A Study on High Quality and Low Cost Peer to Peer Live Streaming over Internet

(インターネット上のP2Pライブストリーミングの高品質化・低コスト化の研究)

by

BAHAA ALDEEN ALGHAZAWY

A dissertation submitted to the Department of Information Engineering, Graduate School of Engineering
in partial fulfillment of the requirements for the degree of

Doctor of Engineering

at the

Hiroshima University

2016

A Study on High Quality and Low Cost Peer to Peer Live Streaming over Internet

(インターネット上のP2Pライブストリーミングの高品質化・低コスト化の研究)

by

BAHAA ALDEEN ALGHAZAWY

Abstract

Recently, Peer to Peer (P2P) systems have been widely leveraged to broadcast a live video stream to a large number of viewers. In P2P live streaming, peers (viewers) not only watch the stream but also contribute their resources such share it. As a result, this collaborative approach reduces the cost and improves the scalability of traditional client-server live streaming systems. However, P2P systems work in a best-effort manner and depend on the voluntary contribution of peers. Hence, it is challenging to provide a high quality streaming service (i.e., smooth playback and short delay) in the P2P live streaming. In order to achieve that, it is important to solve two key issues as follows: 1) how to maximally utilize resources of participant peers to share a live stream, and 2) how to improve the reliability of P2P systems by the assistance of reliable servers with a low cost.

In this thesis, we tackled the above-mentioned two issues and provided answers for them. At first, a scheme is proposed to maximally utilize resources of participant peers. By that, the utmost level of available resources is utilized which either can improve the quality of service or reduce the server cost or a trade-off between that. The scheme allows all peers to be engaged in the live stream distribution process. It is achieved by developing an efficient peering and content scheduling strategy that takes both upload bandwidth bottleneck and content bottleneck issues of P2P systems into account. The scheme is able to construct an efficient overlay structure with a short hop-count delay and attain a maximal streaming rate.

Secondly, the P2P system is assisted by the reliable cloud computing, having a hybrid cloud-P2P, to provide a guaranteed quality of service. Such assistance overcomes the reliability issue of a pure P2P where peers join/leave the system or fail dynamically (peer churns). In the proposed cloud-P2P

model, cloud storage and cloud content delivery network services are exploited to help the peers to maintain a smooth playback. Moreover, peers are referenced to the cloud storage service to recover the overlay quickly in case of peer churns. As cloud services are not free, the model is designed to incur as low cost as possible while guaranteeing the quality of service.

Dedicated
to my parents, Abdulhakim and Rehab,
to my wife, Borouj,
to my children, brothers and sisters
for the endless support, encouragement and love.
Having a PhD. is only possible due to their sacrifices.

Acknowledgments

I acknowledge, with gratitude, my debt of thanks to my supervisor Prof. Satoshi Fujita for his encouragement and patience and whose knowledge and expertise were generously shared throughout my PhD. His guidance helped me in all the time of research and writing of this thesis.

Besides my supervisor, I would like to thank the rest of my thesis committee: Prof. Koji Nakano and Prof. Toru Nakanishi for their influential insights and feedback.

My sincere thanks also go to Assoc. Prof. Sayaka Kamei and Assist. Prof. Satoshi Taoka, who provided me with a precious support in the lab.

Thanks to my colleagues in Distributed Systems Lab. for their discussion and motivation. Also, I would like to thank my friends in the Information Engineering department who made my research and life easier.

Thanks to the faculty members and staff of the department of information engineering for their continuous support.

THIS PAGE INTENTIONALLY LEFT BLANK

Contents

List of Figures

List of Tables

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 1

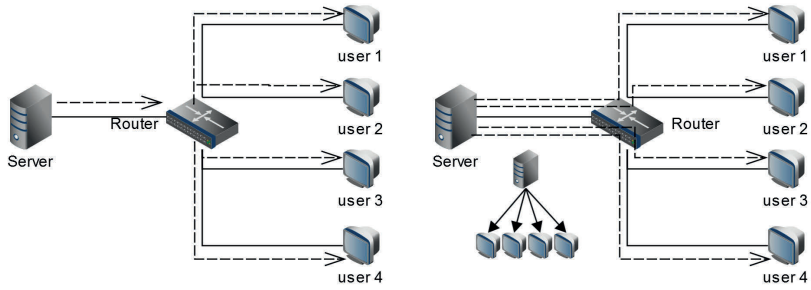
Introduction

Video over internet has attracted a huge number of users in recent years thanks to the widespread of broadband accesses to the internet. Hundreds of millions of video hours are watched on YouTube[1] every day [1]. Along with that, many successful streaming providers have attracted millions of users such as Yahoo video[2], Hulu[3], and others. In fact, it is forecasted that internet video streaming and downloads will grow up to more than 80% of the global internet consumer traffic by 2019 [2].

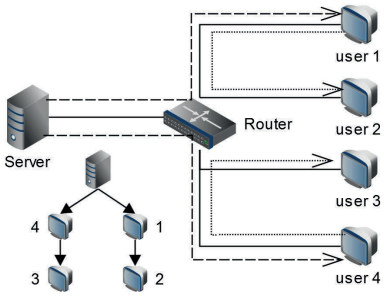
An efficient streaming solution is the IP multicast which can be used to deliver a video stream to many users with a low overhead on the underlying IP network. In IP multicast, the server forwards only one copy of each streaming packet to the nearest network router. Then, the packet is routed to all destinations in such a way that a physical link transfers only one copy of a single packet as illustrated in Figure 1.1(a). Unfortunately, IP multicast is not widely deployed by Internet Service Providers (ISPs) due to different challenges. Some of them are the costly installation and management, the lack of supporting some functionalities such as security and group management and the lack of a good pricing model [3].

Another traditional live streaming solution has been the client-server model. In which, a central streaming server sends identical streaming packets to every user as in 1.1(b). It is not difficult to show that such a solution has different drawbacks such as: 1) it is not scalable where one server can sup-

[1] www.youtube.com
[2] www.yahoo.com
[3] www.hulu.com



(a) IP multicast. (b) Client-server model.



(c) Application layer multicast.

Figure 1.1: A simple example of different multicast solutions.

port a limited number of users, 2) it is highly expensive as a huge number of users requires the investment of many servers and 3) it incurs more overhead (number of packets per physical link) on the underlying IP network. To improve the scalability, availability and performance, content delivery networks (CDNs) are established such as Akamai [4] and Limelight [5]. CDNs deploy a large number of servers in multiple data centers located in separated geographical regions. The streaming content can be delivered to users from nearby servers with a short latency. More recently, the emergence of cloud computing offers an elastic platform to provision resources on demand to support the content delivery process. Furthermore, a cloud content delivery

[4] www.akamai.com
[5] www.limelight.com

network (CCDN) supporting live streaming services has been deployed such as Amazon CloudFront [6] and Microsoft Azure CDN [7].

Attractively, researchers reach to a solution to implement the multicast functionality in the application layer rather than the network layer [4, 5, 6, 7]. In the application layer multicast (Figure 1.1(c)), users form a logic overlay network over the physical IP network. So packets are replicated and routed at the user hosts rather than the network routers. Different works [8, 9, 10, 11] showed that such a solution, called peer-to-peer network (P2P), addresses the scalability issue of the client-server model. In P2P, peers contribute their resources to realize a low cost and scalable content distribution service. Eventually, P2P has become a promising solution for the live streaming service as its collaborative nature allows a server with a low capacity (upload bandwidth) to serve thousands of users(peers). In P2P live streaming, every peer can download and simultaneously also upload a video stream to other peers. That is achieved by having a peering strategy which allows a peer to connect to other peers forming an overlay structure. Along with that, a content scheduling algorithm is necessary for peers to distribute the streaming content among each other [12]. Different P2P live streaming systems use different overlay structures and different content distribution protocols. The challenging question is that how to provide a high quality of streaming service (i.e., smooth playback and a short delay) with a low cost by adopting the P2P.

One point is that by the efficient maximal utilization of peers' resources, we can broadcast a maximal streaming rate with a short end-to-end delay. In case the streaming rate is fixed, the maximal utilization of resources reduces the necessary amount of server's bandwidth to broadcast the video stream to peers, i.e., it reduces the cost. To maximally utilize the resources, we need to guarantee that all peers are engaged in the content distribution process. Then, an efficient peering and content scheduling strategy is needed which is expected to overcome both of the upload bandwidth bottleneck and the content bottleneck in the P2P systems. In more detail, if the aggregated resources of peers, called the *capacity of P2P system*, is enough to broadcast the streaming content then the server should deliver no more than one copy of the video to the peers. A simple example is when every peer has an

[6] aws.amazon.com/cloudfront

[7] azure.microsoft.com

upload capacity greater than the streaming rate. Then, all peers can receive the video by organizing them in a tree where the server is a root and each peer has a single parent and a single child, Figure 1.2. However, such an overlay is not efficient and has a long end-to-end delay. The problem becomes more difficult in the heterogeneous environment where peers have unequal upload capacity. In fact, converging to an efficient overlay that maximize the resource utilization of peers is a challenging research question.

- **Point 1**: Finding an efficient overlay structure by designing a peering and content scheduling strategy is a challenging question. Especially, when the capacity of the P2P system is barely enough to broadcast the streaming content to all peers, i.e., the spare capacity to adopt future peers is very low.

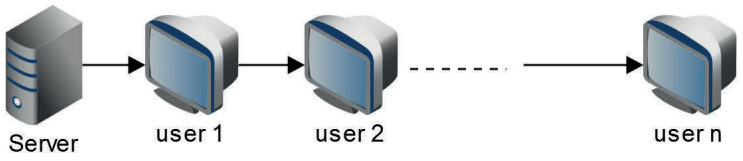


Figure 1.2: A simple overlay example. A server and n peers each has an enough upload capacity to forward the live stream to another peer.

Another point is that P2P live streaming is not reliable enough as it relies on the voluntary contribution of participating peers where peers join or leave the system dynamically (peer churn). So, it is hard to guarantee the quality of streaming service [13] even with a maximal resource utilization. That is in contrast to the reliable but costly server-based solutions. Then, to leverage the low cost and scalable P2P and, at the same time, overcome the reliability issues, hybrid P2P with other infrastructures has attracted considerable attention. For example, the hybrid cloud-P2P has been studied to deliver a guaranteed quality of streaming service [14, 15]. The assistance of P2P live streaming by the cloud is generally realized by on-demand renting of computing, storage and bandwidth resources from the cloud. That incurs

additional cost related to the amount of resources rented. Recently, reaching a low cost cloud-P2P live streaming has been an attractive topic.

- **Point 2**: A hot topic is how to guarantee the quality of service in cloud-P2P live streaming with as low cost as possible.

1.1 Contribution

In this thesis, we provide solutions to improve the quality of P2P live streaming service and reduce the cost of the guaranteed service in cloud-P2P live streaming. The following issues has been addressed.

- How to maximally utilize peer resources and maintain an efficient P2P overlay?.

By doing so, we improve the quality of service as more resources are utilized efficiently. And, at the same time, we reduce the required server bandwidth to deliver the live stream to all peers, i.e., reducing the cost. We discussed how to improve the resource utilization [12] in mesh overlays. Then, we addressed the issue in multiple-tree overlays by proposing a scheduling scheme [16] that works in the challenging condition where no spare capacity is available in the P2P system, i.e., all peers' resources are utilized. The scheme attains the maximal resource utilization while maintaining an efficient overlay of multiple short-delay trees in a distributed manner. A budget-model is used in the scheduling scheme such that each peer has a budget relative to its upload capacity and corresponds to the maximum number of children that peer can have. In the scheme, the role of uploading a substream (a part of video stream) is transferred to another peer by exchanging money (capacity) among peers, provided that the balance of each peer is not below zero. A newly joining peer try to instantly spend its budget to subscribe to substreams in such a way that guarantees a high number of peers forwarding exactly one substream and that the trees have a short hop-count delay. Such a proposed scheme is also able to broadcast the maximal streaming rate as it is able to attain the maximal resource utilization.

- How to guarantee the quality of streaming service with a low cost?.

Improving the quality of service in the first work does not mean a guaranteed quality of service. P2P live streaming systems are vulnerable to peer dynamics. To guarantee the quality of service, we need to improve the reliability of P2P systems. To achieve that, we assisted the P2P by the cloud storage service and cloud content delivery network (CCDN). The assistance of P2P live streaming by the cloud is realized by storing the latest chunks in the live stream to the storage service in the cloud. In addition, each peer is allowed to fetch missing chunks from the storage service through edge locations of the CCDN. These edge locations fetch chunks from the storage service upon receiving requests from peers. The cost of cloud-assistance is comprised of the number of requests issued to the cloud and the amount of data fetched from the cloud. We propose three techniques to reduce the cost of such a cloud-assistance [17]. More concretely, in the proposed method, 1) each peer which lost its parent in the overlay can find a new parent by referring to the information registered in the cloud, 2) several peers which proactively fetch chunks from the cloud are dynamically invested, and 3) the number of requests issued to the cloud is reduced by allowing peers to fetch a collection of chunks using a single request. We were able to reduces the cost of conventional cloud-P2P live streaming schemes by reducing the total amount of data fetched from the cloud by 42% and the total number of requests issued to the cloud by 66% while guaranteeing the quality of live streaming service.

1.2 Publications

- Bahaa Aldeen Alghazawy and Satoshi Fujita, "Low cost cloud-assisted peer to peer live streaming," *KSII Transactions on Internet and Information Systems,* (In press).

- Bahaa Aldeen Alghazawy and Satoshi Fujita, "A scheme for maximal resource utilization in peer-to-peer live streaming," *International journal of Computer Networks & Communications,* vol. 7, no. 5, pages 13-28, September, 2015.

- Bahaa Aldeen Alghazawy and Satoshi Fujita, "Probabilistic packet scheduling scheme for hybrid pull-push P2P live streaming protocols," *Proc. of the Second International Workshop on Advances in Networking and Computing,* pages 248-251, November, 2011.

1.3 Thesis Outline

- In Chapter 2, we present the background of our work by reviewing the state-of-art literature in P2P live streaming. Different types of P2P overlays along with the content distribution and performance bounds are explained. Then, the hybrid CDN-P2P and cloud-P2P are introduced. A special focus is given to the most related work.

- In Chapter 3, we go deeper into details of the maximal resource utilization scheme. We explain how to attain the maximal streaming rate in the challenging condition when the capacity of P2P system is barely enough to broadcast the video stream to peers.

- In Chapter 4, a low cost cloud-assisted P2P live streaming system with a guaranteed quality of service is presented. Three proposed techniques are explained and evaluated to show the important reduction in the monetary cost of the system.

- Chapter 5 concludes the thesis and points out the opportunities of a future work.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 2

P2P Live Streaming: A Literature Review

In this chapter, we will go into deeper details of P2P live streaming pure and hybrid systems by reviewing the state-of-art literature. That is to implement a solid base for understanding the work presented in this thesis. The overlay structure, content distribution and the performance bounds of P2P live streaming are explained. Then, hybrid CDN-P2P and cloud-P2P are introduced where more details are spent on the cloud-P2P and its cost due to the promising rule of cloud in future live streaming.

2.1 P2P Live streaming

As mentioned in Chapter 1, The basic idea of P2P live video streaming is that every peer participating in the P2P overlay contributes their resources to help the distribution of streaming content. P2P live streaming could realize a scalable video streaming with a low cost compared with the classical client-server model. Hence, P2P streaming systems have been widely deployed to deliver a good quality streaming content to a large number of users. As examples of commercial deployment are SopCast[1], PPLive[2], PPStream[3], UUSee[4] and

[1] http://www.sopcast.com
[2] http://www.pplive.com
[3] http://www.ppstream.com
[4] http://www.uusee.com

many others. Different P2P streaming systems use different overlay structures and different content distribution methods. The fundamental types of P2P overlay structures are: mesh structure, tree structure and hybrid structures. On the other hand, the content distribution methods are either pull-based, push-based or hybrid pull-push based.

2.1.1 Overlay Structures

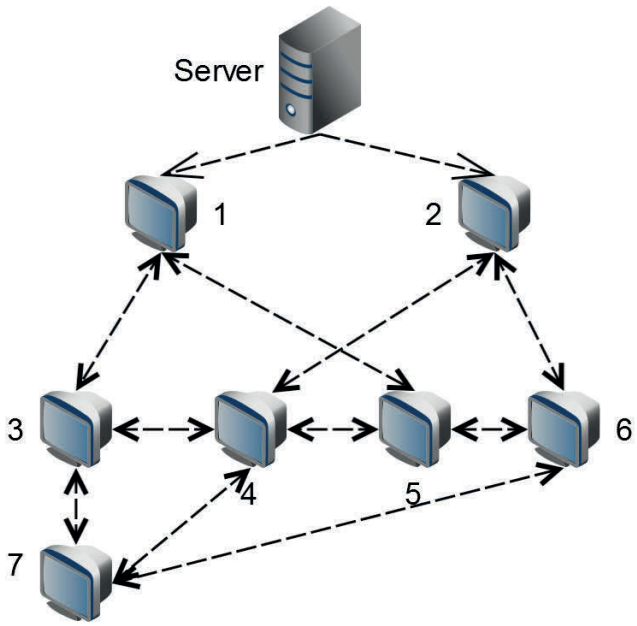
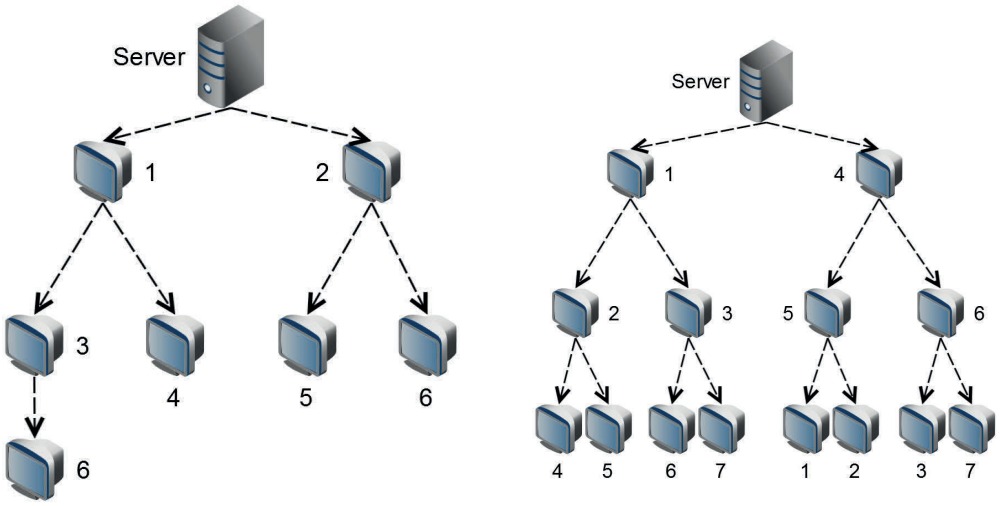


Figure 2.1: Mesh overlay.

P2P overlay structures are either mesh structure, tree structure or a hybrid of them. In mesh overlays, each peer establishes neighborhood relationships to random peers which are selected from the set of peers depending on the upload capacity and the content availability, Figure 2.1. Peers periodically exchanges the content availability with their neighborhood to "pull" missing content from them. Such a content distribution method is called the pull method which is widely used in mesh-based systems like PROMISE [18],

Chainsaw [19], CoolStreaming/DoNet [20] and PULSE [21]. In mesh-based systems, there is a trade-off between the delay and control overhead. More specifically, to achieve a short delay, peers need to exchange the content availability among each other within short intervals. That increases the control overhead and wastes peers' resources. By that reason, researchers are motivated to study the scheduling, i.e., which peers are better neighbors [22] and which missed content is given the priority to be pulled first [23, 24]. Thanks to the dynamic and random construction of the mesh overlay, it is robust against peer churns. However, it does not guarantee the quality of content distribution such as the delay, jitter, and the transmission overhead.



(a) Single Tree (b) Multiple Trees

Figure 2.2: Different types of tree-based overlay.

On the other hand, in tree-based systems, peers are organized in a tree-structured overlay as in Figure 2.2(a). The streaming content, which is "pushed" by the media server located at the root of the tree, is delivered to the downstream peers by repeating store-and-relay operation. Examples of tree-based systems include ESM [4], NICE [25], ZIGZAG [26], SpreadIt [27] and others. Tree-based systems are simple and efficient but are not robust against peer churns. A leave of a parent peer prevents all its descendants from

receiving the stream till they find new parents. In addition, trees could not fully utilize contributed resources since it does not use the upload capacity of leaf peers in the overlay.

Such drawbacks of tree-based systems can be overcome by adopting *multiple trees* as in SplitStream [28], CoopNet [29], ChunkySpread [30] and TURINStream [31]. SplitStream [28], as an example, divides a given stream into multiple substreams and delivers those substreams using a forest of trees; one tree for each substream. Peers are organized trying to use each peer as an internal node in at most one tree and as a leaf node in remaining trees, Figure 2.2(b).

To get the benefit of the two worlds, i.e., the mesh and tree overlays, some works adopted a hybrid structure [32, 33]. A state-of-art work is mTreebone [34] in which stable peers are organized in a tree-structure called treebone; a backbone overlay over which most of the streaming content is distributed. In addition, an auxiliary mesh overlay is organized between the stable and non-stable peers to compensate the weakness of the tree and to have a better resource utilization.

2.1.2 Content Distribution

After forming the overlay network, peers follow a content distribution method to deliver the content to every single peer. Mainly, there are three content distribution methods which are push method, pull method and the hybrid of them. In fact, the overlay structure plays an important role in selecting the distribution method. In tree overlays, it is straightforward to choose the push distribution method where parents forward the content to their children without an explicit request. For that reason, usually tree-based systems are called tree-push systems. In such systems, the main deal is how to construct and maintain the overlay, i.e., the parent-child relationships. In contrast, for the mesh overlay, the representative distribution method is the pull method. That is due to the random neighborhood relations in which the role of the parent or child is not explicit. However, as mentioned in the previous section, the pull method has some drawbacks such as the performance-control tradeoff.

