Peer-to-Peer Netw. Appl. manuscript No.
(will be inserted by the editor)

Peer-to-Peer Live Video Streaming with Rateless Codes for Massively Multiplayer Online Games

Shakeel Ahmad · Christos Bouras · Eliya Buyukkaya · Muneeb Dawood · Raouf Hamzaoui · Vaggelis Kapoulas · Andreas Papazois · Gwendal Simon

Received: date / Accepted: date

Abstract We present a multi-level multi-overlay hybrid peer-to-peer live video system that enables players of Massively Multiplayer Online Games to simultaneously stream the video of their game and watch the game videos of other players. Each live video bitstream is encoded with rateless codes and multiple trees are used to transmit the encoded symbols. Trees are constructed dynamically with the aim to minimize the transmission rate at the source while maximizing the number of served peers and guaranteeing on-time delivery and reliability. ns-2 simulations and real measurements on the Internet show competitive performance in terms of start-up delay, playback lag, rejection rate, used bandwidth, continuity index, and video quality.

Keywords Peer-to-peer networking; Live video streaming; Rateless codes; Online games.

S. Ahmad
Southampton Solent University, United Kingdom;
E-mail: shakeel.ahmad@solent.ac.uk

C. Bouras, V. Kapoulas, and A. Papazois
Computer Technology Institute & Press “Diophantus”, Greece;
E-mail: {bouras, kapoulas}@cti.gr, papazois@ceid.upatras.gr

E. Buyukkaya
Kadir Has University, Turkey;
E-mail: eliya.buyukkaya@khas.edu.tr

M. Dawood
Teesside University, United Kingdom
E-mail: m.dawood@tees.ac.uk

R. Hamzaoui
De Montfort University, United Kingdom;
E-mail: rhamzaoui@dmu.ac.uk

G. Simon
Telecom Bretagne, France;
E-mail: Gwendal.Simon@telecom-bretagne.eu

1 Introduction

Massively Multiplayer Online Games (MMOGs) allow a large number of online users to inhabit the same virtual world and interact with each other in a variety of collaborative and competing scenarios. Gamers within an MMOG typically become members of online communities with shared adventures and common objectives. Players can play against other players or build groups to compete against other groups or against computer-controlled enemies. A survey [1] conducted as part of the FP7-funded CNG project [2] has highlighted that live streaming of screen-captured video of the game is one of the most desirable community tools for MMOG gamers. Players can use it to showcase their skills, share experience with friends, or coordinate missions in strategy games.

To stream user-generated live video, existing commercial platforms (e.g., TwitchTV [3], Ustream TV [4], and Livestream [5]) rely on a centralized architecture. However, even when the streaming system is supported by a Content Delivery Network (CDN), this solution is not cost-effective. Indeed, the popularity distribution of user-generated video poses a major challenge to large-scale streaming systems. On one hand, a large proportion of users are likely to act as sources, so there are many live streams to deal with. On the other hand, each stream is typically watched by a small population consisting of a few friends. Moreover, although the unit price of delivering data over the Internet has significantly decreased, the amount of bandwidth consumed by a single user has grown at a faster rate because of the higher requirements in terms of bitrate, frame rate, and resolution [6].

To reduce the bandwidth and maintenance costs for the service provider, a peer-to-peer (P2P) system can be used instead. While many P2P live video systems have been pro-

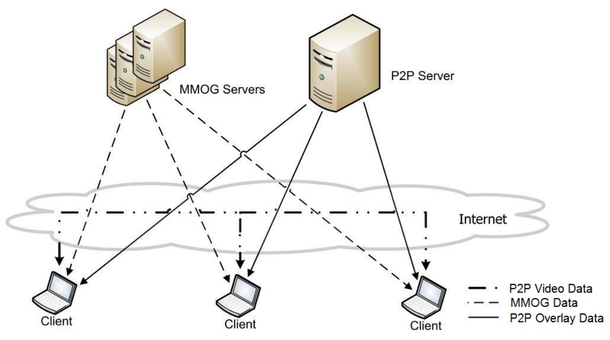

Fig. 1 Overview of the CNG system.

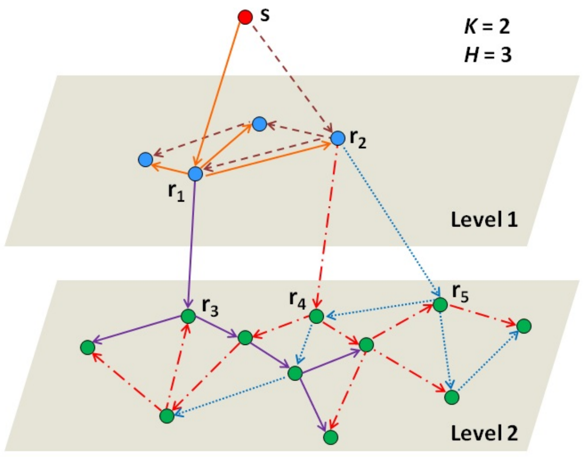

Fig. 2 Video diffusion within and across levels via multicast trees. Five trees are used: two in level 1 (higher level) rooted at r_1 and r_2 and three in level 2 (lower level) rooted at r_3, r_4, and r_5. Here, two packets should be received in order to decode the video ($K = 2$). A packet cannot be forwarded more than three times in a tree ($H = 3$).

posed, none of them was designed to simultaneously fulfill the following requirements:

- The video source can be a casual gamer with limited upload bandwidth. It is therefore critical to minimize the transmission rate at the source.
- The start-up delay and playback lag should be as small as possible to enable interaction between gamers.
- The system should be resilient against packet loss and peer churn.
- Live video streaming should not consume the upload and download bandwidth required for the smooth operation of the MMOG (MMOG client-server game traffic).
- Peers should be arranged in levels so that video is delivered at the same time to all peers in the same level. Moreover, peers in a higher level should be able to watch the video before those in a lower level. A level can be a priority class in a tiered or freemium service (two popular business models where users are charged according to the quality of service received).
- Users should be able to watch several videos simultaneously for, for example, intra-group coordination.

In the CNG project, we developed a multi-level multi-overlay hybrid P2P system that addresses the above requirements. Fig. 1 shows a high-level overview of the architecture. The main components are the clients (peers), a P2P server, and MMOG servers. The P2P server has persistent communication with the peers and is responsible for building and updating the P2P overlays (one overlay for each video source). The MMOG servers are independent of the P2P server and deliver the game content according to a standard client-server model.

Fig. 2 shows peers in one P2P overlay. Height-bounded *multicast trees* in multiple levels are used for fast delivery of video data. Rateless codes [7, 8] are used to provide resilience against packet loss and peer churn. Rateless codes are ideally suited as they (1) have very low computational cost, (2) minimize delivery redundancy when a peer receives data concurrently from multiple peers, (3) can easily be adapted to varying network conditions since one can generate on the fly as many encoded symbols as needed.

Our main contribution are novel algorithms to build and manage the multicast trees in a dynamic environment where peers may join and leave. Our algorithms allow us to efficiently schedule the rateless encoded packets within and across levels. We aim at minimizing the transmission rate for the source while maximizing the number of served peers and guaranteeing on-time delivery and reliability at the peers. We run extensive ns-2 simulations to test our system with respect to scalability, bandwidth heterogeneity, packet loss, and peer churn. Unlike previous work, we provide results for a wide range of metrics, including start-up delay, playback lag, continuity index, peak signal to noise ratio (PSNR), used bandwidth, rejection rate, and impact on the gaming experience. Moreover, we provide results from real measurements over the Internet.

The remainder of the paper is as follows. In §2, we discuss related work. In §3, we describe our video transmission strategy. In §4, we present our overlay construction algorithms. In §5, we report our simulation results. In §6, we present results from real measurements on the Internet. Finally, we give our conclusions and discuss challenges for future work in §7.

2 Related work

In this section, we review previous work that used rateless coding for P2P video streaming.

The first P2P system based on rateless codes was proposed by Wu and Li [9]. As soon as a receiving peer successfully decodes a source block, it becomes a source and applies rateless coding on the decoded source block to generate encoded symbols for other peers. However, this approach delays video delivery because packets are not forwarded to other peers before the decoding is complete.

Grangetto, Gaeta, and Sereno [10] improved this seminal work by introducing a method called *Relay and Encode*, where a receiving peer immediately forwards the received encoded symbols to other peers. The authors show that Relay and Encode results in a smaller playback lag. In [11], peers that successfully decode a source block inform the source so that it stops sending them symbols. However, these works focus on the transmission aspects of the P2P system and do not address important issues such as overlay construction, peer churn, packet loss, and bandwidth fluctuation.

The idea of Relay and Encode was later implemented in a real P2P live streaming application (ToroVerde Streaming (TVS) [12]) and the system was tested on PlanetLab. However, TVS assumes that the source is a powerful video server with huge upload bandwidth.

Oh, Wu, and Song [13] use rateless codes in a P2P video on demand streaming system. The goal is to provide a stable service of high quality with low computational complexity and a short start-up delay. However, the system was designed for a video on demand application and may not be easily adapted to live video streaming. Moreover, the paper does not deal with overlay management, scalability, and peer discovery, and the results are given for a small emulated network of four peers.

