Peer-to-Peer Netw. Appl. (2015) 8:414–431
DOI 10.1007/s12083-014-0263-5

Advanced bootstrap and adjusted bandwidth for content distribution in peer-to-peer live streaming

Jun-Li Kuo · Chen-Hua Shih · Cheng-Yuan Ho · Yaw-Chung Chen

Received: 8 February 2012 / Accepted: 4 March 2014 / Published online: 26 March 2014

Abstract Today peer-to-peer (P2P) systems have been deployed for various Internet multimedia applications such as live video streaming service. However, both peer churn and upload capacity insufficiency are inherent problems causing long start-up delay and unstable playback quality. Therefore, we introduce a scalable application-layer multicast algorithm, ABCD-P2P, which inherits the short end-to-end delay and low control overhead due to the push delivery scheme. The logical overlay is close to the physical topology (i.e., geographic proximity) in ABCD-P2P, so a short streaming multicast delivery path reduces the number of hops and avoids the network bottleneck or far routing. Both an advanced bootstrap mechanism and an adjusted bandwidth mechanism are suitable for the asymmetric bandwidth network. The mathematical analysis and simulation results demonstrate that our proposed scheme can achieve the goals of high playback smoothness, short start-up delay, short end-to-end delay, low control overhead, and short recovery time.

J.-L. Kuo (✉)
NSD 1, CNSBG, Foxconn, Hsinchu, Taiwan, No. 5, Hsin Ann Road, Hsinchu City 300, Taiwan
e-mail: estar.cs95g@nctu.edu.tw

C.-H. Shih
Chung Shan Institute of Science and Technology, Taoyuan, Taiwan
e-mail: shihch@csie.nctu.edu.tw

C.-Y. Ho
Advanced Research Institute, Institute for Information Industry, Taipei, Taiwan
e-mail: cyho@csie.nctu.edu.tw

Y.-C. Chen
Computer Science, National Chiao Tung University, Hsinchu, Taiwan
e-mail: ycchen@cs.nctu.edu.tw

Keywords Peer-to-peer network · Application-layer multicast · Overlay design · Tree push · IPTV · P2P TV

1 Introduction

With the fast growth of broadband access networks, live media streaming services over Internet have become popular in recent years. Nowadays, more and more people watch television through Internet, which is called IPTV (Internet Protocol TeleVision) [1]. YouTube alone hosted 45 terabytes of videos and attracted 1.73 billion views in 2006, then the number of IPTV subscribers grew to 36.8 million and its revenue increased to $4.6 billion in 2009 [2]. IPTV may be considered as the next killer network application.

It is effective to deliver live media streams to the users by content distribution network (CDN) or IP multicast technique, however, the high cost of CDN deployment and the business policies problems in IP multicast obstruct the service deployment. For these reasons, application layer multicast and peer-to-peer (P2P) technology are employed to support the live media streaming today [3]. The live streaming multicast through the P2P network is called P2P IPTV and more studies are interested in this topic [4].

P2P technology has been successfully applied to file sharing [5] and video on demand (VoD) [6, 7] for high scalability and low commercial cost. The major advantage of P2P solution is to efficiently overcome the bandwidth limitations of the traditional client–server model. In addition, P2P solutions could enormously reduce the maintenance cost of entire system due to the peer-paralleling distributed computing. However, the design of live TV is more difficult than that of file sharing and VoD due to the in-time requirement of streaming delivery and the limited availability of future content.

Measurement studies [8] of P2P live streaming system point out that the major limitation is *peer churn*,[1] and the annoying bottleneck of service provisioning is the insufficient upload bandwidth. Measurement studies [9] also point out that the long start-up delay, the high control overhead, and the long end-to-end latency are the shortcomings of all existing P2P IPTV systems. These inherent shortcomings lead to the low quality of service (QoS) and the low quality of experience (QoE).

To offer users satisfactory QoE, we propose a novel P2P streaming scheme, called ABCD-P2P (Advanced Bootstrap and Adjusted Bandwidth for Content Distribution P2P IPTV). The adjusted bandwidth mechanism enhances the playback smoothness, the advanced bootstrap mechanism shortens the start-up delay, and the proximity of content delivery shortens the end-to-end delay and shortens recovery time. ABCD-P2P provides the overlay protocol and integrated scheme for the efficient delivery in dynamic and asymmetric network. To guarantee the scalability and stability, the logical overlay is close to the physical topology for the proximity delivery.

In the next section, we present the related works for a discussion of P2P live streaming. Section 3 introduces our proposed scheme and the design of ABCD-P2P. Section 4 presents the mathematical analysis and a discussion of compared algorithms. In Section 5, the simulation results demonstrate that our proposed scheme works remarkably in large and dynamic scale P2P network. Section 6 concludes the paper.

2 Related works

Due to the limited deployment of IP multicast, application layer multicast has attracted more and more research interests and efforts. Since the overlay live streaming concept was first introduced in Narada [10], many studies, including CoolStreaming [11] and ZigZag [12], followed this trade of live streaming overlay multicast to design the improved P2P protocol for real-time video service in large scale. CoolStreaming is on *mesh* base, and ZigZag is on *tree* base, both they are the successful P2P live streaming systems.

2.1 Tree or mesh

The P2P solutions can be divided simply into the mesh-based overlay and the tree-based overlay. Briefly speaking, the peers of mesh have unfixed parents and children. All peers can contribute upload capacity since there are no *leaf* peers.[2] The most important advantage of mesh is a great churn tolerance and a high scalability. Because mesh scheme usually adopts *pull* algorithm to collect data, mesh scheme is also called as *mesh-pull* scheme. On the other hand, tree-based overlay features some advantages, such as, simple to design, efficient to deliver, and stable to support streams. Because tree scheme usually adopts *push* algorithm to collect data, tree scheme is also called as *tree-push* scheme.

However, mesh-pull or tree-push is not invariable and unalterable. For mesh-based overlay as examples, CoolStreaming [13] was a representative work of mesh-pull scheme, and the new version of CoolStreaming [14] adopted hybrid *pull-push* scheme. AnySee [15] was also a work of mesh-based overlay, and the new version of AnySee [16] constructed the hybrid *tree-mesh* overlay. GridMedia [17] adopted hybrid pull-push scheme to improve mesh-pull scheme. PRIME [18] presented the *mesh-push* scheme to revolutionize the traditional mesh-pull scheme.

On the other hand, typically there are two types of the tree-based overlay: *single-tree* algorithms and *multiple-tree* algorithms. Both NICE [19] and ZigZag [20] were the representative works of single-tree overlay. Afterward, DHCM [21] improved the short *path length*[3] and the stable data rate from NICE, and FollowTree [22] improved the short end-to-end delay from ZigZag. We discover the improvements of single tree and compare with NICE and ZigZag, which are introduced in Section 2.2 and Section 2.3 respectively.

The overlays of Overcast [23] and SpreadIt [24] were both single-tree bases in early periods. On the other hand, SplitStream [25] was the representative work of multiple-tree overlay. BACS [26] improved SplitStream and used cluster tree solve bandwidth instability, unfairness, and asynchrony.

Generally speaking, tree-based scheme pays much attention to the construction of overlay, but mesh-based scheme pays much attention to the data scheduling [27]. How to decide mesh or tree is a tradeoff about design limitations. In summary:

- Single-tree structure is simple and efficient but vulnerable to dynamics;
- Multiple-tree structure is more resilience but more complex than single tree;
- Mesh structure is more robust but occurs to longer delay and more control overhead.

Therefore, the mesh-pull mode can work well with the high churn rate, while the tree-push mode can efficiently reduce the accumulated latency.

[1] Churn means peers arriving and departing at a high rate. The dynamics of peer churn disrupt content delivery and adversely affect the delivered quality to participating peers.

[2] A leaf peer means an end-edge peer without its children like as a leaf of tree.

[3] The path length means the ratio of logical overlay path to physical topology path.

2.2 NICE

NICE (Internet Cooperative Environment) is an application-layer multicast protocol for low-bandwidth data streaming with large receivers to improve end-to-end delay and to reduce control overhead. NICE uses a hierarchical tree-based arrangement to establish the topological clusters and control the data delivery paths, hence NICE must need a central algorithm to derive the data delivery paths and cluster leaders to manage the peers. The cluster leader not only helps to select an optimal path with the shortest end-to-end delay due to the latency-based hierarchical structure, but also reduces the control overhead because of inter-cluster multicast instead of broadcast. The cluster leader has the minimum distance to all other peers in the same cluster. Therefore, the choice of cluster leader is important for a new joining peer to find its appropriate position quickly.