To improve the performance while reducing the overhead in mesh overlays, different works approached a pull-push content distribution method as in [35, 36, 37]. Here, there is a distinction between the neighbor relation-

ship and parent-child relationship. More concretely, peers can select several neighbors as parents according to the upload capacity and the content availability, and establish a parent-child relationship to the selected neighbors. For example, in CoolStreaming [35], the video stream is divided into chunks, and chunks are arranged into several substreams, e.g. Fig 2.3. The content availability is represented by buffer maps storing the latest chunk received in different substreams. After exchanging buffer maps among neighbors, each peer can determine which substream to subscribe to and from which neighbor. To subscribe to a substream, a peer sends a single request (pull mode) to its neighbor. That leads to a parent-child relationship in which the parent will continue pushing the substream chunks to the subscribing peer. In this case, the child peer is responsible for monitoring the quality of the received substream to determine if changing the parent is necessary. Our work [12] is also based on the pull-push content distribution. It uses dynamic and pseudo-random substreams to improve the upload capacity utilization of peers.

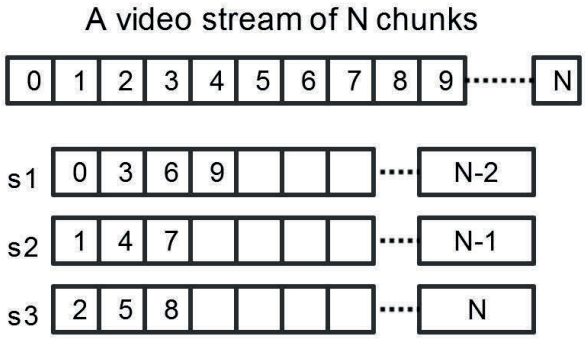


Figure 2.3: A simple example of a stream divided into three substreams.

Going beyond the hybrid pull-push, some works [38, 39, 40], designed a push-only content distribution method for the mesh overlay. That means a peer will autonomously determine which chunk should be pushed to which neighbor without explicit requests. That is to improve the stream continuity and reduce the overhead and delay. Yet, it is a challenging issue to avoid the redundancy of chunks received from different neighbors.

2.2 Performance Bounds

Researchers approached different P2P live streaming problems aiming to strengthen its success. For example, incentive mechanisms [41, 42, 43] are proposed to encourage peers to contribute their resources. That improves the performance and stability of the system. To improve the robustness of P2P live streaming and guarantee a good performance, the network coding is exploited in [44, 45]. Nevertheless, P2P live streaming performance bounds are still a hot topic to approach. The delay bounds are partially studied in [46], and the optimal broadcast time is discussed in [47, 48].

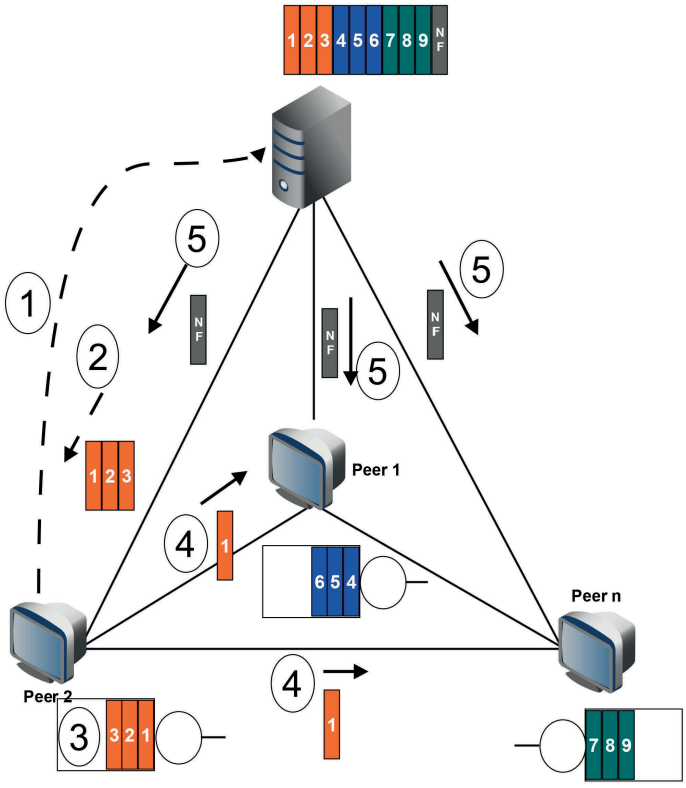


Figure 2.4: AQCS system design.

In relation to the topic of Chapter 3, the maximum achievable stream-

ing rate has been recently studied. In the following, we assume the system contains a media server in addition to n participant peers. In [49], an upper bound on the maximum streaming rate is derived for a fully connected network in which each peer is adjacent with any another peer. Let u_s denote the upload capacity of the media server and u_i denote the upload capacity of the i^{th} peer. Then, an upper bound $r^{\max}$ on the maximum streaming rate is given as

$$r^{\max} = \min\left\{u_s, \frac{u_s + U(P)}{n}\right\} \tag{2.1}$$

where $U(P) = \sum_i u_i$. The above formula indicates that $r^{\max}$ does not exceed the upload capacity of the media server, and in addition, it does not exceed the average upload capacity of peers and the media server. Different schemes have been proposed in [50, 51, 52, 53] to achieve the maximum streaming rate defined in Equation (2.1) for the fully connected overlay. As an example, in AQCS [50], the stream is to divide into several *chunks* of few kilo bytes. Those chunks are pulled/pushed from the media server to the peers, cached at the receivers' forwarding queues, and relayed from the receivers to their $n-1$ neighbors, see Figure 2.4. The upload capacity of each peer is "inferred" from the occupancy of its forwarding queue. When the occupancy is less than a threshold, the peer sends a pull signal to the server to acquire a new chunk (step 1 in the figure), and upon receiving the request, the server sends back three chunks to the requester (step 2), which will be stored in the forwarding queue and relayed to other peers (steps 3 and 4). When the server responds to all pull signals, it uploads one chunk to all peers (step 5) marked by "NF", which will not be relayed to the other peers.

However, it is an impractical solution in large scale systems as every peer needs to connect with all other peers. Moreover, those schemes cause either an excessive transmission overhead or long delivery delay as the number of peers increases. Then, more practical solutions are theoretically presented by Liu et al. with bounded peer out-degree per-tree, i.e., the number of children a peer can have in a single tree, in Snowball algorithm [54] and bounded total peer out-degree in Bubble algorithm [55]. A cluster-tree algorithm is proposed by same authors as a practical implementation of their theoretical study. However, the cluster-tree algorithm is not a fully distributed algorithm where a tracker is used to construct Bubble trees. Furthermore, the minimum number of peers in each cluster is kept a little bit large to achieve a higher

streaming rate, and hence a long fan-out delay. Authors in [56] studied the optimal streaming rate over general overlays with peer degree bounds (number of active connections) by using central solutions. Network coding and video coding schemes are also used in this regard. In [57], authors studied a network-coding based distributed solution to maximize the streaming rate for arbitrarily overlays and under peer degree bounds. In [58], the scalable video coding SVC is used to maximally utilize peers' resources.

Back to the multiple-tree systems, SplitStream [28] adopted a strategy to use each peer as an internal node in at most one tree and as a leaf node in remaining trees. By that reason, SplitStream highly utilizes the upload capacity of each peer, i.e., can broadcast a maximal streaming rate. However, it results in an inefficient overlay of degenerate trees when trying to maximally utilize the upload capacity of each peer. That will be discussed in more details in Chapter 3 where we proposed a new scheme [16] to attain the maximal resource utilization while maintaining an efficient overlay of multiple short-delay trees in a distributed manner. In our scheme, each peer contacts with only a small number of neighbors in the overlay network. Then, it autonomously subscribes to substreams according to a budget-model in such a way that realizes an efficient multiple-tree overlay.

2.3 Hybrid P2P Live Streaming

No thing is complete! The scalable and low cost streaming service offered by P2P never made it reliable. Peers join, leave and fail randomly and frequently, and contribute their resources voluntary. Such a dynamic nature makes it hard to guarantee the quality of streaming service (QoS) such as playback continuity and short playback delay [13]. On the other hand, content delivery networks (CDNs) are used to deliver guaranteed quality of video streaming, e.g., Youtube. But that comes with a high cost incurred by the huge number of dedicated servers needed to meet the demand of users. That motivates researchers to approach the hybrid of P2P with other infrastructures such as CDN or cloud computing. Such an approach inherits the reliability from the cloud or CDN and the scalability from the P2P. In fact, the hybrid architecture is applied into two kinds; 1) in the first kind, the P2P is augmented to the CDN or cloud to reduce the costs of dedicated servers and improve the scalability of the system. 2) in the second kind, the CDN is exploited to

assist the P2P to improve the reliability and guarantee the QoS.

2.3.1 Hybrid CDN-P2P Live Streaming

The hybrid of P2P with CDN is approached by many researchers [59, 60, 61]. A representative example of CDN-P2P is LiveSky [62] which is commercially deployed. In LiveSky, the CDN servers are organized in a tree-based overlay, Figure 2.5. The users are either served from the CDN only (legacy users) or directed to the P2P overlay. That is the decision of edge servers which also manage the P2P operations. The P2P overlay is localized per edge server and is a hybrid tree-mesh overlay. Another example is PROSE [27] which efficiently utilizes CDN resources by proactively injecting them into the P2P. More concretely, it divides CDN resources into two parts, and uses the first part to serve streaming content on demand. The second part is used to proactively push streaming content to several selected peers (super-peers) to improve the overall performance. A detailed analysis and comparison of the CDN-P2P systems is found in [64]. Unfortunately, provisioning resources in the CDN is semi-static, and such resources could be either insufficient (affects the QoS) or underutilized (extra cost) due to the highly dynamic P2P demand [65], and flash crowd cases [66].

2.3.2 Hybrid Cloud-P2P Live Streaming

Recently, cloud computing has become a promising platform offering an elastic resource allocation and cost-effective (pay-as-you-go) model. Some of the important resources offered by the cloud computing are, but not limited to, computing instances (virtual machines), storage service and cloud content delivery network CCDN. Different cloud providers are available and witnessing a huge success such as Amazon web services (AWS) [5], Microsoft Azure [6], IBM cloud [7] and others.

Researchers have not missed the chance and started to leverage the cloud computing for providing different services. A cost-effective content distribution is discussed in [67, 68]. Mobile users are supported by the cloud to improve the quality of streaming service in [69, 70]. Authors in [71] designed a

[5] www.aws.com
[6] azure.microsoft.com/
[7] www.ibm.com/cloud-computing

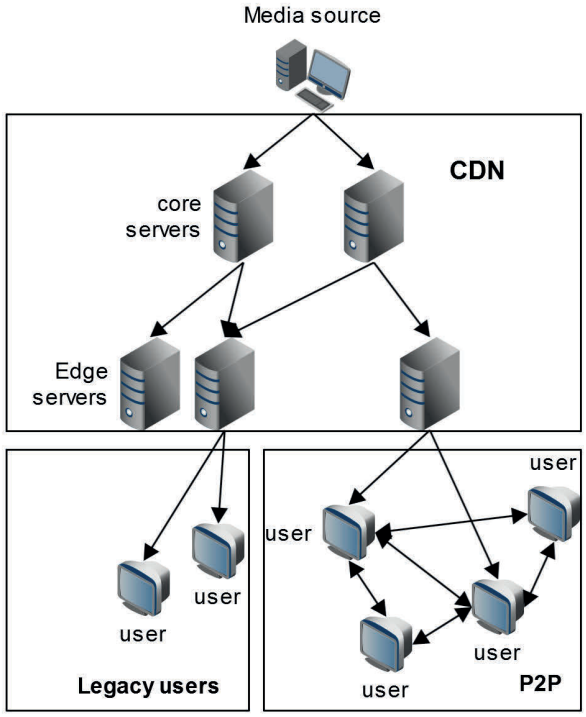


Figure 2.5: Hybrid CDN-P2P model in LiveSky.

migration strategy for video-on-demand (VoD) providers to partially migrate their videos to the cloud trying to minimize the cost. In [72], a mechanism is proposed to retrieve video segments' metadata over the cloud for VoD. CloudStraem [73] transcoded the videos to SVC (scalable video coding) over the cloud and delivered them in an adaptive manner to network dynamics through a cloud-based SVC-proxy. That lead researchers to focus on the efficient scheduling of user media requests to cloud servers. To reduce the response time and cost, the allocation of virtual machines to physical servers by considering the requirements of the cloud media service is discussed in [74]. Authors in [75, 76] developed a blind scheduling algorithm for mobile media cloud where no prior knowledge is available on the request rate and service time. Jointly, the algorithm ensures simplicity, fairness among servers and minimized user waiting time. To minimize the cost and assure the streaming

quality from a video provider perspective, researchers studied different mechanisms to allocate cloud resources based on the dynamic demand. In [77, 78] provisioned resources in the cloud are dynamically scaled up and down to meet the user demands in VoD. Authors in [79] studied how to optimally procure virtual machines for VoD where the cost and quality of experience trade-off is investigated under Amazon pricing models.

The research community also showed a great interest in exploiting the flexible resource allocation of the cloud to solve the issues of CDN-P2P live streaming. A general abstraction of the cloud-p2p model is illustrated in Figure 2.6. In general, the cloud assistance is realized by storing latest chunks in the live stream to the storage service in the cloud. In addition, each peer is allowed to fetch missing chunks from the storage service through edge locations of the CCDN. These edge locations fetch chunks from the storage service upon receiving requests from peers. Another way for the assistance of P2P by the cloud is to rent computing instances and to use them as virtual peers to disseminate live stream to the existing peers. The cost of cloud assistance is comprised of the amount of data fetched from the CCDN and the number of requests handled by the cloud. The latter cost is further divided into requests handled by the storage service and requests handled by the CCDN. In the case of renting computing instances, the cost is comprised of the amount of data fetched and number of hours instances are run. A main goal of the work in this direction is to minimize the monetary cost of the cloud-P2P live streaming while providing a guaranteed QoS.

CALMS [80] presented a general framework to migrate the live streaming service to the cloud. The cloud servers are rented dynamically from different regions based on a demand prediction mechanism. AngleCast [14] is a typical P2P-assisted cloud live streaming system. It deploys computing instances (EC2) called angels on-demand to maintain a predefined high streaming rate. These angels are augmented with an optimized multiple-tree P2P overlay to reduce the cost of cloud resources. A central entity "registrar" is used to orchestrate the multiple-tree overlay. Clive [15] is a cloud-assisted P2P based on a mesh-structured P2P overlay. To guarantee the streaming quality, it rents helpers from the cloud, which are either active or passive (i.e., computing instances or storage service). Clive determines which and how many helpers to be rented so as to maximize the system performance while bounding the cloud cost. Most recently, VMCAST [81] is proposed as a stability enhancing solution for the tree-based multicast overlays based on

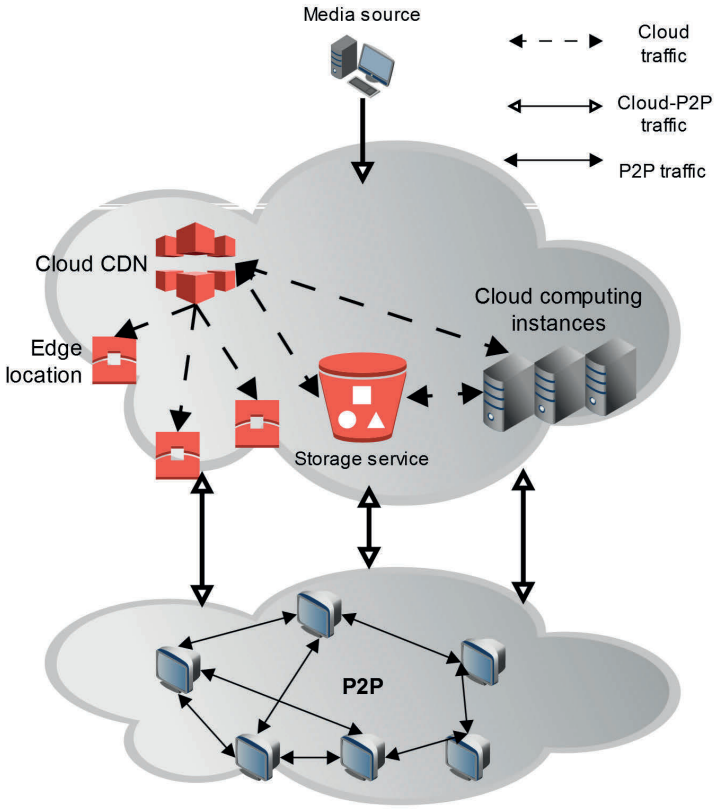


Figure 2.6: Hybrid Cloud-P2P model.

the cloud virtual machine assistance.

However, the rent of computing instances is less flexible than the rent of upload bandwidth (e.g., the minimum rent period of computing instances is generally one hour). Thus to refine the cost of cloud assistance while keeping the high performance, we exploited only the storage service and the CCDN and proposed a cost effective cloud-assisted P2P live streaming system [17]. The details of our method are presented in Chapter 4 where three techniques are proposed to reduce the cost of such a cloud assistance based on a multiple-tree P2P overlay. Unlike mesh-based P2P overlay as in Clive [15], multiple-tree P2P overlay has the following flaws while it has

advantages such as the short playback delay and low overhead: 1) the chunk size is smaller than in mesh-based overlays which increases the number of requests submitted to the cloud to acquire missing chunks; and 2) the leave of a peer causes suspension of a video stream at descendants of the leaving peer which also causes the increase in the number of requests submitted to the cloud. The proposed method scales the bandwidth resources flexibly by asking more/fewer peers to proactively fetch the streaming content from the CCDN rather than renting computing instances. What is more is that the overlay is maintained in a distributed manner with the assistance of the storage service in contrast to the central management of AngleCast [14].

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 3

Maximal Resource Utilization

A key issue to realize an efficient content distribution in the P2P environment is how to maximize the utilization of resources contributed by the participants. Note that by the efficient maximal utilization of resources, we can improve the quality of streaming service, for instance, broadcast a maximal streaming rate with a short end-to-end delay. While if the streaming rate is fixed, the higher utilization of peers' resources reduces the required server bandwidth to deliver the stream to the whole population which reduces the cost. To maximally utilize the resources, we need to guarantee that all peers are engaged in the content distribution process. Then, an efficient peering and content scheduling strategy is needed which is expected to overcome both of the upload capacity bottleneck and the content bottleneck in the underlying P2P systems. Finding such a strategy becomes more difficult when the capacity of the P2P system is barely enough to broadcast the streaming content to all peers, i.e., the spare capacity to adopt future peers is very low.

In this chapter, we present a new scheme for the maximal resource utilization. We adopt the multiple trees approach and designed a scheduling scheme that works in the challenging condition where no spare capacity is available. The scheme attains the maximal resource utilization while maintaining an efficient overlay of multiple short-delay trees in a distributed manner. For that purpose, a budget-model is used such that each peer has a budget relative to its upload capacity and corresponds to the maximum number of children that peer can have. In the scheme, the role of uploading a substream is transferred to another peer by exchanging money among peers, provided that the balance of each peer is not below zero. A newly joining peer tries to instantly

spend its budget to subscribe to substreams in such a way that guarantees a high number of peers forwarding exactly one substream and that the trees have a short hop-count delay. The proposed scheme is also able to broadcast the maximal streaming rate as it is able to attain the maximal resource utilization.

By simulation, we show that the proposed scheme overcomes the drawbacks of conventional live streaming schemes. The simulation result indicates that under the proposed scheme, the overlay network certainly converges to an efficient structure with a short hop-count delay. Moreover, it indicates that the proposed scheme gives nice features in the homogeneous case, i.e., when all peers have an equal upload capacity.

The remainder of this chapter is organized as follows. Section 3.1 explains the background of the maximal resource utilization problem. It introduces the problem in both mesh and tree overlays. Then, Section 3.2 describes the proposed scheme in details. Section 3.3 presents simulation results, and, finally, Section 3.4 concludes the chapter.

3.1 Problem Definition & Background

As mentioned in Chapter 2, the upper bound on the maximum streaming rate was derived by Kumar *et al.* [49] in a fully connected overlay. The fully connected overlay is an ideal P2P overlay in which each peer is adjacent with any another peer in the system. Let us assume that the system contains a media server in addition to n participant peers. Let u_s denote the upload capacity of the media server and u_i denote the upload capacity of the i^{th} peer. Then, an upper bound $r^{\max}$ on the maximum streaming rate is given as

$$r^{\max} = \min\left\{u_s, \frac{u_s + U(P)}{n}\right\} \tag{3.1}$$

where $U(P) = \sum_i u_i$.

Kumar *et al.* proposed an optimal scheme which tightly attains that upper bound. In the scheme, the media server divides the given stream into several uneven substreams and feeds each substream to a designated peer in the system. Each substream fed by the server will be propagated to all peers in the system by repeating store-and-relay operation. The detail of the scheme is as follows.

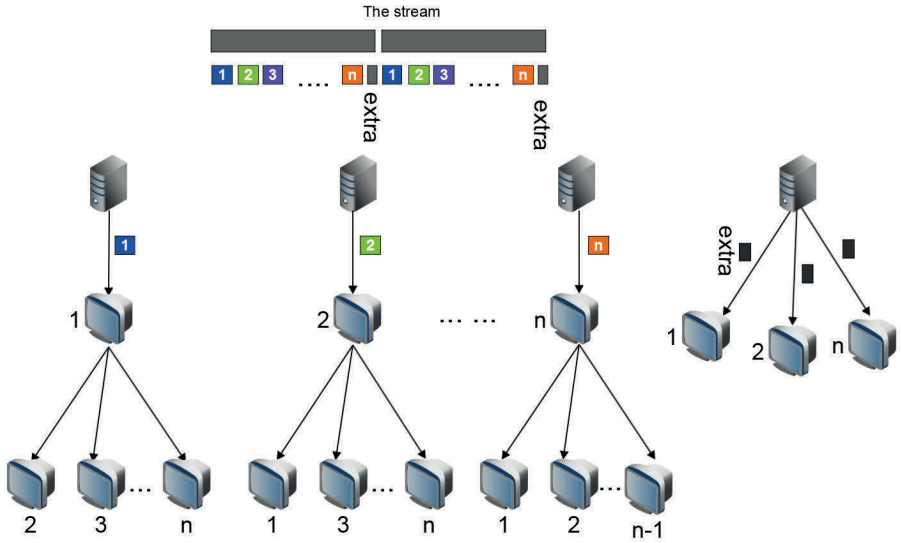


Figure 3.1: Trees constructed by Kumar *et al.* scheme over fully connected overlay network.

