATIONS OF CDNPATCH

The implementations of CDNPatch are straightforward as described in the previous sections except for 2 parts: the *Substitute* assignment mentioned in Section 4.2, and evaluation of streaming quality constraints mentioned in Section 4.4. This section describes their implementations.

5.1 | Substitute assignment

Sudden failure of any arbitrary peer will cause all of its children to simultaneously lose a parent. An effective approach to facilitate P2P failover recovery from sudden peer failures is to let every peer send to its children a recommended *Substitute* of itself. By doing so, the child can use this *Substitute* as a backup-parent option. We formulate the assignment of *Substitute* at an arbitrary peer, say $v_{x,t}$, as the following optimization problem with the objective of maximally matching the free upload capacity of $(\mathcal{N}_{x,t} \setminus \mathcal{C}_{x,t})$ to the upload demand of $\mathcal{C}_{x,t}$, assuming that $v_{x,t}$ departed from the system.

***Substitute* assignment problem:** Given a peer $v_{x,t}$, the weight function $W : \mathcal{C}_{x,t} \times (\mathcal{N}_{x,t} \setminus \mathcal{C}_{x,t}) \to \mathbb{R}$ defines the maximum amount of free upload capacity that $(\mathcal{N}_{x,t} \setminus \mathcal{C}_{x,t})$ can provide to meet the demand of $\mathcal{C}_{x,t}$ in case that $v_{x,t}$ suddenly fails. The objective is to find a bijection function $F : \mathcal{C}_{x,t} \to (\mathcal{N}_{x,t} \setminus \mathcal{C}_{x,t})$ such that $\sum_{v_{z,t} \in \mathcal{C}_{x,t}} W(v_{z,t}, F(v_{z,t}))$ is maximized.

Our solution to the aforementioned *Substitute* assignment problem is shown in Figure 7, which is developed based on Kuhn-Munkres algorithm. At the beginning, the neighbor set $\mathcal{N}_{x,t}$ is split into 2 disjoint sets: the children set U and the non-children set V (line 1). The *FillDummy* function fills U or V with dummy vertices to make them of an equal size (line 2). Next, our solution sets the weight value for each element in the matrix $U \times V$ (lines 3 to 7). In case that an element is associated with a dummy vertex or that it leads to violation of constraint (1) or (2), its weight value is set to 0 (lines 4 and 5). Otherwise, its weight value is the minimum of the upload-capacity demand and the upload-capacity supply associated with that element (line 6). Finally, after Kuhn-Munkres algorithm is invoked to compute the bijection mapping F (line 8), our solution retrieves the result from F for each child (lines 9 to 13).

Substitute_Assignment(peer $v_{x,t}$):
1. $U \leftarrow \mathcal{C}_{x,t}$; $V \leftarrow (\mathcal{N}_{x,t} \setminus \mathcal{C}_{x,t})$
2. *FillDummy*(U, V)
3. For each $v_i \in U$ and each $v_j \in V$ do
4. If (Constraint(1) or (2) is violated) $w[i][j] = 0$
5. Else If (v_i or v_j is dummy) $w[i][j] = 0$
6. Else $w[i][j] = \min(r/K, r_{jx,t} + \mu_{j,t})$
7. End-For
8. $F = KuhnMunkresMAX(U, V, W)$
9. For each $v_i \in U$ do
10. $v_j = F(v_i)$
11. If ($w[i][j]$) *Substitute*[i] = v_j
12. Else *Substitute*[i] = NULL
13. End-For

FIGURE 7 Pseudo code of *Substitute* assignment algorithm

5.2 | Streaming-quality estimation

In mesh-based P2P live streaming, every peer keeps track of how many video blocks it has received from each of its parents during the last second. This count information allows *CDNPatch maintenance* process to evaluate if constraint (4) is violated. When this process is invoked at peer $v_{x,t}$, it iteratively checks whether or not $r_{\text{ix},t} \geq \frac{r}{K}, \forall v_{i,t} \in \mathcal{P}_{x,t}$. Any failed or unqualified parents will be replaced by qualified ones or by some auxiliary CDN servers.

For evaluation of constraint (3) at peer $v_{x,t}$, we need to determine the smallest value of $\hat{K}$, $\hat{K} \geq K$, so the probability that at least $(K - |\mathcal{S}_{x,t}|)$ out of $(\hat{K} - |\mathcal{S}_{x,t}|)$ parents or backup parents survive after τ_m seconds exceeds $(1 - \epsilon)$, where $|\mathcal{S}_{x,t}|$ is the number of qualified parents in $\mathcal{P}_{x,t}$ with a unique *Substitute*. Based on the survival analysis in Section 4.3, we know that $\hat{K}$ is the smallest integer such that $F(K - |\mathcal{S}_{x,t}| - 1; \hat{K} - |\mathcal{S}_{x,t}|, S(\tau_m)) \leq \epsilon$. The corresponding implementation is similar to the server provisioning method in Figure 4. It first obtains the count information about $|\mathcal{S}_{x,t}|$ and then computes $S(\tau_m)$ and $\hat{K}$. When $|\hat{\mathcal{P}}_{x,t}|$ reaches $(\hat{K} - K)$, our evaluation method will return TRUE as the result; otherwise, it will return FALSE.