NICE can improve the end-to-end delay because NICE uses end-to-end latency as the distance metric between peers. NICE also improves the quality of data path because NICE optimizes the latency-based clusters. Moreover, NICE carries messages hierarchically through cluster leaders to reduce the control overhead. However, the major shortcoming of NICE is the high complexity and high overhead of cluster maintenance and refinement.[4]

To derive data delivery paths is relative to the maintenance and refinement of clusters, NICE splits and merges the clusters to guarantee the optimal data multicast. In addition, when a peer leaves or joins, NICE executes the selection of cluster leader and ensures that there is no loop in all data paths. As a result, we can know that peer churn and cluster refinement lead to the most control overhead. According to the analyses [19], NICE has a worst-case node degree O(log n) with n peers, a worst-case control overhead is O(log n), an average control overhead is O(k) while k is the cluster size, a worst-case join message overhead is O(k log n), and an average join latency is O(log n).

2.3 ZigZag

ZigZag is a peer-to-peer technique for single-source media streaming. ZigZag is similar with NICE, ZigZag also uses a hierarchical tree-based cluster to improve end-to-end delay and to reduce control overhead. In addition, ZigZag takes the load balance into account. Each cluster has a *head* and an *associate head* to organize the multicast tree, and they are responsible for the orphan to find new parent when peer leaving. Because ZigZag efficiently solves the bottleneck around the *rooter*[5] and cluster leader, ZigZag performs shorter end-to-end delay and less control overhead than NICE. Although the cluster refinement cost of ZigZag is more than the cost of NICE, ZigZag has the shorter delivery path and more balancer structure than NICE.

ZigZag consists of two important entities: *administrative organization* and *multicast tree*. Administrative organization represents the logical relationships among the peers, and multicast tree represents the physical relationships to make peer link together to receive real content. The peers play different and complex roles to organize the clusters in the hierarchical arrangement of administrative organization. The roles include *subordinate*, head, *foreign head*, and associate head. However, the major shortcoming of ZigZag is that the ratio of leaf peers to intermediate peers is too high to contribute upload capacity sufficiently.

As above assumption, network size is n (peers) and cluster size is between k and $3k-1$, according to the analyses [20], the worst-case node degree is $O(k^2)$, the height of the multicast tree is $O(\log_k n)$, the worst-case control overhead is $O(k \log_k n)$, the join latency is $O(\log_k n)$, the join overhead is $O(k^2 \log_k n)$, and the worst-case refinement overhead is $O(k^2)$.

2.4 Summary of tree-based systems

We take NICE and ZigZag as examples to discuss about the tree-based overlays. We can discover four shortcomings in the follow discussion: (1) High refinement overhead. (2) Too many leaf peers. (3) The traffic bottleneck of leader. (4) Long recovery time.

- Both NICE's cluster leader and ZigZag's cluster header periodically check the size of their clusters, and cluster is sometimes split and merged appropriately. When the cluster becomes oversize or undersize due to peer churn, both NICE and ZigZag execute cluster maintenance to keep the size in $[k, 3k-1]$. In tree-based structure, most control overhead comes from the refinement operator.
- In experiment analysis of both NICE and ZigZag, there is a large number of leaf peers which leads to the low upload capacity. The leaf peer can be seen as the *free-rider* which is difficult to provide its upload capacity.
- The leader is an important role in cluster-based overlay and a leader manages everything of its cluster. A fixed constant for peer degree may lead to the traffic bottleneck. Moreover, the overhead of leader is much larger than the overhead of ordinary peer.

[4] Cluster split and merge belong to the NP-hard problem. The cluster leader must load the complexity of computing and handle many control messages to initiate split and merge operator.

[5] The rooter is a root node of tree-based P2P network. A rooter is always a P2P tracker, data server, or content provider.

- The departure of peer causes the broken delivery path, and the crash due to departure of cluster leader is more serious than the crash due to departure of ordinary peer. The recovery mechanism spends too much time on the arrangement of new path and the reselection of cluster leader.

In this paper, we try to overcome these shortcomings, and we present our proposed scheme in next Section.

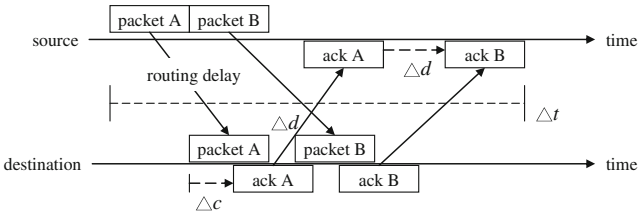

Fig. 1 Delay estimator

3 Our proposed scheme

The characteristic of overlay proximity assists in the advanced bootstrap mechanism and adjusted bandwidth mechanism, so we design five components and a set of protocols to achieve the high proximity for the efficient delivery and recovery. The integrated solution works well to get the high QoE in the large-scale and asymmetric netwotk.

3.1 The components of proposed scheme

In this paper, we propose a novel cooperation scheme for P2P live streaming. According to the designed issues of P2P IPTV consisting of (1) delay sensitivity, (2) bandwidth bottleneck, (3) initiation process, (4) leave legacy, and (5) lifetime expectation, we design five components to take charge of respective issues as shown in Table 1.

The five components cooperate to maintain P2P overlay. Delay Estimator has the highest priority among them when a conflicting contradiction happened, because the time sensitivity for delay tolerance is the important metric in P2P streaming service. The conflicting contradiction often happens on the dynamic and unbalanced overlay, which leads to the low correlativity of components. For example, peer P has high upload capability and short lifetime, and peer P' has low upload capability and long lifetime; peer P is selected because Bandwidth Estimator has a priority.

3.1.1 Delay sensitivity

P2P live streaming service is a real time application that cannot tolerate a long latency. We design a Delay Estimator to evaluate the end-to-end delay. The end-to-end delay can indicate the length of routing path and the available bandwidth. Figure 1 illustrates the execution of Delay Estimator. The source sends two continuous packets to the destination. When receiving the packet, the destination sends back the acknowledgement (Ack) to the source. Because of network routing time and finite bandwidth, the time of two continuous packets arriving has a delay like a gap denoted as $\triangle d$. Because of finite computing resource for handling packets, there is a computing delay $\triangle c$ between receiving packet and sending Ack. If a peer serves many neighbors and is very busy, $\triangle c$ is large. From the first packet sending to the last Ack returning, the total time $\triangle t$ is evaluated for delay. Therefore, $\triangle t$ can be briefly estimated for routing delay, network dynamics, available bandwidth, and computing ability between two peers. We use $\triangle t$ as the end-to-end delay evaluation in Delay Estimator. Note that the packet size must be big enough to sense $\triangle c$ and $\triangle d$.

3.1.2 Bandwidth bottleneck

In this paper, a peer owns its neighbors depending on its uploading capacity. In some approaches, such as NICE, ZigZag, DHCM, FollowTree, SplitStream, and BACS, a peer decides the connections of its neighbors depending on a constant. Those methods were simple to organize their overlay, but they did not consider the bandwidth utilization to support the continuous stream. Hence, a bandwidth bottleneck was usually revealed in the cluster leader or the intermediate node in those above methods. In our proposed scheme, how many neighbors a peer connects depends on its upload bandwidth. The Bandwidth Estimator can measure the upload bandwidth capability via the system call of operating system

Table 1 The components of our proposed scheme

Components	Issues	Responsibility	Priority
Delay estimator	Delay sensitivity	To evaluate the end-to-end delay	1
Bandwidth estimator	Bandwidth bottleneck	To measure the upload capability	2
Joining helper	Initiation process	To shorten the start-up delay	3
Leaving helper	Leave legacy	To recover the broken path	4
Lifetime looker	Lifetime expectation	To measure the lifetime	5

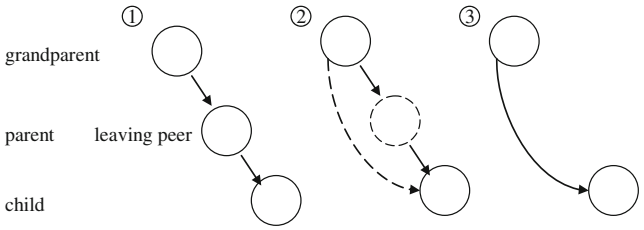

Fig. 2 Leaving helper

or the network protocol. Unlike download best effort in pull scheme, we adopt the push scheme, so we take account of upload bandwidth due to most asymmetric digital subscriber lines. The bandwidth capability of upstream node influences the performance of entire overlay. In our proposed scheme, Bandwidth Estimator is used to avoid bandwidth bottleneck in upstream. Therefore, ABCD-P2P can locate the peers with low upload capacity in the leaves.