- When $r^{\max} = u_s$, the stream of bit rate $r^{\max}$ $(= u_s)$ is divided into n substreams so that the bit rate of the i^{th} substream is $s_i = \frac{u_i}{U(P)} \times r^{\max}$. Then the server feeds the i^{th} substream to the i^{th} peer using the first n trees in Figure 3.1. The reader could easily verify that the aggregate rate of n substreams is $r^{\max}$ and the i^{th} peer can simultaneously feed its substream to the other $(n-1)$ peers since $(n-1)s_i \leq u_i$.

- When $r^{\max} < u_s$, the given stream of bit rate $r^{\max}$ $(= \frac{u_s + U(P)}{n})$ is divided into $n+1$ substreams so that $s_i = \frac{u_i}{n-1}$ for each $1 \leq i \leq n$ and $s_{n+1} = (u_s - \frac{U(P)}{n-1})/n$. Then the server feeds two substreams to peer i, i.e., the i^{th} substream using the first n trees in Figure 3.1, and a copy of the $(n+1)$st substream using the last tree in Figure 3.1. The reader could easily verify that the i^{th} peer can simultaneously feed its substream to the other $(n-1)$ peers, since the aggregated bit rate does not exceed u_i.

With this scheme, each peer can successfully receive the full stream without

causing an overload of the participant peers. However, as the number of peers increases, the bit rate of each substream becomes quite low. Hence, especially for the peers with low upload capacity, it incurs an excessive transmission overhead due to a large fraction of packet headers. AQCS [50], explained in Chapter 2, also presented a different method to attain the maximum streaming rate under the fully connected overlay. However, the assumption of each peer is connected to all other peers is not practical and does not scale up. A more practical model is when the out-degree of each peer (number of neighbors) is bounded. In the next two subsections, we will discuss the bounded peer's out-degree in both mesh and tree overlays. Then, we reach to our final prospective design in subsection 3.1.3

3.1.1 Mesh-based Overlays

As explained in Chapter 2, in mesh based protocols such as CoolStreaming/DoNet [20], the live stream is divided into equal chunks. Each peer establishes a neighborhood relation with a bounded number of peers. Then, it exchanges the availability of chunks in its buffer using buffer maps, and tries to "pull" missing chunks from its neighbours. By decreasing the time interval for the exchange of buffer maps, the delay is reduced. However, it significantly increases the control overhead per chunk. To overcome such drawbacks of conventional protocols, hybrid pull-push protocols have been proposed. In Coolstreaming [35] proposed by Xie et al., data chunks are arranged into several substreams, and each peer can acquire substreams from its neighbours by establishing parent-child relationships. Here, buffer maps stores the latest chunk received in different substreams. After exchanging buffer maps among neighbors, each peer can determine which substream to subscribe to and from which neighbor. To subscribe to a substream, a peer sends a single request (pull mode) to its neighbor. That leads to a parent-child relationship in which the parent will continue pushing the substream chunks to the subscribing peer. In Coolstreaming [35], the substream rate is fixed and neighborhood relations are random. Such a scenario causes a problem described as follows. Since each peer can have a limited amount of upload capacity, it could accommodate a limited number of children. Thus, it easily causes a competition among children, and the loser of such a competition should try to find another parent after being refused by the former parent. Such a wandering behavior of peers will not stabilize the overlay,

which will significantly degrade the overall performance and waste resources of the P2P streaming system.

To overcome such an instability issue of the overlay network and improve the resource utilization, we developed a chunk scheduling scheme for pull-push method based on the notion of randomization [12]. In contrast to CoolStreaming [35], substreams are dynamically (and randomly) generated by each peer so that the load of parents with respect to the upload capacity will be balanced. In our work [12], each chunk is associated with two important information. The first one is the sequence number which is necessary to order received chunks in the buffer. The second one is a pseudo random number in the range [1; 100] which is generated by the server (in the following, we will say that the chunk is "tagged with" a random number). The attached tag is used to deliver the chunk through different routes while avoiding duplication and starvation. Forwarding of tagged chunks is conducted by referring to a forwarding table which is a set of rules designating "which chunk should be forwarded to which child". Each entry of the table provides two information, i.e., a subrange of [1; 100] and a set of children associated with the subrange. The selection of chunks to be forwarded to a child is conducted in a probabilistic manner at the time of constructing a forwarding table as follows:

At first, each peer calculates the upload capacity availability, i.e., the ability of forwarding a substream to the children. It is calculated by dividing the number of children which can be supported without delay by the number of children which are currently being accommodated. For example, if a peer can accommodate 3 children with a full substream rate and it currently accommodates 10 children then the upload availability is calculated as 0.3 (= 3/10).

After receiving upload availabilities from parents, a child partitions the range [1; 100] into subranges so that each subrange is associated with a parent and has a length proportional to the upload availability corresponding to the parent. That is done by calculating the weight of each parent so that it is proportional to the upload availability. More concretely, by using upload availability p_i of parent i, the weight of parent i is calculated as

$$w_i := \frac{p_i}{\Sigma_i p_i}.$$

For example, if the bandwidth availability of four parents are $(p_1, p_2, p_3, p_4) =$

$(0.2, 0.25, 0.3, 0.3)$, the weight of those parents are calculated as

$$(w_1, w_2, w_3, w_4) = (0.19, 0.238, 0.286, 0.286).$$

After that, the peer divides range $[1, 100]$ so that the length of each subrange is proportional to the weight of each parent, and identifies the first integer in each subrange. In the above example, those integers are identified as $1, 20, 42$ and 72.

Finally, the peer sends tuples of the first and the last integers of the subrange to the corresponding parent; e.g., parent 2 receives $(20, 41)$ from the peer. After receiving those ranges from children, each peer constructs (or updates) a forwarding table. By adopting such a probabilistic, dynamic decision making, we could effectively avoid a high concentration of requests to specific neighbor and could improve the efficiency of resource utilizations.

3.1.2 Tree-based Overlays

A basic idea to bound the peer's out-degree in tree overlays while trying to maximize the resource utilization is to divide the given stream into $\lceil \frac{r^{\max}}{s} \rceil$ substreams with fixed bit rate s. Here, the bit rate of the last substream is given as $s' = r^{\max} - (\lceil \frac{r^{\max}}{s} \rceil - 1)s \leq s$. Let N denote the number of resulting substreams, i.e., $N \stackrel{\text{def}}{=} \lceil \frac{r^{\max}}{s} \rceil$. Those substreams are delivered to the participants through different spanning trees, i.e., N edge-disjoint spanning trees $T_1, T_2, \ldots, T_N$ is prepared. Then, the server feeds the i^{th} substream to the root v_i of the i^{th} spanning tree T_i. The set of spanning trees is designed so that 1) the majority of peers are internal peers in only one tree and leaf peers in the remaining trees and 2) any internal peer in each tree has a bounded out-degree.

Let $V = \{0, 1, 2, \ldots, n-1\}$ be the set of peers. We will construct a set $\mathcal{T}$ of spanning trees in such a way that each peer i is responsible to forward only one substream to $\eta(i) \stackrel{\text{def}}{=} \frac{u_i}{s}$ neighboring (downstream) peers, where u_i is the upload capacity of i and s is the bit rate of a substream. A scheme to construct the set $\mathcal{T}$ is described as follows. At first, it prepares N empty bins $I_1, I_2 \ldots, I_N$, and tries to pack as many peers in V into those bins as possible, subject to:

- each peer is packed into at most one bin and

- $\sum_{i \in I_j} \eta(i)$ does not exceed $n-1$, for each $1 \leq j \leq N$.

Let $\tilde{V} := V - \bigcup_{1 \leq j \leq N} I_j$ be the set of residual peers after the packing. By definition, for any I_j and for any $k \in \tilde{V}$, it holds

$$\sum_{i \in I_j} \eta(i) + \eta(k) > n - 1. \tag{3.2}$$

Let us assume $\sum_{i \in I_j} \eta(i) < n-1$ holds for any j, without loss of generality (otherwise, we can easily construct a tree to have I_j as the set of internal peers using a procedure described later). Thus in order to construct a tree with internal peers I^* with $I_j \subset I^*$ and $\sum_{i \in I^*} \eta(i) = n-1$ for each j, we need to recruit peer(s) from $\tilde{V}$ which provide an upload capacity of amount $\Delta_j \stackrel{\text{def}}{=} n-1-\sum_{i \in I_j} \eta(i)$ to tree T_j. In a centralized environment, we can accomplish such a recruitment using a simple greedy algorithm. More particularly, we can attain an assignment such that each peer in $\tilde{V}$ contributes to at most two trees as an uploader.

Given I_j and a peer $k \in \tilde{V}$ contributing an upload capacity of amount Δ_j, T_j is constructed in the following manner.

- Peers in $I_j \cup \{k\}$ organize a tree structure T'_j such that: 1) each peer i has at most $\eta(i)$ children and 2) the distance from the root to the furthest peer is as short as possible. A reasonable heuristic to realize such a configuration is to start from a tree consisting of a single peer with a maximum upload capacity and to successively connect peers to the tree in an descending order of the upload capacity.

- Since the upload capacity of $I_j \cup \{k\}$ is $n-1$, after organizing such a tree with $|I_j|+1$ peers, there remains an upload capacity of amount $n-1-|I_j|$. Thus, T_j can be constructed from T'_j by connecting peers in $|V|-(I_j \cup \{k\})$ to peers in T'_j until the upload capacity is exhausted.

If $u_s = r^{\max}$, it is possible to pack N bins (i.e., trees). Then, we can attain the maximum streaming rate by feeding N substreams from the media server to the roots of N resulting trees. The aggregated bit rate $r^{\max}$, since those substreams can be forwarded to the downstream peers in each tree by repeating store-and-relay operation. However, if $u_s > r^{\max}$, such a basic approach does not work well, since the derivation of the upper bound $r^{\max}$

assumes that the media server "fully" utilizes its upload capacity. That means the capacity of size $u_s - r^{\max}$ is not used in such a simple approach, and hence, we can pack $N'(< N)$ trees only.

To overcome this issue, when $u_s > r^{\max}$, we virtually separate the role of the media server into two sub-servers, say σ_a and σ_b, and use those sub-servers in the following manner:

- σ_a packs N' trees and feeds N' substreams with bit rate s to the roots of packed trees, where each substream is disseminated to $n-1$ peers according to the basic scheme, and
- σ_b disseminates the remaining substream to all peers using the remaining upload capacity at the server and the capacity of peers in $\tilde{V}$ (after packing N' trees), if any.

In this case, the server packs N' bins only where the value of N' is determined as follows. Let u_a and u_b denote the upload capacity of σ_a and σ_b, respectively. Note that $u_a + u_b = u_s$ by definition. The value of u_a is determined to satisfy

$$u_a = \frac{u_a + U(P)}{n},$$

i.e., $u_a := \frac{U(P)}{n-1}$, and given bit rate s, N' is determined as

$$N' := \left\lceil \frac{u_a}{s} \right\rceil.$$

Note that $N - N'$ is generally small for large n's but is not negative since

$$N - N' \;=\; \left\lceil \frac{u_s + U(P)}{sn} \right\rceil - \left\lceil \frac{U(P)}{s(n-1)} \right\rceil \;\geq\; 0$$

for any $n \geq 1$ where the last inequality is due to $u_s > \frac{U(P)}{n-1}$.

Clearly, a larger $\eta(i)$ reduces the height of the resulting tree, which results in a short end-to-end delay and a high resilience to "peer churns" as it reduces the number of ancestors of each peer. By taking into account the heterogeneity of the upload capacity, we bound the maximum out-degree of peers as $\max_i\{\eta(i)\} \leq d_{max}$ and bound the minimum out-degree as $\min_i\{\eta(i)\} \geq d_{min}$. Then, we have $d_{max} = \frac{\max_i u_i}{\min_i u_i} d_{min}$. By choosing appropriate bounds d_{min} and d_{max}, the bit rate of substreams is determined as

$$s \;=\; \frac{\max_i u_i}{d_{max}} \;=\; \frac{\min_i u_i}{d_{min}}.$$

3.1.3 Final Prospective Design

In Section 3.1.1, the hybrid pull-push protocol in mesh overlays can efficiently reduce both the delay and the overhead. Moreover, mesh based protocols are robust against peer churns. Yet, the dynamic construction of random overlay does not guarantee the quality of content distribution such as the delay, resource utilization, and the transmission overhead. On the other hand, multiple-tree overlays are structured, simple and efficient, and exhibit a good performance with respect to the broadcasting time [47, 48] and the maximum streaming rate [46, 16]. So, researchers' preference will be for multiple-tree overlays when a maximal resource utilization or a guaranteed service is the target to be achieved.

Although the previous design of multiple-tree overlay (in Section 3.1.2) can attain the maximum streaming rate, it is still hard to be implemented as the degree of the server is still unbounded; that is when $u_s > r^{\max}$. Another problem is that the cost of adding a new peer to the overlay is generally high since it would significantly change the structure of the overlay (the reader can imagine a situation in which by adding a new peer, the decreasing order of peers in a bin k is changed, which causes a significant change of the structure of the kth tree). Moreover, such a design needs a global knowledge of the whole population, i.e., requires a central solution, to construct the set of trees. To reach an implementable model, we fix the number of substreams N by selecting a fixed substream rate s. Here, we assume that the maximum streaming rate is fixed over time. In fact, this is a soft assumption as, from equation 3.1, the maximum streaming rate converges to a fixed value as the number of peers increases. In this case, when the upload capacity of the server u_s is greater than the streaming rate $N \times s$, the extra capacity, i.e., $u_s - N \times s$ will be wasted. So, the server does not broadcast a substream rate of $(u_s - N \times s)/n$ to all peers to bound the out-degree of the server. However, such a substream rate limits to zero when the number of participants n is high. Then, the goal is to construct a set of spanning trees in such a way that maximize the utilization of peers' resources in a distributed manner. Furthermore, the overlay should be stable, i.e., with a minimal change due to new arrivals, and efficient in terms of delay.

SplitStream [28], as a related work, with the aid of Scribe [82] ensures that trees have a disjoint set of internal peers. When a peer joins the SplitStream system, it selects random peers, and asks them to be their parents. If a peer

can not adopt more children and receives a join request from another peer, one of its children will be rejected according to the value of a utility function. Rejected child seeks for another parent by referring to its previous siblings or to a set of peers with excess upload capacity called the spare capacity group. Thus, SplitStream is able with a high probability to reorganize a collection of trees even when the upload capacity of each peer is fully used, and hence broadcasting the maximal streaming rate for a large population. However, especially in our case of research when trying to maximally utilize the upload capacity of each peer, i.e., the spare capacity is low, the mechanism of rejecting children will frequently happen. So, in many cases, peers will not find a peer as a parent before asking the spare capacity group; this will be verified later in this chapter. Thus, the search for spare capacity peers happens frequently which is a time consuming process. The more important consequence of asking the spare capacity peers is that those peers will have a small number of children in different trees leading to degenerate trees.

To overcome this problem, we need to propose a new scheme specifically optimized to utilize the peer's resources. That scheme should be able to maintain an efficient overlay structure by allowing peers to change their position or their set of children.

3.2 Proposed Scheme

3.2.1 Preliminaries

Notions used in the proposed scheme are summarized in Table 3.1. In the proposed scheme, we divide the given video stream with bit rate r into N substreams of bit rate $s = r/N$ each, and deliver those substreams through different spanning trees as in SplitStream. Thus in the following, we will use terms "tree" and "substream" interchangeably.

Let V be a set of n peers and $\mathcal{T} = \{T_1, T_2, \ldots, T_N\}$ be a variable set of N trees (substreams). Each peer $i \in V$ can have different number of children in each tree in $\mathcal{T}$, while the total number of children should not exceed a value $m(i)$ determined by the upload capacity of the peer. In the following, we call $m(i)$ the **budget** of peer i, and will design a scheme such that the role of uploading a substream is transferred to another peer by exchanging money among peers, provided that the balance of each peer is not below zero.

Given collection of trees $\mathcal{T}$, the **price** of peer i with respect to the k^{th} tree T_k is defined as the number of children of i in T_k plus one. Such N prices of peer i are locally stored in the form of a price vector C_i of length N. Note that $C_i[k] \geq 1$ for any i and k. Given collection of trees $\mathcal{T}$, a peer is said to be **saturated** if it has the maximum number of children in only one tree. More particularly, peer i is saturated with respect to the k^{th} tree T_k if $C_i[k] = m(i) + 1$ and $C_i[h] = 1$ for all $h \neq k$. T_k is said to be a **dominant substream** for peer i if $C_i[k] = \max_k C_i[k]$.

Table 3.1: Main notions of Chapter 3.

n:	number of peers
N:	number of substreams (trees)
r:	streaming rate
s:	substream rate
$m(i)$:	budget of peer i (maximum number of children)
C_i:	price vector of peer i
$C_i[k]$:	price of peer i in kth tree
$\beta(i)$:	balance of peer i
U_i:	set of neighbors of peer i
D:	number of neighbors
P_i:	set of substreams not subscribed by peer i

3.2.2 Tree Reconfiguration

Suppose that each peer is associated with a set of D random peers (neighbors) by the tracker. Let U_i be a subset of peers associated with peer i. In the proposed scheme, i can subscribe to a (new) substream by communicating with peers j in U_i. The concrete scheduling algorithm, the details of which will be described in Section 3.2.3, is based on three ways of re-configuring trees in $\mathcal{T}$ (see Figure 3.2 for illustration). If i could not finish the scheduling due to the lack of resources in U_i, it contacts peers in a set of peers with free capacity, the detail of which is described in Section 3.2.5.

Way-1: The first way of re-configuring $\mathcal{T}$ is to use the free upload capacity of a peer. More concretely, if peer $j \in U_i$ is subscribing to the k^{th}

substream and has a free upload capacity, then peer i can subscribe to the k^{th} substream by making itself as a child of j in T_k (note that such an action decreases the balance of j by one). See Figure 3.2 (A) for illustration. If there are several such pairs of parent and substream, i prefers to a pair of j and k such that k is a dominant substream of j. That makes j closer to the saturation since the join of i to j in T_k increases the value of $C_j[k]$ by one.

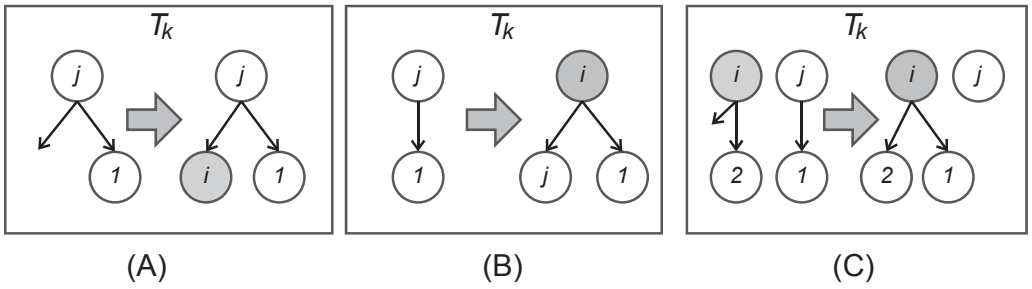


Figure 3.2: Three ways for reconfiguring tree T_k. (A) peer i can be a child of j simply because j has a free capacity. (B) peer i can buy a substream T_k from peer j by paying a price of 2, and hence peer i has two children. (C) both peers i and j have one child in the same tree T_k so peer i asks j to swap his child.

Way-2: The second way is to buy the right of uploading substreams by paying money. More particularly, if j is subscribing to the k^{th} substream and i has a positive balance, then by paying money of amount m (≥ 1) to j, i can subscribe to the k^{th} substream by taking the place of j in T_k and is granted the right of uploading the substream to j and its $m-1$ children. Figure 3.2(B) illustrates this case. To not reduce the number of saturated peers, the only restriction in this case is that j must not be saturated with respect to the transferred substream.

Way-3: The third way is to swap children with other peers. Suppose that there are two peers i and j subscribing to the k^{th} substream, where: 1) i has a free upload capacity while T_k is its dominant substream and 2) j has at least one child in T_k but it is not a dominant substream for j. Then, i asks j to hand over the right of uploading the k^{th} substream to one child of j in T_k by paying one unit of money. This way is exemplified in Figure 3.2(C).

3.2.3 Scheduling Process

In the proposed algorithm, peers subscribe to substreams through three phases. The role of the first is to subscribe to substreams taking into account to increase the number of peers that are internal in only one tree, i.e., to increase the number of saturated peers. The role of the second phase is to complete the subscription to all substreams. If the peer has a balance after the second phase, it executes the third phase to increase the number of its children for the dominant substream.

First Phase: Let P_i be a variable representing the set of substreams which is not subscribed by the peer i. For each $q \in P_i$, peer i counts the neighbors who have children in tree q but q is not their dominant substream. Then, those neighbors are willing to reject children in q and adopt children in their dominant substream. Let $q^* \in P_i$ be a substream with a maximum count and U_i^* ($\subseteq U_i$) be the set of neighbors contributing to the count of q^*. Both q^* and U_i^* could be calculated as in Scheduling Alg. 3.1. To increase the number of peers uploading exactly one substream, peer i conducts three steps of the first phase as follows (pseudo-code is shown in Scheduling Alg. 3.2 where all details related to hop-count are omitted for simplicity).

Step 1: Let $\beta(i)$ be the balance of peer i. This step is executed only when $\beta(i) \geq 1$. Peer i selects a peer $j \in U_i^*$ with a shortest depth from the root in the tree corresponding to q^*, and buys q^* from j by paying money. Peer i either pays one unit of money and subscribes to q^* (Way-2) or pays more than one unit of money and subscribes to the dominant substream of j (Way-1) in addition to q^* (Way-2). In the latter case, the paid money should not exceed $\beta(i) - (|P_i| - 2)$, since we need to reserve money for the remaining $|P_i| - 2$ unsubscribed substreams. The reader should note that q^* is the potential dominant substream for peer i.

Step 2: For each $j' \in U_i^* \setminus \{j\}$, peer i gets one child of j' for substream q^* (Way-3) and subscribes to the dominant substream of j' (Way-1), if i has not yet subscribed to it. Note that substream q^* should be commonly subscribed by i and j' but is not a dominant substream of j'.

The last option for peer i to increase the number of children for the dominant substreams of peers in U_i is to look for peers that have a free capacity and subscribe to their dominant substreams. Thus, we have the third step:

Scheduling Alg. 3.1: Calculating q^* and U_i^* by peer i

1: $N \leftarrow$ Number of substreams (subs.).
2: $C_j[N] \leftarrow$ cost vector of peer j.
3: $P_i \leftarrow N$ set of unsubscribed subs.
4: $U_i \leftarrow$ set of neighbors of peer i.
5: $\beta(i) \leftarrow$ balance of peer i.
6: $F[N] \leftarrow$ Array represents frequency of non-dominant subs.
7: $I[N] \leftarrow$ Array of neighbor sets. // I[k] is a set of neighbors who have sub. k as non-dominant.

