Multimedia Tools and Applications (2021) 80:20929–20970
https://doi.org/10.1007/s11042-020-10198-9

QoS monitoring in real-time streaming overlays based on lock-free data structures

Franco Tommasi[1] · **Valerio De Luca**[1] · **Catiuscia Melle**[1]

Received: 6 February 2020 / Revised: 7 October 2020 / Accepted: 24 November 2020 /
Published online: 11 March 2021

Abstract
Peer-to-peer streaming is a well-known technology for the large-scale distribution of real-time audio/video contents. Delay requirements are very strict in interactive real-time scenarios (such as synchronous distance learning), where playback lag should be of the order of seconds. Playback continuity is another key aspect in these cases: in presence of peer churning and network congestion, a peer-to-peer overlay should quickly rearrange connections among receiving nodes to avoid freezing phenomena that may compromise audio/video understanding. For this reason, we designed a QoS monitoring algorithm that quickly detects broken or congested links: each receiving node is able to independently decide whether it should switch to a secondary sending node, called "fallback node". The architecture takes advantage of a multithreaded design based on lock-free data structures, which improve the performance by avoiding synchronization among threads. We will show the good responsiveness of the proposed approach on machines with different computational capabilities: measured times prove both departures of nodes and QoS degradations are promptly detected and clients can quickly restore a stream reception. According to PSNR and SSIM, two well-known full-reference video quality metrics, QoE remains acceptable on receiving nodes of our resilient overlay also in presence of swap procedures.

Keywords Real-time streaming · P2P streaming · Peer churning · QoS · Playback continuity · Lock-free

1 Introduction

Peer-to-peer (P2P) streaming has been introduced to distribute live or on-demand multimedia content over the Internet: end systems receiving a video stream simultaneously

✉ Valerio De Luca
valerio.deluca@unisalento.it

Franco Tommasi
franco.tommasi@unisalento.it

Catiuscia Melle
catiuscia.melle@unisalento.it

[1] Department of Engineering for Innovation, University of Salento, Lecce, Italy

upload that stream or some parts of it to other nodes, becoming peers in an application layer structure called overlay. P2P streaming systems can be typically tree-based or mesh-based. Tree-based overlays propagate multimedia contents through a tree graph rooted in a streaming source. On the other hand, mesh-based overlays do not rely on a static graph connecting nodes: the multimedia stream is divided into small temporal sequences, called "chunks", which are exchanged among several nodes. A detailed comparison between these two topology categories can be found in [35], which presents also a survey on other transmission schemes such as scalable video coding [40] and multiple description coding (MDC) [54].

A key factor for P2P streaming is the Quality of Service (QoS), which consists in a set of parameters (such as bandwidth, packet loss, delay and jitter) measuring the overall performance of the service. Real-time scenarios require a low end-to-end delay between the streaming source and any receiving node. A particular subcategory of real-time streaming applications consists in interactive applications, which are the main target of our research work. They impose even more strict requirements on delay and playback continuity, which means the ability to play a stream with no interruption or frame freezing. This is the case, for instance, of e-learning scenarios. A very low delay is a key factor to give users the possibility to ask questions within a few seconds, before the lecturer moves on to a new subject. At the same time, playback continuity is very important to allow a good understanding of a lecture: unfortunately, network congestion and physical link problems may cause interruptions or severe QoS degradations in the stream propagation over the Internet.

In our previous work [47] we presented a survey on some of the most efficient solutions (based on meshes, trees, network coding and MDC) with a specific focus on real-time streaming. In particular, we highlighted that mesh-based protocols are generally not suitable for real-time interactive streaming due to their long and unpredictable delays [17], despite their better resource utilization [28]. A previous work [59] proved that mesh-based overlays, where each node establishes connections with several neighbours, are nearly optimal in terms of peer bandwidth utilization and system throughput even without intelligent scheduling and bandwidth measurement. However, such analysis does not take into consideration the low rate connections across different ISPs and the limitations deriving from the presence of NATs and firewalls.