3.1.3 Initiation process

A successful live streaming service should shorten the start-up delay. Focus on P2P overlay, an initiation process of each peer implies a new joining process. The new joining peer visits the rooter at first, and then the rooter gives the candidates to the peer to connect. We design a Joining Helper to help the new joining process, and the major task is to evaluate the network ability of candidates. It uses neither the complex cluster merge/split algorithm nor the complex stream merge/split algorithm. New joining process only considers the minimum start-up delay.

3.1.4 Leave legacy

A leave legacy means that a leaving peer can depart gracefully and share its resource before disconnecting to help to recover P2P overlay. We design a Leaving Helper to let peer leave gracefully to recover the broken path. When a peer closes our P2P application, it connects its upstream and downstream actively. So, the downstream peer does not become an orphan when its parents leave as Fig. 2 shown. However, this active connection cannot always work well because the ability of grandparent is not equal to the ability of parent. The refined algorithm helps to solve the problem.

3.1.5 Lifetime expectation

GridMedia [28] pointed out "there was a statistically positive correlation between the elapsed online duration and the expected remaining online time." PRIME [29] pointed out "a peer remains longer, and it churns less." There is a strong correlation between the lifetime and the probability of leaving. This means a peer has the longer time remaining, and it has the less probability of leaving. Therefore, we design a Lifetime Looker to know the lifetime of each peer, and try to arrange long lifetime peer at the top, but arrange short lifetime peer at the leaf.

3.2 The protocol of proposed scheme

The logical overlay derives the data paths to be close to the physical topology, thus the overlay is crucial for scalability and proximity. Instead of cluster-based approach [20] or depth-balanced approach [25], we use end-to-end delay as the evaluation for proximity to construct a P2P overlay. Integrated with the advanced bootstrap mechanism and the adjusted bandwidth mechanism, the proposed protocol can pursue the short start-up delay and high playback smoothness for the high QoE.

3.2.1 New peer joining

Figure 3 explains how a new peer joins in our proposed system:

(1) When a new peer p_{new} joins the P2P network, it connects the rooter firstly and sends a *New Join Message* to the rooter.
(2) When rooter receives the *New Join Message*, it executes the Delay Estimator to estimate the delay between itself and p_{new}, denoted as T_{r2p}.[6]
(3) Rooter executes the Joining Helper to response the candidates of parents according to the similar T_{r2p} and sends a *Join Accept Message* to p_{new}. If rooter cannot find the suitable candidates, rooter can provide video streaming to p_{new} temporarily. Algorithm 1 describes how to complete the new-joining process at rooter.
(4) When p_{new} receives the *Join Accept Message*, it executes the Delay Estimator to estimate T_{p2p} (denoted as the latency between two peers) for all candidates, and then p_{new} executes the Joining Helper to select a candidate with minimum T_{p2p} to become its child as Algorithm 2 described.
(5) After this parent knows the new peer p_{new} via a *Notice Message*, parent sends *Update Message* to inform rooter for a new overlay and pushes video streaming to p_{new}.

In summary, to shorten the delay of data delivery and message routing, a peer selects its parents according with short delay. The delay estimation not only indicates the logical distance, but also estimates indirectly available bandwidth. In our proposed algorithm, rooter gives p_{new} the candidates of parents depending on similar T_{r2p}, then p_{new} find the parent among these candidates with the shortest T_{p2p}. T_{r2p} estimation ensures the new peer knowing its proximity, and T_{p2p} estimation ensures the new peer finding the suitable parents with the shortest transmitting delay.

[6] T_{r2p} means time delay between rooter and a peer (from rooter to peer unidirectionally).

Algorithm 1: New Peer Joining at Rooter

while a *New Join Message* is received **do**
    // estimate round-trip time between rooter and new joining peer
    T_{r2p} = DelayEstimator.RTT(*rooter*, p_{new});
    query time = T_{r2p} + *range*;[7]
    // select candidates with shorter round-trip time than *query time*
    candidate list = JoiningHelper.getCandidates(*query time*);
    if *candidate list* exists **then**
        related info = JoiningHelper.query(p_{new});
        candidate list.set(*related info*);
        Join Accept Message.set(*candidate list*);
    else
        Join Accept Message.set(*rooter*);
    sends *Join Accept Message* to the new joining peer;
end while

Algorithm 2: New Peer Joining at Peer

while a *Join Accept Message* is received **do**
    // estimate round-trip time between new joining peer and candidates
    // and select the candidate with the shortest round-trip time
    min $T_{p2p} = \infty$;
    related info = null;
    candidate list = *Join Accept Message*.get();
    for all $p_i \in$ *candidate list*
        T_{p2p} = DelayEstimator.RTT(p_{new}, p_i);
        related info.add(JoiningHelper.query(p_i));
        if *min* $T_{p2p} > T_{p2p}$ **then**
            min $T_{p2p} = T_{p2p}$;
    end for
    // notice parent and inform *rooter*
    sends *Notice Message* to the candidate for parent;
end while

3.2.2 Data pushing

In pull scheme, a peer always requests to pull a video chunk, so the control overhead of pull scheme is heavier than that of push scheme. In addition, GridMedia calculated the formula of *pull delay* which was equivalent to $3\tau/2+3\delta$ while τ was the *request interval*[7] and δ was the average end-to-end delay, but *push delay* is $\delta + c$ while c was the minor constant [17]. Because of the low traffic overhead and the short waiting delay, we adopt the push scheme in this paper. Every chunk has a unique *sequence number* (SN) to be sequential and identified. Algorithm 3 describes the data pushing in every peer.

Algorithm 3: Data Pushing at Peer

```
while a chunk is received do
    // check the sequence number of this chunk
    SN = chunk.getSN();
    // if this chunk is ready, receive it and forward it
    if SN < Now or SN > Now + buffer size or buffer.get(SN) == true then
        drops chunk;
    else
        data = chunk.getData();
        // send chunk to its children
        for all p_i ∈ children
            sends chunk to p_i;
    end for
    puts data in buffer;
end while
```

Our proposed scheme permits a peer to have many parents. Receiver sends a request with attribute (D, R) to its parents, D means *divisor* and R means *remainder*. When the chunk SN % D is R, the chunk is forwarded to the receiver. For example, when receiving request with (2, 0), the sender forwards just even chunks to the receiver; when receiving request with (1, 0), the sender forwards all chunks to the receiver.

3.2.3 Peer leaving

The dynamics of peer departure disrupt content delivery and deprave the delivered quality to participating peers. P2P systems must suffer recovery overhead to rearrange the overlay after most peers leaving.

In our proposed scheme, when a user closes the application, the peer p_{leave} executes the Leaving Helper to finish the following steps:

(1) As Fig. 4 illustrated, p_{leave} (peer B) sends the *Leave Message* to the rooter to start peer leaving.[8] At the same time, p_{leave} sends the *Connect Help Message* to both its parent (peer A) and child. Later, p_{leave} sets a timer and waits the *Disconnect Message* from its child (peer C).

[7] Request interval is between two buffer map packets and two request packets.

[8] *Leave Message* is sent at first step to let rooter cancel the registration, this can avoid any candidate message or refine message from rooter.

(2) When peer A and C receive the *Connect Help Message*, peer A establishes a new connection to peer C via exchanging *Connect Message*.
(3) Both peer A and C check meta-information of *Connect Help Message* and *Connect Message* to confirm each other.
(4) When the new connection is established successfully, peer C should receive two upstreams with the same chunks (i.e. one from peer A, the other from peer B). Peer C sends *Disconnect Message* to disconnect the old connection from peer B.
(5) When peer B receives the *Disconnect Message*, it sends the *Disconnect Message* to peer A. Peer B disconnects the old connection to finish peer leaving.

In traditional multicast trees, rooter executes the central algorithm for recovery to help a peer finding new parents when its original parents leave. This method cannot react to peer leaving quickly in large scale or heavy churn, and the inefficient recovery leads to a bad QoE. In our proposed scheme, a peer p_{leave} executes the graceful departure when it wants to leave. As Algorithm 4 and 5 describe, the leaver p_{leave} and its child can use the Leaving Helper to recover the breaking path.