8: **for each** sub q **in** P_i **do**
9: **for each** peer j' **in** U_i **do**
10: **if** $C_{j'}[q] > 1$ and q is not dominant **then**
11: $F[q] \leftarrow F[q] + 1$
12: $I[q] \leftarrow I[q] \cup \{j'\}$
13: **end if**
14: **end for**
15: **end for**
16: $q^* \leftarrow max_q F[q]$
17: $U_i^* \leftarrow I[q^*]$

Scheduling Alg. 3.2: First Phase

Step 1
Ensure: $\beta(i) > 0$
Ensure: $U_i^* \neq \emptyset$
1: select a peer j from U_i^* with shortest depth.
2: i subscribes to q^* by buying j (Way-2)
3: $P_i \leftarrow P_i \setminus \{q^*\}$
4: $\beta(i) \leftarrow \beta(i) - 1$
5: **while** $\beta(i) - |P_i| - 1 > 0$ and $C_j[q^*] > 1$ **do**
6: i gets a new child from j for q^* (Way-3)
7: $\beta(i) \leftarrow \beta(i) - 1$
8: $DO_j \leftarrow$ is the dominant sub of j
9: **if** $DO_j \notin P_i$ **then**
10: i subscribes to DO_j from j (Way-1)
11: $P_i \leftarrow P_i \setminus \{DO_j\}$
12: **else**
13: $\beta(j) \leftarrow \beta(j) + 1$
14: **end if**
15: **end while**
16: **IF** $\beta(i) = 0$ **goto Step 3**
Step 2
Ensure: $\beta(i) > 0$
Ensure: $U_i^* \neq \emptyset$
17: **for each** peer j' **in** $U_i^* \setminus \{j\}$ **do**
18: **if** $C_{j'}[q^*] > 1$ **then**
19: i gets a new child from j' for q^* (Way-3)
20: $\beta(i) \leftarrow \beta(i) - 1$
21: **if** $DO_{j'} \in P_i$ **then**
22: i subscribes to $DO_{j'}$ from j' (Way-1)
23: $P_i \leftarrow P_i \setminus \{DO_{j'}\}$
24: **else**
25: $\beta(j') \leftarrow \beta(j') + 1$
26: **end if**
27: **IF** $\beta(i) = 0$ **goto Step 3**
28: **end if**
29: **end for**
Step 3
Ensure: $P_i \neq \emptyset$
30: **for each** peer j' **in** U_i **do**
31: **if** $\beta(j') > 0$ and $DO_{j'} \in P_i$ **then**
32: i subscribes to $DO_{j'}$ from j' (Way-1)
33: $P_i \leftarrow P_i \setminus \{DO_{j'}\}$
34: $\beta(j') \leftarrow \beta(j') - 1$
35: **end if**
36: **end for**

Step 3: If there is a peer $j \in U_i$ such that j has a free capacity and i has not subscribed to a "dominant" substream q of j, then i becomes a child of j with respect to q. This operation is repeated until there is no such peer j in U_i.

Scheduling Alg. 3.3: Second Phase

Step 4
Ensure: $\beta(i) > 0$
Ensure: $P_i \neq \emptyset$
$P_{oq} \leftarrow$ represents a peer with the lowest cost of a sub q
1: **for each** sub q **in** P_i **do**
2: select a peer P_{oq} from U_i
3: **if** $\beta(i) \geq C_{P_{oq}}[q]$ and $DO_{P_{oq}} \neq q$ **then**
4: i subscribes to q by buying P_{oq} (Way-2)
5: $P_i \leftarrow P_i \setminus \{q\}$
6: $\beta(i) \leftarrow \beta(i) - C_{P_{oq}}[q]$
7: **IF** $\beta(i) = 0$ **goto Step 5**
8: **end if**
9: **end for**
Step 5
Ensure: $P_i \neq \emptyset$
10: **for each** sub q **in** P_i **do**
11: **for each** peer j' **in** U_i **do**
12: **if** $\beta(j') > 0$ **then**
13: i subscribes q from j' (Way-1)
14: $P_i \leftarrow P_i \setminus \{q\}$
15: $\beta(j') \leftarrow \beta(j') - 1$
16: **end if**
17: **end for**
18: **end for**
19: **IF** $P_i \neq \emptyset$ **contact the free set.**

Second Phase: The second phase, Scheduling Alg. 3.3, is executed when peer i could not subscribe to all substreams after the first phase being finished. At any step in this phase, if P_i becomes empty, peer i proceeds to the third phase to increase the number of children for its dominant substream (q^*). The steps are as follows.

Step 4: For each unsubscribed substream $q \in P_i$, peer i seeks a peer $j \in U_i$ such that $C_j[q]$ is the cheapest among all peers in U_i and q is not the dominant substream of j. Then, peer i buys q from j by paying money (Way-2). A draw in prices is resolved by the hop-count delay. Peer i looks for the cheapest price to save the money to get more children in its dominant substream q^*.

Step 5: At this point, the budget of peer i is exhausted. Thus in order to subscribe to a new substream in P_i, i needs to use the free capacity of other peers in U_i (Way-1). Recall that the use of the free capacity of peer j does not decrease the balance of i, but it decreases the balance of j because it reduces the amount of free capacity of j. If there is no peer with an available free capacity in U_i, as a last resort, peer i asks peers in the free set until P_i becomes empty (the way of maintaining the free set is described in section 3.2.5).

Third Phase: (Post Processing) If $\beta(i) > 0$ and $U_i^* \neq \emptyset$ at this point, peer i tries to collect as many children for substream q^* as possible from peers in U_i^*. We need to notice that this is a special case of the swap process (Way-3) so that no new subscription occurs, and is conducted only once. Third phase is shown in Scheduling Alg. 3.4.

Scheduling Alg. 3.4: Third Phase: Post Processing

Step 6
Ensure: $U_i^* \neq \emptyset$
Ensure: $\beta(i) > 0$
1: **for each** peer j' **in** U_i^* **do**
2: **while** $C_{j'}[q^*] > 1$ **do**
3: i get a child from j' for q (Way-3)
4: $\beta(i) \leftarrow \beta(i) - 1$
5: **IF** $\beta(i) = 0$ **break**
6: **end while**
7: **end for**

3.2.4 Scheduling Example

In the example, all peers have a uniform upload capacity of four, i.e., $u_j = 4$ for each j, and the given video stream is divided into four substreams of unit

bit rate ($s = 1$), i.e., $r = 4$ and $N = 4$. Peer i, with an upload capacity $u_i = 4$, wants to join the system. The budget of peer i is determined as $m(i) := u_i/s = 4$ and the set of neighbors is given as $U_i = \{a, b, c, d\}$. Given a collection of trees $\mathcal{T}$ shown in Figure 3.3(A), price vectors are calculated as shown on top of the figure. From such vectors, we notice that: 1) peers b and c are saturated in fourth and third trees, respectively, and 2) peers a and d have the second substream as a dominant one.

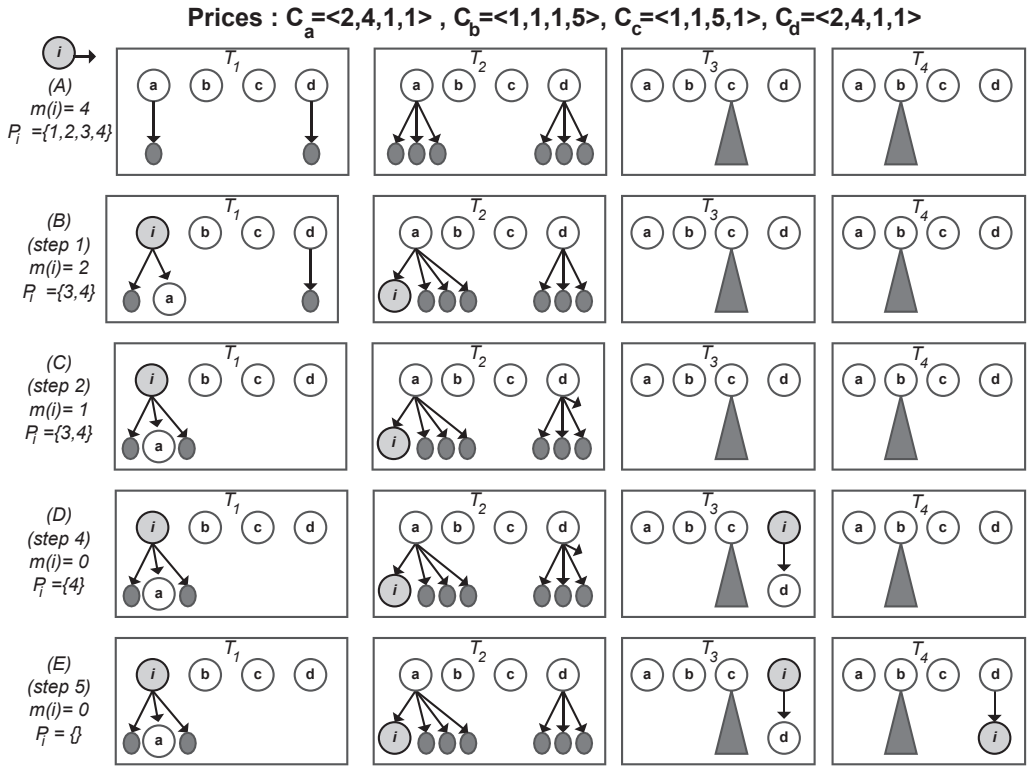


Figure 3.3: A scheduling example. The triangle means that the peer has all its children in this tree.

From the Figure 3.3(A), we notice that the first substream is not dominant for both peers a and d with a price equals to two. Thus, peer i selects the first substream as q^* along with peers a and d be the members of U_i^*. According to step 1, Figure 3.3(B), peer i buys q^* from the peer $a(\in U_i^*)$ by paying

two units of money. That means peer i will replace peer a in the first tree and adopt both peer a and his child. Then, peer a has got a free capacity by receiving money from peer i. That allows peer a to adopt peer i in the second tree corresponding to the dominant substream of peer a.

In Figure 3.3(C), representing step 2, peer i asked peer d to swap its child in first tree. However, it could not subscribe to the dominant substream of peer d, which is the second tree, as it is already subscribed to. Note that by this action the balance of peer i is reduced by one and peer d has got a free capacity. At this point, as peer d is the only peer that has a free capacity and its dominant substream is not required by peer i, the step 3 will have no effect on the overlay.

In the second phase of the algorithm, peer i starts with step 4, illustrated in Figure 3.3(D). There are three peers to have a price equals to one in the third tree, and peer i chooses peer d as the seller of the third substream (note that hop-counts are not illustrated in this figure for simplicity). By choosing peer d, peer i will replace it in the third tree and adopt it by paying one unit of money. Finally, Figure 3.3(E) illustrates the case of step 5 in which peer i becomes a child of peer $d \in U_i$ in the fourth tree. Peer i's balance is zero and hence will skip the post processing phase.

3.2.5 Managing the Free Set

To implement the free set in a distributed manner, we get benefit from the forest of trees organized in the proposed scheme. In this subsection, we describe a concrete way to realize three operations used for the free set, i.e., join, leave and find.

To join the free set, a peer i tells all its parents about that (recall that it has at most N parents in $\mathcal{T}$). After receiving a message from a child in a tree, each peer forwards the information to the parent in the tree unless it is the root. As a result, we have a path from i to the root in each tree so that *all peers on the path are aware of that i is a member of the free set*. This operation takes at most $N \times d$ messages provided that the maximum depth of trees is bounded by d. The leave from the free set is conducted in a similar way. If peer i in the free set changes the parent in a tree, which frequently occurs in the scheduling process, such an update must be propagated to all peers on the paths by the old and new parents of i, i.e., the old parent initiates the propagation of leave message and the new parent initiates the

propagation of join message.

If peer j wants to find a peer in the free set, it sends a request message to one of its parents selected randomly. The request is forwarded up in the corresponding tree until it finds a peer that knows about one of the free set peers. Note that such a forwarding process can always find a peer in the free set in at most d hops (if any), since the root of any tree knows all members of the free set. The reader should note that in the above process, the root of a tree does not become a bottleneck in many cases, because: 1) the tree is randomly selected from N candidates in $\mathcal{T}$ and 2) it is likely that a request path and a join path will meet at a deep level of the selected tree. If a peer in the free set receives several requests from different peers, it serves its upload capacity in the first-come and first-serve basis.

3.3 Evaluation

To evaluate the performance of the proposed scheme, we conducted extensive simulations based on OPSS [83]. The performance of the scheme is compared with SplitStream, where not to reinvent the wheel, we used an OPSS simulation package developed for SplitStream [28] in the evaluation. The efficiency of constructed multiple-tree overlays is evaluated through the following three metrics:

A) **Saturation fraction** of a peer is the ratio of the number of children for its dominant substream to the maximum number of children of the peer (i.e., budget). It takes a value in range $[0, 1]$ where a higher value implies that the leave of the peer affects its descendants for a smaller number of substreams.

B) The hop-count delay of a peer in a tree is the number of links on the unique path connecting the peer to the root (source) in the tree. The **average hop-count** of a peer indicates the average of the hop-count delay over all trees.

C) **Free set requests** represent the total number of requests received by the free set to complete a scheduling. We are interested in this metric due to the fact that the maintenance cost of the free set and the cost required for seeking subscribers heavily affect the overhead of the scheme. More importantly, by asking the peers of the free set, those peers will have a small number of children in different trees leading to degenerate trees.

3.3.1 Setup

In the following, we represent the upload capacity of peers in terms of the budget and the rate of video streams in terms of the number of substreams, i.e., we normalize actual values by the bit rate of substreams. Table 3.2 summarizes all scenarios examined in the evaluations, where in each scenario, the download capacity of each peer is assumed to be sufficiently large. Scenario's name in the table is encoded by the environment type, HM (homogeneous) or HT (heterogeneous), followed by the bit rate of given video stream (e.g., 4 means that the stream is divided into four substreams), and the resource index where 1 stands for $R = 1.0$ and 2 stands for $R = 1.25$. The reader should note that the resource index, R, is defined as the ratio of the available capacity in the system to the streaming rate times the number of peers as in [28].

Table 3.2: Simulation scenarios.

	HM4-1	HM4-2	HM8-1	HM8-2	HT4-1	HT4-2	HT8-1	HT8-2
Server's capacity	4	5	8	10	4	4	8	8
Peer's capacity	4	5	8	10	1,3,8	1,4,10	2,6,16	3,7,20
Stream rate	4	4	8	8	4	4	8	8
Resource Index	1	1.25	1	1.25	1	1.25	1	1.25

Heterogeneous settings follow the setting used in [88]. More concretely, we adopt three types of upload capacities low, medium and high which correspond to the bit rate of 128 [Kbps], 384 [Kbps] and 1000 [Kbps], respectively, and we fix the substream rate to either 64 [Kbps] or 128 [Kbps]; thus in the former case, the upload capacity of each type is normalized to 2, 6 and 16, respectively. The fraction of each type of peers in the population is fixed as in Table 3.3.

For each scenario, we ran the proposed scheme and SplitStream by fixing the number of peers to $n = 10000$, where we did not consider churn to make a fair comparison of the schemes. The proposed scheme is evaluated for different values of D (the number of peers in set U_i). Although D was chosen to be a multiple of the number of substreams (namely, N, $2N$ or $4N$) in the simulation, any value can be used for D. The saturation fraction and the average hop-count delay are calculated for each peer and the cumulative

distributions are plotted for only some scenarios to save the space. On the other hand, the average value over all peers is presented in tables for all scenarios.

Table 3.3: Fraction of each type in the population.

Type	Fraction
Low (128 Kbps)	37%
Medium (384 Kbps)	27%
High (1000 Kbps)	36%

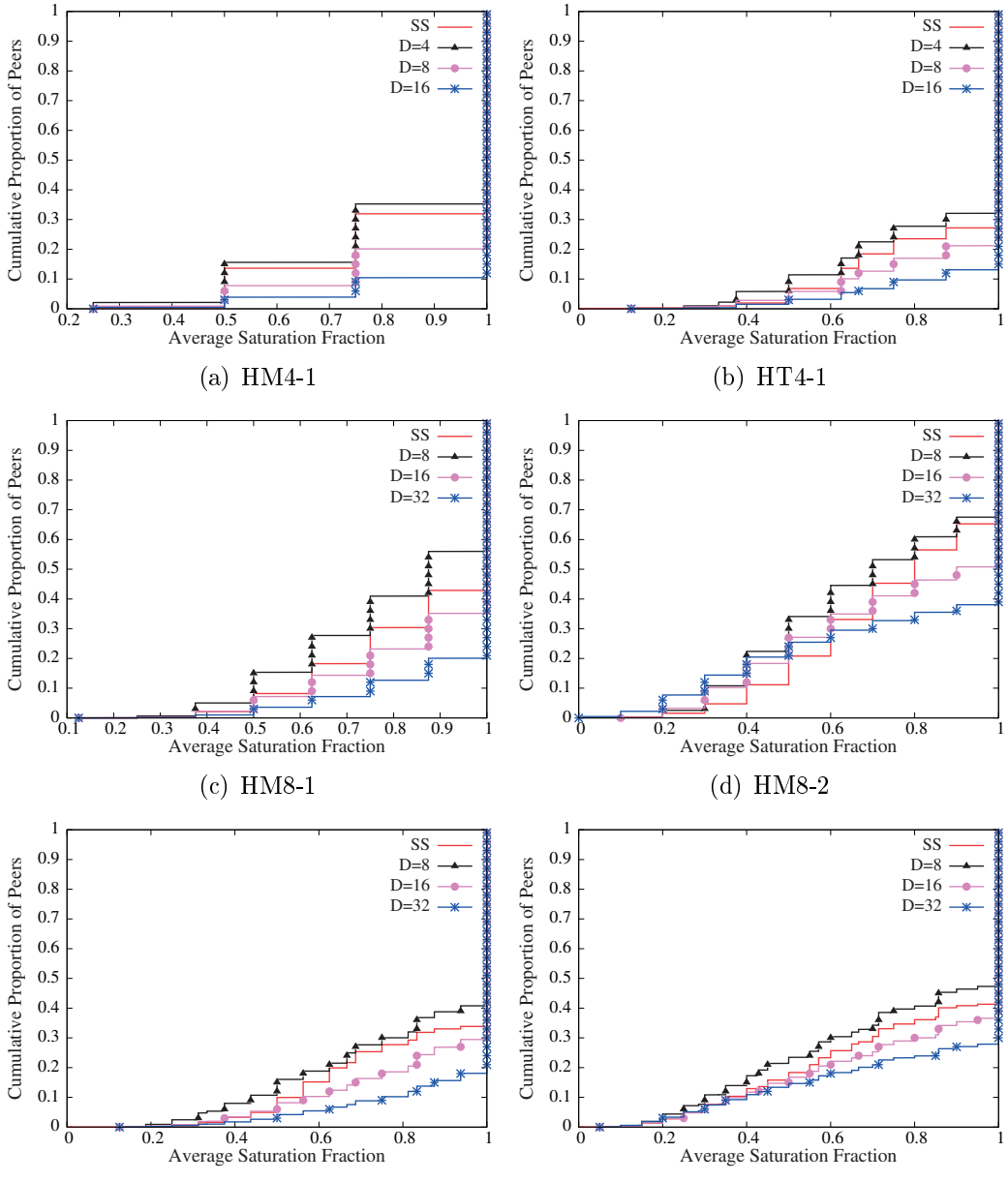
3.3.2 Results

Saturation Fraction

As was mentioned, a peer with a high saturation fraction means a lower number of substreams to be lost in case of its leave. However, as will be seen in the next section, a higher saturation fraction does not necessarily mean a shorter hop-count delay, since a nearly-saturated peer might have few children as leaves in other trees.

Table 3.4: Average saturation fraction.

	SS	$D = 4$	$D = 8$	$D = 16$
HM4-1	0.89	0.87	0.93	0.96
HM4-2	0.76	0.73	0.78	0.80
HT4-1	0.90	0.88	0.93	0.96
HT4-2	0.82	0.80	0.85	0.87
	SS	$D = 8$	$D = 16$	$D = 32$
HM8-1	0.87	0.82	0.90	0.94
HM8-2	0.76	0.70	0.77	0.79
HT8-1	0.87	0.84	0.90	0.94
HT8-2	0.81	0.78	0.84	0.86



(a) HM4-1 (b) HT4-1

(c) HM8-1 (d) HM8-2

(e) HT8-1 (f) HT8-2

Figure 3.4: Cumulative distribution of saturation fraction.

Figure 3.4 shows the cumulative distribution of the saturation fraction with different number of neighbors, D, and Table 3.4 summarizes the average saturation fraction in each scenario, where (SS) stands for SplitStream.