Eittenberger [14] discusses the feasibility of using rateless codes to increase the upload throughput of mobile devices for P2P applications in cellular networks. In [15], trade-offs between different rateless code parameters in P2P streaming applications are studied. However, both works ([14, 15]) do not propose a complete P2P live video system.

A demo of the proposed system was presented at the 2012 IEEE International Conference on Peer-to-Peer Computing [16]. A simplified version of the system that targets static scenarios in which all peers join before the start of the video session and remain subscribed until the end of the session was described in a conference paper [17]. Simulations showed that the system can sustain high performance as long as the number of peers that leave the system is moderate. In contrast to this paper, our new work does not impose any restriction on peer activity and proposes algorithms that build the overlay dynamically as peers join and leave. This fundamental change in overlay design and management offers a substantively novel contribution and leads to a completely different set of experimental results.

3 Video transmission

In this section, we introduce the different actors in the P2P system and describe their interactions. Details about overlay construction and management are left to the next section.

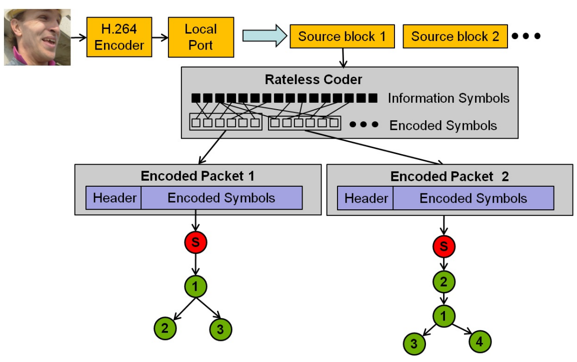

Fig. 3 Video coding and streaming. Node s denotes the source. Packet 1 is sent over a multicast tree to peers 1, 2, and 3. Packet 2 is sent to peers 2, 1, 3, and 4.

3.1 Initialization

A peer wishing to broadcast its video asks the P2P server to advertise it. This peer is called a *source*. If another peer is interested in the advertised video, it sends a request to the P2P server. Like a *tracker* (a server of *peerlists* in traditional P2P systems), the P2P server is in charge of updating the overlay information and informing the participating peers. The overlay information consists of peer assignments to levels and sets of multicast trees for each level (see §4). The overall overlay is denoted by $G(V, E)$ where V is the set of peers and E is the set of links. A link between u and v is denoted by (u, v). The connection between the P2P server and the peers is checked through regular keep alive messages. Failure of the connection triggers removal of the peer.

3.2 Video and Channel Coding

As soon as the source peer receives the overlay information from the P2P server, it captures the video and compresses it with the H.264 video coder [18]. The resulting bitstream is partitioned into source blocks, where each source block corresponds to one GOP (Group of Pictures) and is an independent unit of fixed playback duration Δ.

Then the source peer applies rateless coding on each source block and sends the resulting encoded symbols in successive UDP packets. Packets are transmitted in an interval of duration Δ with a uniform inter-departure time, and one packet is sent on each multicast tree (Fig. 3).

3.3 Inter-level Communication

A root of a level 1 multicast tree (r_1 and r_2 in Fig. 2) immediately forwards packets directly received from the source to the level 2 multicast trees associated to it (one tree for r_1 and two trees for r_2 in Fig. 2). Moreover, as soon as it successfully decodes a source block, it sends an acknowledgment to

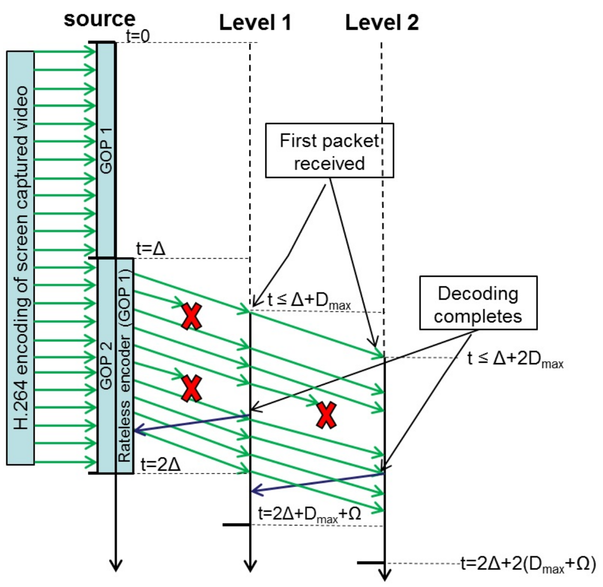

Fig. 4 Transmission strategy. The source builds the source block corresponding to the first GOP over a period of length $\varDelta$. After rateless coding, it transmits the source block over the next period of length $\varDelta$.

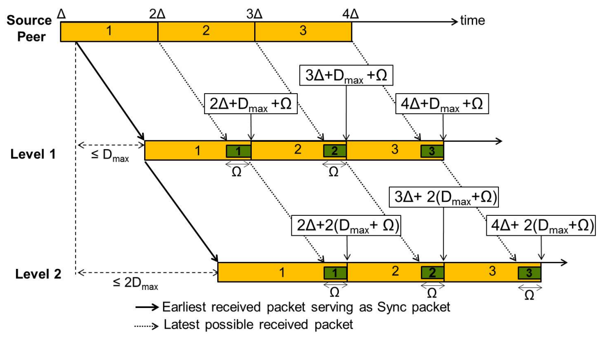

Fig. 5 Playback synchronization. Peers at level L start playing back the first GOP at time $2\varDelta + LD_{\max} + (L-1)\varOmega$.

the source, so that the source stops sending it packets, and it creates new encoded packets by applying rateless coding on the decoded source block. Then it sends these new encoded packets to level 2 peers over the multicast trees associated to it (ignoring those already used). To reduce the probability that a level 2 peer receives duplicate packets, level 1 root peers use randomly chosen rateless code *seeds* when they encode a source block. The value of the seed is sent as part of the packet header. The number of packets sent by a level 1 root peer to level 2 is set not to exceed the number of level 2 multicast trees associated to this root peer. The procedure described above for two levels is repeated for the next levels.

3.4 Level-Aware Video Delivery.

To ensure that all peers in the same level have the same playback lag, and peers in a higher level have a shorter playback lag than those in a lower one, the following procedure is followed. All peers are synchronized in time. This can be achieved, for example with the Network Time Protocol [19]. A time stamp is inserted in each UDP packet to indicate the start time of the current source block.

All level 1 peers play back the first GOP at time $2\varDelta + D_{\max}$. Here $D_{\max} = (H+1) \times l_{\max}$, where H is the maximum height of a multicast tree and $l_{\max}$ is an estimation of the maximum latency between two nodes in the overlay. Thus $2\varDelta + D_{\max}$ is the latest possible arrival time for any packet from the first source block (Fig. 4).

When a level 1 root peer completes the decoding of the first source block, it enters the re-encoding phase up to time $2\varDelta + D_{\max} + \varOmega$. Here, $\varOmega$ gives sufficient time to send re-encoded packets in case the decoding of the source block is delayed. As the packet loss rate increases, the decoding completion time shifts towards $2\varDelta + D_{\max}$ and more time is needed for sending re-encoded packets. We used $\varOmega = p_{\max}\varDelta$ where $p_{\max}$ is an estimation of the maximum packet loss rate between two nodes in the overlay (Fig. 5).

All level 2 peers play back the first GOP at time $2\varDelta + 2D_{\max} + \varOmega$, which is the latest possible arrival time of any packet for the first source block.

More generally, all peers at level L play back the kth GOP at time $(k+1)\varDelta + LD_{\max} + (L-1)\varOmega$.

4 Overlay construction and management

In §4.1, we formulate the problem of constructing an optimal forest for the diffusion of source blocks in *one* level. In §4.2, we explain the inter-level interactions in a multi-level overlay. We detail our forest construction algorithm in §4.3.

4.1 Intra-Level Multicast Trees

For simplicity, we first describe the problem for one level only. The main idea is to use multicast trees for video diffusion. Each multicast tree is a rooted tree used to transmit *one* packet of encoded symbols from the source s to a number of peers. The *root* of the tree receives the packet directly from the source.

To recover the source block, a peer must receive at least K packets of encoded symbols. In other words, a peer should belong to at least K trees.

Every peer v has an *upload capacity*, denoted by c_v, which is the number of packets the node v can transmit. This capacity constraint limits the number of children a node can have. In addition, to guarantee on-time delivery and reliability at the peers, the end-to-end delay and the packet loss rate on the path from the source to each peer should be bounded. This requirement imposes a bound on the height of each tree.

Our objective is to minimize the transmission rate at the source while guaranteeing recovery of the source block. The more trees are used to ensure that all peers receive at least K packets, the higher is the source transmission rate. Thus, minimizing the source transmission rate is equivalent to minimizing the number of trees. In light of this, we formulate the video diffusion problem in the overlay as a *Height-Bounded Spanning Forest problem with Capacity constraint* (HBSFC). The goal of HBSFC is to find a set of trees (a forest) F with minimum cardinality such that

– the number of trees is limited by the source capacity c_s.
– for each node v, the sum of its out-degrees in all trees of the forest is not greater than its capacity c_v.
– each node is in at least K trees. A node that cannot be inserted in K trees, is rejected from the overlay.
– the height of each tree is limited by a bound H.

4.2 Multi-level Overlay Management

When the number of levels is greater than one, we have to make sure that the video can be relayed from one level to the next. We create *inter-level connections* for that purpose.