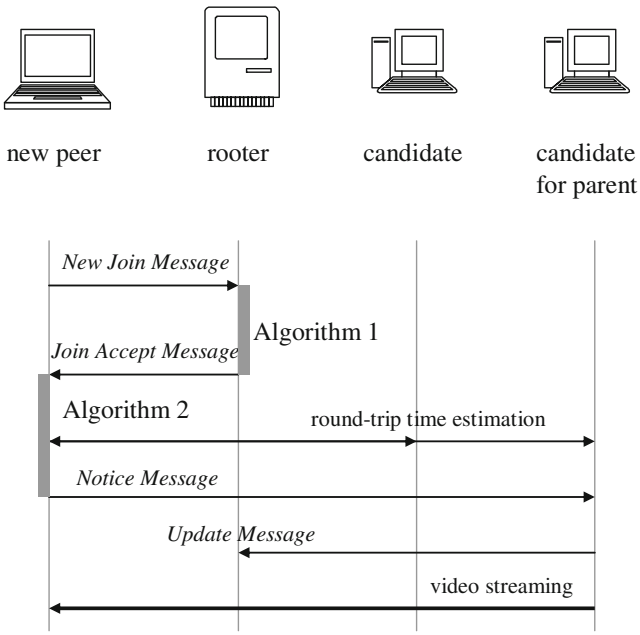

Fig. 3 Our proposed scheme for new peer joining

Algorithm 4: Peer Leaving at Leaver

```
sends Leave Message to rooter;
sends Connect Help Message to parent;
sends Connect Help Message to child;
timer = LeavingHelper.getTimer();
bool isOK = false;
// timer counts down
while timer > 0 do
        // wait for Disconnect Message
        if a Disconnect Message is received
                isOK = true;
                break;
        timer – –;
end while
if isOK
        // recovery is successful
else
        sends Disconnect Message to child;
sends Disconnect Message to parent;
end while
```

Algorithm 5: Peer Leaving at Resident

```
while a Connect Help Message is received do
        // judge whether father or son
        if Connect Help Message.getSon() equals this;
                sends Connect Message to Connect Help Message.getFather();
        else
                sends Connect Message to Connect Help Message.getSon();
end while
while a Connect Message is received do
        if Connect Message.getSon() equals this;
                sends Disconnect Message to parent;
        else
                LeavingHelper.connect(Connect Message.getSon());
                sends Update Message to inform rooter for a new overlay;
end while
```

If time is up before *Disconnect Message* comes, p_{leave} still disconnect all connections. Two reasons leads to this result: (1) New connection has connected but *Disconnect Message* is not back. (2) New connection is not connected. In former case, the new connection can work after the leaver departing, and the parent (peer A) also sends *Update Message* to rooter to inform a new overlay. In latter case, the child (peer C) becomes an orphan and reconnects via the peer adaptation or reselects parent through rooter.

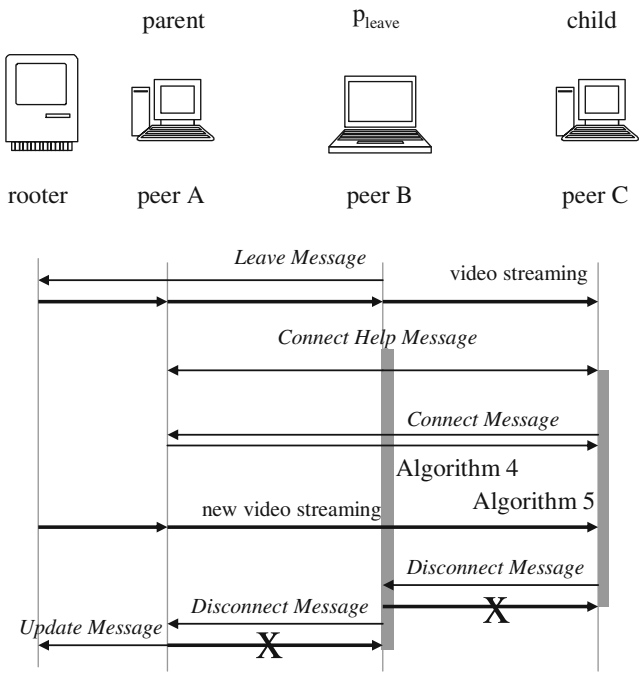

Fig. 4 Our proposed scheme for peer leaving

3.2.4 Peer adaptation

In general, the Internet service provider (ISP) gives users more download bandwidth than upload bandwidth [28]. Therefore, the bottleneck of P2P live streaming system is the outgoing bandwidth of major asymmetric connections. In our proposed scheme, each peer evaluates its accessible outgoing bandwidth to schedule data delivery, instead of best effort for download.

Each peer can execute *local peer adaptation* to replace an unsuitable parent. As Algorithm 6 and Fig. 5 illustrated, a child sends *Interchange Query Message* to its parent while its incoming bit rate is not enough. When receiving *Interchange Query Message*, a parent executes Bandwidth Estimator to compute its *contribution ratio*, which equals the amount of uploads divided by the amount of downloads.

$$\text{Contribution ratio} = \frac{\sum \text{uploads}}{\sum \text{downloads}}$$

If the contribution ratio is smaller than 1, the parent accepts this interchange and then sends *Interchange Accept Message* to execute the interchange of peers; else, the parent rejects this interchange and then sends *Interchange Reject Message*.

While receiving *Interchange Accept Message*, the child waits for the interchange for its parent; on the contrary, the child gives up its parent and finds other candidates to become a new parent while receiving *Interchange Reject Message*.

Algorithm 6: Local Peer Adaptation

```
while incoming bit rate becomes low or QoS is bad do
    sends Interchange Query Message to parent;
    // wait for the response
    while a message is received do
        if message is Interchange Accept Message then
            interchange(this, parent);
        else message is Interchange Reject Message then
            disconnection(this, parent);
    end while
end while
```

3.2.5 Overlay refinement

We design the overlay refinement algorithm to achieve the following goals:

(1) To refine an optimal overlay: According to the previous studies [16, 19, 21], we consider that the first factor is short latency, the second factor is high upload capacity, and the third factor is low probability of leaving in priority sequentially. The rooter sends the *Refine Message* to the peers on the recommendable path. The peers can judge whether accept the recommendation or not depending on their QoS.
(2) To keep the smooth delivery path: This work is similar with the above method. In addition, the overlay refinement uses Lifetime Looker to arrange the peers with long lifetime on several backbone paths. This method results in a stable backbone path supporting its branches to multicast smoothly.
(3) To balance each peer's load: This method is like as cluster splitting in NICE and ZigZag to balance load. However, the number of members triggers cluster splitting in NICE and ZigZag. In ABCD-P2P, the peers can judge whether accept the recommendation or not depending on their Delay Estimator and Bandwidth Estimator.

In summary, our proposed overlay refinement can bring three advantages: (1) The rooter only sends the recommendation to some peers, and each peer judge whether accepts the recommendation or not by itself. (2) Much load is moved from the new joining process to the local peer adaptation and the global overlay refinement in order to speed up the start-up delay. (3) The overlay refinement is not hierarchical, and the