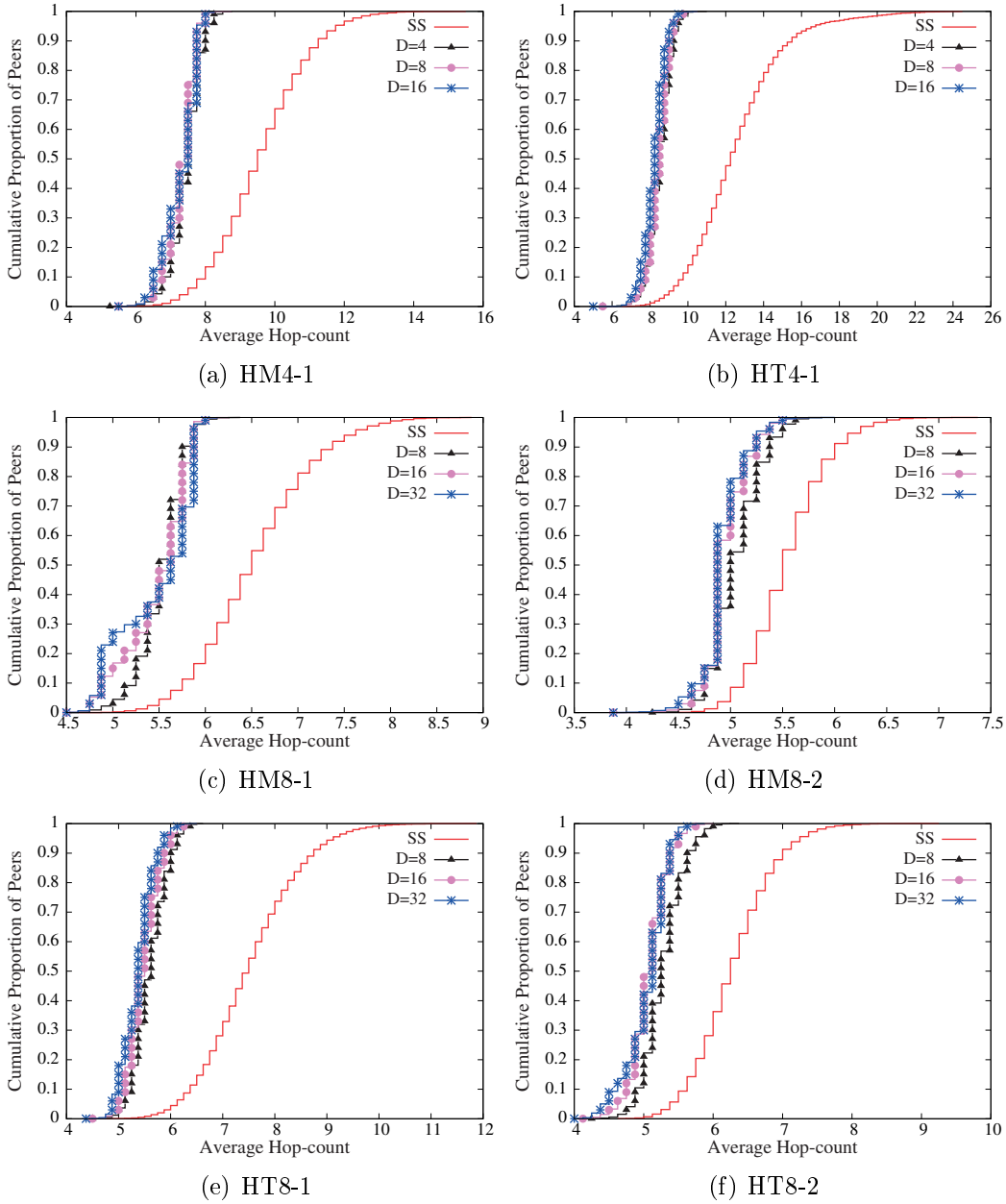
The average saturation fraction of the proposed scheme increases as D increases since it increases the chance of buying or swapping substreams with other peers, whereas it is worse than SplitStream when D takes the smallest value N. From the figure, we can also confirm that the number of peers that have their children in more than one tree is 10% in HM4-1 and 20% in HM8-1 for $D = 4N$. It means that the saturation fraction is higher for a lower number of substreams. It should also be noted that the saturation fraction degrades by increasing the resource index R from 1.0 to 1.25. In fact, if R is sufficiently large, it is possible to attain the given streaming rate without fully utilizing upload capacities, which prevents many peers from being saturated.

Average Hop-Count Delay

Next, we evaluate the average hop-count delay of the proposed scheme. Figure 3.5 shows the cumulative distribution of the average hop-count delay and Table 3.5 shows its average in each scenario, as before. We can observe that the proposed scheme outperforms SplitStream for all scenarios, and in seven out of eight scenarios, it attains a shorter hop-count delay than SplitStream by at least 1.0 even when $D = N$ (note that the difference becomes large for large D's). Figure 3.5 also clarifies that the proposed scheme outperforms SplitStream with respect to the "maximum" average hop-count delay.

Such a positive effect of parameter D reduces for large resource index R. In fact, in scenario HM4-2 with $R = 1.25$, the average hop-count *increases* as D increases in contrast to other scenarios with $R = 1.25$. One possible conjecture to explain such a phenomenon is that for large R's, as D increases, the average hop-count decreases up to a limit related to the number of substreams and after that limit, the delay increases again due to the (unnecessary) join to other trees as leaves of deeper level.

To verify this conjecture, we conducted additional simulation for HM8-2 and HM8-1 and increased D up to 48. As a result, we found that the average hop-count of HM8-2 increases from 4.99 to 5.31 by increasing D from 40 to 48, and that of HM8-1 does not change from 5.47 regardless of the increase of D.



(a) HM4-1 (b) HT4-1

(c) HM8-1 (d) HM8-2

(e) HT8-1 (f) HT8-2

Figure 3.5: Cumulative distribution of average hop-count.

Table 3.5: Average hop-count.

	SS	D=4	D=8	D=16
HM4-1	9.58	7.43	7.31	7.29
HM4-2	7.96	6.62	6.47	7.03
HT4-1	12.51	8.52	8.43	8.22
HT4-2	9.95	8.05	7.60	7.45
	SS	D=8	D=16	D=32
HM8-1	6.53	5.52	5.43	5.47
HM8-2	5.54	5.05	4.95	4.93
HT8-1	7.52	5.60	5.47	5.37
HT8-2	6.28	5.28	5.07	5.05

Another important issue we need to address is that why the proposed scheme outperforms SplitStream even under a low saturation fraction? To clarify this point, we analyzed the difference of the structure of the resulting multiple-trees to an optimum multiple-tree, which can be obtained for homogeneous cases as follows. Since the number of children of each peer is bounded by N, an optimum tree contains $N^{\ell-1}$ peers at the ℓ^{th} level for each ℓ (e.g., the first level consists of the root of the tree, the second level consists of N children of the root, and so on) except for the deepest level of the tree, where the depth d of the tree can be obtained by solving

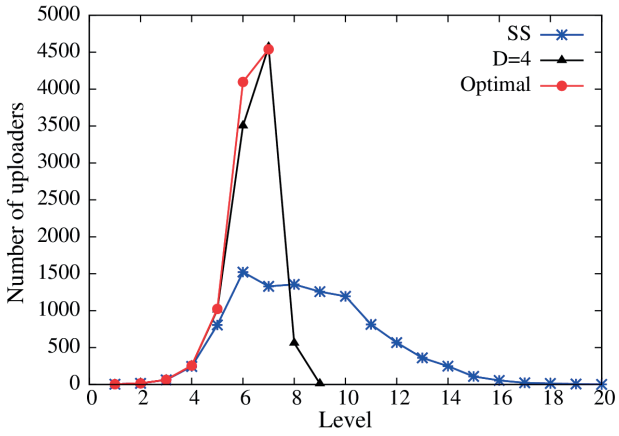
$$\sum_{i=1}^{d-1} N^{i-1} < n \leq \sum_{i=1}^{d} N^{i-1},$$

which is approximately

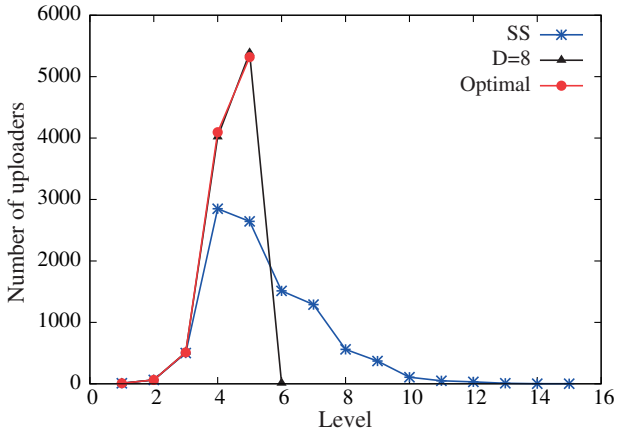
$$\frac{\log\{n(N-1)+1\}}{\log N}$$

The number of uploaders (peers with at least one child) at the ℓ^{th} level of the optimum tree can thus be calculated as follows:

1. for $1 \leq \ell < d-1$, it is N^{ℓ}, and



(a) HM4-1



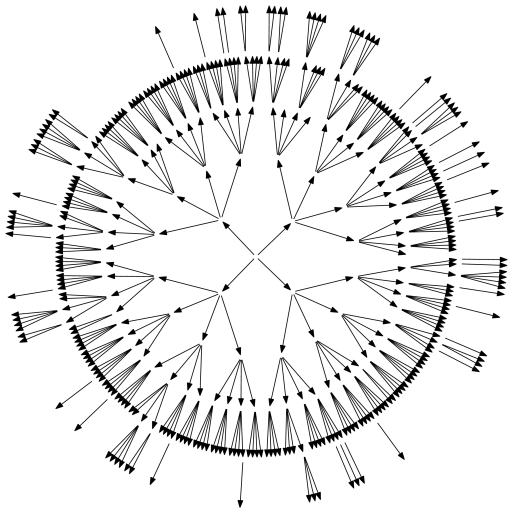
(b) HM8-1

Figure 3.6: Number of uploaders in different levels.

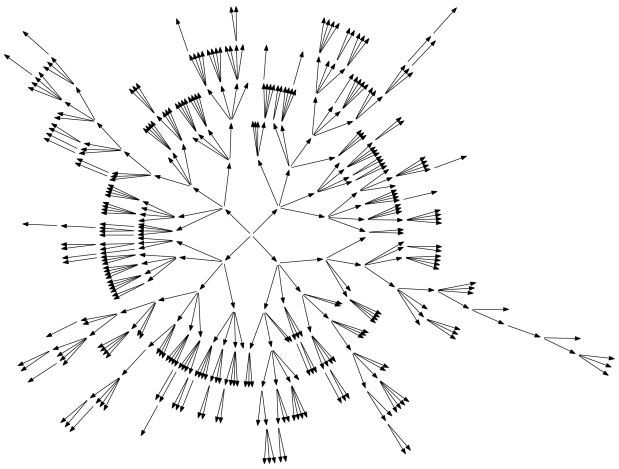
2. for $\ell = d-1$, it is $\left\lceil \frac{n-\sum_{i=1}^{d-1} N^{i-1}}{N} \right\rceil$. Consequently, the number of uploaders at the ℓ^{th} level across all trees in an optimum multiple-tree is given as

$$N * \min\{N^{\ell-1}, \left\lceil \frac{n - \sum_{i=1}^{\ell} N^{i-1}}{N} \right\rceil\}.$$

Figure 3.6 compares the resulting multiple-trees with an optimum one,



(a) The proposed scheme, HM4-1.



(b) SplitStream, HM4-1.

Figure 3.7: Radial graphs of the proposed and SplitStream overlays.

for scenarios HM4-1 and HM8-1. The horizontal axis of the figure is the level of the tree and the vertical axis is the number of uploaders at each level. The proposed scheme matches the optimum tree up to the fifth level, and it is nearly optimal even for deeper levels. On the other hand, SplitStream goes

far from optimal with a large gap (e.g., the gap which is 2500 uploaders at the sixth level in HM4-1) and with the existence of many uploaders at deeper levels. Recall that the proposed scheme has been designed to increase the number of saturated peers in all trees. Moreover, as peers prefer to buy or swap other peers that have a short hop-count delay in case of a price draw, the proposed scheme can maintain short depth trees. As for SplitStream, the random selection of parents according to Pastry Id can not guarantee an efficient overlay construction with a short hop-count delay and leads to a high use of the peers in the free set capacity, as will be verified in the next subsection, resulting in this kind of degenerate trees. To exemplify that, a snapshot of the constructed multiple-tree overlays by the proposed scheme and SplitStream are compared in Figure 3.7. Only 100 homogeneous peers are simulated for this purpose to simplify the figures. It is clear from the figure that SplitStream trees have extra hops.

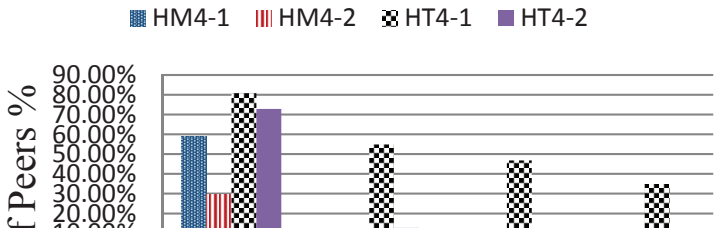
Free Set Requests

Finally, we evaluate the amount of free set requests issued by the participants. Figure 3.8 shows the fraction of peers which issued (at least one) free set request before completing the scheduling. Recall that such a request is issued when it does not have enough balance or it can not find a neighbor which has enough upload capacity. In homogeneous scenarios, the proposed scheme causes no free set request, whereas the fraction of peers which issue a free set request in SplitStream is 60% for $R = 1.0$ and 30% for $R = 1.25$. This means that in homogeneous environment, the proposed scheme is remarkably efficient compared with SplitStream with respect to the overhead for the maintenance of free set.

The superiority of the proposed scheme to SplitStream can be observed even under heterogeneous scenarios provided that $R = 1.25$ and such an effect is enhanced for larger D's. For example, in HT8-2, exactly one peer (among 10000 peers) issued a free set request for $D = 32$.

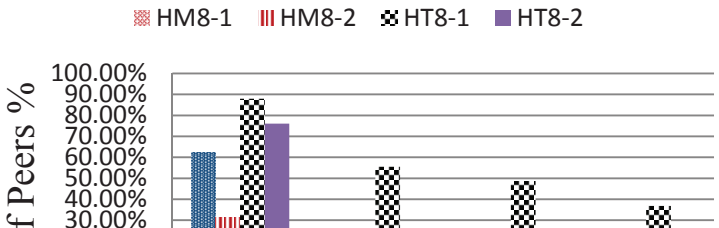
3.4 CONCLUDING REMARKS

This chapter presented a scheduling scheme for P2P streaming systems which attains the maximal resource utilization in a distributed manner. The pro-



	SS	D=4	D=8	D=16
HM4-1	59.14%	0.00%	0.00%	0.00%
HM4-2	29.47%	0.00%	0.00%	0.00%
HT4-1	81.02%	54.98%	46.84%	34.96%
HT4-2	72.79%	12.68%	3.79%	0.40%

(a) Four substreams.



	SS	D=8	D=16	D=32
HM8-1	62.21%	0.00%	0.00%	0.00%
HM8-2	31.23%	0.00%	0.00%	0.00%
HT8-1	87.99%	55.70%	48.88%	36.84%
HT8-2	76.02%	5.55%	0.64%	0.01%

(b) Eight substreams.

Figure 3.8: Free set requests.

posed scheme is able to build an efficient multiple-tree overlay with a short hop-count delay even when no spare capacity is available. The result of simulation proves that:

- The constructed multiple-trees certainly converge to an efficient overlay with a short hop-count delay
- The proposed scheme outperforms SplitStream with respect to the average hop-count in all scenarios examined in the experiments.
- the proposed scheme outperforms SplitStream in regard to the number of peers who are internal in only one tree (saturated peers) provided that the number of allowed neighbors is more than the number of substreams.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 4

Low Cost Cloud-P2P Live Streaming

In the previous chapter, a higher streaming rate with a shorter delay is attained by maximally utilizing resources of P2P. Yet, the P2P live streaming has a high dynamism in peer population (called peer churn) which affects the reliability of pure P2P live streaming systems. Without a guaranteed reliability, no quality of service can be guaranteed.

In this chapter, we provide a guaranteed quality of service by assisting the pure P2P live streaming by the cloud. That assistance is achieved by exploiting a storage service and a cloud content delivery network (CCDN). The latest chunks in the live stream are stored to the storage service in the cloud, and each peer is allowed to fetch missing chunks from the storage service through edge locations of the CCDN. These edge locations fetch chunks from the storage service upon receiving requests from peers.

The cost of cloud assistance is comprised of the amount of data fetched from the CCDN and the number of requests handled by the cloud. The latter cost is further divided into requests handled by the storage service and requests handled by the CCDN. Accordingly, we propose three techniques to reduce the cost of such a cloud assistance and evaluate them through extensive simulations.

The remainder of this chapter is organized as follows. In Section 4.1, the research problem is introduced. Section 4.2 formally describes a model consisting of multiple-tree overlay assisted by the cloud. Section 4.3 describes the baseline model over which we proposed our method in Section 4.4. The

results of simulations are summarized in Section 4.5. Finally, Section 4.6 concludes the chapter.

4.1 Problem Description & Contribution

P2P system can not standalone and needs the support from other reliable services when 1) the streaming service has to be guaranteed. That is due to the dynamic nature and voluntary contribution in P2P, and 2) when the streaming rate expected by peers exceeds the capacity of the P2P system. This is widely expected today as the download bandwidth offered by Internet providers is normally higher than the upload bandwidth.

To solve these issues, P2P should be assisted by a server or a farm of servers to disseminate the video stream to all peers. The own servers and content delivery networks (CDNs) are expensive solutions due to the cost of dedicated infrastructures in the former one and the semi-static provision of resources (i.e., servers) in the latter one. On the other hand, the cloud emergence has offered a flexible and cost-effective platform to provision resources on-demand. Cloud computing platform offers different services that could be exploited for this purpose such as the storage service, cloud CDN and computing instances (virtual servers). Then, the assistance may be achieved by two different ways. At first, by exploiting a storage service and a cloud content delivery network (CCDN). The latest chunks in the live stream are stored to the storage service in the cloud, and each peer is allowed to fetch missing chunks from the storage service through edge locations of the CCDN. These edge locations fetch chunks from the storage service upon receiving requests from peers. Another way for the assistance of P2P by the cloud is to rent computing instances and to use them as virtual peers to disseminate the live stream to other peers [14, 15]. However, the rent of computing instances is less flexible than the rent of upload bandwidth (e.g., the minimum rent period of computing instances is generally one hour). Moreover, tenants of computing instances share the network infrastructure where the bandwidth allocated for an instance is not predictable nor guaranteed. Thus to refine the cost of cloud assistance while maintaining the high performance, we take the former approach to propose a cost effective cloud-assisted P2P live streaming system.

4.1.1 Quick Recovery

In this chapter, we adopt the multiple tree structure. Such a structure is weak against peer churn compared with other structures. However, it is efficient in terms of the latency and the message overhead, and it exhibits a good performance with respect to the broadcasting time [47, 48] and the maximum streaming rate [46, 16]. In other words, the multiple-tree structure is an appealing structure if we can overcome its shortcomings (e.g., stability and robustness) by combining it with other services such as the cloud. The baseline model is multiple-tree overlay adopted in SplitStream [28]. In this model, a given live stream is equally divided into substreams by the sequence number of chunks contained in the stream. Then, each substream is delivered to all peers (subscribers) through a different delivery tree. We assume that sets of internal nodes of the trees must be mutually disjoint as in SplitStream. In such a multiple-tree-structured P2P network, the leave of an internal node suspends the forwarding of a substream to descendants which affects the quality of the stream. The lost part of streaming data should be fetched from the cloud which significantly increases the cost. Then, we need to have a solution with the help of cloud to improve the robustness of the system.

- Contribution Point 1
  Firstly, we propose a technique to reduce such a cost by exploiting the cloud storage service to explicitly register "orphaned" peers which lost their parents in delivery trees. Match-making between orphaned peers and internal peers with enough capacity can be done by referring to the registered information. That significantly reduces the time before an orphaned peer becomes a child of a new parent in the delivery tree. However, it incurs additional cost due to requests handled by the storage service.

4.1.2 Proactive Bandwidth Investment

Assume the case where the capacity of the P2P system is not enough to disseminate the video stream to all peers. In fact, this case is a main interest of our study in this thesis. Considering the number of peers as n, the stream rate as r, the aggregated capacity of all peers including the source as U, if $n \times r > U$ then there is a shortage in the P2P system's capacity. Let

$\theta = n \times r - U$ denote that shortage. In the hybrid system of P2P assisted by the cloud, such a shortage will be fetched by the peers from the cloud to maintain the quality of service. The fetch will occur when some peers notice the delay of stream chunks, for example when two seconds remain to the chunk playback point.

However, it would be a more efficient solution to estimate the shortage θ, and proactively invest a corresponding bandwidth into the P2P system. That could be done by selecting some peers to proactively fetch the chunks and be roots of the multiple-tree overlay. The invested bandwidth should be balanced among the trees, i.e., by investing θ/s peers in each tree where s is the substream rate delivered through every tree. Such a solution will improve the quality of stream as chunks are fetched earlier. Moreover, the capacity of the system will be increased and the depth of trees will be shorter. That will reduce the number of descendants affected by the leave of their parent.

- Contribution Point 2
  We propose a second technique where the key idea is to proactively fetch chunks by several internal nodes selected from each tree. The selected node (peer) plays the role of a root for the corresponding tree, which reduces the height of the delivery tree and the load of other internal nodes. The number of peers to be selected is controlled by referring to the number of orphaned peers registered to the storage service (first technique). That is by the reason if the number of selected peers is too large, the amount of proactively fetched chunks becomes excessive and if it is too small, the cost due to orphaned peers increases.

4.1.3 Number of Requests

In the previous subsection, the system invests a bandwidth correspondent to the shortage, θ, to proactively fetch the chunks from the cloud. Remember, the cost does not only depend on the amount of bandwidth invested but also on the number of requests to fetch it. Normally, the chunk size is small in tree-based systems to be quickly forwarded to children where a chunk is forwarded only after being completely downloaded by the parent. Then, with the increase of θ, the number of requests to the cloud increases which is a costly behavior. Accordingly, an efficient solution should overcome this issue to minimize the cost.

- Contribution Point 3
  The third proposed technique reduces the number of requests handled by the CCDN in the second technique by allowing peers to request a collection of chunks instead of individual chunks. More concretely, when proactively fetching chunks from the CCDN as a selected peer for tree t, it acquires consecutive chunks associated with the whole stream (instead of a particular substream) in the form of frames of chunks, and serves as a root for tree t.

4.1.4 Achievement

In this chapter, we propose a method to reduce the cost of the cloud-assisted P2P systems as explained in contribution points in the previous subsections. The performance of the proposed method is evaluated by simulation. We implement our simulator on top of an event-driven P2P simulator [85]. The simulation results are summarized as follows.

1. In comparison with the baseline model, the first technique is able to save 44% of the total amount of data fetched from the cloud and 40% of the total number of requests issued to the cloud.

2. The second technique achieves extra save by reducing the number of orphaned peers per failure due to the short height of the delivery tree.

3. The third technique has the least monetary cost where it saves up to 42% of the total amount of data fetched from the cloud and 66% of the total number of requests issued to the cloud.