6 | OVERHEAD ANALYSIS

6.1 | Communication overhead

CDNPatch's communication overhead is mainly determined by 2 factors: the size of the neighbor set and the size of

the backup-parent set. CDNPatch requires each peer to send to its neighbors a 9-byte neighbor-status message every τ_m seconds, so the communication overhead because of the neighbor-status messages is $\Theta(m/\tau_m)$ bytes per second per peer. In comparison, in an ordinary mesh-based P2P system, each redundant parent of a peer must be maintained as its neighbor and every second they have to exchange a buffermap message of hundred bytes.

Regarding the communication overhead because of the backup-status messages, each backup parent of a peer $v_{x,t}$ should send a 7-byte backup-status message to $v_{x,t}$ every τ_m seconds, so the associated communication overhead is $\mathcal{O}(\hat{K}/\tau_m)$ bytes per second per peer, where $\hat{K}$ is determined by $F(K-1;\hat{K},S(\tau_m)) \leq \epsilon$. Hoeffding's inequality yields an upper bound on $F(K-1;\hat{K},S(\tau_m))$: $F(K-1;\hat{K},S(\tau_m)) \leq e^{-2\frac{(p\hat{K}-K)^2}{\hat{K}}}$, where $p = S(\tau_m)$. Now, solving $e^{-2\frac{(p\hat{K}-K)^2}{\hat{K}}} = \epsilon$ for $\hat{K}$ on condition that $\hat{K} \geq K$, we obtain a lower bound on $\hat{K}$: $\frac{-\ln\epsilon+4pK+\sqrt{(\ln\epsilon)^2-8pK\ln\epsilon}}{4p^2}$. For $\tau_m \ll L$, $p \approx 1$ and ϵ is a small constant fraction, so CDNPatch's communication overhead because of the backup-status messages is $\mathcal{O}(K/\tau_m)$ bytes per second per peer. Summing up, because K is less than m and the neighbor-status messages, compared with the backup-status messages, contribute to most of CDNPatch's communication overhead, CDNPatch's overall communication overhead is $\Theta(m/\tau_m)$ bytes per second per peer.

6.2 | Computation overhead

Failover recovery is the only computation that copes with recovery of P2P live streaming quality on the fly, whereas the computations of *backup maintenance* and *Substitute assignment* are executed in the background every τ_m seconds. Because the *failover recovery* method checks K parent connections iteratively, its time complexity is $\Theta(K)$ even though its actual execution time is less than a second.

Our implementation of the *backup maintenance* method yields a time complexity of $\mathcal{O}(m \log m)$ as explained below. When backup-status messages arrive at a peer, a preprocessing module will be invoked to check the received backup-status information and to store the qualified ones in a sorted list named *QualifiedBackup* in subject to constraints (1) and (2). When the *backup maintenance* method begins, it determines the minimum size of the backup-parent set by looking up a precomputed reference table instead of computing the minimum size on the fly. Suppose this minimum size is k, then *backup maintenance* simply uses the first k peers in the *QualifiedBackup* list as the backup parents. The aforementioned procedure takes only a time complexity of $\mathcal{O}(1)$. However, it takes a time complexity of $\mathcal{O}(m \log m)$ for *backup maintenance* to sort the neighbor set, so the overall time complexity of *backup maintenance* is $\mathcal{O}(m \log m)$.

The *Substitute assignment* method is developed based on Kuhn-Munkres algorithm with time complexity of $\mathcal{O}(m^3)$. On a personal computer with a 2.0 GHz CPU, it takes less than 0.2 seconds to solve *Substitute* assignment problem for $m \leq 100$. Note that $m = 100$ is a reasonable upper bound on the neighbor-set size according to P2P literature.[10]

TABLE 2 Summary of CDNPatch's overheads

CDNPatch Overhead	Asymptotic Bound
Communication Overhead	$\Theta(m/\tau_m)$
- *neighbor-status message*	$\Theta(m/\tau_m)$
- *backup-status message*	$\mathcal{O}(K/\tau_m)$
Computation Overhead (on-the-fly)	$\Theta(K)$
- *substitute assignment* (background)	$\Theta(m^3)$
- *backup maintenance* (background)	$\mathcal{O}(m \log m)$
- *failover recovery* (on-the-fly)	$\Theta(K)$

Table 2 summarizes the complexity of CDNPatch's communication and computation overheads.

6.3 | Auxiliary CDN capacity consumption

The mesh-based P2P live streaming system is built upon a randomized overlay network, so it is very difficult to analyze how many *Substitute* backup parents a peer may find in τ_m seconds and hence how much upload capacity a peer may need from the auxiliary CDN servers to meet the service-level agreement. However, the analysis in Section 4.3 shows that the auxiliary CDN server capacity consumed per peer is upper bounded by a fraction $(1 - \frac{K'}{K})$ of r kbps, where K' is the largest integer such that $F(K'-1;K,S(\tau_m)) \leq \epsilon$. In the next section we will evaluate CDNPatch's performance for auxiliary CDN capacity consumption by simulation methodology.

7 | PERFORMANCE EVALUATION

In this section, we study through simulation the effectiveness and the efficiency of our CDNPatch proposal. We first explain the simulation setup that is used to conduct the simulation experiments, then present some representative results and discuss their implications.