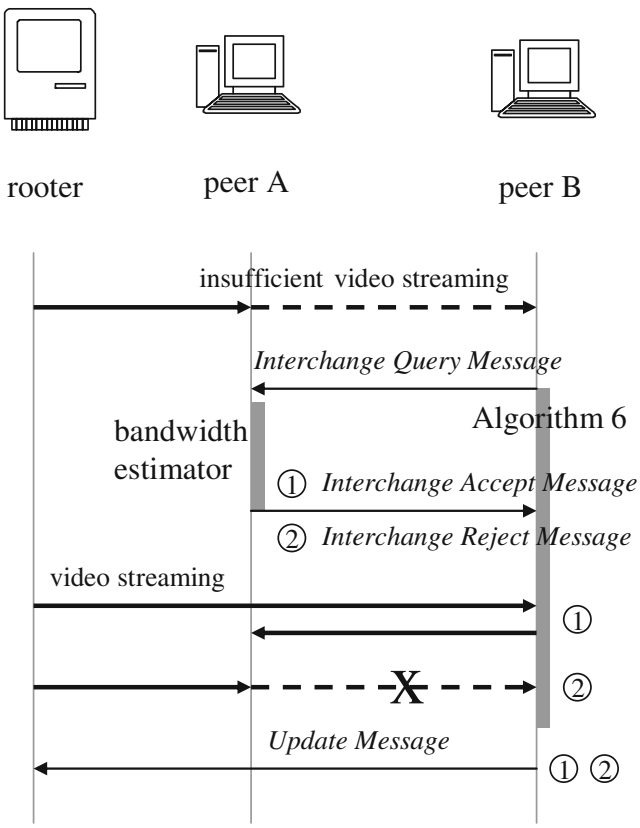

Fig. 5 Our proposed scheme for local peer adapting

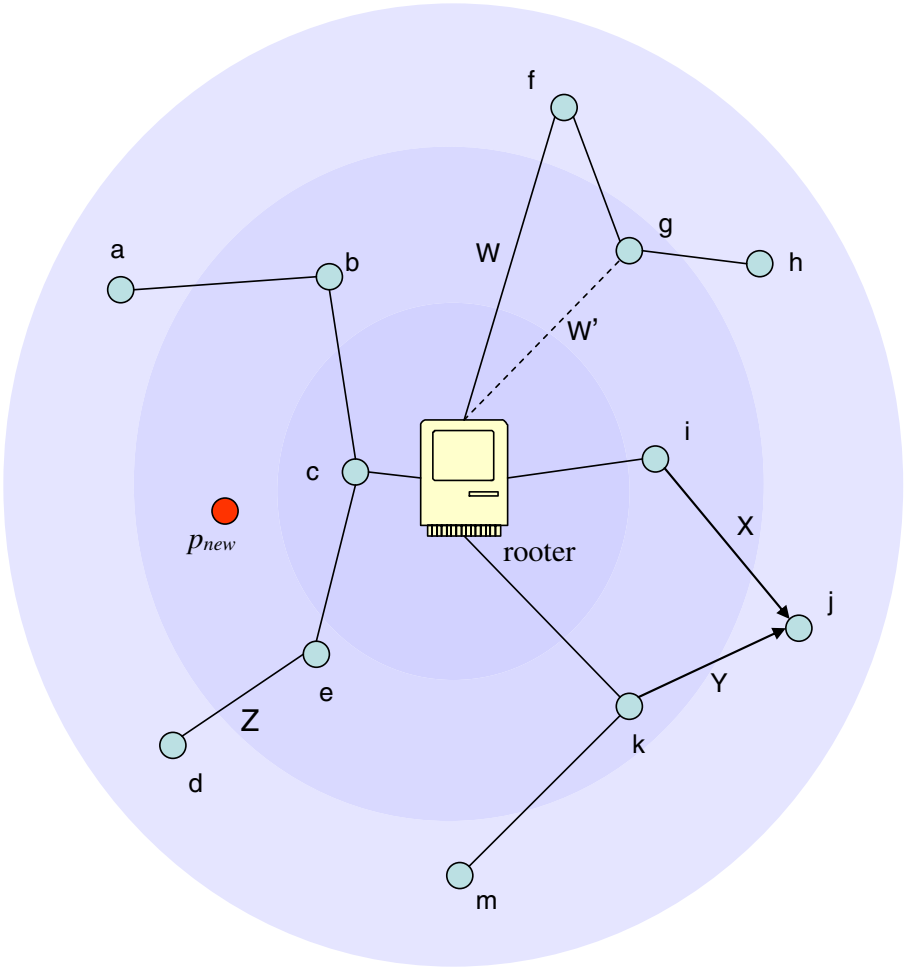

Fig. 6 Our proposed P2P network

load of local peer adaptation is more than the load of global overlay refinement to reduce rooter's overhead.

3.3 An example

For example, P2P network can be defined as a graph $G = (V, E)$ as Fig. 6 shown. V is the set of nodes $\{a, b, c \ldots\}$ and E is the set of edges $\{Z, Y, X \ldots\}$, and an edge can be also defined as a pair of nodes, e.g. edge $X = (d, f)$. When a new peer p_{new} joins the P2P overlay network, rooter sends *Join Accept Message* with a candidate list to p_{new}. The candidate list should include b, c, e, g, i, k because their T_{r2p} is shorter than T_{r2p}+range of p_{new}. Next, p_{new} estimates T_{p2p} for all candidates b, c, e, g, i, k. Then, p_{new} maybe select node e to become e's child and sends *Notice Message* to e. Finally, the new overlay topology is sent to rooter.

After three messages passing successfully, the bootstrap process is completed. The new peer always acts the leaf to speed up the bootstrap process.[9] For above example, p_{new} becomes e's child and becomes the leaf of tree as Fig. 7(a) shown, p_{new} gets video streaming from node e quickly. However, node e maybe has a poor upload capability to push streaming to two children difficultly. Thus, node e can disconnect edge Z and forward node d to p_{new} as Fig. 7(b) shown. This method for disconnecting and forwarding must be executed only when p_{new} is a leaf without any child initially.

On the other hand, p_{new} thinks of replacing e with another parent when quality of streaming bit rate is bad for several seconds. Then p_{new} can execute peer adaptation to select the suitable parent instead of peer e. Each peer checks the bit rate of its incoming stream, and the bit rate should be stable and keep fixed as similar as the data rate of video quality. If the bit rate of its incoming stream is much higher than the data rate of video quality, the data delivery includes the needless duplications, and partners check sub-stream map again to avoid the duplication; if the bit rate is lower, the upload bandwidth of its parents is insufficient to provide the full data, and the peer adaptation is executed.

For one example, p_{new} wants to execute local peer adaptation with peer e, and if peer e accepts this adaptation as

[9] Other bootstrap processes of multicast trees spend much time and much cost, for examples, In NICE, the leader is reselected when a new peer joins; In ZigZag, the cluster is reorganized when a new peer joins; In SplitStream, the new peer is arranged at an appropriate internode.

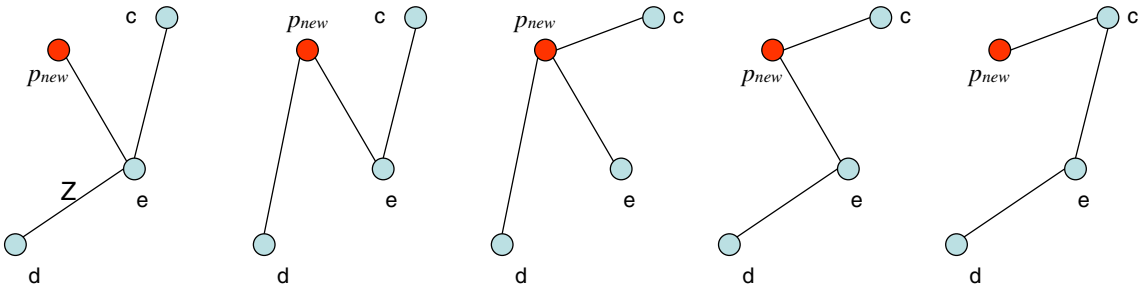

Fig. 7 The overlay of new peer joining

Fig. 7(c) shown. Originally, data path is $c \rightarrow e$, $e \rightarrow p_{new}$, $e \rightarrow d$. After peer p_{new} and peer e exchange their roles, data path becomes $c \rightarrow p_{new}$, $p_{new} \rightarrow e$, $p_{new} \rightarrow d$. In another case, after the local peer adaptation, the new overlay is shown as Fig. 7(d). After repeating local peer adaptation, a new overlay may become as Fig. 7(e). For this example, Fig. 7(d) and (e) both are stable overlays, and the local peer adaptation can coverage in the stable overlay via the demonstration afterward of our simulated results.

For another example, peer f, g, h joins in the P2P system in order. Through new joining steps, the path $f \rightarrow g \rightarrow h$ is established and the P2P overlay is shown in Fig. 6. Maybe, less overhead and satisfied QoS let peer f, g, h never execute local peer adaptation. Rooter periodically executes global overlay refinement to find that edge W' is batter than edge W, and then rooter sends *Refine Message* to peer g. If peer g agrees the recommendation, it executes local peer adaptation with peer f and h. Finally, any change is updated to rooter via *Update Message*.

4 Analysis

For the ease of exposition, we evaluate the performance of ABCD-P2P via the mathematical analyses.

4.1 Reach

Every peer must be *reachable* from rooter, or else content cannot be multicast.

Theorem 1 *Every peer must be reachable*

Proof

(1) A hypothetical assumption: P2P network can contain *finite* k peers, and there is one rooter and zero peers initially.

(2) When k=1, 1st peer must connect to rooter.

When k=2, 2nd peer may connect to rooter or 1st peer.

To reason by analogy, when k=3, 4, 5 …, k^{th} peer can be connectable.

(3) By induction on n. Inductive hypothesis is proved when $k = n$, it means n^{th} is connectable to rooter or other $n-1$ peers, and all n peers are reachable.