To simplify the management of the whole system, root peers in a level are in charge of transmitting data to the next level. To this effect, a root peer immediately forwards a packet it received *directly* from the previous level to a root peer in the next level. Therefore, a root peer in level l acts as a source for some root peers in level $l + 1$. This strategy has three advantages: (i) there is no delay between the reception of a packet and its transmission to the next level, (ii) the inter-level connections are well distributed over peers because root peers are well distributed over the population, and (iii) the management of inter-level connections is easy for the P2P server: it only has to inform the root peers of every level about the trees in the next level.

We denote by R_l the set of level l root peers. If there is no capacity issue, each root peer in level l becomes a source of $\left\lfloor \frac{|R_{l+1}|}{|R_l|} \right\rfloor$ trees in level $l + 1$. The remaining trees are randomly allocated to the root peers.

Fig. 2 gives an example of a two-level overlay. Each peer in a level is spanned in two different trees, with height no more than three. Peers r_1 and r_2 are the root peers in level 1, while r_3, r_4 and r_5 are the root peers in level 2, i.e., $R_1 = \{r_1, r_2\}, R_2 = \{r_3, r_4, r_5\}$. Therefore, $|R_1|$ is equal to 2, while $|R_2|$ is equal to 3. Peers r_1 and r_2 are the sources for trees in level 2. Peers r_3 and r_4 are connected to r_1 and r_2, respectively. The tree for r_5 is randomly allocated to r_2.

The management of peer capacity is a critical issue for the inter-level links. We explain how we solve this issue hereafter.

For any peer v, we distinguish the upload capacity c_v^{cur} that can be used to serve peers in the same level from the upload capacity c_v^{next} that is secured to serve peers in the next level. Clearly, $c_v^{cur} + c_v^{next} \leq c_v$ for any peer v, and $c_v^{next} = 0$ if v is not a root peer or is a root peer at the lowest level.

Algorithm 1 Overlay construction algorithm - Insertion

```
Require: Complete graph $G(V, E)$,
      $v$          :  *unspanned* peer sending join request,
      $L$          :  number of levels in the overlay,
      $L_{max}$    :  maximum number of levels the overlay
                      can have,
      $K$          :  minimum number of packets to be received
Ensure: Forest $F_l$ in each level $l$
 1: for $l = 1$ to $L$ and $v$ is unspanned do
 2:     insert $v$ in forest $F_l$
 3: end for
 4: if $v$ is unspanned then
 5:     if $L < L_{max}$ and $C_L^{next} \geq K$ then
 6:         insert $v$ in forest $F_{L+1}$
 7:     else if $c_v > c_u$ where $u \in V$ with minimum capacity then
 8:         replace $u$ by $v$ in all trees containing $u$
 9:         reject $u$
10:     else
11:         reject $v$
12:     end if
13: end if
```

A root peer should reserve some upload capacity to serve some peers in the next level. The number of trees that are constructed in a level l depends on the amount of resources that have been secured by the root peers in the previous level. If we denote by F_l and V_l, the forest and the set of peers in level l, respectively, then the number of trees in F_l is $|F_l| = \sum_{v \in V_{l-1}} c_v^{next}$.

Once the joining request of a peer is received, we should both update the trees and determine the amount of resources that must be secured for the next level. On the one hand, we aim to maximize the number of peers that are covered in level l, i.e., the number of peers that are spanned in at least K trees of F_l. *This objective calls for a high* $\sum_{v \in V_l} c_v^{cur}$. On the other hand, we have to reserve enough resources for the next level. *This objective calls for a high* $C_l^{next} = \sum_{v \in V_l} c_v^{next}$. The two objectives conflict. Algorithm 1 and Algorithm 2 use heuristics to find a compromise between these two objectives when the overlay structure is updated following peer insertion or removal.

4.2.1 Overlay construction - Insertion

In Algorithm 1, we propose a heuristic to insert into the overlay a peer that sends a join request. The algorithm tries to insert the peer into the nearest level to the source (lines 1-3). The peer is inserted into K trees of the forest in a level with Algorithm 4 (line 2). If the peer cannot be inserted into any level, one of the following occurs:

– A new level is added to the overlay and the peer is inserted into the new level if the overlay can have an addi-

Algorithm 2 Overlay construction algorithm - Removal

```
Require: Complete graph $G(V, E)$,
    $v$        : spanned peer in level $l'$ sending removal request,
    $F_l$      : height-bounded forest in level $l$,
    $T_l^k$    : tree $\in F_l$,
    $L$        : number of levels in the overlay
Ensure: Forest $F_l$ in each level $l$
 1: remove $v$ from forest $F_{l'}$
 2: for $l = l'$ to $L$ and $|F_l| > C_{l-1}^{next}$ do
 3:     sort trees in $F_l$ in increasing size order
 4:     for $k = 1$ to $|F_l|$ and $|F_l| > C_{l-1}^{next}$ do
 5:         $W \leftarrow$ all nodes in $T_l^k$
 6:         sort nodes in $W$ in increasing height order (i.e., leaf nodes first)
 7:         for $i = 1$ to $|W|$ and $v_i$ is a leaf $\in W$ do
 8:             find a tree $T_l^m \in F_l$ into which $v_i$ can be inserted
 9:             if $v_i$ is inserted in $T_l^m$ then
10:                 remove $v_i$ from $T_l^k$
11:             end if
12:         end for
13:         if $T_l^k$ does not contain any node then
14:             $F_l \leftarrow F_l \backslash \{T_l^k\}$
15:         end if
16:     end for
17:     if $|F_l| > C_{l-1}^{next}$ then
18:         remove node $u \in V_l$ with minimum capacity from $F_l$
19:         end of the algorithm
20:     end if
21: end for
```

tional level, i.e., if the number of levels has not reached the maximum number of levels the overlay can have and the peers in the last level have enough resources to serve the new level (lines 5-6).

– If the peer has more capacity than the peer u having the minimum capacity in the overlay, the peer replaces u in all trees containing u (lines 7-9).
– The peer is rejected (line 11).

4.2.2 Overlay construction - Removal

In Algorithm 2, we propose a heuristic to remove a peer from the overlay. The peer sending the request is removed from the level in which the peer is residing with Algorithm 5 (line 1). The peer removal from a level may result in the creation of new trees in the level. If the peers residing in the previous level cannot gather enough resources to serve all trees in the level, the algorithm tries to decrease the number of trees by removing some trees from the level (lines 2-16). The algorithm first removes trees having the smallest number of nodes (lines 3-4). The removal of a tree is based on the insertion of tree nodes into other trees not containing the nodes. The removal process starts from the leaves of the tree and stops if a leaf cannot be inserted into any other tree (lines 5-7). If a leaf can be inserted into another tree, it is removed

Algorithm 3 Tree construction algorithm - Insertion

```
Require: Complete graph $G(V, E)$,
    $v$        : unspanned peer sending join request,
    $c_u^s$    : spare capacity of a node $u \in V$,
    $F_l$      : height-bounded forest in level $l$,
    $T_l^k$    : tree $\in F_l$,
    $H$        : maximum tree height
Ensure: updated tree $T_l^k$
 1: $W \leftarrow$ all nodes in $T_l^k$
 2: $u \leftarrow$ a leaf $\in W$ with maximum spare capacity and $depth = H$
 3: sort nodes in $W$ in increasing depth order (i.e., root node first)
 4: for $i = 1$ to $|W|$ and $v_i \in W$ with $depth(v_i) < H$ do
 5:     if $c_{v_i}^s > 0$ then
 6:         insert $v$ into $T_l^k$ as a child of $v_i$
 7:         update $c_{v_i}^s$
 8:         succeed - end of the algorithm
 9:     else if $c_v^s > 1+$number of children of $v_i$ in $T_l^k$ then
10:         replace $v_i$ by $v$ and insert $v_i$ as a child of $v$ in $T_l^k$
11:         update $c_{v_i}^s$, $c_v^s$
12:         succeed - end of the algorithm
13:     else if $\exists u$ and $c_u^s > 1+$number of children of $v_i$ in $T_l^k$ then
14:         replace $u$ by $v$ in $T_l^k$
15:         replace $v_i$ by $u$ and insert $v_i$ as a child of $u$ in $T_l^k$
16:         update $c_{v_i}^s$, $c_u^s$
17:         succeed - end of the algorithm
18:     end if
19: end for
20: fail - end of the algorithm
```

from the current tree (lines 8-11). The tree is removed from the forest in the level if it does not contain any node (lines 13-15). After the tree removal process, if the peers in the previous level still cannot support the level, the peer with minimum capacity is removed from the level (lines 17-20).

4.3 Resource-Aware Multicast Trees

This section explains the algorithms for the construction of *one* intra-level overlay forest in case of peer insertion and peer removal. These algorithms consider the constraints described in §4.1 (about the trees) and §4.2 (about the sharing of physical resources).

In §4.3.1, we describe our tree construction algorithm in case of peer insertion. In §4.3.2 and §4.3.3, we detail our forest construction algorithm in case of peer insertion and peer removal, respectively.

4.3.1 Tree construction - Insertion

Algorithm 3 is a key routine called by Algorithm 2 in line 9 and by Algorithm 4 in lines 10 and 20. The algorithm either fails, if the peer cannot be inserted into the tree, or succeeds by returning the tree containing the peer.