7.1 | Simulation setup

We have implemented an OMNeT++ simulation that captures the major features of a generic hybrid CDN-P2P system as well as the functionality of mesh-based P2P live streaming and the design of CDNPatch. This simulation only serves as a proof of concept, so we make a few simplifications in its setup: (1) our simulation considers end users' Internet access bandwidth be the bottleneck to P2P live media streaming in a regional P2P network; (2) we simulate the end-to-end propagation delay in a regional P2P network by a random value from 10 to 150 milliseconds according to the previous work[10];

(3) existing hybrid CDN-P2P literature did not provide an explicit design of dynamic server provisioning and failover mechanisms comparable to the design of CDNPatch, so in the simulation experiments we use the pure mCache mechanism, which does not have access to an extra auxiliary CDN server capacity, to set the baseline performance for evaluation of the failure recovery performance of CDNPatch, and evaluate the cost effectiveness of CDNPatch against the cost of the pure CDN scheme; (4) for fair comparison with CDNPatch design, we simulate the mCache mechanism by a stochastic process that uniformly samples the active peers in the system every τ_m seconds; (5) to prevent the system from dying out, our simulation makes the failed peers rejoin immediately as newly initialized peers.

Table 3 lists the default settings of our simulation experiments that are similar to those used in the study of Li et al.[10] At the beginning of the simulation, 1000 peers join a regional P2P swarm of the hybrid CDN-P2P system as described in Section 3, and peers hear from the tracker that peer lifetimes follow a general distribution function $L(u)$ with an average of L seconds. Our simulation adopts several peer-lifetime models, including (1) an exponential CDF: $L(u) = 1-e^{-(e/393)}$ with $L = 393$ seconds, and (2) 3 Weibull peer-lifetime models from P2P measurement literature,[41] namely, PPTV (PPLive) Lifetime: $L(u) = 1 - 2.01 \cdot e^{-(u/12.262)^{0.24}}$ with $L = 393$ seconds and $u \geq 2.8$ seconds, PPStream Lifetime: $L(u) = 1 - 1.20 \cdot e^{(u/322.07)^{0.39}}$ with $L = 1222$ seconds and $u \geq 4.1$ seconds, and SOPCast Lifetime: $L(u) = 1-1.08 \cdot e^{(u/993.79)^{0.45}}$ with $L = 1861$ seconds and $u \geq 3.4$ seconds. Each peer can maintain at most $m = 20$ neighbors. We use the 2-class model from the study of Kumar, Liu, and Ross[42] to simulate the heterogeneity in peers' upload capacity. In our 2-class model, a small fraction 5% of the peers are classified as super peers each with 1-Mbps upload capacity, and the remaining peers are ordinary peers each with 500-kbps upload capacity. The media source in the system has an upload capacity of 10 Mbps and can serve up to 20 children. The media source streams the live media at a rate of $r = 450$ kbps with $K = 10$ sub-streams and with a uniform block size of 10 kilobits, which leaves only 50-kbps free upload capacity per peer that can be used for other information exchange. Although long start-up buffering helps in reducing some playback interruptions, most of the live media audience perceives a long start-up delay as an indication of poor QoS. Because of this reason, we fix the peers' start-up delay to merely 10 seconds, so peers can buffer up to 20 seconds' media content and start playing the live media immediately when the playback buffer is half loaded. Peers can store a list of 100 peers in their mCache. There is 1 CDN edge server acting as a media source to prime this regional P2P network; it also hosts the CDNPatch tracker/manager for (1) assisting new peers in the joining process and (2) provisioning the minimally required amount of auxiliary CDN server capacity for its regional P2P network. The CDNPatch maintenance timeout period is $\tau_m = 30$ seconds, and the playback discontinuity probability ϵ is 1%.

TABLE 3 Default simulation settings

Parameter	Value
Media Source Capacity	10 Mbps
Media Streaming Rate	450 kbps
Video Block Size	10 kilo-bits
Number of Sub-streams	10
Peer Upload Capacity	500 kbps (95%)
	1 Mbps (5%)
Expected Peer Lifetime	393 s (Exponential)
	393, 1222, 1861 s (Weibull)
Neighbor-set Size	20 peers
Start-up Buffering	10 s
mCache Capacity	100 peers
End-to-end Delay	10 to 150 ms
CDNPatch Maintenance Period (τ_m)	30 s
Playback Discontinuity Probability (ϵ)	1%

The following metrics are used to evaluate the system's failure recovery performance:

1. *Failure recovery time:* is the time elapsed from the moment a failure event occurs to the moment the affected peer has subsequently recovered its playback rate.
2. *Failure recovery success rate:* is the fraction of the affected peers that have recovered their playback rate within a limited time period.
3. *Playback continuity index (PCI):* is 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. ($PCI \equiv 0$) means the worst video playback quality; ($PCI \equiv 1$) means the best video playback quality without any interruption.
4. *Communication overhead:* is the amount of upload bandwidth per peer that is consumed by exchange of buffermaps or CDNPatch messages.
5. *Auxiliary CDN capacity consumption:* is the average amount of the auxiliary CDN server capacity that has been provisioned and used for each peer.

7.2 | CDNPatch vs mCache

The first objective of our simulation experiments is to compare the failure recovery performance of a hybrid CDN-P2P live streaming system with and without using CDNPatch. During the simulation run-time of each case, we uniformly sample 30 active peers, make a fraction η of their parents simultaneously leave the system, and then measure their failure recovery performance during the following-up period of 300 seconds. We test the P2P system's failure recovery performance of both cases with a set of failure rates, $\eta \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$. The failure rate $\eta = 0.5$ is high enough for our system settings because with an average peer lifetime of about 7 minutes, the chance that a peer loses more than half of its parents in 30 seconds is slim.