4. Due to the assistance of the cloud, all techniques are able to guarantee the quality of live streaming service.

4.2 System Model

4.2.1 Overview

In this chapter, we focus on the hybrid of P2P and cloud computing platform. The role of P2P is to deliver live streams issued by a media server to the peers through a P2P overlay. Along with that, the role of cloud platform is to assist

the delivery by using a storage service and a cloud content delivery network (CCDN). In our model, we consider a P2P overlay consisting of N trees which span all peers. Live stream is a sequence of chunks which are given unique sequence numbers starting from 0. Chunks are equally divided into N substreams so that the ith substream consists of chunks with sequence number $j \equiv i(mod\ N)$. The reader should note that by letting r be the bit-rate of the given stream, the bit-rate of each substream is given as r/N. As will be described later, chunks are delivered to the peers in such a way that the ith substream is delivered through the ith tree for each $0 \leq i \leq N-1$.

Let $c(u)$ denote the upload bandwidth of a peer u. Since the bit-rate of each substream is r/N, u can accommodate at most $s(u) \stackrel{\text{def}}{=} \lfloor c(u)N/r \rfloor$ peers as the "children" in the P2P overlay. $s(u)$ is called the capacity of u and we say that u has a free capacity if it has less than $s(u)$ children. The resource index R of a P2P concerned with the delivery of the given stream is the ratio of the available capacity of the P2P to the capacity which is necessary to deliver N substreams to all peers in the P2P, i.e.,

$$R \stackrel{\text{def}}{=} \frac{\sum_{u \in V} s(u)}{N \times |V|},$$

where V is the set of peers in the P2P. In the following, we are particularly interested in cases such that $R \simeq 1$. Note that in hybrid live streaming systems, the resource index of the P2P indicates the amount of contribution of the cloud platform; i.e., the amount of contribution of the cloud should be large as the value of resource index decreases.

As for the mechanism to detect the leave of adjacent peers in the P2P, we assume that each peer exchanges hello messages with its neighbors for every τ seconds. Similarly, each peer detects the leave of parent in a delivery tree by monitoring the delay of chunks contained in the corresponding substream.

4.2.2 Cloud Computing Platform

Figure 4.1 illustrates the behavior of the cloud platform. At first, it continuously receives a stream of chunks from the media server, and keeps chunks issued in the recent m seconds in the storage service. Each peer in the P2P can request these chunks at any time, and requested chunks are delivered to the requester through edge locations in the CCDN. Such a request is issued by a peer when it detects the missing of a chunk in a substream. A

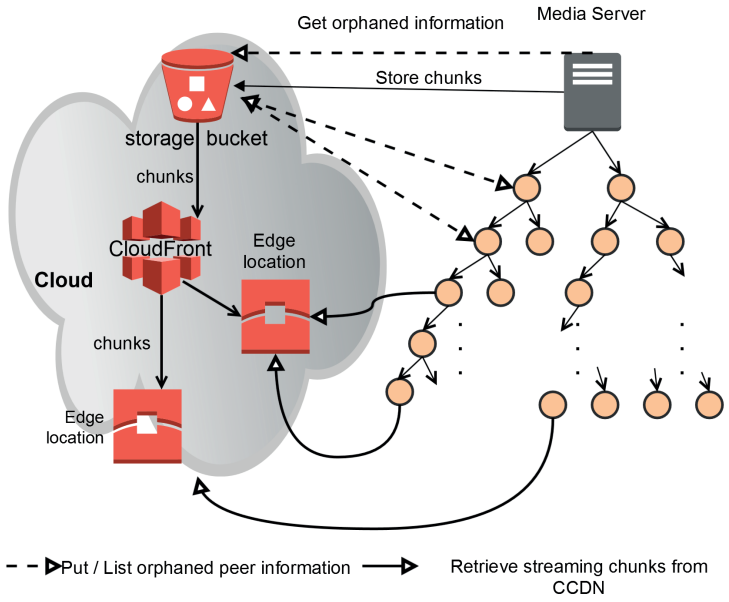


Figure 4.1: Cloud-assisted P2P live streaming model.

chunk is considered to be missed when the remaining time to its playback point meets a certain threshold. Then, a peer and its descendants may consider same chunks as missed in a close time and acquire them through the CCDN. Moreover, when a peer acquires the missed chunk from the CCDN, it will be busy in forwarding the latest chunks to its children. Hence, in this case, chunks acquired through the CCDN are not forwarded to the children to bound the overhead and redundant data. A different case, as in Section 4.5.2, is that a subset of peers (proactively) issue requests to fetch some (not-yet missed) chunks from CCDN, i.e., to increase the system throughput. Forwarding these chunks to children is necessary to get the benefit of the proactive design.

In addition to the recently issued chunks, in the proposed method, the storage service keeps information concerned with "orphaned" peers. We say that a peer is orphaned if the connection to the parent is lost. See Section 4.4 for the details.

In the following, we measure the cost of such a cloud assistance by using a pricing model used in actual cloud computing platform. More concretely, we

use Amazon's pricing model [86, 87] in which the customer should pay for: 1) the amount of data bandwidth fetched from the CCDN, 2) the number of requests issued to the CCDN and 3) the number of requests issued to the storage service. We omit the other costs such as the storage cost, as they are common for all cloud assisted schemes.

4.3 Baseline Model

This section introduces a model of hybrid P2P systems which will be used as a baseline in succeeding sections. This model is a variant of the SplitStream live streaming system [28] in which N delivery trees are organized so that internal nodes of the trees are mutually disjoint. In other words, each peer can join at most one tree as an internal node and the other trees as a leaf node. Let T be the set of N trees. At the time of participation, each peer determines the role of the peer in each tree; i.e., internal or not. Let $\sigma : V \times T \rightarrow \{0, 1\}$ be a function indicating the selection made by the peers, where $\sigma(p, t) = 1$ if p joins tree t as a candidate of internal node. Note that in the SplitStream, such a selection is conducted in a random manner.

Assume that peer q submits peer p a request to subscribe to a substream through tree t, where p is a peer with $\sigma(p, t) = 1$. Peer p accepts the request if it has a free capacity. Moreover, p accepts the request if $\sigma(q, t) = 1$ and it has a child c with $\sigma(c, t) = 0$. Then p admits q as a new child after *pushing out* c from the tree (thus c is orphaned at this time)(Figure 4.2-(A)). If it does not accept the request, p forwards the request to a child c' with $\sigma(c', t) = 1$, and such a forwarding is repeated until it reaches a peer which accepts the request or fails to be forwarded to a child, see Figure 4.2-(B). Note that this process always succeeds for q with $\sigma(q, t) = 1$ since any internal node with a deepest level of the tree either has a free capacity or has a leaf child k'' with $\sigma(k'', t) = 0$, while it might fail for q with $\sigma(q, t) = 0$. In such a case, q should request the free set, see Figure 4.2-(C).

In the baseline model, to reduce the length of such forwarding steps, we assume that the first peer p receiving the request from q with $\sigma(q, t) = 1$ is selected in the following manner:

1. if q is an orphaned peer and the depth of q in the tree was two or more before being orphaned, the former grandparent of q is selected as p, and

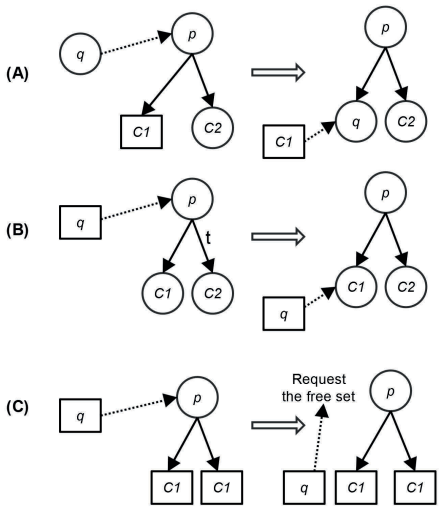


Figure 4.2: Baseline P2P overlay configuration: circle peers belong to tree t and square peers belong to other trees. Peer q requesting peer p for tree t. In (A), $\sigma(p,t) = \sigma(q,t) = \sigma(C2,t) = 1$ while $\sigma(C1,t) = 0$ so peer p accepts q and rejects $C1$ which is forwarded to p for example. In (B), $\sigma(p,t) = \sigma(C1,t) = \sigma(C2,t) = 1$ so peer p forwarded q to $C1$ for example. In (C), $\sigma(q,t) = \sigma(C1,t) = \sigma(C2,t) = 0$ so peer q reaches the leaves level without being adopted. Hence, it should request the free set.

2. otherwise, a random peer is selected as p.

In addition, if $\sigma(q,t) = 0$, q does not request any p but registers itself to a *pool of orphaned peers* concerned with tree t, and waits for a response from peers with a free capacity. In the baseline model, the registration message issued by q is collected to the root of tree t' with $\sigma(q,t') = 1$ (i.e., it is sent to the parent of q in tree t' and is forwarded up to the root of the tree), and the collected messages are forwarded down from the root of tree t, so that they are received by an internal node of t with a free capacity, if any. If q receives several responses, it accepts only one and discards others.

Thus the cost of the baseline model concerned with the cloud assistance is comprised of the number of requests issued to the CCDN and the amount of chunks fetched from the CCDN.

4.4 Proposed Method

The proposed method consists of three techniques; 1) quick recovery from the status being orphaned with the aid of cloud storage services, 2) proactive fetching of chunks from the CCDN, and 3) the reduction of the number of requests to the CCDN with the notion of frames. In the succeeding subsections, we explain each technique in detail.

4.4.1 First Technique: Quick Recovery with Storage Service

In hybrid P2Ps, once a peer becomes orphaned, the delivery of chunks to the peer is suspended until it becomes a child of a new parent. During such a suspension, the orphaned peer should fetch missing chunks from the CCDN to keep the playback of the live stream. Thus, a short suspension time will significantly reduce the amount of fetched data.

The basic idea of the first technique is to reduce such a suspension time with the aid of cloud storage services (note that it is experimentally evaluated that general cloud storage services can serve as many requests as it receives [15]). More concretely, when a peer p becomes orphaned in a tree t where $\sigma(p,t) = 0$, it sends a PUT request to a bucket in the cloud storage to add a file concerned with the event, where the directory of buckets and the way of authentication should be known to all peers in advance. The file name encodes peer's IP, port, and the name of substream, i.e., it needs to add different files for each tree. For example, if peer a is orphaned for a substream k, it puts a file in the storage bucket with the following name:

$$aIP\_PortNumber\_k.txt$$

On the other hand, any peer with a free capacity can find orphaned peers by sending a LIST request to the bucket which returns a list of file names with necessary metadata. After obtaining it, the free capacity peer tries to accommodate orphaned peers as new children until its capacity is exhausted,

and each orphaned peer which becomes a child of a new parent deletes the corresponding file in the bucket by sending a DELETE request.

Such a match-making mechanism is also used to balance the load of trees. More concretely, we modify the selection of a tree conducted by each newly arrived peer in the baseline model in such a way that it joins the tree as an internal node with the largest number of orphaned peers (recall that such a number of orphaned peers can be easily obtained by sending a LIST request to the bucket).

Additional cost due to the first technique is the number of requests handled by the storage service. Usually, requests to the storage service are more expensive than requests to the CCDN (see [86, 87] for the example of cloud price in Amazon), but as will be evaluated later, the additional cost incurred by the first technique is smaller than the benefit of quick recovery.

4.4.2 Second Technique: Proactive Bandwidth Investment

The key idea of the second technique is to allow several peers selected from each tree to conduct a proactive fetch of chunks from the CCDN so that the delayed chunks due to the shortage of P2P capacity does not occur. Such selected peers, called cloud peers hereafter, plays the role of a root concerned with the delivery of the corresponding substream. See Figure 4.3 for illustration. As will be evaluated later, with the notion of cloud peers, we could reduce the depth of the overlay and enlarge the available capacity of the delivery tree.

The number of cloud peers is related to P2P shortage and determined as follows (see Step 1 of Algorithm 4.1). At first, the server periodically issues a LIST request to the storage service to acquire the latest list O of orphaned peers. A request and computational overhead is incurred when the length of such a period is short (few seconds). On the other hand, a long period might make it difficult to follow the dynamic change in the P2P shortage. In our implementation we found a period of 30 seconds as a good balance. Since the list O might not accurately reflect the shortage of the P2P capacity (i.e., $N \times |V| - \sum_{u \in V} s(u)$), the server identifies a subset L ($\subseteq O$) which have been orphaned for a time longer than τ_n, and regards it as a subset of actual orphaned peers due to P2P shortage. A typical value of τ_n is four seconds,

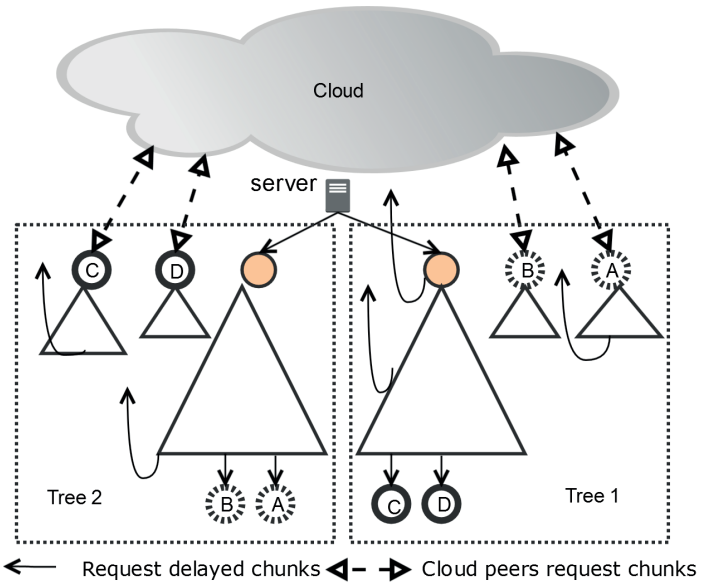


Figure 4.3: Proactive bandwidth investment. Assume the estimated P2P shortage is 4. Then, two peers are selected in each tree to act as cloud peers, where peers A and B fetch substream 1 from the CCDN, and peers C and D fetch substream 2.

while as for the selection of the value of τ_n, there is a tradeoff between the cost of proactive fetching and the cost of reactive fetching. Automatic adjustment of parameter τ_n is left as a challenging future work.

If $|O| > \theta$ holds for a predetermined threshold θ, and all trees in T contain at least one peer belonging to subset L, then the server selects $|L|/N$ random peers from each tree and asks them to act as cloud peers (see Step 2 of Algorithm 4.1). The reader should note that the latter condition is necessary to exclude extreme cases in which the distribution of peers in L across trees is highly imbalanced since such an imbalance could be naturally resolved by accepting more peers to the system without investing cloud peers. If $|O| < \theta'$ for some $\theta' < \theta$, on the other hand, the server eliminates a set of cloud peers to reduce the cost of proactive fetching. Eliminated cloud peers rejoin their tree, while keeping to fetch chunks from the CCDN till they become a child of new parent. A typical value of threshold θ is $10N$, which implies that it allows

Algorithm 4.1: Selection of Cloud Peers

$O \leftarrow$ the set of all orphaned peers.
$L \leftarrow \emptyset$
$N \leftarrow$ number of trees (substreams).
$C[N] \leftarrow$ Array of the set of cloud peers in each tree.
$I[N] \leftarrow$ Array of the set of internal peers in each tree. //Note $I[i] \supset C[i]$ ($0 \leq i \leq N-1$).

Step 1: Determine the number of cloud peers
for each peer j **in** O **do**
  $t_j \leftarrow$ the time when j is registered as orphaned peer.
  if $current\_time - t_j \geq \tau_n$ **then**
    $L \leftarrow L \cup \{j\}$
  end if
end for
// $|L|$ represents the estimated shortage of P2P capacity.

Step 2: adding/removing cloud peers
$\theta \leftarrow 10N$.
if $|O| > \theta$ **and** every tree contains at least one peer in L **then**
  for $i := 0$ **to** $N-1$ **do**
    // select new $|L|/N$ cloud peers in each tree
    $U \leftarrow$ a set of $|L|/N$ peers randomly chosen from $I[i] \setminus C[i]$
    $C[i] \leftarrow C[i] \cup U$
  end for
else
  if $|O| < \theta'$ **then**
    for $i := 0$ **to** $N-1$ **do**
      // eliminate cloud peers from each tree
      $U \leftarrow$ set of N peers randomly chosen from $C[i]$
      $C[i] \leftarrow C[i] \setminus U$.
    end for
  end if
end if

each tree to have ten orphaned peers on average. In the implementation, $5N$ cloud peers are eliminated when $|O| < \theta'$. Thus, each tree will have new five orphaned peers to be adopted by the free capacity peers. Selecting

these values as a multiple of N allows us to have a fair comparison between the second and third technique (Section 4.5.3) by having the same ratio of selected to eliminated cloud peers.

The cost due to the second technique is similar to the first technique, while it has an apparent advantage so that the depth of the trees becomes much smaller and a disadvantage so that the server should manage the set of cloud peers.

4.4.3 Third Technique: Less Requests to the CCDN

The objective of the third technique is to reduce the number of requests issued to the CCDN with the notion of frames. More concretely, when a cloud peer proactively fetches chunks from the CCDN, it requests a frame consisting of F consecutive chunks instead of individual chunks (see Figure 4.4 for illustration). In addition, after being selected as a cloud peer for tree t, it fetches chunks corresponding to the full stream from the CCDN, while it forwards chunks corresponding to a substream associated with tree t to the children in the tree. Then to keep the amount of data fetched from the CCDN, we decrease the number of cloud peers from $|L|/N$ to $|L|/N^2$. See Figure 4.5 for illustration.

Similar effect to the notion of frames could be obtained by increasing the size of each chunk. However, in tree-based streaming systems, it is preferable to have small chunks since each peer needs to receive the whole chunk before forwarding it to the children. In fact, IP packet of 1 Kbyte length is used as the basic chunk in the SplitStream [28]. This is in contrast to mesh-based systems which use large chunks (60 Kbytes in [20], 14 Kbytes in some PPLive channels [88]) to avoid excessive overhead of signaling per chunk.

The increase of frame size F reduces the number of requests issued to the CCDN inverse proportionally. For example, assume that the stream carries 20 chunks per second and the server invested 20 cloud peers, i.e., the CCDN should receive 400 requests per second from cloud peers under the second technique. If the frame size is set to 10 chunks, which corresponds to 0.5 second, the number of requests reduces to 40 per second (90 % save), and if the frame size is set to 40, it reduces to 10 per second (97.5 % save). A disadvantage of the large frame size is the waste of chunks which occurs when the set of cloud peers changes. For example, by scaling the number of cloud peers down, some of them need to rejoin all trees by finding new

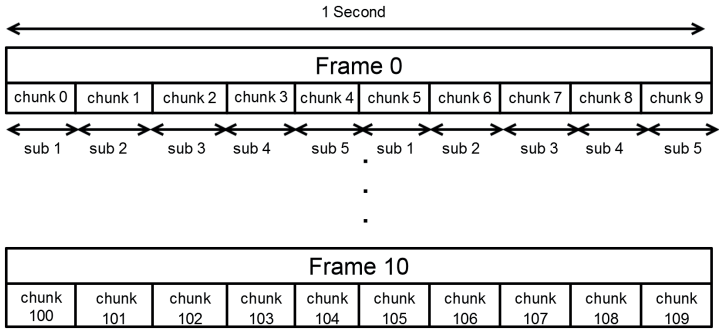


Figure 4.4: The structure of frames consisting of 10 consecutive chunks. In this example, each frame corresponds to a part of the stream of 1 sec.

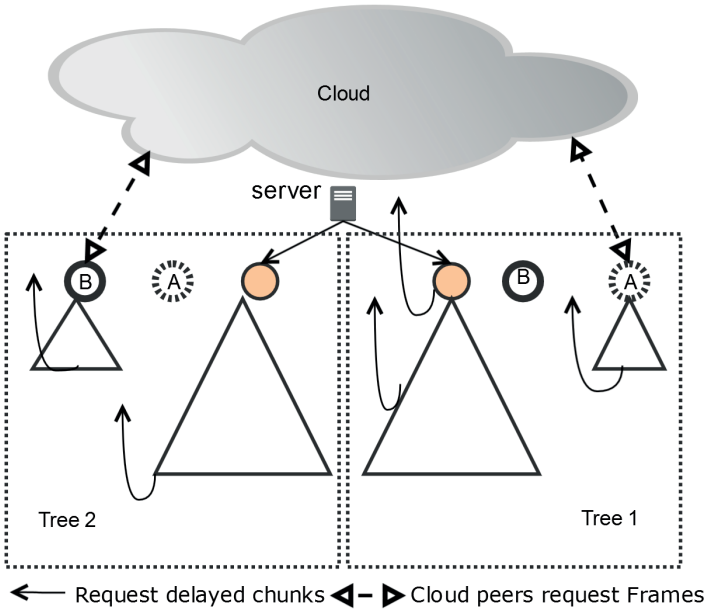


Figure 4.5: Third Technique. Assume $N = 2$ and the P2P shortage is four substreams. One peer is selected as a cloud peer for each tree. Peer A (resp. peer B) fetches the full stream from the CCDN and acts as a root in the first tree (resp. the second tree).

parents. Such new parents may not have received the latest chunks yet so they will forward duplicated chunks. Also, when asking a peer to act as a cloud peer, that peer will request the latest frame which may include some of already received chunks. Another way of wasting the chunks is when a cloud peer leaves the system after asking a large frame from the CCDN. In the evaluation given in the next section, we set the value of F to two seconds.

4.5 Evaluation

To evaluate the performance of the proposed method, we conducted extensive simulations using an event-driven packet level simulator encoded in C++. The simulator is implemented on top of a peer to peer simulator [85]. It uses a real world node-to-node latency matrix measured on the Internet [89] with the average end-to-end delay 79 ms, while it does not consider the queuing management in routers.

In the following, we compare the following four schemes:

- cloud-assisted P2P live streaming scheme Baseline which is based on the baseline model;
- scheme Orphan which is based on the first technique;
- scheme Proactive which is based on the first and second techniques; and
- scheme Frame which is based on the first, second and third techniques.

As for the metrics for evaluation, we consider: 1) the amount of data fetched from the CCDN (Section 4.5.2), 2) the number of requests handled by the cloud (Section 4.5.3), and 3) the quality of live stream including the delivery ratio and the average playback delay (Section 4.5.5). We also calculate the monetary cost of the schemes by assuming the cloud price of AWS (Section 4.5.4).

4.5.1 Setup

In the simulations, we consider a video stream of 1000 Kbps which is the bit-rate recommended by YouTube for the 480p video quality[1]. The chunk size

[1] https://support.google.com/youtube/

is 6250 bytes and the number of substreams is $N = 5$ (i.e., each substream carries 4 chunks per second). Each peer starts playback after buffering the stream for 15 seconds, and it sends a request for a chunk to the CCDN when the remaining time before playing back the chunk becomes 2 seconds. The frame size in Frame is set to $F = 40$, i.e., each frame corresponds to a part of the stream of 2 seconds.