The algorithm aims to insert the peer as close as possible to the tree root. To do so, we traverse the tree nodes from top to bottom except for the leaves with maximum depth (lines 3-4). During the tree traversal, one of the following occurs:

Algorithm 4 Forest construction algorithm - Insertion

```
Require: Complete graph $G(V, E)$,
    $v$ : unspanned peer sending joining request,
    $F_l$ : forest in level $l$ where peer is to be inserted,
    $T_l^k$ : tree $\in F_l$,
    $K$ : minimum number of packets to be received
Ensure: Height-bounded forest $F_l$
1: $F \leftarrow \emptyset$
2: for $k = |F_l| + 1$ to $C_{l-1}^{next}$ and $v$ is not spanned in $K$ trees do
3:     $p \leftarrow$ a root node with spare capacity in $V_{l-1}$
4:     create $T_l^k = (\{v\}, \emptyset)$ with $p$ as source
5:     $F \leftarrow F \cup \{T_l^k\}$
6:     secure one unit of capacity from root node $v$ if $v$ has spare capacity
7: end for
8: sort trees in $F_l$ in increasing size order
9: for $k = 1$ to $|F_l|$ and $v$ is not spanned in $K$ trees do
10:    insert $v$ in $T_l^k$
11: end for
12: $F_l \leftarrow F_l \cup F$
13: $F \leftarrow$ trees in $F_l$ containing $v$
14: $F' \leftarrow$ trees in $F_l$ not containing $v$
15: sort trees in $F$ in increasing size order
16: sort trees in $F'$ in decreasing size order
17: for $k = 1$ to $|F|$ and $v$ is not spanned in $K$ trees do
18:     for $m = 1$ to $|F'|$ and $v$ is not spanned in $K$ trees do
19:         $u \leftarrow$ a leaf with maximum spare capacity, in $T_l^m$, but not in $T_l^k$
20:         if $u$ is inserted in $T_l^k$ then
21:             replace $u$ by $v$ in $T_l^m$
22:             $F \leftarrow F \cup \{T_l^m\}$
23:             $F' \leftarrow F' \backslash \{T_l^m\}$
24:         end if
25:     end for
26: end for
27: if $v$ is not spanned in $K$ trees then
28:     remove $v$ from trees in $F_l$
29: end if
```

- If there exists a node with spare capacity, the peer is inserted into the tree as a child of this node (lines 5-8).
- If there exists a node such that the peer can support the node and all its children, the peer replaces the node and the node is attached to the peer as its child. The peer thus becomes the parent of the node and its children (lines 9-12).
- If a deepest leaf u with maximum spare capacity in the tree with maximum height (line 2) exists and this leaf u can support an inner node and its children, then the peer replaces u, u replaces the inner node, and the inner node is attached to u as its child (lines 13-18).

The algorithm fails if the peer cannot be inserted into the tree (line 20).

Algorithm 5 Forest construction algorithm - Removal

```
Require: Complete graph $G(V, E)$,
    $v$ : spanned peer in level $l$ sending removal request,
    $F_l$ : forest in level $l$,
    $T_l^k$ : tree $\in F_l$
Ensure: Height-bounded forest $F_l$
1: $F \leftarrow$ trees in $F_l$ containing $v$
2: for $k = 1$ to $|F|$ do
3:     if $v$ is a leaf in $T_l^k$ then
4:         remove $v$ from $T_l^k$
5:         if $T_l^k$ does not contain any node then
6:             $F_l \leftarrow F_l \backslash \{T_l^k\}$
7:         end if
8:     else
9:         $u \leftarrow v$'s child node with maximum height in $T_l^k$
10:        replace $v$ by $u$ in $T_l^k$, i.e., children of both $v$ and $u$ become $u$'s children
11:        while $u$ does not have enough capacity to support all its children in $T_l^k$ do
12:            form a new tree consisting of $u$'s child with maximum height
13:        end while
14:    end if
15: end for
```

4.3.2 Forest construction - Insertion

Algorithm 4 constructs a forest in one level in case of peer insertion. The algorithm is a key routine called by Algorithm 1 in lines 2 and 6. It returns a forest containing the required number of trees such that all nodes are spanned in K trees. The algorithm is based on the following three steps:

- *Creation of new trees*: Depending on the amount of resources secured for level l by the peers residing in the previous level ($l - 1$), we first create new trees at level l rooted at the peer v sending the join request. We link the root peers in levels $l - 1$ to v and secure capacity for the next level ($l + 1$) (lines 2-7).
- *Insertion into existing trees*: If the peer has not been spanned in K trees, it is inserted into existing trees in the level according to Algorithm 3 (lines 8-11). The peer is first inserted into trees having the smallest number of nodes in order to minimize peer connection changes in trees as a result of peer insertion. This also simplifies the completion of the peer's trees by leaves in other trees in the third step of the algorithm.
- *Replacing leaves in other trees after inserting them into the peer's trees*: If the peer still has not been spanned in K trees, we use Algorithm 3 to insert leaves with maximum spare capacity into the peer's trees with smallest size. These leaves are taken from trees with maximum size that do not contain the peer (lines 13-20). The leaves are then replaced by the peer (line 21).

Finally, if the peer has not been inserted into K different trees, the initial state of the forest is kept (lines 27-29).

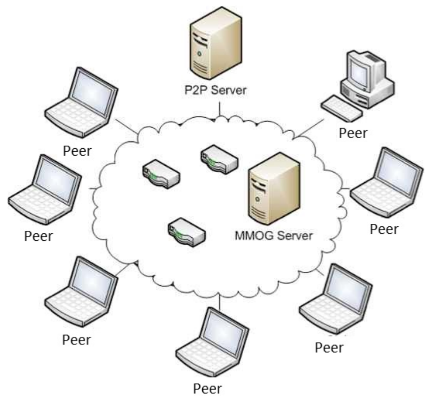

Fig. 6 Network topology of the simulation environment.

4.3.3 Forest construction - Removal

Algorithm 5 constructs a forest in one level in case of peer removal. The algorithm is a key routine called by Algorithm 2 in line 1. It returns a forest containing the required number of trees such that all nodes excluding the peer (and maybe some other peers due to capacity constraints) are spanned in K trees.

If the peer sending a removal request is a leaf, we simply delete it (lines 3-4). We delete the tree from the forest if the tree does not contain any node (lines 5-7).

On the other hand, if the peer has children, the child u with maximum height replaces the peer in the tree (lines 9-10). In addition to its own children, u now becomes the parent of the children of the peer. If u does not have enough capacity to support all its children, u's children with maximum height are transformed into new trees in the forest (lines 11-13).

5 Simulation results

We used the ns-2 network simulator to test our system with respect to scalability, bandwidth heterogeneity, packet loss, and peer churn. The simulations were run on a PC with an Intel Core i7-2600K 3.4 GHz processor and 16 GB RAM.

We modeled the MMOG traffic as constant bit rate (CBR) over TCP because most popular MMOGs, e.g., World of Warcraft, use TCP as the transport protocol [20]. We followed measurements in [20] and set the peer upload and download MMOG traffic to 5 kbps and 14 kbps, respectively. A low-rate CBR background traffic (10 kbps) over TCP was added to consider additional user online activities such as Web browsing.

The simulation environment consisted of three main components: P2P server, peers, and MMOG server. Each system component was connected to the Internet via its access link. This configuration leads to a large-scale star topology (Fig. 6). Each peer, represented by a node in ns-2, has one MMOG client, one P2P source agent, up to four P2P sink agents depending on how many videos it is receiving simultaneously, and one TCP background traffic agent.

The peer download capacity was set to 10 Mbps for all peers. For the peer upload capacity, we followed [21] and used a log-normal distribution. In the simulations where a peer was included in only one overlay (§5.2), the mean upload capacity was set to 1024 kbps, and the second parameter of the distribution, σ, was set to 0.385, giving upload capacities ranging from 256 kbps to 4.5 Mbps. In the simulations where a peer was included in up to four overlays (§5.3), the mean upload capacity was increased to 4096 kbps while σ was not changed. To avoid that a source peer gets a smaller upload capacity than the video bit rate, all source peers were assigned an upload capacity of 1024 kbps and were allowed to use up to 512 kbps for video streaming.

The video sharing service was constrained to exploit only half the upload bandwidth of a peer. This provided a safety margin against bandwidth fluctuations of up to 50% and ensured that video sharing is not affected as long as the available upload bandwidth does not fall by more than 45%. This takes into account the MMOG bandwidth requirement which is 0.3% to 5% of the peer upload capacity (256 kbps to 4.5 Mbps).

For the link latencies, we followed measurements in [22] and used a log-normal distribution with mean 17.19 ms and variance 0.0029.

Peer churn was modeled with the following parameters:

– ratio of peers initially in the system to the maximum number of peers. We set the ratio to 0.05.
– peer playing session length distribution. Measurements in [23] and [24] show that the player session length in MMOGs has a heavy tail characteristic. We followed the measurements in [24] and modeled the playing session length with a Weibull distribution. We used $scale = 50$ and $shape = 0.5$ to have a mean session duration of 100 s. The minimum session duration was set to 20 s.
– peer arrival. Peers join the system according to a Poisson process with rate $\lambda = \frac{1.3n}{T}$, where n is the number of peers that want to join the system and T is the simulation duration.