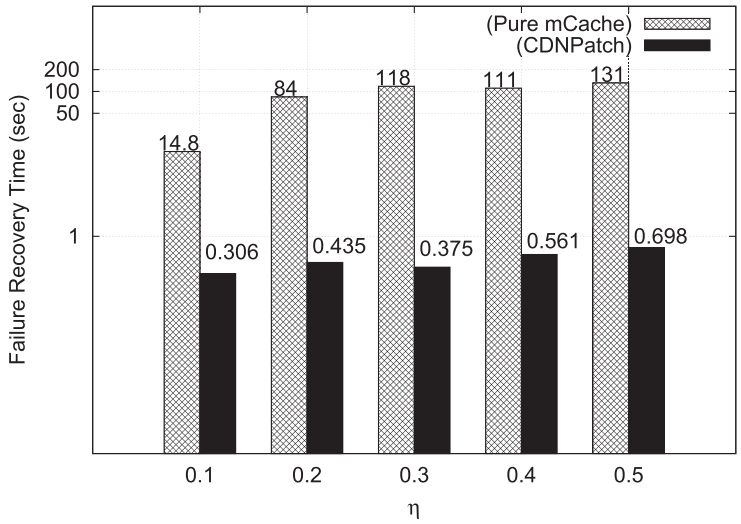

FIGURE 8 Failure recovery time against failure rate η: with and without using CDNPatch under the system settings of $(\tau_m, \epsilon) = (30s, 1\%)$. CDN, content delivery network

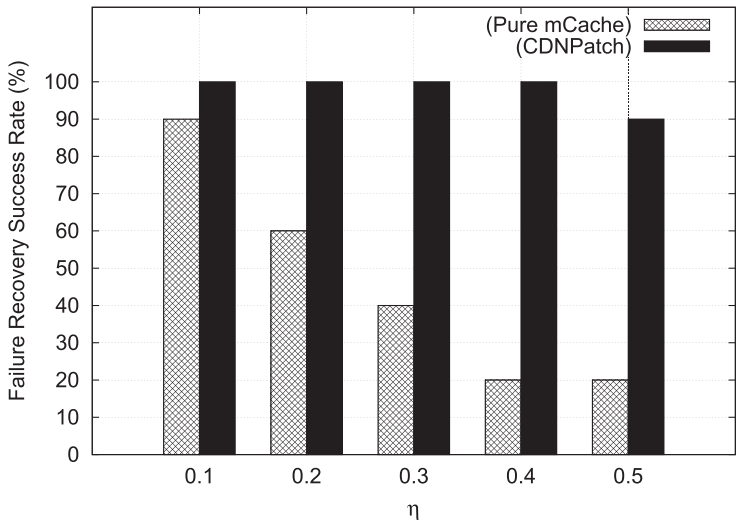

FIGURE 9 Failure recovery success rate against failure rate η: with and without using CDNPatch under the system settings of $(\tau_m, \epsilon) = (30s, 1\%)$. CDN, content delivery network

We plot in Figures 8 and 9 the resultant failure recovery performance of failure recovery time and failure recovery success rate. These results reveal that (1) CDNPatch effectively reduces the task of failure recovery to merely failover to the backups that takes less than a second to complete. For the experiments with $\eta \leq 0.4$, CDNPatch delivers a remarkable 100% failure recovery success rate and a failure recovery time less than 1 second at a cost of P2P communication overhead of less than 1 kbps, while consuming 92.1-kbps, 138-kbps, 184-kbps, and 322-kbps auxiliary CDN server capacity per peer under the impact of the exponential, SOPCast, PPStream, and PPTV peer-lifetime model, respectively. Even for $\eta = 0.5$, CDNPatch still delivers a 90% failure recovery success rate. (2) In comparison, the hybrid CDN-P2P system that relies on the pure mCache likely yields a poor failure recovery success rate and cannot deliver timely failure recovery. Even at a low failure rate of $\eta = 0.1$, the hybrid CDN-P2P system on average takes more than 10 seconds to find a qualified parent candidate from the mCache. As the failure rate η increases to more than 0.2, the averaged failure recovery time increases to more than 80 seconds and the averaged failure recovery success rate quickly drops from 90% to 40% then to 20%.

For the experiment with $\eta = 0.5$, we plot in Figures 10 and 11 the averaged PCI of the peers that were affected by the failure event to be able to see how much CDNPatch improves the stability of the system's failure recovery performance. It shows that CDNPatch delivers a timely and stable failure recovery—there is only 1 transient glitch on the curves a few seconds after the occurrence of the failure event, but CDNPatch rapidly restores the playback quality of the affected peers back to PCI ≥ 0.9. This is because with the setting of $\epsilon = 1\%$, each peer has at most 4 backup parents or an equivalent amount of auxiliary CDN server capacity