Table 4.1: Distribution of the upload bandwidth of peers.

Bandwidth [Kbps]	384	512	768	1024	3000
Share [%]	32.9	14.7	8	28.1	16.3
Capacity	1	2	3	5	15

The number of peers is 1000 and the upload bandwidth of the server is 2 Mbps. To reflect the heterogeneity of peers, we consider the distribution of upload bandwidth summarized in Table 4.1(based on [90]); e.g., the upload bandwidth of 147 peers is 512 Kbps which indicates that the capacity of these peers is $\lfloor 512/200 \rfloor = 2$. Each peer possesses a download bandwidth exceeding 1000 Kbps to enjoy the live streaming. The resource index calculated from the above parameters is $R = 0.944$, which is slightly less than 1.00. Thus the shortage 0.056 ($= 1.000 - 0.944$) must be compensated by the CCDN.

The simulation time is fixed to 900 seconds. To realize a dynamic behavior of peers, we use real P2P traces as in [36], in which the number of online peers gradually increases from 0 to around 1000 and then reaches a stable state in which 1% of peers join/leave in each second. See Figure 4.6 for illustration. Among leaving peers, 95% of them gracefully leave and 5% of them ungracefully leave [91], where the latter is detected by observing the lag of chunks received from the parent and the missing of heartbeat messages transmitted by the children every 5 seconds. We conduct such a simulation run 16 times and take an average. The total download time per simulation run is 13025 min on average (note that it is smaller than 15000 min ($= 900 \times 1000/60$) due to churn) and the total amount of chunks downloaded by the peers is 93.16 GB, which implies that 5.21 GB of chunks corresponding to 0.056 ($= 1 - 0.944$) of downloaded chunks must be fetched from the CCDN, where 0.944 is the resource index of the hybrid P2P. A summary of parameters' values is presented in Table 4.2.

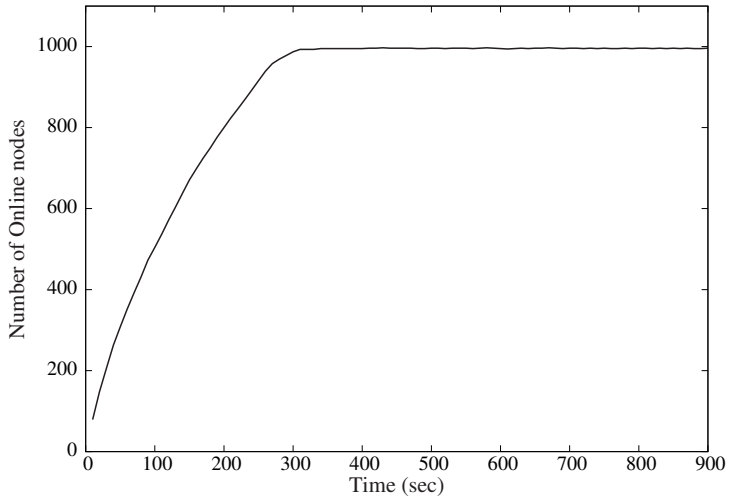


Figure 4.6: Online peers.

Table 4.2: Parameters' value in the simulation setting.

Number of peers	1000
Churn rate[per sec.]	1%
Streaming rate [Kbps]	1000
Number of substreams	5
Chuck size [Kbyte]	6.25
Frame size [chunks]	40
Buffering [sec.]	15
Server capacity [kbps]	2000
Simulation length [sec.]	900
Heartbeats interval [sec.]	5

4.5.2 Amount of Data Fetched from the CCDN

At first, we evaluate the amount of data fetched from the CCDN. Figure 4.7 summarizes the results, where the horizontal axis is the elapsed time and the vertical axis is the fetch rate per second. The temporal variance of the

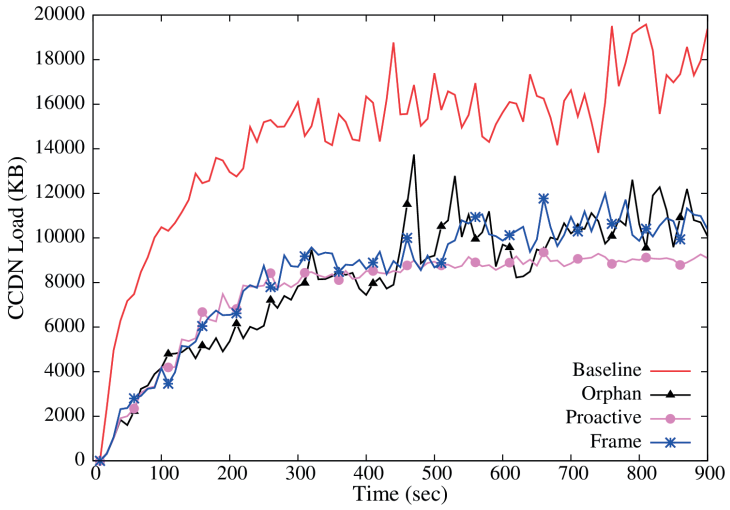


Figure 4.7: Time transition of the fetch rate from the CCDN.

fetch rate is due to the join/leave of peers and we could observe that the proposed schemes reduce the fetch rate of Baseline by about 40 %. The total amount of fetched data during simulation is shown in Figure 4.8. Since at least 5.21 GB of chunks must be fetched from the CCDN, the proposed techniques certainly mitigate additional fetch due to churn, e.g., although it is 7.28 ($= 12.49 - 5.21$) GB under Baseline, it significantly reduces to 1.33 ($= 6.54 - 5.21$) GB under Proactive.

The badness of Frame with respect to additional fetches is because of the redundancy mainly caused by the large frame size. Figure 4.9 shows the fraction of duplicated chunks among received chunks, which indicates that under Frame, 18 chunks among received 10000 chunks are duplicated. As a reason for the redundancy is the leave of cloud peers which frequently occurs in Proactive and Frame. More specifically, when a cloud peer leaves, its child who has received most recent chunks from the cloud peer might connect to a new parent which did not receive the latest chunks yet. In such a case, the new parent forwards duplicated chunks to the new child. The large frame size in Frame increases the redundancy as explained in Section 4.5.3.

To clarify the difference of Proactive and Frame in more detail, we separately evaluate data fetches conducted by cloud peers and the other peers.

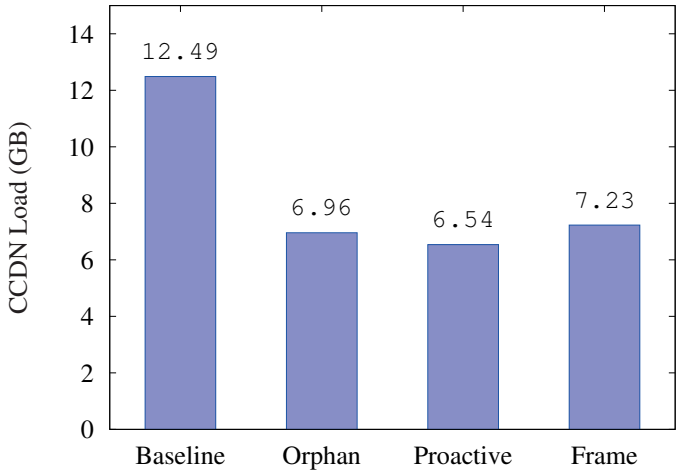


Figure 4.8: Total amount of data fetched from the CCDN.

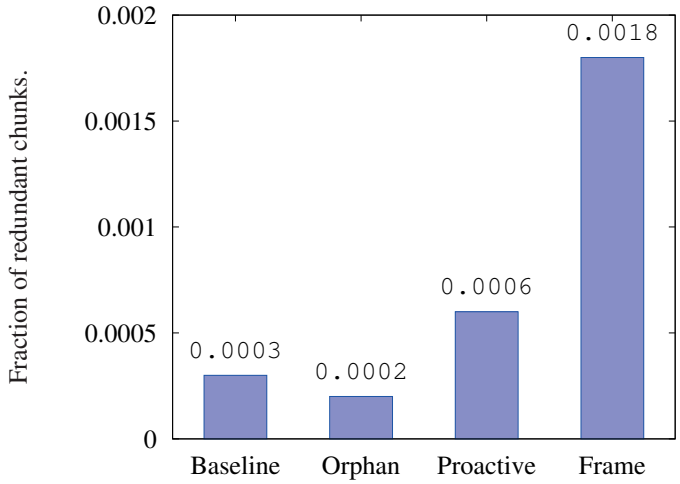


Figure 4.9: Fraction of redundant chunk.

Figure 4.10 summarizes the result. The result shows that the amount of fetch conducted by cloud peers is almost the same, i.e., 3.87 GB for Proactive and

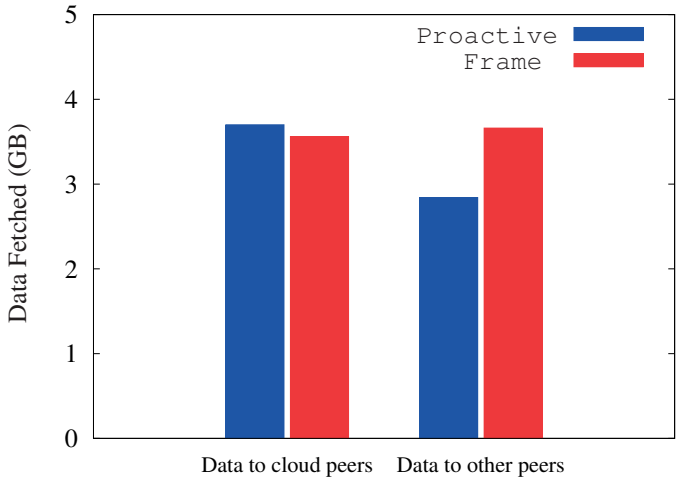


Figure 4.10: The amount of fetches conducted by cloud peers and the other peers.

3.73 GB for Frame, but the amount of fetches conducted by the other peers to mitigate churn differs, i.e., it is 2.98 GB for Proactive which is apparently less than 3.84 GB for Frame. Such a difference is due to: 1) the redundancy of Frame as was observed in Figure 4.9 and 2) the difference of average hop-count from root to the peers which is 3.07 in Proactive and 5.43 in Frame (recall that Frame invests less cloud peers than Proactive).

4.5.3 Number of Requests Handled by the Cloud

The number of requests handled by the cloud storage service is shown in Figure 4.11. The proposed schemes issue more requests than Baseline since they exploit the storage service to realize a quick match-making between orphaned peers and free capacity peers. The difference of three proposed schemes is mainly due to the difference of average hop-count, since shorter hop-count implies that fewer peers are affected by churn. In fact, the result shows that a scheme with a smaller average hop-count issues fewer requests; e.g., Proactive issues the least and Frame issues the second least.

The number of requests handled by the CCDN is shown in Figure 4.12.

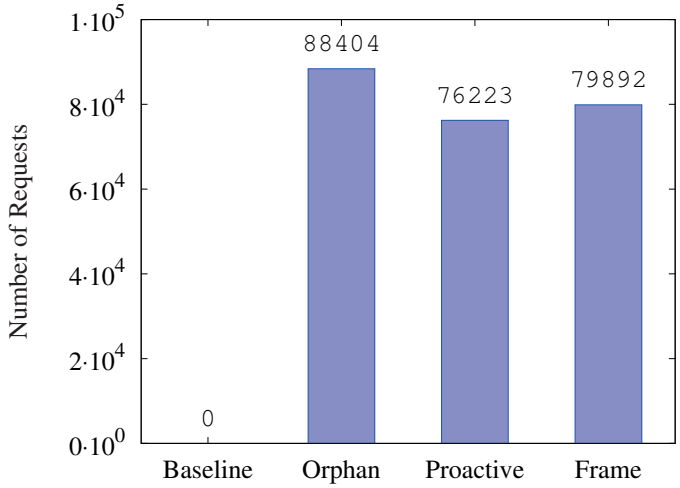


Figure 4.11: The number of requests handled by the cloud storage service.

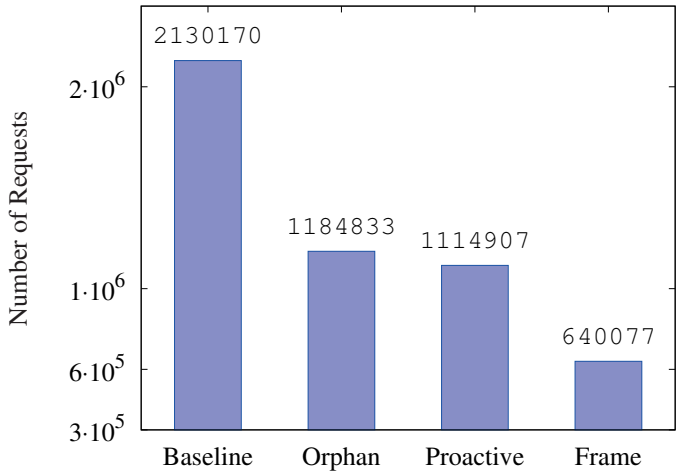


Figure 4.12: The number of requests handled by the CCDN.

The difference among Baseline, Orphan and Proactive is comparable to the difference of the amount of fetched data observed in Figure 4.8. However,

the number of requests issued in Frame is much smaller than the others (e.g., it saves 42.5% of requests compared with Proactive), which is apparently because of the effect of introducing frames.

4.5.4 Total Monetary Cost of the Cloud-Assistance

Table 4.3: AWS prices in Tokyo region.

Requests	**Price**
To the storage service	\$0.0047 per 1000 requests
To the CCDN	\$0.0090 per 10000 requests
Data Transfer out of CCDN	**Price**
$<$ 10 TB/month	\$0.140 per GB

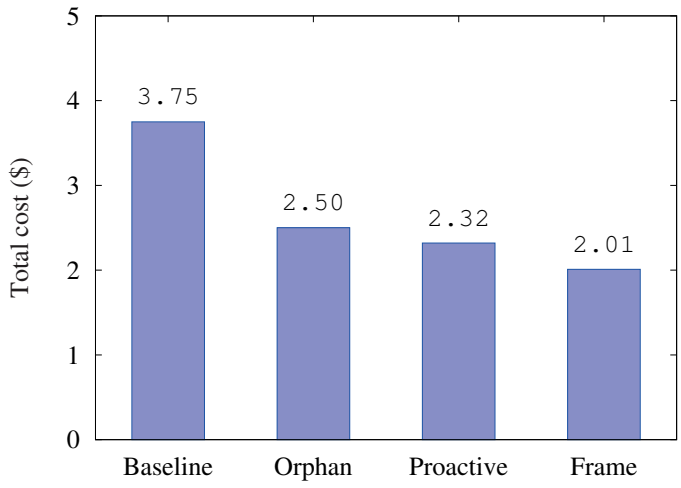


Figure 4.13: Total cost (\$).

In this subsection, we evaluate an example of the monetary cost of the cloud-assistance for each scheme. In the example, we adopted the billing

model of AWS [86, 87] [2] and assumed that all peers contact cloud services in Tokyo region. Table 4.3 shows the AWS prices in Tokyo region, which indicate that: 1) requests to the cloud storage service take $\$4.7 \times 10^{-6}$ per request, 2) requests to the CCDN take $\$0.9 \times 10^{-6}$ per request, and 3) data fetch takes \$0.14 per GB. Let R_s be the number of requests handled by the storage service, R_c the number of requests handled by the CCDN, and D the amount of fetched data in GB. Then the total monetary cost of a scheme is calculated as [3]

$$C(\$) = 4.7 \times 10^{-6} \times R_s + 0.9 \times 10^{-6} \times R_c + 0.14 \times D.$$

By substituting values obtained in previous subsections, we have the monetary cost of each scheme as shown in Figure 4.13. From the figure, we can make the following observations:

1. Orphan saves the cost of Baseline by 33.3%. This implies that the benefit of quick recovery is larger than the cost of requests handled by the cloud storage service.

2. Proactive saves the cost of Baseline by 39.5%. The improvement of Proactive is because of the proactive fetch conducted by the cloud peers and the fewer orphaned peers due to the short average hop-count.

3. Frame saves the cost of Baseline by 46%. This is due to the reduction of the number of requests handled by the CCDN.

4.5.5 Playback Delay & Delivery Ratio

Finally, we evaluate the quality of live stream delivered to the peers in terms of the average delivery ratio and the average playback delay. The**delivery ratio** is the ratio of chunks received by the playback deadline among chunks transmitted by the server. The **playback delay** is the delay between the time when a chunk is sent out from the server and the time when it is played by the peer. The maximum playback delay is the time by which the chunk

[2] billing model and prices may change over time

[3] Only major costs are listed in the example. Some costs are neglected, such as the storage cost, due to its small value and other costs are skipped, such as internal communication among cloud services, assuming they are common among all simulated schemes.

is played by all peers. Values of these metrics are sampled every 10 seconds and averaged at the end of each simulation run.

Table 4.4 summarizes the results. The results indicate that all schemes attain a sufficiently high delivery ratio 0.999 and a short maximum playback delay of 13.5 seconds, which is due to the behavior of peers such that they fetch chunks from the cloud when the time before playback becomes 2 seconds. Note that 13.5 seconds is shorter than the buffering time which is set to 15 seconds in the simulation. As for the average playback delay, we can make the following observations. At first, Proactive attains an average delay of 5.45 seconds thanks to the short depth of the delivery tree. However, Frame is worse than Proactive by 1.37 seconds and Orphan is worse than Frame by 1.47 seconds. The badness of Orphan is due to the behavior of peers so that they fetch chunks "after" detecting that the time before playback is 2 seconds, and the badness of Frame is due to the longer depth of the delivery tree and the longer time taken by cloud peers to download a frame before starting the forward of the chunks to the children. Figure 4.14 shows the time transition of the average delay of each scheme.

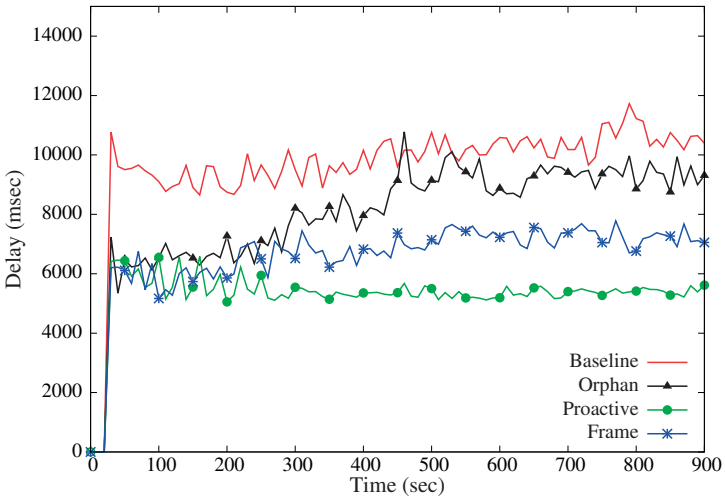


Figure 4.14: Time transition of the average playback delay.

Table 4.4: Quality of live stream in each scheme.

Metric	Baseline	Orphan	Proactive	Frame
Avg. delivery ratio	0.999	0.999	0.999	0.999
Max. playback delay [sec.]	13.561	13.548	13.377	13.447
Avg. playback delay [sec.]	9.944	8.297	5.457	6.826

4.6 CONCLUDING REMARKS

This chapter presented three techniques to reduce the cost of cloud-assistance in hybrid P2P live streaming while guaranteeing the quality of service. At first, we exploit the cloud storage service to speed up the match-making between orphaned peers and peers with a free upload capacity. Then, we invest a set of cloud peers to proactively fetch chunks from the cloud, where the number of cloud peers is dynamically adjusted based on the estimated shortage of the P2P. Finally, to reduce the number of requests handled by the cloud, we introduce the notion of frame to allow cloud peers to fetch a collection of chunks using a single request. The simulation results indicate that the resulting scheme reduces the total amount of data fetched from the cloud by 42% and the total number of requests issued to the cloud by 66% while guaranteeing a high streaming quality.

Chapter 5

Conclusions

In this thesis, the P2P live streaming is studied focusing on the quality and the cost of the service. At first the problem of maximizing the resource utilization of the participant peers is approached to improve the QoS in pure P2P. Then, to guarantee the QoS, the hybrid of P2P with the reliable cloud platform is studied. In addition, the cost of the hybrid scheme is reduced without affecting the QoS. We achieved that by proposing new schemes and methods as follows:

Firstly

The maximal of resource utilization is studied in Chapter 3 where a new scheme is proposed to attain the maximal streaming rate in pure P2P. That is done by exploiting the peer's resources (upload capacity) in the maximal manner. The peers are organized in a multiple-tree overlay with the aid of a budget-model. In the scheme, peers exchange the money (upload capacity) by transferring the children to each other. The newly joined peer will quickly contribute to the distribution process, i.e., spend its money. Exchanging the children is done in such a way that guarantees a high number of peers forwarding exactly one substream and that the trees have a short hop-count delay.

By delivering a maximal streaming rate with a short delay, we have improved the quality of service in pure P2P. That is verified by simulation where the result indicates that under the proposed scheme, the overlay network certainly converges to an efficient structure with a short hop-count delay. Moreover, it indicates that the proposed scheme gives nice

features in the homogeneous case and overcomes conventional schemes in all simulated scenarios.

Secondly

To guarantee the quality of service, the pure P2P is assisted by the cloud storage service and the cloud content delivery network (CCDN) in Chapter 4. The computing instances are avoided to attain a flexible resource renting which results in a lower cost. The assistance is achieved by storing the latest streaming chunks in the cloud storage service. Then, each peer is allowed to fetch missing chunks from the storage service through edge locations of the CCDN.

Such assistance incurred additional cost comprised of the amount of data fetched from the CCDN and the number of requests handled by the cloud. We proposed three techniques to reduce the cost of such a cloud assistance and evaluate them through extensive simulations. In the first technique, we exploited the the cloud storage service to reduce the time for orphaned peers to find a new parent in the delivery tree. That significantly reduces the system cost while incurs additional requests handled by the storage service. In the second technique, several internal nodes are selected from each tree to proactively fetch chunks from the CCDN. The selected node (peer) plays the role of a root for the corresponding tree, which reduces the height of the delivery tree and the load of other internal nodes. Finally, the third technique reduces the number of requests handled by the CCDN in the second technique by allowing peers to request a collection of chunks, called frames, instead of individual chunks.