We used the CIF Foreman video sequence and encoded it with the H.264 encoder at 30 frames per second (fps) and 320 kbps. Each GOP had one I frame followed by 29 P frames. The playback duration of each source block was $\Delta = 1$ s. We exploited the accurate Raptor code model proposed in [25] to simulate rateless coding. With this model, a redundancy of 5% gives a high probability of successful decoding [25]. However, to take the effect of peer churn into account, redundancy was set to 50%. For the Raptor code, the symbol size was 1 byte and there were 938 symbols in each UDP packet. The maximum tree height in the overlay

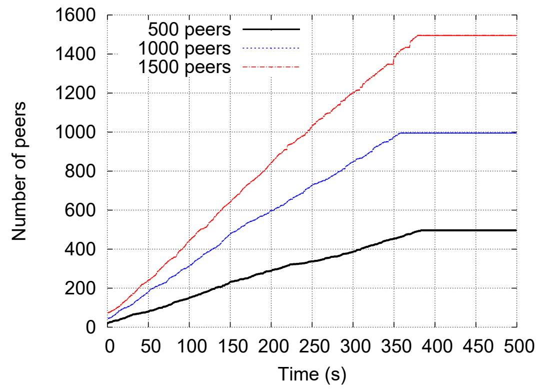

Fig. 7 Number of peers in the system as a function of time. The number of peers changes according to a churn model determined by the ratio of peers initially in the system (0.05), the playing length distribution (Weibull model), and peer arrival (Poisson process). Three curves are shown, each of which corresponds to a different maximum number of peers (500, 1000, and 1500).

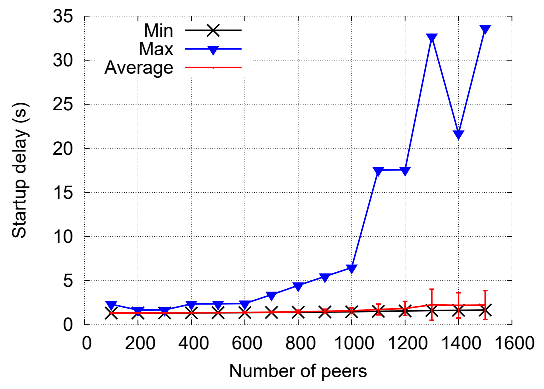

Fig. 8 Start-up delay vs. maximum number of peers. The error bars represent one standard deviation on each side of the average delay.

was $H = 3$. The maximum latency between two nodes in the overlay, $l_{\max}$, was set to the maximum link latency of the log-normal distribution, and the maximum packet loss rate between two nodes, $p_{\max}$, was set to the input packet loss rate.

5.1 Metrics

To evaluate our system, we used the following metrics:

- **Start-up delay**: interval between the time a user joins a P2P system and the time it starts playing back the video.
- **Playback lag**: difference between the playback time of the source peer and that of the receiving peer.
- **Rejection rate**: probability that a user is rejected when it tries to join the system.
- **Continuity index** [26]: ratio of the number of source blocks that were available at their due playback time to the number of source blocks that should have been played back by that time. This is a measure of in-time delivery of video content.
- **PSNR**: measure of the objective quality of the reconstructed video with respect to the original one. For an original video frame f_1 and a reconstructed one f_2, each containing $N \times N$ pixels with values in $\{0, \ldots, 255\}$, the PSNR is computed as

$$PSNR(f_1, f_2) = 10 \log_{10} \left[\frac{255^2 \times N^2}{\sum_{i=1}^{N} \sum_{j=1}^{N} (f_1^{i,j} - f_2^{i,j})^2} \right]$$

  where $f_1^{i,j}$ and $f_2^{i,j}$ are the pixel values at row i and column j in the original and reconstructed frame, respectively.
- **Average upload and download bandwidth**.
- **Delay penalty**: percentage increase in the round trip time (from a gamer's machine to the MMOG server) caused by video streaming.
- **Bandwidth penalty**: percentage reduction in the available bandwidth for MMOG traffic caused by video streaming.

  The delay and bandwidth penalties are used to measure whether video streaming affects the gaming experience.

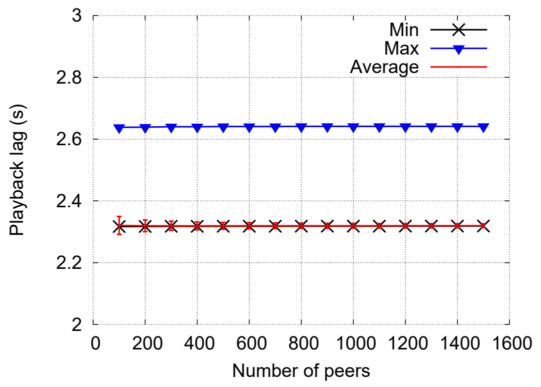

Fig. 9 Playback lag vs. maximum number of peers. The error bars represent one standard deviation on each side of the average playback lag

5.2 Single Overlay

In the first experiment, we used a single video source and studied the scalability of the system with respect to the maximum number of peers in the overlay (from 100 to 1500, with an increment of 100). For each value of the maximum number of peers, we repeated the simulations 50 times as this number was sufficient to obtain stable results. The simulations were run independently, with a duration of 500 s each.

Fig. 7 shows the average number of active peers as a function of time for three selected values of the maximum number of peers $(500, 1000, 1500)$.

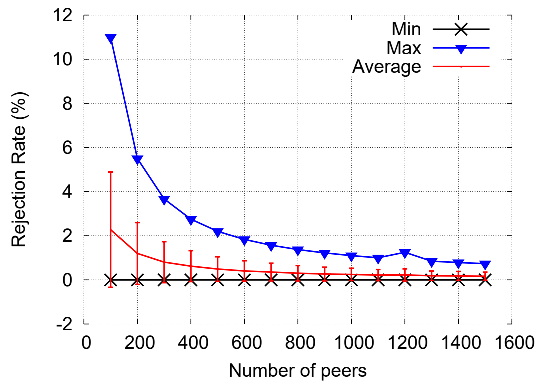

Fig. 10 Rejection rate vs. maximum number of peers. The error bars represent one standard deviation on each side of the average rejection rate.

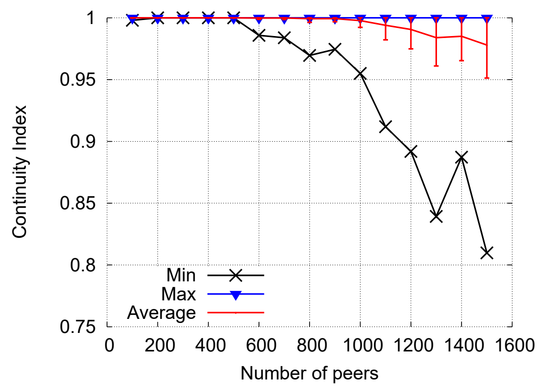

Fig. 11 Continuity index vs. maximum number of peers. The error bars represent one standard deviation on each side of the average continuity index.

Fig. 8 and Fig. 9 show the start-up delay and playback lag, respectively, for each value of the maximum number of peers. Most of the peers experienced a start-up delay of less than 2 s and a playback lag of less than 2.4 s. Moreover, the average values were very close to the minimum values. In very rare cases, the start-up delay was high. This happened when, repeatedly, sending peers left the overlay before the receiving peer decoded the first source block successfully.

Fig. 10 shows the rejection rate. A peer requesting to view the video could be rejected by the P2P Server if its inclusion causes under-provisioning, i.e., the total required download bandwidth becomes higher than the total upload capacity. When the maximum number of peers was increased, the probability of finding peers to upload the video to a new peer increased and the rejection rate decreased.

Fig. 11 shows that the average continuity index of peers was very close to 1. Fig. 12 shows that the average received PSNR remained close to the average PSNR of the transmitted video (35.07 dB). Note that a few peers had a PSNR above 35.07 dB. This is because a peer can be active for only a few seconds and receive that part of the video that has higher PSNR than the average. The rare cases of peers

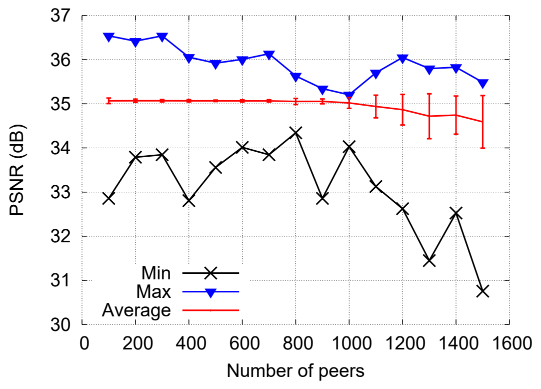

Fig. 12 PSNR vs. maximum number of peers. The error bars represent one standard deviation on each side of the average PSNR.

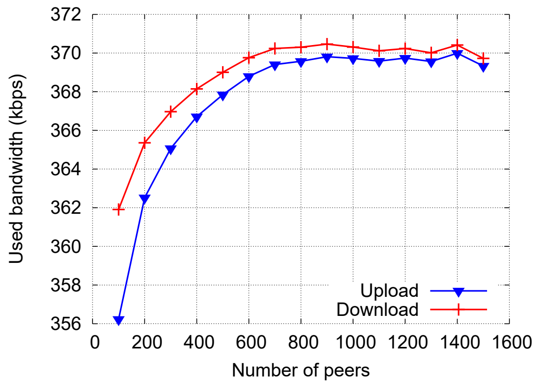

Fig. 13 Average used bandwidth vs. maximum number of peers.