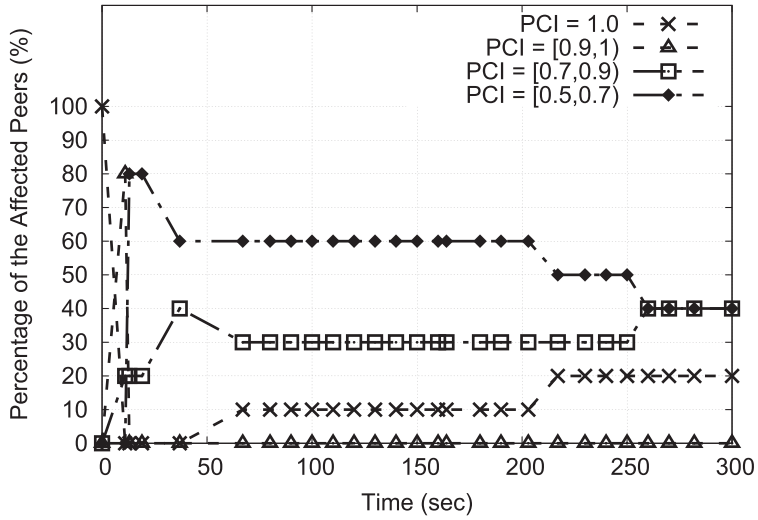

FIGURE 10 Playback continuity of the peers affected by the failure event against time: without using CDNPatch (pure mCache) under the system settings of $(\eta,\tau_m, \epsilon) = (0.5, 30s,1\%)$. PCI, playback continuity index

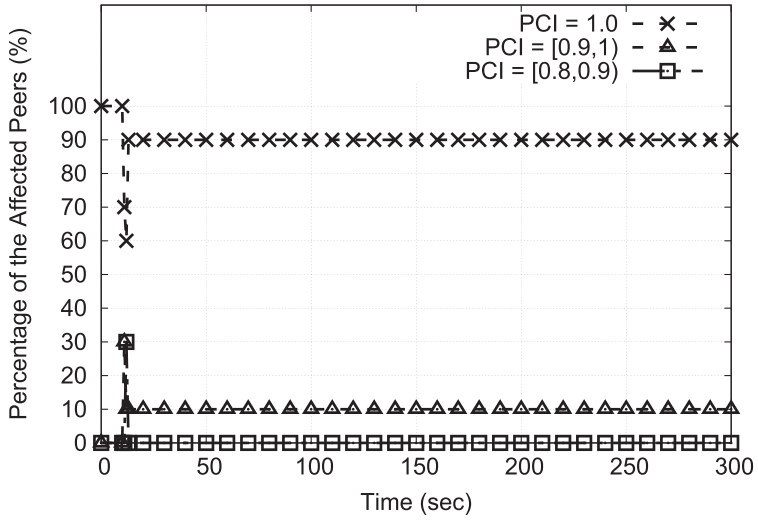

FIGURE 11 Playback continuity of the peers affected by the failure event against time: using CDNPatch under the system settings of $(\eta,\tau_m, \epsilon) = (0.5, 30s,1\%)$. PCI, playback continuity index

in reserve, which are not enough to handle the failure scenario of $\eta = 0.5$ where some peers may simultaneously lose at most 5 of their parents. On the other hand, the failure recovery performance of the pure mCache scheme is poor—after the 300-second measurement period, only 20% of the affected peers have been fully recovered (PCI = 1.0), 40% of the peers have a playback rate between 300 and 400 kbps (PCI = (0.7, 0.9)), and the remaining peers have a very poor playback rate below 300 kbps (PCI = (0.5,0.7)).

7.3 | Impact of playback discontinuity probability and maintenance period

To examine the impact of the playback discontinuity probability ϵ, we rerun the previous simulation experiment with $(\tau_m,\epsilon) = (30s,0.1\%)$, and plot CDNPatch's failure recovery performance in Figure 12. With this setting, CDNPatch can completely mask the impact of a 50% parent-failure rate and deliver a remarkable failure recovery performance with a 100% failure recovery success rate and a failure recovery time less than 1 second while consuming a communication overhead of less than 1 kbps and an amount of 138-kbps, 184-kbps, 230-kbps, and 368-kbps auxiliary CDN capacity per peer under the exponential, SOPCast, PPStream, and PPTV peer-lifetime model, respectively.

Next, we examine the impact of the maintenance period τ_m on the failure recovery performance. We use the setting of $(\tau_m,\epsilon) = (10s,1\%)$[¶] and a couple of failure rates, $\eta \in \{0.1, 0.2\}$, to test the P2P system's failure recovery performance

[¶]It becomes unrealistic to expect that a P2P system could refresh every peer's mCache content every τ_m<10 seconds.

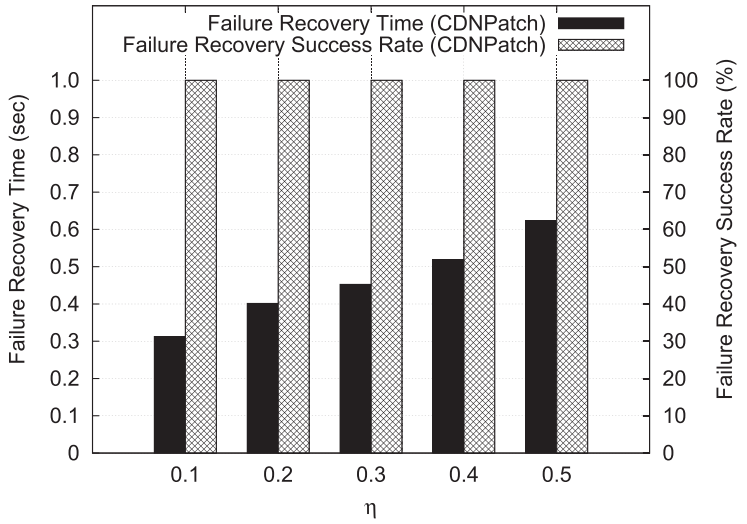

FIGURE 12 Failure recovery time and failure recovery success rate against failure rate η: with using CDNPatch under the system settings of $(\tau_m,\epsilon) = (30s, 0.1\%)$.