The simulation results indicate that the proposed method reduces the total amount of data fetched from the cloud by 42% and the total number of requests issued to the cloud by 66%. Along with that, due to the assistance of the cloud, the proposed method is able to guarantee the quality of live streaming service.

Future Work

In this thesis, the video stream is divided into substreams with an equal rate, and the upload capacity of a peer is considered as its upload bandwidth

divided by that rate. That may result in a waste of some bandwidth in some peers, i.e., the reminder of the division. This problem is more server when the substream rate is high like when the stream rate is high or the number of substreams is low. Such a case is natural nowadays due to the high video quality expected by the users. Then, to have a better resource utilization with a fine granularity, it may be interesting to divide the stream into unequal substream rates. Then, each peer is matched to a substream rate in such a way to reduce the wasted bandwidth. That incurs additional complexity though. Another issue is that other metrics like stability of peers, location, underlying network delay could be considered to refine the proposed methods.

In the hybrid cloud-P2P systems, the algorithm for estimating the shortage of the P2P plays a key role in reducing the cost of the cloud assistance. It will be a healthy direction to extend the presented algorithm to consider adjusting its parameters dynamically. Also, it will be a nice idea to consider the payment per-peer rather than the per provider considered in this thesis. That means to design a system in which peers cooperate to pay and disseminate the live stream among each other with the assistance of the cloud. Furthermore, bringing the proposed method to the real work by implementing it will open the door for more empirical studies on the cost of the cloud-assistance. Finally, we need to mention that the cost-performance tradeoff is still a hot topic in the cloud-P2P live streaming.

THIS PAGE INTENTIONALLY LEFT BLANK

Bibliography

[1] YouTube, Statistics. `https://www.youtube.com/yt/press/statistics.html`, [accessed: December 2015]

[2] Cisco Systems Inc. Cisco Visual Networking Index: Forecast and Methodology, 2014-2019. `http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white_paper_c11-481360.pdf` [accessed: May 2015]

[3] C. Diot, B. N. Levine, B. Lyles, H. Kassem and D. Balensiefen. Deployment issues for the IP multicast service and architecture. *IEEE Network*, Vol. 14, No. 1, pages 78 - 88, 2000.

[4] Y. Chu, S. Rao and H. Zhang. A case for end system multicast. *Proc. of ACM SIGMETRICS,* pages 1-12, June, 2000.

[5] P. Francis. Yoid: Extending the Multicast Internet Architecture. White paper `http://www.aciri.org/yoid/`, 1999.

[6] J. Jannotti, D.K. Gifford, K.L. Johnson, M.F. Kaashoek and J.W. O'Toole, Jr. Overcast: Reliable Multicasting with an Overlay Network. *Proc. of the 4th Symp. on Operating Systems Design and Implementation*, pages 197-212, October, 2000.

[7] D. Pendarakis, S. Shi, D. Verma and M. Waldvogel. ALMI: An Application Level Multicast Infrastructure. *Proc. of 3rd USENIX Symp. on Internet Technologies and Systems*, March, 2001.

[8] I. Stoica, R. Morris, D. Karger, F. Kaashoek and H. Balakrishan. Chord: A Scalable peer-to-peer Lookup Service for Internet Applications. *Proc. of ACM SIGCOMM,* pages 149-160, August, 2001.

[9] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. *Proc. of IFIP/ACM Middleware,* pages 329-350, November, 2001.

[10] S. Ratnasamy, M. Handley, R. Karp and S. Shenker. Application-Level Multicast Using Content-Addressable Networks. *Proc. of the 3rd Int. COST264 Workshop on Networked Group Communication (NGC'1),* pages 14-29, November, 2001.

[11] S. Zhuang, B. Zhao, A. Joseph, R. Katz and J. Kubiatowicz. Bayeux: an architecture for scalable and fault-tolerant wide-area data dissemination. *ACM Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV),* pages 11-20, June, 2001.

[12] B.A. Alghazawy and S. Fujita. A scheme for maximal resource utilization in peer-to-peer live streaming. *International journal of Computer Networks & Communications,* vol. 7, no. 5, pages 13-28, 2015.

[13] C. Wu, B. Li and S. Zhao. Diagnosing Network-wide P2P Live Streaming Inefficiencies. *Proc. of IEEE INFOCOM,* pages 2731-2735, April, 2009.

[14] R. Sweha, V. Ishakian and A. Bestavros. AngelCast: Cloud-based Peer-Assisted Live Streaming Using Optimized Multi-Tree Construction. *Proc. of the 3rd Multimedia Systems Conf.,* pages 191-202, February, 2012.

[15] A. H. Payberah, H. Kavalionak, V. Kumaresan, A. Montresor and S. Haridi. CLive: Cloud-assisted P2P live streaming. *Proc. of the IEEE Int. Conf. on Peer-to-Peer Computing,* pages 79-90, September, 2012.

[16] B.A. Alghazawy and S. Fujita. Low cost cloud-assisted peer to peer live streaming. *KSII Transactions on Internet and Information Systems,* to appear.

[17] B.A. Alghazawy and S. Fujita. Probabilistic packet scheduling scheme for hybrid pull-push P2P live streaming protocols. *Proc. of 2nd Int. Conf. on Networking and Computing,* pages 248-251, December, 2011.

[18] M. Hefeeda, A. Habib, B. Botev, D. Xu and B. Bhargava. PROMISE: peer-to-peer media streaming using CollectCast. *Proc. of 11th ACM Int. Conf. on Multimedia,* pages 45-54, November,2003.

[19] V. Pai, K. Kumar, K. Tamilmani, V. Sambamurthy and A. Mohr. Chainsaw: eliminating trees from overlay multicast. *Proc. of 4th Int. Conf. on Peer-to-Peer Systems,* pages 127-140, February, 2005.

[20] X. Zhang, J. Liu, B. Li and P. Yum. CoolStreaming/DONet: A Data-driven Overlay Network for Peer-to-Peer Live Media Streaming. *Proc. of IEEE INFOCOM,* pages 2102-2111, March, 2005.

[21] F. Pianese, D. Perino, J. Keller and E. Biersack. Pulse: an adaptive, incentive based, unstructured p2p live streaming system. *IEEE Transaction on Multimedia,* vol. 9, no. 8, pages 1645-1660, 2007.

[22] J. Ghosha, M. Wang, L. Xu and B. Ramamurthy. Variable neighbor selection in live peer-to-peer multimedia streaming networks. *5th Int. Conf. on Broadband Communications, Networks and Systems,* pages 344-346, September, 2008.

[23] C. Liang and Y. Guo and Y. Liu. Is Random Scheduling Sufficient in P2P Video Streaming?. *28th Int. Conf. on Distributed Computing Systems (ICDCS),* pages 53-60, June 2008.

[24] L. Abeni, C. Kiraly and R. Lo Cigno. On the Optimal Scheduling of Streaming Applications in Unstructured Meshes. *Networking 2009,* pages 117-130, 2009.

[25] S. Banerjee, B. Bhattacharjee, C. Kommareddy and G. Varghese. Scalable application layer multicast. *Proc. of ACM SIGCOMM,* pages 205-220, October, 2002.

[26] D. Tran, K. Hua and T. Do. Zigzag: An efficient peer-to-peer scheme for media streaming. *Proc. of IEEE INFOCOM,*, pages 1283-1292, April, 2003.

[27] H. Deshpande, M. Bawa and H. Garcia-Molina. Streaming live media over peer-to-peer network. *Technical Report, Stanford University,* 2001.

[28] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron and A. Singh. SplitStream: High-bandwidth content distribution in cooperative environments. *ACM Symp. on Operating Systems Principles (SOSP),* pages 298-313, October, 2003.

[29] V. Padmanabhan, H. Wang, P. Chou and K. Sripanidkulchai. Distributing Streaming Media Content using Cooperative Networking. *ACM Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV),* pages 177-186, May, 2002.

[30] V. Venkataraman, K. Yoshida and P. Francis. Chunkyspread: Heterogeneous unstructured tree-based peer-to-peer multicast. *Proc. of 14th IEEE Int. Conf. on Network Protocols (ICNP),* pages 2-11, November, 2006.

[31] A. Magnetto, R. Gaeta, M. Grangetto and M. Sereno. TURINstream: A Totally pUsh, Robust, and effIcieNt P2P Video Streaming Architecture. *IEEE Transactions on Multimedia,* vol. 12, no. 8, pages 901-914, 2010.

[32] Q. Huang, H. Jin and X. Liao. P2P Live Streaming with Tree-Mesh based Hybrid Overlay. *Int. Conf. on Parallel Processing Workshops,* pages 55, September, 2007.

[33] T. Kouchi and S. Fujita. How to Tolerate Simultaneous Leave of Peers in Tree-Structured P2p Live Streaming Systems. *Proc. of the Int. Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA),* pages 149-155, July, 2014.

[34] F. Wang, Y. Xiong and J. Liu. mTreebone: A Collaborative Tree-Mesh Overlay Network for Multicast Video Streaming. *IEEE Transactions on Parallel and Distributed Systems,* vol. 21, no. 3, pages 379 - 392, 2010.

[35] S. Xie, B. Li, Y. Keung and X. Zhang. Coolstreaming: Design, Theory, and Practice. *IEEE Transactions on Multimedia,* Vol. 9, No. 8, pages 1661-1671, 2007.

[36] M. Zhang, Q. Zhang, L. Sun and S. Yang. Understanding the Power of Pull-Based Streaming Protocol: Can We Do Better? *IEEE Journal on Selected Areas in Communications,* vol. 25, no. 9, pages 1678-1694, December, 2007.

[37] C.Y. Keong, P.K. Hoong and C.-Y. Ting. Efficient hybrid push-pull based P2P media streaming system. *IEEE 17th Int. Conf. on Parallel and Distributed Systems,* pages 735-740, December, 2011.

[38] A. Ouali, B. Kerherve and B. Jaumard. A packet-loss resilient push scheduling for mesh overlays. *Proc. of IEEE Consumer Communications and Networking Conference (CCNC),* pages 611-615, January, 2011.

[39] L. Bracciale, F. Lo Piccolo, D. Luzzi, S. Salsano, G. Bianchi and N.Blefari-Melazzi. A push-based scheduling algorithm for large scale P2P live streaming. *Proc. of 4th Int. Telecommunication Networking Workshop on QoS in Multiservice IP Networks,* pages 1-7, February, 2008.

[40] N. Liu, J. Yang, H. Cui, G. Zheng and H. Chen. Efficient push-based packet scheduling for Peer-to-Peer live streaming. *Cluster Computing,* vol. 16, no. 4, pages 767-777, December, 2013.

[41] A. Payberah, J. Dowling, F. Rahimian and H. Haridi. Sepidar: Incentivized market-based p2p live-streaming on the gradient overlay network. *Proc. of IEEE International Symposium on Multimedia,* pages 1-8, December, 2010.

[42] G. Tan and S. Jarvis. A payment-based incentive and service differentiation scheme for peer-to-peer streaming broadcast. *IEEE Transaction on Parallel and Distributed Systems,* vol. 19, no. 7, pages 940-953, 2008.

[43] S. Kanda and S. Fujita. Incentive Scheme for P2p Live Streaming Systems Being Aware of the Upload Capability of the Participants. *Proc. of the Int. Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA),* pages 91-97, July, 2014.

[44] M. Wang and B. Li. R2: Random Push with Random Network Coding in Live Peer-to-Peer Streaming. *IEEE Journal on Selected Areas in Communications,* vol. 25, no. 9, pages 1655 - 1666, 2007.

[45] C. Feng and B. Li. On large-scale peer-to-peer streaming systems with network coding. *Proc. of 16th ACM int. conf. on Multimedia,* pages 269-278, October, 2008.

[46] Y. Liu. On the Minimum Delay Peer-to-Peer Video Streaming: how Realtime can it be?. *Proc. of the 15th int. conf. on Multimedia,* pages 127-136, September, 2007.

[47] G. Bianchi, N.B. Melazzi, L. Bracciale, F. Lo Piccolo and S. Salsano. Streamline: An Optimal Distribution Algorithm for Peer-to-Peer Real-Time Streaming. *IEEE Transactions on Parallel and Distributed Systems,* vol. 21, no. 6, pages 857-871, 2010.

[48] S. Fujita. Optimal Serial Broadcast of Successive Chunks. *Theoretical Computer Science,* vol. 574, pages 3-9, 2015.

[49] R. Kumar, Y. Liu and K. W. Ross. Stochastic Fluid Theory for P2P Streaming Systems. *Proc. of IEEE INFOCOM,* pages 919-927, May, 2007.

[50] Y. Guo, C. Liang and Y. Liu. AQCS: Adaptive Queue-Based Chunk Scheduling for P2P Live Streaming. *Proc. of IFIP Networking,* pages 433-444, May, 2008.

[51] L. Massoulie, A. Twig, C. Gkantsidis and P. Rodriguez. Randomized decentralized broadcasting algorithm. *Proc. of IEEE INFOCOM,* pages 1073-1081, May, 2007.

[52] J. Li, and P. A. Chou and C. Zhang. Mutualcast: an efficient mechanism for content distributionin a p2p network. *Microsoft Research, MSR-TR-2004 100,* 2004.

[53] T. Nguyen, K. Kolazhi, R. Kamath, S. Cheung and D. Tran. Efficient Multimedia Distribution in Source Constraint Networks. *IEEE Transactions on Multimedia,* vol. 10, no. 3, pages 532-537, 2008.

[54] S. Liu, R. Zhang-Shen, W. Jiang, J. Rexford and M. Chiang. Performance bounds for peer-assisted live streaming. *Proc. of ACM SIGMETRICS,* pages 313-324, June, 2008.

[55] S. Liu, M. Chen, S. Sengupta, M. Chiang, J. Li and P. A. Chou. P2P Streaming Capacity under Node Degree Bound. *Proc. of 30th Int. Conf. on Distributed Computing Systems (ICDCS),* pages 587-598, June, 2010.

[56] S. Sengupta, S. Liu, M. Chen, M. Chiang, J. Li and P. A. Chou. Peer-to-Peer Streaming Capacity. *IEEE Transactions on Information Theory,* vol. 57, no. 8, pages 5072-5087, 2011.

[57] S. Zhang, Z. Shao, M. Chen and L. Jiang. Optimal Distributed P2P Streaming under Node Degree Bounds. *IEEE/ACM Transactions on Networking,* vol. 22, no. 3, pages 717-730, 2014.

[58] M. S. Raheel, R. Raad and C. Ritz. Achieving maximum utilization of peer's upload capacity in p2p networks using SVC. *Peer-to-Peer Networking and Applications*, online, 2015.

[59] D. Xu, S.S. Kulkarni, C. Rosenberg and H.-K. Chai. Analysis of a CDN-P2P hybrid architecture for cost-effective streaming media distribution. *Multimedia Systems,* vol. 11, no. 4, pages 383-399, 2006.

[60] Z. Chen, H. Yin, C. Lin, X. Liu and Y. Chen. Towards a trustworthy and controllable peer-server-peer media streaming: an analytical study and an industrial perspective. *Proc. of IEEE GLOBECOM,* pages 2086-2090, November, 2007.

[61] X. Liu, H. Yin, C. Lin, Y. Liu, Z. Chen and X. Xiao. Performance analysis and industrial practice of peer-assisted content distribution network for large scale live video streaming. *Proc. of the 22nd Int. Conf. on Advanced Information Networking and Applications,* pages 568-574, March, 2008.

[62] H. Yin, X. Liu, T. Zhan, V. Sekar, F. Qiu, C. Lin, H. Zhang and B. Li. Design and Deployment of a Hybrid CDN-P2P System for Live Video Streaming: Experiences with LiveSky. *Proc. of the 17th ACM Int. Conf. on Multimedia,* pages 25-34, October, 2009.

[63] Z.-H. Lv, L.-J. Chen, J. Wu, D. Deng, S.-J. Huang and Y. Huang. PROSE: Proactive, Selective CDN Participation for P2P Streaming. *Journal of Computer Science and Technology,* vol. 28, no. 3, pages 540-552, May, 2013.

[64] Z. Lu, Y. Wang and Y.R. Yang, An Analysis and Comparison of CDN-P2P-hybrid Content Delivery System and Model. *Journal of Communications,* vol. 7, no. 3, March, 2012.

[65] E. Veloso, V. Almeida, W. Meira, A. Bestavros and S. Jin. A Hierarchical Characterization of a Live Streaming Media Workload *IEEE/ACM Transactions on Networking,*, vol. 14, no. 1, pages 133-146, 2006.

[66] B. Li, G.Y. Keung, S. Xie, F. Liu, Y. Sun and H. Yin. An Empirical Study of Flash Crowd Dynamics in a P2P-Based Live Video Streaming System. *Proc. of IEEE GLOBECOM,* pages 1-5, December, 2008.

[67] A. Montresor and L. Abeni. Cloudy Weather for P2P, with a Chance of Gossip. *Proc. of the IEEE Int. Conf. on Peer-to-Peer Computing,* pages 250-259, September, 2011.

[68] P. Michiardi, D. Carra, F. Albanese and A. Bestavros. Peer-assisted content distribution on a budget. *Computer Networks,* vol. 56, no. 7, pages 2038-2048, May, 2012.

[69] X. Jin and Y.-K. Kwok. Cloud Assisted P2P Media Streaming for Bandwidth Constrained Mobile Subscribers. *Proc. of 16th IEEE Int. Conf. on Parallel and Distributed Systems (ICPADS),* pages 800-805, December, 2010.

[70] X. Wang, T.T. Kwon, Y. Choi, H. Wang and J. Liu. Cloud-assisted adaptive video streaming and social-aware video prefetching for mobile users. *IEEE Wireless Communications,* vol. 20, no. 3, pages 72-79, June, 2013.

[71] H. Li, L. Zhong, J. Liu, B. Li and K. Xu. Cost-Effective Partial Migration of VoD Services to Content Clouds. *Proc. of IEEE Int. Conf. on Cloud Computing (CLOUD),* pages 203-210, July, 2011.

[72] V. Rocha, F. Kon, R. Cobe and R.a Wassermann. A hybrid cloud-P2P architecture for multimedia information retrieval on VoD services. *Computing,* online, September, 2014.

[73] Z. Huang, C. Mei, L.E. Li and T. Woo. CloudStream: delivering high quality streaming videos through a cloud-based SVC proxy. *Proc. of IEEE INFOCOM,* pages 201-205, April, 2011.

[74] M.M. Hassan, B. Song, A. Almogren, M.S. Hossain, A. Alamri, M. Alnuem, M.M. Monowar and M.A. Hossain. Efficient Virtual Machine Resource Management for Media Cloud Computing. *KSII Transactions on Internet and Information Systems,* vol. 8, no. 5, pages 1567-1587, May, 2014.

[75] L. Zhou and H. Wang. Toward Blind Scheduling in Mobile Media Cloud: Fairness, Simplicity, and Asymptotic Optimality. *IEEE Transaction on Multimedia,* vol. 15, no. 4, pages 735-746, June, 2013.

[76] L. Zhou, Z. Yang, J.J.P.C. Rodrigues and M. Guizani. Exploring blind online scheduling for mobile cloud multimedia services. *IEEE Wireless Communications,* vol. 20, no. 3, pages 54-61, June, 2013.

[77] Y. Wu, C. Wu, B. Li, X. Qiu and F.C.M. Lau. Cloudmedia: When cloud on demand meets video on demand. *Proc. of the Int. Conf. on Distributed Computing Systems (ICDCS),* pages 268-277, June, 2011.

[78] D. Niu, H. Xu, B. Li and S. Zhao. Quality-assured cloud bandwidth auto-scaling for video-on-demand applications. *Proc. of IEEE INFOCOM,* pages 460-468, March, 2012.

[79] J. He, Y. Wen, J. Huang and D. Wu. On the cost-qoe tradeoff for cloud-based video streaming under amazon ec2's pricing models. *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 24, no. 4, pages 669-680, April, 2014.

[80] F. Wang, J. Liu and M. Chen. CALMS: Cloud-assisted live media streaming for globalized demands with time/region diversities. *Proc. of IEEE INFOCOM,* pages 199-207, March, 2012.

[81] W. Gu, X. Zhang, B. Gong, W. Zhang and L. Wang. VMCAST: A VM-Assisted Stability Enhancing Solution for Tree-Based Overlay Multicast. *PLoS ONE,* vol. 10, no. 11, online, November, 2015.

[82] M. Castro, P. Druschel, A. Kermarrec, and A. Rowstron SCRIBE: A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communications,* vol. 20 no. 8, pages 1489-1499, October, 2002.

[83] L. Bracciale, F. Lo Piccolo, D. Luzzi and S. Salsano. OPSS: an overlay peer-to-peer streaming simulator for large-scale networks. *SIGMETRICS Performance Evaluation Review,* vol. 35, no. 3, pages 25-27, December, 2007.

[84] S. Saroiu, K. P. Gummadi and S. D. Gribble. A Measurement Study of Peer-to-Peer File Sharing Systems. *Proc. SPIE,* vol. 4673, pages 156-170, 2001.

[85] Meng Zhang. Peer to Peer streaming simulator. `http://media.cs.tsinghua.edu.cn/~zhangm/` [accessed: May 2015].

[86] Amazon CloudFront pricing. `http://aws.amazon.com/cloudfront/pricing` [accessed: May 2015].

[87] Amazon S3 Pricing. `http://aws.amazon.com/s3/pricing/` [accessed: May 2015].

[88] X. Hei, C. Liang, J. Liang, Y. Liu and K.W. Ross. A Measurement Study of a Large-Scale P2P IPTV System. *IEEE Transactions on Multimedia,* vol. 9, no. 8, pages 1672-1687, December, 2007.

[89] Meridian project. Meridian node to node latency matrix (2500x2500). `http://www.cs.cornell.edu/People/egs/meridian/data.php` [accessed May: 2015].

[90] Z. Liu, Y. Shen, K.W. Ross, S.S. Panwar and Y. Wang. Substream Trading: Towards an Open P2P Live Streaming System. *Proc. of the IEEE Int. Conf. on Network Protocols,* pages 94-103, October, 2008.

[91] B. Li, S. Xie, G.Y. Keung, J. Liu, I. Stoica, H. Zhang, and X. Zhang. An Empirical Study of the Coolstreaming+ System. *IEEE Journal on Selected Areas in Communications,* vol. 25, no. 9, pages 1627-1639, December, 2007.