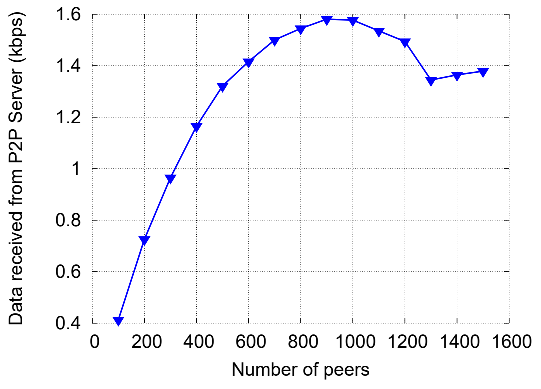

Fig. 14 Control information vs. maximum number of peers.

with poor PSNR were due to peer churn (peers present in the P2P system only when the video PSNR was low or peers receiving video from peers that left and were replaced by peers that left immediately).

Fig. 13 shows the average used upload and download bandwidth for video sharing. The average used upload bandwidth was slightly lower than the average used download bandwidth because the source peer is contributing its upload bandwidth which reduces the burden on peers. The differ-

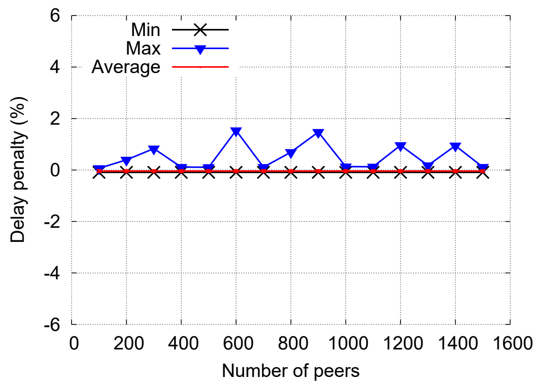

Fig. 15 Delay penalty vs. maximum number of peers. The error bars represent one standard deviation on each side of the average delay penalty.

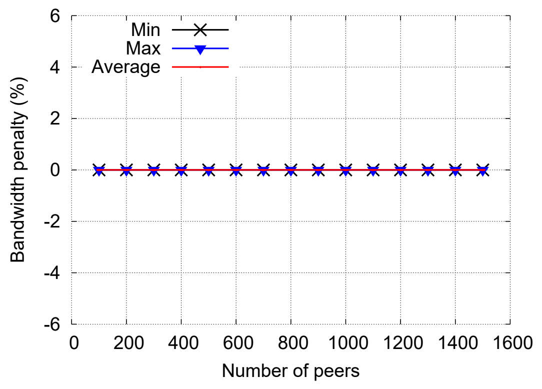

Fig. 16 Bandwidth penalty vs. maximum number of peers. The error bars represent one standard deviation on each side of the average bandwidth penalty.

ence between the average used upload and download bandwidth is larger when the number of peers is small.

Whenever a peer leaves or joins, the P2P server sends overlay update messages to the affected peers. As shown in Fig. 14, this overhead was negligible compared to the video bit rate. By increasing the maximum number of peers, the average degree of the overlay (i.e., the average number of peers associated to a peer) increased and therefore the overhead also increased. However, beyond a certain threshold (900 in this experiment), the average degree saturated as new peers were placed in lower levels. By further increasing the number of peers, the overhead decreased because of the increase in the number of leaves.

Fig. 15 and Fig. 16 show that the P2P traffic had a negligible effect on the MMOG operation: the average response time from the MMOG server increased by only 0.04% and the average bandwidth available to the MMOG was not reduced.

We now evaluate the effect of packet loss on the received video quality. The maximum number of peers was 800. To simulate packet loss, we used an independent and identically

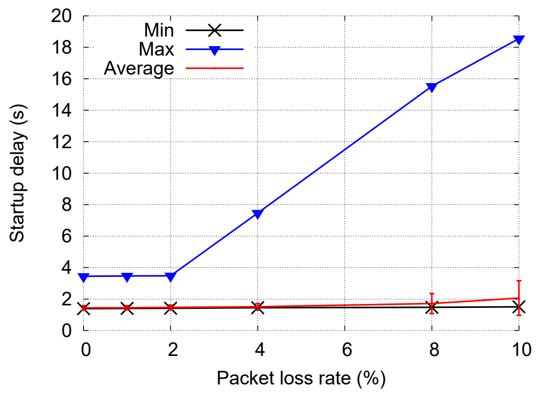

Fig. 17 Startup delay vs. packet loss rate. The error bars represent one standard deviation on each side of the average start-up delay.

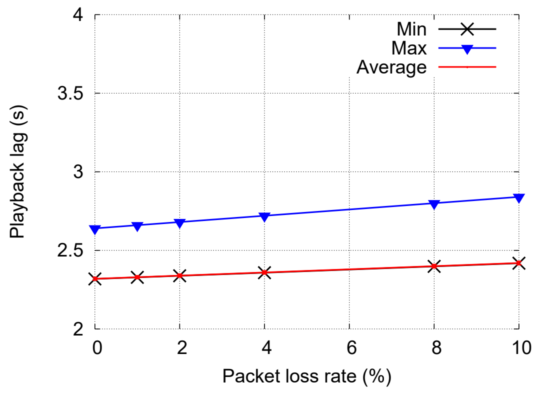

Fig. 18 Playback lag vs. packet loss rate. The error bars represent one standard deviation on each side of the average playback lag.

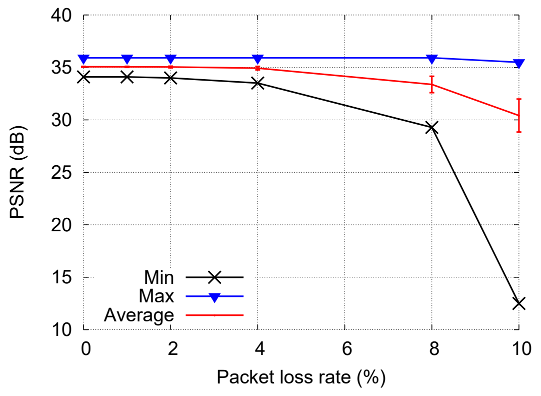

Fig. 19 PSNR vs. packet loss rate. The error bars represent one standard deviation on each side of the average PSNR.

distributed (iid) packet loss model where packets are lost with a given packet loss probability in the physical links. Fig. 17 and 18 show that the average start-up delay and playback lag increased slightly when the packet loss rate was increased. This is because a peer has to wait slightly longer before it receives enough packets for successful decoding.

Fig. 19 shows that the average PSNR of the system stayed almost constant for packet loss rates below 4%. This

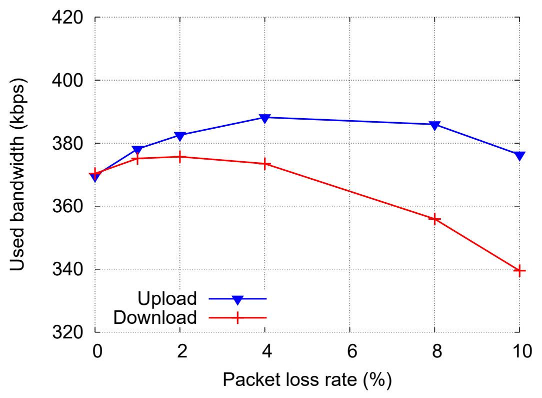

Fig. 20 Average used bandwidth vs. packet loss rate.

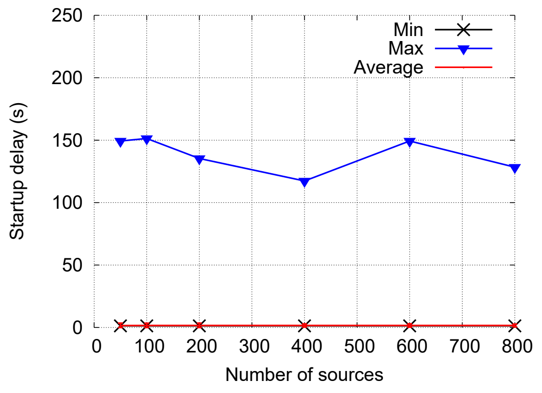

Fig. 21 Startup delay vs. number of video sources. The error bars represent one standard deviation on each side of the average startup delay.

was because the rateless code was able to recover almost all losses. However, the PSNR dropped significantly beyond that value because the redundancy used for the rateless code is not enough to cope with such a high packet loss rate. Note that due to the existence of multiple hops between the source and the receiver, the end to end packet loss rate may be much higher than the physical link loss rate.

At 0% loss rate, both the average used upload and download bandwidths are approximately the same. As the packet loss rate increased, peers received fewer packets and hence forwarded fewer packets as well, which resulted in a decrease in the used upload and download bandwidth (Fig. 20).

5.3 Multiple Overlays

This experiment was designed to study the scalability of the system with respect to the number of live videos. The maximum number of users was 500 and the number of video sources was varied. Each user was allowed to watch up to four videos such that the number of peers watching a particular video follows a Zipf distribution. When a peer participated in more than one P2P overlay, its resource was equally allocated among the overlays.