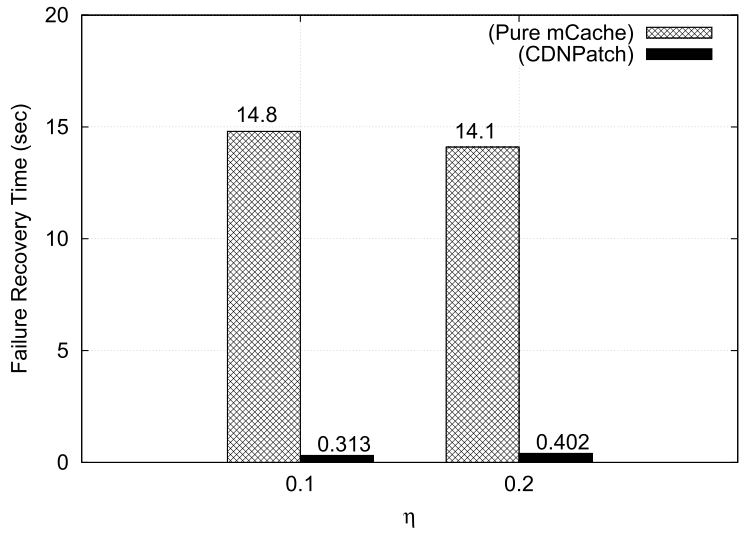

FIGURE 13 Failure recovery time against failure rate η: with and without using CDNPatch under the system settings of $(\tau_m, \epsilon) = (10s,1\%)$.

because with an average lifetime of about 7 minutes, the chance that a peer loses more than 20% of its parents in 10 seconds is comparable to the chance that a peer loses more than half of its parents in 30 seconds. The experimental results are plotted in Figures 13 and 14. The figure shows that CDNPatch delivers an outstanding failure recovery performance, the same as before, while consuming a communication overhead of less than 1 kbps and 46-kbps, 46-kbps, 92-kbps, and 230-kbps auxiliary CDN capacity per peer under the exponential, SOPCast, PPStream, and PPTV peer-lifetime model, respectively. The figure also shows that with the use of a small value of τ_m=10 seconds, the failure recovery time of the pure mCache scheme has been improved to less than 15 seconds, but its failure recovery success rate is still quite low—it achieves a 90% and a 60% failure recovery success rate at $\eta = 0.1$ and 0.2, respectively. We have conducted another experiment with $(\tau_m,\epsilon) = (10s, 0.1\%)$, but the results look almost identical to those shown in Figures 13 and 14.

We finally list in Table 4 the CDN capacity consumption and the P2P communication overhead under different settings of the playback discontinuity probability ϵ and the maintenance period τ_m. As outlined in Table 4, CDNPatch's gain in the failure recovery performance comes at a small per-peer communication overhead of less than 1 kbps. By using CDNPatch with the setting of τ_m=10 seconds, a hybrid CDN-P2P system can mask the peer dynamics' impact of 3 real P2P systems, namely, SOPCast, PPStream, and PPTV, with 100% failure recovery success rate and a failure recovery time less than 1 second at a cost of small per-peer communication overhead of less than 1 kbps per second, while using only about 10%, 21%, and 51%, respectively, of the pure CDN scheme's server-capacity consumption.

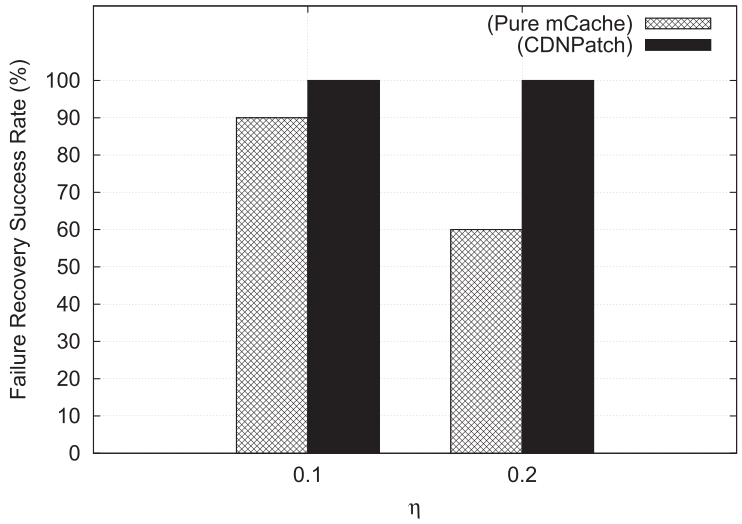

FIGURE 14 Failure recovery success rate against failure rate η: with and without using CDNPatch under the system settings of $(\tau_m, \epsilon) = (10s, 1\%)$

TABLE 4 Summary of the CDN capacity consumption and the P2P communication overhead under different settings of $\epsilon$and τ_m