(4) When $k = n+1$, $(n+1)^{th}$ peer may be connectable to rooter, or connects to other n peers. The hypothesis that n peers are reachable is given by (3), so $(n+1)^{th}$ peer arbitrarily connects some one at random to be reachable. The $(n+1)^{th}$ peer is reachable that is true.

(5) Therefore, by the principle of mathematical induction, n peers in ABCD-P2P can be reachable for all positive integers n. Every peer must be reachable.

Theorem 2 *Every peer must be reachable when other peer depart*

Proof

(1) A hypothetical assumption: there are x peers leaving from P2P network simultaneously. Without loss of generality, x peers are i_x^{th} ancestors of residents.

(2) The Algorithm 4 defines i_x^{th}, and Algorithm 5 lets $i_x^{th}-1$ peer reachable to $i_x^{th}+1$ peer.

(3) Let n is the range of i. When n=1, i.e. a left departs, all residents are reachable.

Table 2 Summary of mathematical analyses about the comparison of ABCD-P2P, ZigZag, and NICE

		ABCD-P2P	ZigZag	NICE
-	Average query time	O(log n)	O(log n)	O(log n)
×	Worst query time	O(n)	O(log n)	O(log n)
○	Average join latency	O(c)	O(k log n)	O(log n)
○	Average join overhead	O(c)	O(k^2 log n)	O(k log n)
-	Average node degree	O(k)	O(k)	O(k)
△	Worst node degree	O(n)	O(k^2)	O(log n)
-	Average recovery overhead	O(k)	O(k)	O(k)
△	Worst recovery overhead	O(c+k)	O(k^2)	O(log n)
-	Worst control overhead	O(log n)	O(k log n)	O(log n)
×	Worst refinement overhead	O(log n+k^2)	O(k^2)	O(k^2)

n is the number of peers; k is the number of neighbors per peer; c is a constant

○ Better, − Same, △ Unknown, × Worse

Table 3 Peer upload bandwidth distribution in Substream Trading (kbps)

Total upload bandwidth (kbps)	256	320	384	448	512	640	768	1024	1500	> 3000
Distribution (%)	10.0	14.3	8.6	12.5	2.2	1.1	6.6	28.1	1.4	14.9
Contributed upload bandwidth	150	250	300	350	400	500	600	800	1000	1000

(4) When $n < x < k$, all $k - x$ peers are reachable by induction on n according to Theorem 1.
(5) When i are continuous, $i_x^{\text{th}} - 1$ peer are reachable to $i_x^{\text{th}} + n$ peer. Therefore, by the principle of mathematical induction, every peer in ABCD-P2P can be reachable for all positive integers n. Every peer must be reachable.

Theorem 3 *The average time of every query for any peer from rooter is* O(log *n*), *while n is the network size*

Proof Without loss of generality, a peer has m children in j^{th} layer among all n peers, and query time should equals approximately to O(j). For the leaf, average query time increases with the depth of multicast tree $O(\log_m n)$. However, if the structure of overlay is similar with a chain, the worst query time is O(n).

4.2 Waiting time of new joining

As Fig. 3 illustrates, the waiting time of new joining process is spent at Algorithm 1 and Algorithm 2. The major task of Algorithm 1 is to select the candidates, and the major task of Algorithm 2 is to decide the parents among the candidates.

Theorem 4 *The average waiting time of new joining peer is* O(c), *while c is a constant*[10]

Proof
(1) The Algorithm 1 spends O(c) time selecting the candidates according to the proximity via Joining Helper.
(2) The Algorithm 2 spends O(c) time deciding the parents according to the proximity via Delay Estimator. The RTT of message is O(1) due to direct proximity.
(3) The waiting time = The execution time of Algorithm 1 and Algorithm 2 + the time of visiting candidates = O(c) + O(c) + O(1) = O(c)

The overhead means how many messages are produced in the interval. The overhead of new joining process influences the traffic load.

[10] c can be seen as the range of selecting candidates. c affects the number of candidates and the proximity of new joining peer.

Theorem 5 *The average overhead of new joining peer is* O(c)

Proof
(1) The Algorithm 1 sends O(1) message to a new joining peer.
(2) The Algorithm 2 sends O(c) message informing the candidates.
(3) The overhead = The overhead of Algorithm 1 and Algorithm 2 + the overhead of visiting candidates = O(1) + O(c) + O(c) = O(c)

4.3 Node degree

The node degree can indicate the balance, load, and complexity of a peer.

Theorem 6 *The average node degree is* O(k), *k is the number of neighbors*

Proof Every peer via the Bandwidth Estimator can control the number of incoming streams and outgoing streams, as a result, the average node degree is O(k).

Theorem 7 *The worst node degree is* O(n)

Proof The worst case is that all peers are leaves or form a *full mesh*, as a result, the worst node degree is O(n).

4.4 Recovery overhead of peer leaving

When the parents leave, a child becomes an orphan and gains no data suddenly. It needs a recovery mechanism to connect the new parents. We define the recovery overhead as an overhead produced in the interval of changing parents. Because of the leave legacy of Leaving Helper, the recovery overhead includes *Connect Help Message*s, *Connect Message*s, and *Disconnect Message*s in Algorithm 4 and Algorithm 5.

Table 4 Peer upload bandwidth distribution in our simulation

Set	Upload bandwidth (kbps)	Download bandwidth (kbps)	Distribution (%)
A	3000	3000	1
B	900	2100	10
C	300	1500	75
D	50	600	14

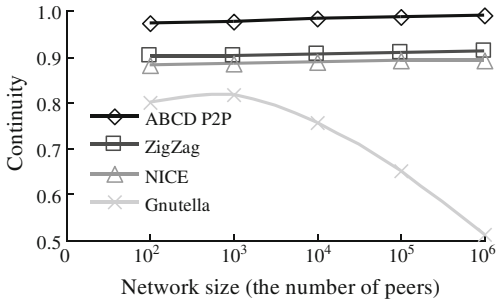

Fig. 8 Continuity for scalability

Theorem 8 *The average recovery overhead is* O(k)

Proof

The overhead
= Sending *Connect Help Message*s
+ exchanging *Connect Message*s
+ sending *Disconnect Message*s
= O(k) + 2O(k) + O(k)
= O(k)

Theorem 9 *The worst recovery overhead is* O(c + k)

Proof

The overhead
= The overhead as Theorem 8
+ the rejoin overhead as Theorem 5
= O(k) + O(c)

4.5 Control overhead

During data delivery, the messages between rooter and peers are transmitted to maintain the P2P overlay. The overhead of these messages is called as control overhead.

Theorem 10 *The worst control overhead is* O(log n)

Proof The height of tree structure of ABCD-P2P is O(log n), the worst control overhead is to control a path per trigger. Hence, the worst control overhead is equivalent to visiting all peers on the same path, i.e. O(log n).

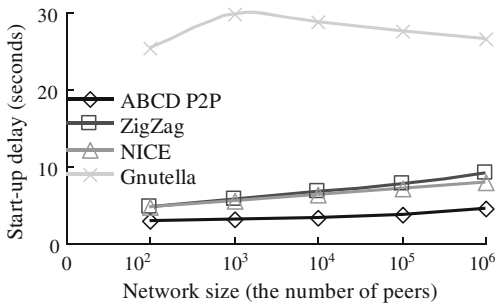

Fig. 9 Start-up delay for scalability

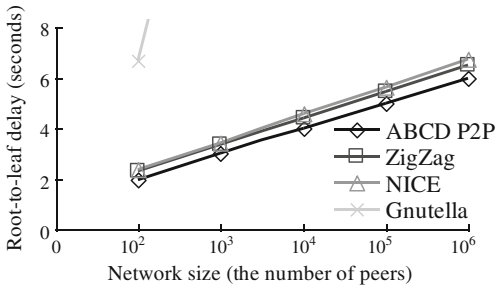

Fig. 10 Root-to-leaf delay for scalability

4.6 Refinement overhead

The algorithm of refinement influences the performance of all P2P services. In addition, the complexity and overhead of refinement algorithm influence the efficiency of refinement. The complexity of our proposed refinement scheme is difficult to analyze via mathematical model. The ideal overhead of refinement equals the overhead of global overlay refinement adds the overhead of local peer adaptation. Hence, the ideal overhead = O(log n) + O(k).