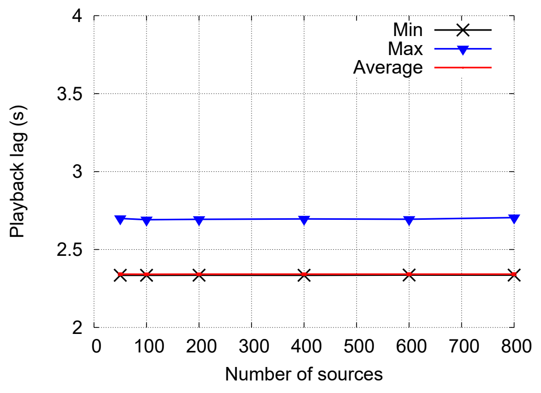

Fig. 22 Playback lag vs. number of video sources. The error bars represent one standard deviation on each side of the average playback lag.

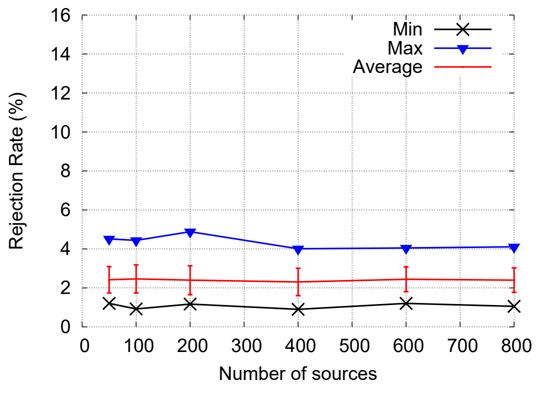

Fig. 23 Rejection rate vs. number of video sources. The error bars represent one standard deviation on each side of the average rejection rate.

Fig. 21 and 22 show that the average start-up delay and playback lag remained almost stable with increasing number of sources.

Fig. 23 shows that the rejection rate did not change significantly by increasing the number of sources.

The average PSNR remained constant and close to the maximum PSNR when the number of sources was increased (Fig. 24). Moreover, more than 99.27% of peers had a PSNR higher than 34.6 dB, and more than 92.74% of peers had a PSNR higher than 35 dB.

Fig. 25 shows the average used upload and download bandwidth as a function of the number of sources. Since a source adds its upload capacity to the overlay without consuming the download capacity, increasing the number of sources reduces the upload contribution requirement on peers. This explains why the average used upload bandwidth was lower than the average used download bandwidth and the difference between them increased when the number of sources was increased. Similarly, when the number of sources was increased, the average overlay population (i.e., the number of peers per overlay) became smaller. In a smaller overlay, a peer depends on relatively fewer send-

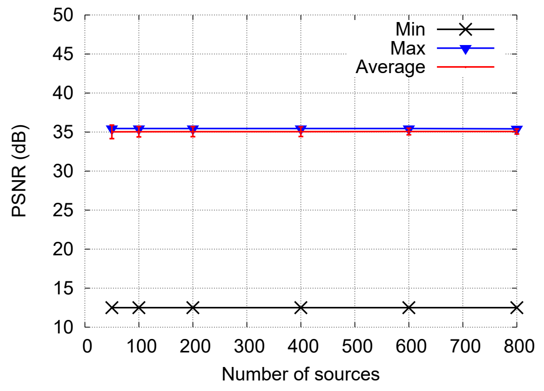

Fig. 24 PSNR vs. number of video sources. The error bars represent one standard deviation on each side of the average PSNR.

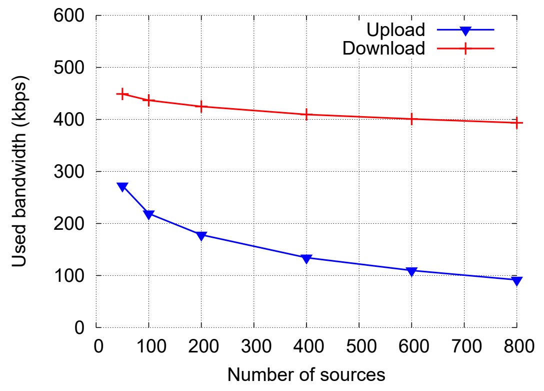

Fig. 25 Average used bandwidth vs. number of video sources.

ing peers, thus peer churn may reduce the number of packets received by a peer. This explains the slight decrease in the average download bandwidth. However, peers can still get maximum PSNR because of redundancy. When a peer receives fewer packets, it forwards fewer packets as well, which reduces the average used upload bandwidth.

6 Online tests

This section presents the results of online testing. Video from *The Missing Ink* MMOG [27] was captured in real time and encoded at a constant bit rate of 128 kbps. A frame rate of 25 fps and a resolution of 320x240 were used. Before a test, each user synchronized its PC clock with an Internet time server. This was necessary to measure the playback lag. The default upload and download bandwidths were set to 900 kbps and 1800 kbps, respectively. Users were asked to estimate their upload and download bandwidths with the *SpeedTest* tool [28]. If the measured upload bandwidth (respectively download bandwidth) was smaller than 1100 kbps (respectively 2000 kbps), the default value was replaced by the measured value minus 200 kbps (to cater for the MMOG and other background traffic).

Table 1 Results from First Online Test.

Metric	Average	Minimum	Maximum
Start-up delay (s)	28.63	18.17	34.85
Playback lag (s)	1.92	0.87	3.42
Continuity Index	0.991	0.972	1

Table 2 Results from Second Online Test.

Metric	Average	Minimum	Maximum
Start-up delay (s)	34.45	11.404	81.21
Playback lag (s)	3.265	0.97	5.51
Continuity Index	0.88	0.41	1

When a user joined a session, the P2P Client process created log files for start-up delay, playback lag and continuity index. Each log file was uniquely identifiable by the user name and video ID. At the end of the session, each tester uploaded the generated log files to an FTP server. Then a program was run to process all log files and provide the average, minimum and maximum values of all metrics. Nine users participated in the first test: three were located in Leicester (UK), two in Patras (Greece), three in Petah-Tikva (Israel), and one in Tel Aviv (Israel). Each user advertised its video between 15 to 25 min. At the same time, and in successive slots, this user selected an advertised video and watched it for at least 5 min. Table 1 shows the results of the measurements.

The start-up delay was higher than that of the ns-2 simulations. This is due to the time needed for Network Address Translation (NAT) traversal. The NAT traversal module used in our implementation took between 10 to 20 s to establish a single connection. Moreover, when a peer needed to communicate with multiple peers, the NAT module established the connections successively, aggregating the delay. The playback lag was small. Its fluctuation is due to the variation in the network propagation delay, forwarding delay, as well as lack of precision in the synchronization with the Internet time server. The average continuity index was 0.991, meaning that on average a user failed to decode fewer than 1% of the source blocks.

In a second test, a similar procedure was followed with the difference that each user was allowed to watch up to three videos simultaneously. Eight users participated in the test. Four were located in Leicester (UK), one in London (UK), and three in Patras (Greece). Each user advertised its video for 30 min. Each user watched simultaneously up to three videos selected randomly. Each of the selected videos was watched for at least 5 min before switching to another video. The results are summarized in Table 2.

Because in this test a peer watched up to three videos simultaneously, more connections needed to be established between this peer and the other peers. Since the NAT traversal module establishes connections one by one, the start-up

Table 3 Performance of Popular P2P Systems.

	Playback lag (s)	Start-up delay (s)	Experimental Setup	NAT Traversal
TVS [12]	-	25 (avg)	PlanetLab	No
PPLive [29]	150 (max)	20 to 120	Internet	Yes
Coolstreaming [26]	-	21 to 25	Trace driven simulation	Weak support
Anysee [30]	20 to 30	20	Trace driven simulation	No
SopCast [30]	60 (avg)	60 to 300	PlanetLab	-
CLive [31]	25 (avg)	-	Simulation	No
NAPA-WINE [32]	6 (min)	-	Controlled Lab Network	No
VUD [33]	5 to 20	5 to 18	Simulation (PlanetLab)	No
MATIN [34]	6 to 10	10	OMNET++	No
Transit [35]	-	5 to 50	Trace driven simulation	No

delay increased. The slower NAT traversal process also explains why the continuity index decreased.

A comparison between the performance of our P2P live streaming system with that of state of the art systems is not straightforward. This is because an implementation of those systems is not readily available to allow testing under identical conditions. Therefore, our comparison is based on performance measurements reported in the literature. Table 3 shows statistics on start-up delay and playback lag for popular P2P live streaming systems. Although this comparison is not done under identical conditions, it provides useful information.

Previous systems have a higher playback lag because they use a pull-based overlay construction mechanism. Peers store segments (of up to a few minutes of video) in their buffer. These segments may be fetched by other peers minutes after they were stored. In contrast, our system is push-based with peers sending packets on trees as soon as they receive them.

The start-up delay is mainly due to two factors: (1) overlay construction, i.e., time spent in finding a suitable set of sending peers and (2) NAT traversal. When a peer joins our system, the P2P server updates the overlay and informs all relevant peers who start using the new overlay within one second. This helps reducing the start-up delay. In other systems, usually a joining peer contacts peers chosen randomly from a list of peers to request video segments. The effect of NAT traversal can be seen in Table 3 by observing that the start-up delay of systems not supporting NAT traversal is much lower than that of systems supporting it. Our system supports full NAT traversal and still achieves a competitive start-up delay.