	Overhead (τ_m=30*s*)	**Overhead (τ_m=10*s*)**
Buffermap Message	44 kbps	44 kbps
CDNPatch Message		
– $\epsilon = 1\%$	0.052 kbps	0.151 kbps
– $\epsilon = 0.1\%$	0.054 kbps	0.151 kbps
Auxiliary CDN Capacity Consumption		
– exponential (L = 393 s)		
$\epsilon = 1\%$	92.1 kbps	46 kbps
$\epsilon = 0.1\%$	138 kbps	46 kbps
– PPTV (L = 393 s)		
$\epsilon = 1\%$	322 kbps	230 kbps
$\epsilon = 0.1\%$	368 kbps	276 kbps
– PPStream (L = 1222 s)		
$\epsilon = 1\%$	184 kbps	92 kbps
$\epsilon = 0.1\%$	230 kbps	138 kbps
– SOPCast (L = 1861 s)		
$\epsilon = 1\%$	138 kbps	46 kbps
$\epsilon = 0.1\%$	184 kbps	92 kbps

8 | CONCLUSIONS

We have shown in this paper that it takes a combination of dynamic server provisioning and failover mechanisms to ensure that an unreliable hybrid CDN-P2P live streaming system can perform timely failure recovery in a cost-effective manner. To enhance P2P live streaming quality under the impact of peer dynamics, we have designed a hybrid CDN-P2P failover solution named *CDNPatch*, which enables peers to quickly recover from parent failures by failover to backup parents or to auxiliary CDN servers. From our simulation experiments, CDNPatch was found to be effective, efficient, and stable against peer dynamics of real P2P systems. There are a couple of open issues that need further examinations. First, we believe that it is possible to further improve CDNPatch's hybrid service strategy and reduce its CDN capacity consumption. Second, it would be of interest to find the Pareto frontier of the hybrid CDN-p2p failure recovery strategies in subject to different QoS and cost constraints. Furthermore, some possible future work for CDNPatch includes supporting the QoS negotiation with multiple CDN service providers and experimenting with large-scale scenarios for Internet-based CDNs.

REFERENCES

1. Michael A, Thomson L, Jay P. Hybrid content delivery network (CDN) and peer-to-peer (P2P) network. *US Patent 8332484*, Assignee: Akamai Technologies, Inc, December 2012.
2. Yin H, Liu X, Zhan T, Sekar V, Qiu F, Lin C, Zhang H, Li B. LiveSky: enhancing CDN with P2P. *ACM T Multim Comput*. 2010;6:16–34.
3. Magharei N, Rejaie R, Guo Y. Mesh or multiple-tree: a comparative study of live P2P streaming approaches. *Proceedings of IEEE INFOCOM*, Anchorage, Alaska, USA; 2007:1424–1432.
4. Vassilakis C, Stavrakakis I. Minimizing node churn in peer-to-peer streaming. *Elsevier Comput Comm*. 2010;33(14):1598–1614.
5. Yang D, Zhang YX, Zhang HK, Wu TY, Chao HC. Multi-factors oriented study of P2P churn. *Int J Commun Syst*. 2009;22:1089–1103.
6. Lu ZH, Gao XH, Huang SJ, Huang Y. Scalable and reliable live streaming service through coordinating CDN and P2P. *Proceedings of ICPADS*, Tainan, Taiwan; 2011.
7. Zhang G, Liu W, Hei X, Cheng W. Unreeling Xunlei Kankan: understanding hybrid CDN-P2P video-on-demand streaming. *IEEE T Multimedia*. 2015;17:229–242.
8. Birkos K, Andriopoulou F, Papageorgiou C, Dagiuklas T, Kotsopoulos S. Enhanced failover mechanisms for tree-based peer-to-peer streaming. *Proceedings of IEEE ICC*, London, UK; 2015:7024–7029.
9. Wu J, Zhou Y, Chiu DM, Zhu Z. CDN bandwidth allocation in weakly interconnected networks. *Proceedings of IEEE Symposium on Computers and Communication (ISCC)*, Larnaca, Cyprus; 2015:857–862.
10. Li B, Xie S, Qu Y, Keung GY, Lin C, Liu J, Zhang X. Inside the new coolstreaming: principles, measurements and performance implications. *Proceedings of IEEE Infocom*, Phoenix, AZ, USA; 2008.
11. Mol J, Epema D, Sips H. The Orchard Algorithm: building multicast trees for P2P video multicasting without free-riding. *IEEE T Multimedia*. 2007;9(8):1593–1604.