Theorem 11 *The worst refinement overhead is* O(log $n + k^2$)

Proof

(1) Rooter executes global overlay refinement and sends *Refine Message*, and this method leads to a worst overhead O(log n).
(2) Peer executes local peer adaptation and sends *Interchange Query Message*, and this method leads to a worst overhead O(k^2) due to k partners for k queries.
(3) The worst refinement overhead
= Worst global overlay refinement overhead
+worst local peer adaptation overhead
$= O(\log n) + O(k^2)$
$= O(\log n + k^2)$

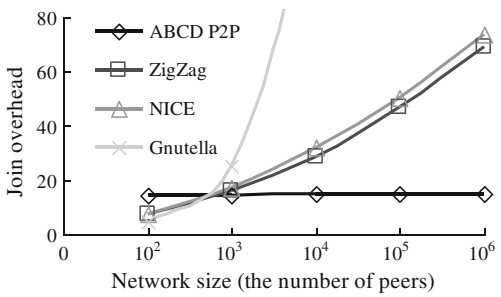

Fig. 11 Join overhead for scalability

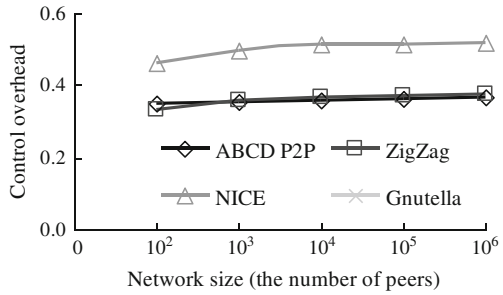

Fig. 12 Control overhead for scalability

4.7 Summary

Through the above mathematical analyses, we can demonstrate that ABCD-P2P is a scalable system because the delay and overhead increase with the linearity or logarithm of the number of peers. We compare ABCD-P2P with ZigZag, and NICE as shown in Table 2. Usually $k << n$, we can induce the comparison: ○ represents that ABCD-P2P is better than Zigzag and NICE; − represents it is same; ▵ represents it is unable to compare; × represents ABCD-P2P is worse than Zigzag and NICE. We can discover the outstanding advantage of ABCD-P2P is short start-up delay (join latency) and little join overhead.

5 Simulation results

5.1 Simulation environment

We use OMNet++ 4.0 and INET module [30] to simulate the P2P live streaming service on tree-based NICE, ZigZag, and mesh-based Gnutella.[11] We refer to the precondition of simulation situation in Substream Trading [31], and the upload bandwidth distribution is listed in Table 3. We jointly consider the data from TWNIC [32] and Table 3 to set the upload bandwidth distribution of our simulation as shown in Table 4.

In Section 5.2, we discuss the scalability characteristic of ABCD-P2P. In Section 5.3, we discuss the overlay interruption and recovery after peer departure.

5.2 Scalability

We use the different numbers of peers to discuss the scalability problem. The number of peers increases with the growing exponent: 10^2, 10^3, 10^4, 10^5, and 10^6 in our simulation experiments.

[11] 95 % of peers in the Gnutella system could be reached within 7 hops by pure flooding. We set 3 hops as the time-to-life of Gnutella message to avoid infinite flooding.

5.2.1 Continuity

The continuity ratio[12] is defined as the number of video chunks that arrive before playback deadlines over the total number of video chunks. The continuity ratio can reflect the playback smoothness and QoE. The higher continuity ratio represents the smoother playback. As Fig. 8 illustrated, we can discover that ABCD-P2P gets the best performance among the compared algorithms, and the continuity ratio increases with the number of peers increasing. More peers in P2P network can provide more upload bandwidth to cooperate each other, this leads to better performance. However, Gnutella has the worst performance because it adopts the flooding algorithm to query and pull data, this situation results in the bad continuity in large scale.

5.2.2 Start-up delay

All users expect to watch video frames as fast as possible when launching the application. Hence, a start-up delay can test users' tolerance and directly reflect the QoE. As Fig. 9 illustrated, ABCD-P2P has the shortest start-up delay among the compared algorithms. Although the start-up delay increases with the number of peers increasing, the start-up delay can be tolerated when 10^6 peers are in the network due to a logarithmic rise. We can demonstrate that the Joining Helper can speed the joining process up.

In essence, mesh-based algorithms occur longer start-up delay than tree-based algorithms do this, because: (1) a new joining peer locates itself via a long-time and complex process of exchanging messages in mesh-based algorithm, but rooter locates a new joining peer in tree-based algorithm; (2) a new joining peer uses the bigger buffer to execute mesh-pull but uses the smaller buffer to execute tree-push in two different algorithms. This reason explains that mesh-based Gnutella can speed up the start-up delay when the number of peers is large, because a new joining peer can get more data into buffer from more peers.

5.2.3 Root-to-leaf delay

As Fig. 10 illustrated, ABCD-P2P has the shortest root-to-leaf delay[13] among the compared algorithms. Through Delay Estimator and Joining Helper, the P2P overlay is proximate to the physical topology. The proximity shortens the routing path and root-to-leaf delay.

[12] Continuity ratio in this paper is equivalent to *continuity index* in CoolStreaming [13] and *received chunk ratio* in Substream Trading [31].

[13] Root-to-leaf delay is the same with *source-to-end delay* defined in AnySee [15] and *delivery delay* defined in OPSS [33].

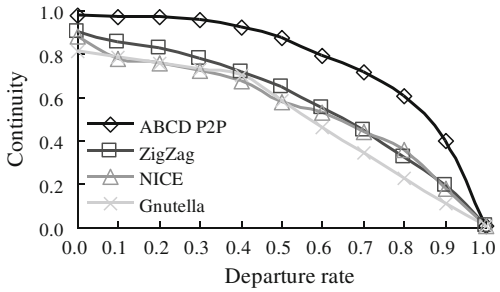

Fig. 13 Continuity for departure

5.2.4 Join overhead

We use the total number of messages produced by a new peer joining to evaluate the join overhead. As Fig. 11 illustrated, ABCD-P2P produces almost the same number of messages which is not various with the number of peers, because the number of messages is irrelative with the number of peers in the new joining process of our proposed scheme. This is an outstanding advantage in the large scale. In both NICE and ZigZag, rooter locates the new joining peer through level by level, thus their join overhead increases with the number of peers increasing. With regard to Gnutella, the flooding scheme produces the excessive join overhead in a large scale. In essence, there is much joining overhead in mesh-based algorithms because a new peer needs many exchanging messages to locate itself into P2P overlay and then organize the local overlay.

5.2.5 Control overhead

We define that the control overhead equals the average of the number of control messages which is handled by each peer per second. As Fig. 12 illustrated, the control overhead of tree-based algorithms is very small, every peer only loads one minus message per second. However, peers in Gnutella need much control overhead to maintain overlay and pull data. Gnutella produces several messages to hundreds messages with the number of peers increasing.

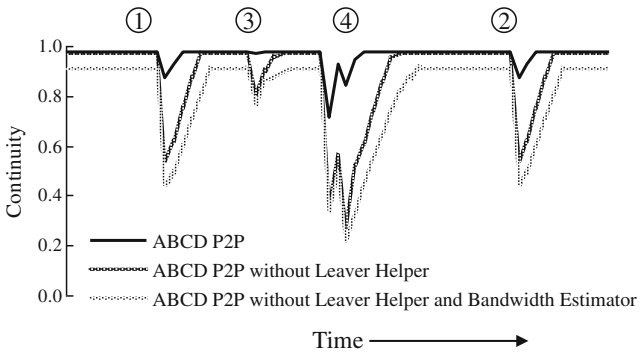

Fig. 14 Our proposed scheme for departure

5.3 Departure

In P2P network, a peer leaves possibly in any time. The peer departure always breaks off the P2P overlay, this departure leads to the interruption of the original stream. We define that the *departure rate* is the probability of peer leaving. The number of peers is 10^4.

5.3.1 Continuity failure

A departure of upstream peer must influence downstream peer, and then downstream peer cannot get data suddenly. The lack of upstream data leads to the continuity ratio declining as Fig. 13 illustrated. ABCD-P2P has the highest continuity ratio among the compared algorithms, and the continuity ratio keeps 95 % plus when departure rate is smaller than 0.3. This statistics expresses ABCD-P2P can handle the small churn because our special algorithm of peer leaving (Section 3.2.3). In ABCD-P2P, a leaving peer prepares a backup path for its downstream to recover the breaking stream before departing (Fig. 2). When departure rate is 0.8 that means the P2P network is very dynamic, the continuity ratio is still 60 % approximately in ABCD-P2P Scheme.