7 Conclusion

We presented a multi-overlay multi-level hybrid P2P live video streaming system for MMOGs. The system uses multiple trees and rateless codes to stream screen-captured video of the game. A P2P server is responsible for the overlay construction and dynamically adapts trees to peer arrival and departure. The main problem addressed by the paper is how to construct trees such that the transmission rate at the source is minimized, the number of served peers is maximized, and all peers receive enough encoded symbols to decode the video on time. We proposed algorithms to dynamically build such trees and provided extensive experimental results with the ns-2 network simulator to study the performance of our system with respect to scalability, bandwidth heterogeneity, packet loss, and peer churn. The results showed that the system has a small start-up delay, short playback lag, low rejection rate, and provides high video quality to most peers. We also presented results from real measurements over the Internet, which confirmed that our system is competitive compared to existing state of the art systems.

In the remainder of this section, we highlight challenges for future work.

Our implementation relies on an LT code [7] for channel coding. The performance can be improved by replacing this code with RaptorQ [36], a rateless code with better coding efficiency than LT codes.

A peer which does not contribute the upload bandwidth requested by the P2P server can affect the Quality of Experience (QoE) of other peers. This may happen if the peer advertises an upload capacity that is higher than the actual one or the ISP limits the upload bandwidth. To deal with this situation, the P2P client could send periodic reports to the P2P server. These reports should contain the packet delivery status from different senders and enable the P2P server to infer the actual contribution of each peer.

The system asks users to measure the available bandwidth and report it to the P2P server. A wrong estimation of bandwidth may lead to inefficient overlay construction. This issue could be addressed by implementing a mechanism to provide a more accurate estimation of the available bandwidth without involving user interaction.

When a peer joins or leaves the system, the P2P server sends update overlay information to all peers. This communication overhead could be reduced by sending update messages to affected peers only. Also, the P2P server reacts to each new event (join request or departure of a peer) independently by updating the overlay and sending updates to each peer. A better solution would be to group all events produced within a certain time frame (e.g., one second) and process them simultaneously to generate a single update message for all events generated in this time frame. This will reduce the computational load as well as the communication overhead of the P2P server.

References

1. E. Ferrari, J. Lessiter, and J. Freeman (2011) Users and uses of multiplayer games and community activities. In: Proc. NEM Summit, Turin.
2. Online. `http://www.cng-project.eu/`. Last accessed 29 Feb. 2016.
3. Online. `http://www.twitch.tv/`. Last accessed 29 Feb. 2016.
4. Online. `http://www.ustream.tv`. Last accessed 29 Feb. 2016.
5. Online. `http://www.livestream.com/`. Last accessed 29 Feb. 2016.
6. Online. `http://themittani.com/news/own3dtv-shuts-down`. Last accessed 29 Feb. 2016.
7. M. Luby (2002) LT codes. In: Proc. 43rd Annual IEEE Symposium on Foundations of Computer Science, Vancouver, pp. 271–280.
8. A. Shokrollahi (2006) Raptor codes. IEEE Trans. Inf. Theory 52(6): 2551–2567.
9. C. Wu and B. Li (2008) rstream: Resilient and optimal peer-to-peer streaming with rateless codes. IEEE Trans. Parallel Distrib. Syst. 19(1): 77–92.
10. M. Grangetto, R. Gaeta, and M. Sereno (2009) Rateless codes network coding for simple and efficient P2P video streaming. In: Proc. IEEE ICME 2009, New York, pp. 1500-1503.
11. M. Grangetto, R. Gaeta, and M. Sereno (2009) Reducing content distribution time in P2P based multicast using rateless codes. In: Proc. Italian Networking Workshop, Bologna, pp. 1–12.
12. A. Magnetto, R. Gaeta, M. Grangetto, and M. Sereno (2010) P2P streaming with LT codes: a prototype experimentation. In: Proc. ACM Workshop on Advanced Video Streaming Techniques for Peer-to-Peer Networks and Social Networking, Florence, pp. 7–12.
13. H. R. Oh, D. O. Wu, and H. Song (2011) An effective mesh-pull-based P2P video streaming system using Fountain codes with variable symbol sizes. Computer Networks 55(12): 2746–2759.
14. P. M. Eittenberger (2012) RaptorStream: boosting mobile peer-to-peer streaming with Raptor codes. In: Proc. ACM SIGCOMM 2012, Helsenki, pp. 291–292.
15. P. M. Eittenberger, T. Mladenov, and U. R. Krieger (2012) Raptor codes for P2P streaming. In: Proc. 20th IEEE Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), Garching, pp. 327–332.
16. S. Ahmad, C. Bouras, E. Buyukkaya, R. Hamzaoui, A. Papazois, A. Shani, G. Simon, and F. Zhou (2012) Peer-to-peer live streaming for massively multiplayer online games. In: Proc. IEEE 12th International Conference on Peer-to-Peer Computing, Tarragona, pp. 67–68.
17. E. Buyukkaya, S. Ahmad, M. Dawood, J. Liu, F. Zhou, R. Hamzaoui, and G. Simon (2012) Level-based peer-to-peer live streaming with rateless codes. In: Proc. IEEE International Symposium on Multimedia, Irvine, pp. 249–254.
18. T. Wiegand, G. Sullivan, G. Bjontegaard, and A. Luthra (2003) Overview of the H.264/AVC video coding standard. IEEE Trans. Circuits and Systems for Video Technology 13(7), pp. 560–576.
19. D. Mills. Network Time Protocol (Version 3) Specification, Implementation and Analysis, Request for Comments 1305. Internet Engineering Task Force, 1992. `http://www.rfc-editor.org/rfc/rfc1305.txt`.
20. P. Svoboda, W. Karner, and M. Rupp (2007) Traffic analysis and modelling for World of Warcraft. In: Proc. IEEE ICC, Glasgow, pp. 1612-1617.
21. A. Bharambe, J. R. Douceur, J. R. Lorch, T. Moscibroda, J. Pang, S. Seshan, and X. Zhuang (2008) Donnybrook: enabling large-scale, high-speed, peer-to-peer games. In: Proc. SIGCOMM, Seattle, pp. 389–400.
22. A. Hernandez and E. Magana (2007) One-way delay measurement and characterization. In: Proc. ICNS'07 Third International Conference on Networking and Services, Athens.
23. K.-T. Chen, P. Huang, and C.-L. Lei (2006) Game traffic analysis: an MMORPG perspective. Computer Networks 50(16), pp. 3002–3023.
24. W.-C. Feng, D. Brandt, and D. Saha (2007) A long-term study of a popular MMORPG. In: Proc. NetGames, Melbourne, pp. 19–24.
25. M. Luby, T. Gasiba, T. Stockhammer, and M. Watson (2007) Reliable multimedia download delivery in cellular broadcast networks. IEEE Trans. Broadcasting 53(1): 235–246.
26. B. Li, S. Xie, Y. Qu, G. Y. Keung, C. Lin, J. Liu, and X. Zhang (2008) Inside the new coolstreaming: principles, measurements and performance implications. In: Proc. INFOCOM'08, Phoenix.
27. Online. `http://www.redbedlam.com/missing-ink-mmorpg/`. Last accessed 29 Feb. 2016.
28. Online. `http://www.speedtest.net/`. Last accessed 29 Feb. 2016.
29. X. Hei, C. Liang, J. Liang, Y. Liu, and K.W. Ross (2007) A measurement study of a large-scale P2P IPTV system. IEEE Transactions on Multimedia 9(8): 1672–1687.
30. A. Sentinelli, G. Marfia, M. Gerla, L. Kleinrock, and S. Tewari (2007) Will IPTV ride the peer-to-peer stream. IEEE Communications Magazine 45(6): 86–92.
31. A. H. Payberah, H. Kavalionak, V. Kumaresan, A. Montresor, and S. Haridi (2012) CLive: Cloud-assisted P2P live streaming. In: Proc. IEEE 12th International Conference on Peer-to-Peer Computing, Tarragona, pp. 79–90.
32. S. Traverso, L. Abeni, R. Birke, C. Kiraly, E. Leonardi, R. Lo Cigno, and M. Mellia (2012) Experimental comparison of neighborhood filtering strategies in unstructured P2P-TV systems. In: Proc. IEEE 12th International Conference on Peer-to-Peer Computing, Tarragona, pp. 13–24.
33. D. Wu, C. Liang, Y. Liu, and K. W. Ross (2010) Redesigning multi-channel P2P live video systems with View-Upload Decoupling. Computer Networks 54(12): 2007–2018.
34. B. Barekatain, D. Khezrimotlagh, M.A. Maarof, H.R. Ghaeini, S. Salleh, A.A. Quintana, B. Akbari, and A.T. Cabrera (2013) MATIN: A random network coding based framework for high quality peer-to-peer live video streaming. PloS ONE 8(8).
35. M. Wichtlhuber, B. Richerzhagen, J. Rückert, and D. Hausheer (2014) TRANSIT: Supporting transitions in Peer-to-Peer live video streaming. In: Proc. IFIP Networking Conference, Trondheim, pp. 1–9.
36. M. Luby, A. Shokrollahi, M. Watson, T. Stockhammer, and L. Minder, Forward Error Correction Scheme for Object Delivery, Internet Engineering Task Force (IETF) Request for Comments: 6330, Aug. 2011.