12. PPTV. URL http://www.pptv.com/. [Accessed 21 September 2016]
13. Sopcast. URL http://www.sopcast.org/. [Accessed 21 September 2016]
14. Akamai Technologies, Inc.URL http://www.akamai.com/. [Accessed 21 September 2016]
15. Chen Z, Yin H, Lin C, Liu X, Chen Y. Towards a trustworthy and controllable peer-server-peer media streaming: an analysis study and an industrial perspective. *Proceedings of IEEE Globecom*, Washington, D.C., USA; 2007:2086–2090.
16. Xu D, Kulkarni S, Rosenberg C, Chai H. Analysis of a CDN-P2P hybrid architecture for cost-effective streaming media distribution. *Springer Multimedia Syst*. 2006;11(4):383–399.
17. Alasaad A, Gopalakrishnan S, Leung VCM. A hybrid approach for cost-effective media streaming based on prediction of demand in community networks. *Springer Telecom Syst*. 2014;59:329–343.
18. Kao YC, Lee CN. Proxy-assisted P2P and multicast transmission schemes for layered-video streaming over wireless networks. *Int J Commun Syst*. 2010;23:1167–1188.
19. Esposito F, Cerroni W. Integrating piece and peer selection in content distribution networks. *Proceedings of IEEE Globecom*, San Diego, CA, USA; 2015:1–6.
20. Rouibia S, Parrein B. P2PWeb-live: a client/server and P2P hybrid protocol for IPTV delivery. *Proceedings of 2015 International Conference on Protocol Engineering (ICPE) and International Conference on New Technologies of Distributed Systems (NTDS)*, Paris, France; 2015:1–6.
21. Li Z, Wu Q, Salamatian K, Xie G. Video delivery performance of a large-scale VoD system and the implications on content delivery. *IEEE T Multimedia*. 2015;17(6):880–892.
22. Lin CS, Lee IT. Applying multiple description coding to enhance the streaming scalability on CDN-P2P network. *Int J Commun Syst*. 2010;23:553–568.
23. Zhou Y, Xu Y, Zhang S. Exploring coding benefits in CDN-based VoD systems. *IEEE T Circ Syst Vid*. 2014;24(11):1969–1981.
24. Milani S, Calvagno G. Distributed multiple description video transmission via noncooperative games with opportunistic players. *IEEE T Circ Syst Vid*. 2015;25(1):125–138.
25. Mandal U, Habib MF, Zhang S, Lange C, Gladisch A, Mukherjee B. Adopting hybrid CDN-P2P in IP-over-WDM networks: an energy-efficiency perspective. *IEEE/OSA J Opt Commun Netw*. 2014;6(3):303–314.
26. Lawey AQ, El-Gorashi TEH, Elmirghani JMH. BitTorrent content distribution in optical networks. *J Lightwave Technol*. 2014;32(21):4209–4225.
27. Meskovic M, Kos M, Meskovic A. Optimal chunk scheduling algorithm based on Taboo search for adaptive live video streaming in CDN-P2P. *Proceedings of Software, Telecommunications and Computer Networks (SoftCOM)*, Split - Bol, Croatia; 2015:205–209.
28. Jeon HS, Jung H, Chun W. ID based Web Browser with P2P property. *Proceedings of International Conference on Future Generation Communication and Networking (FGCN)*, Jeju Island, Korea; 2015:41–44.
29. Nam Y, Lee C, Kang S, Park J. Synchronization among CDN edge severs using P2P networking. *Proceedings of International Conference on Information and Communication Technology Convergence (ICTC)*, Jeju Island, Korea; 2015:466–468.
30. Sales LMD, Sales TBMD, Almeida H, Perkusich A, Gorgonio K. Generalized connections and incentives for supporting CE devices in live streaming systems. *IEEE Trans Consum Electron*. 2014;60(4):605–613.
31. Zhang M, Luo JG, Zhao L, Yang SQ. A peer-to-peer network for live media streaming using a push-pull approach. *Proceedings of ACM International Conference on Multimedia*, Singapore; 2005:287–290.
32. Ghanbari A, Rabiee HR, Khansari M, Salehi M. PPM—a hybrid push-pull mesh-based peer-to-peer live video streaming protocol. *Proceedings of ICCCN*, Munich, Germany; 2012:1–8.
33. Goga O, Teixeira R. Speed measurements of residential Internet access. *Proceedings of the 13th international conference on Passive and Active Measurement*, Vienna, Austria; 2012:168–178.
34. Sweha R, Ishakian V, Bestavros A. AngelCast: cloud-based peer-assisted live streaming using optimized multi-tree construction. *Proceedings of ACM Multimedia Systems*, Chapel Hill, North Carolina, USA; 2012:191–202.
35. Wang F, Liu J, Chen M. CALMS: cloud-assisted live media streaming for globalized demands with time/region diversities. *Proceedings of IEEE INFOCOM*, Orlando, FL, USA; 2012:199–207.
36. Strauss J, Katabi D, Kaashoek F. A measurement study of available bandwidth estimation tools.*Proceedings of IMC*, Miami, Florida, USA; 2003.
37. Network time protocol. URL http://www.ntp.org/. [Accessed 21 September 2016]
38. Wang X, Yao Z, Loguinov D. Residual-based estimation of peer and link lifetimes in P2P networks. *IEEE/ACM Trans Netw*. June 2009;17(3):726–739.
39. Stutzbach D, Rejaie R, Duffield N, Sen S, Willinger W. On unbiased sampling for unstructured peer-to-peer networks. *IEEE/ACM Trans Netw*. April 2009;17(2):377–390.
40. Liang J, Kumar R, Ross KW. The fasttrack overlay: a measurement study. *Comput Netw*. April 2006;50(6):842–858.
41. Silverston T, Fourmaux O. Measuring P2P IPTV systems. *Proceedings of ACM NOSSDAV*, Urbana-Champaign, IL, USA; 2007.
42. Kumar R, Liu Y, Ross K. Stochastic fluid theory for P2P streaming systems. *Proceedings of IEEE Infocom*, Anchorage, Alaska, USA; 2007.

How to cite this article: Wang C-C, and Lin Y-D, (2014), CDNPatch: a cost-effective failover mechanism for hybrid CDN-P2P live streaming systems. *Int. J. Commun. Syst.*, doi: 10.1002/dac.3193.