5.3.2 Side effect of departure

In this experiment, we try to discover why ABCD-P2P performs well in dynamic P2P network. We compare ABCD-P2P with whole components to ABCD-P2P with part components. Three types are compared: ABCD-P2P, ABCD-P2P without Leaver Helper, and ABCD-P2P without Leaver Helper and Bandwidth Estimator as Fig. 14 shown. ABCD-P2P has the best performance in this experiment: (1) the performance of ABCD-P2P declines lightly when churn happens. (2) ABCD-P2P fast recovers the broken stream. (3) The small churn leads to no continuity failure in ABCD-P2P. (4) ABCD-P2P can keep a good performance when facing big and continuous churn.

6 Conclusion

This paper discusses the existing problems, challenges and limitations of P2P live media streaming in a large cooperated network. We focus on only one single source to look for the resolutions suitable for P2P live multicast. Because of the consideration of short start-up delay and short root-to-leaf delay, we design the advanced bootstrap mechanism and adjusted bandwidth mechanism. We propose a novel P2P scheme, ABCD-P2P, adopts the push scheme to deliver chunks fast, efficiently, and stably. The keys of ABCD-P2P are five special components which are designed for the characteristics of P2P users' experience in real world: Delay

Estimator, Bandwidth Estimator, Joining Helper, Leaving Helper, and Lifetime Looker. Our proposed algorithms are developed to meet the motivations and can be demonstrated to achieve the goals. Via both theoretical analysis and simulation results, ABCD-P2P can contribute the following desirable performances, such as, high continuity, short start-up delay, short end-to-end delay, low control overhead, and short recovery time.

References

1. Kerpez K, Waring D, Lapiotis G, Lyles JB, Vaidyanathan R (2006) IPTV service assurance. IEEE Commun Mag 44(9):166–172
2. Shihab E, Cai L, Wan F, Gulliver A, Tin N (2008) Wireless mesh networks for in-home IPTV distribution. IEEE Netw 22(1):52–57
3. Tran DA, Hua KA, Do TT (2004) A peer-to-peer architecture for media streaming. IEEE J Sel Areas Commun 22(1):121–133
4. Hei X, Liu Y, Ross KW (2008) IPTV over P2P streaming networks: The mesh-pull approach. IEEE Commun Mag 46(2):86–92
5. Johnsen JA, Karlsen LE, Birkeland SS (2005) Peer-to-peer networking with BitTorrent
6. Nafaa A, Murphy S, Murphy L (2008) Analysis of a large-scale VOD architecture for broadband operators: a P2P-based solution. IEEE Commun Mag 46(12):47–55
7. Cheng B, Stein L, Jin H, Liao X, Zhang Z (2008) GridCast: Improving peer sharing for P2P VoD. ACM Trans Multimed Comput Commun Appl 4(4):26
8. Hei X, Liu Y, Ross KW (2007) Inferring network-wide quality in P2P live streaming systems. IEEE J Sel Areas Commun 25(9):1640–1654
9. Hei X, Liang C, Liang J, Liu Y, Ross KW (2007) A measurement study of a large-scale P2P IPTV system. IEEE Trans Multimed 9(8):1672–1687
10. Chu Y, Rao SG, Seshan S (2002) A case for end system multicast. IEEE J Sel Areas Commun 20(8):1456–1471
11. Zhang X, Liu J, Li B (2005) On large-scale peer-to-peer live video distribution: Coolstreaming and its preliminary experimental Results. IEEE International Workshop on Multimedia Signal Processing
12. Tran DA, Hua KA, Do TT (2003) ZIGZAG: An efficient peer-to-peer scheme for media streaming. IEEE Conference on Computer Communications
13. Xie S, Li B, Keung GY, Zhang X (2007) Coolstreaming: Design, theory, and practice. IEEE Trans Multimed 9(8):1661–1671
14. Bo L, Susu X, Keung GY, Jiangchuan L, Stoica I, Hui Z, Zhang X (2007) An empirical study of the coolstreaming plus system. IEEE J Sel Areas Commun 25(9):1627–1639
15. Liao X, Jin H, Liu Y, Ni LM, Deng D (2006) AnySee: Peer-to-peer live streaming. IEEE International Conference on Computer Communications
16. Huang Q, Jin H, Liao X (2007) P2P live streaming with tree-mesh based hybrid overlay. International Conference on Parallel Processing Workshops
17. Zhang M, Zhao L, Tang Y, Luo JG, Yang SQ (2005) Large-scale live media streaming over peer-to-peer networks through global internet. ACM workshop on Advances in peer-to-peer multimedia streaming
18. Magharei N, Rejaie R (2009) PRIME: Peer-to-peer receiver-drIven MEsh-based streaming. IEEE/ACM Trans Networking 17(4):1052–1065
19. Banerjee S, Bhattacharjee B, Kommareddy C (2002) Scalable application layer multicast. ACM Conf Appl Tech Archit Protoc Comput Commun 32(4):205–217
20. Tran DA, Hua KA, Do TT (2004) A peer-to-peer architecture for media streaming. IEEE J Sel Areas Commun 22(1):121–133
21. Yu SS, Zheng XW, Zhou JL (2006) A P2P scheme for live media stream multicast. International Multi-Media Modeling Conference Proceedings
22. Tan X, Datta S (2005) Building multicast trees for multimedia streaming in heterogeneous P2P networks. International Conference on Systems Communications
23. Jannotti J, Gifford D, Johnson KL, Kaashoek MF (2000) Overcast: Reliable multicasting with an overlay network. USENIX Association Proceedings of Symposium on Operating Systems Design and Implementation
24. Deshpande H, Bawa M, Garcia-Molina H (2002) Streaming live media over a peer-to-peer network. ACM/IEEE Workshop on Network and OS Support for Digital Audio and Video
25. Castro M, Druschel P, Kermarrec AM, Nandi A, Rowstron A, Singh A (2003) SplitStream: High-bandwidth content distribution in cooperative environments. ACM Symposium on Operating Systems Principles
26. Lee J, Hoang X, Lee Y (2006) BACS: Split channel based overlay multicast for multimedia streaming. Information Networking: Advances in Data Communications and Wireless Networks
27. Magharei N, Rejaie R, Guo Y (2007) Mesh or multiple-tree: A comparative study of live P2P streaming approaches. IEEE International Conference on Computer Communications
28. Tang Y, Luo J, Zhang Q, Zhang M (2007) Deploying P2P networks for large-scale live video-streaming service. IEEE Commun Mag 45(6):100–106
29. Magharei N, Rejaie R (2005) Peer-to-peer receiver-driven mesh-based streaming. ACM SIGCOMM
30. OMNet++: http://www.omnetpp.org/
31. Liu Z, Shen Y, Ross KW, Panwar SS, Wang Y (2008) Substream Trading: Towards an Open P2P Live Streaming System. IEEE International Conference on Network Protocols
32. Taiwan Network Information Center (TWNIC): http://map.twnic.net.tw/
33. Bracciale L, Lo Piccolo F, Luzzi D, Salsano S, Bianchi G, Blefari-Melazzi N (2008) A push-based scheduling algorithm for large scale P2P live streaming. International Telecommunication Networking Workshop on QoS in Multiservice IP Networks

Jun-Li Kuo
(estar.cs95g@nctu.edu.tw) works in Communication & Network Solution, Foxconn, Hsinchu, Taiwan. He received his Ph.D. in Computer Science from National Chiao Tung University, Hsinchu, Taiwan in 2013. His research interests include peer-to-peer network, wireless multimedia, ad-hoc network, and clouding computing.

Chen-Hua Shih
(shihch@csie.nctu.edu.tw) works in Chung Shan Institute of Science and Technology, Taoyuan, Taiwan. He received his Ph.D. in Computer Science from National Chiao Tung University, Hsinchu, Taiwan in 2012. His research interests include peer-to-peer network, QoS, mobile IP, wireless networks, and network protocols.

Cheng-Yuan Ho
(cyho@csie.nctu.edu.tw) is a R&D manager in the Advanced Research Institute, Institute for Information Industry, Taipei, Taiwan. His research interests include the design, analysis, and modeling of the congestion control algorithms, mobile and wireless networks, high speed networking, P2P networks, real-flow test, and quality of service. Ho received his Ph.D. in Computer Science from National Chiao Tung University, Hsinchu, Taiwan in 2008.

Yaw-Chung Chen
(ycchen@cs.nctu.edu.tw) received his Ph.D. degree from Northwestern University, Evanston, Illinois, USA. In 1986 he joined AT & T Bell Laboratories, where he worked on various exploratory projects. He is currently a professor in the Department of Computer Science, National Chiao Tung University, Hsinchu, Taiwan. His research interests include Internet Protocols, fast mobility management, P2P systems and green computing.