More generally, meshes are characterized by a tradeoff between control overhead and delay [59]. Nodes need to exchange information about available chunks (the so-called "buffer maps") very frequently to retrieve packets quickly and reduce delay. Unfortunately, this produces a significant increase of the control overhead. Moreover, frequent playback discontinuities caused by missing chunks can severely affect the lecture understanding. Users' disappointment has been proven to increase exponentially with respect to the number of consecutive video frame losses [33]. However, users mainly perceive video freezing during more than 200 ms and frame losses during more than 80 ms [9]. A larger receiving buffer size could help facing with playback discontinuities, but it would also increase the lag with a real-time event.

Also peer churning has serious consequences on playback continuity: peers are not stable during a streaming session, since they join and leave the overlay in an unpredictable manner. Mesh-based overlays exhibit a better resilience in presence of peer churning due to the multitude of neighbour connections from which a peer receives data chunks, but they have also a higher playback delay, which further increases with the network size [58]: this drawback makes them not suitable for interactive real-time applications, which have very strict requirements in terms of playback lag. A typical example of interactive real-time

streaming is synchronous e-learning, where learners should have the possibility to ask pertinent questions in real-time about a topic in a certain moment during the lesson [47]: in this case, besides the end-to-end delay, another important factor is the intelligibility of the speech, which derives from playback continuity.

For this reason, we chose a tree-based overlay to meet the requirements of an interactive e-learning scenario, where students should have the possibility to ask questions in real-time during a lesson. In such overlay, each node forwards packets as they arrive, without any particular scheduling. This approach can reduce the possibility of frame losses, even though the availability of more neighbours in mesh-based overlays can assure a lower loss ratio in dynamic churning conditions [58].

The assumption of our work is that the performance of tree-based overlays can be improved even under dynamic churning by assigning each node a secondary parent in advance (proactive approach). Under such hypothesis, an optimized procedure for a quick switch to the pre-assigned secondary parent is the key performance factor for a fast recovery of the stream reception.

More specifically, this work describes:

- a QoS monitoring algorithm, which is able to quickly detect both QoS degradations and node departures;
- a switch-to-fallback procedure, which allows a receiving node to switch to a secondary relayer as soon as reception problems are detected by the QoS algorithm.

The whole architecture is based on a multithreaded design: we improved the system performance by adopting lock-free data structures, which allows multiple threads a concurrent and safe access to shared data without synchronization. A good responsiveness of the overall system is indeed a key factor to reduce playback discontinuities.

In this paper we are focusing on a single-tree case, but our design could be also extended to multiple tree overlays, where each tree is used to distribute a different description of the same stream generated by multiple description coding. We considered a single-tree overlay as a pilot study, but the experiments we have conducted could provide useful performance insights also for other topologies (such as meshes): our approach based on lock-free data structures can provide significant performance improvements every time there is the need to rearrange the connections among nodes, regardless of the employed overlay topology.

The rest of the paper is structured as follows: Section 2 summarizes the related work; after a brief summary of our previous works in Sections 3.1 and 3.2 extends the above concepts to resilient overlay design; Sections 3.4 and 3.5 describe our QoS monitoring algorithm and the evaluation of QoS parameters; Section 3.6 describes the switch-to-fallback procedure; Section 4 gives details about our lock-free implementation; Section 5 gives some hints about the generalization of the proposed approach to multi-tree overlays; Sections 6 and 7 present the experimental testbed and the related results respectively; Section 8 concludes the paper and gives hints about some possible future developments.

2 Related work

Since we found no other work focusing on the optimization of the swap procedure, in this section we will mention other studies proposing optimized strategies for the overlay construction and maintainance.

Some authors tried to minimize the consequences of a node leaving the overlay. To this aim, they tried to model the behaviour of nodes and base the overlay construction and maintenance on such information. An example is the reconstruction method proposed in [3] to cope with peer churning in tree overlays. It requires each new joining node to know in advance the time it will leave the overlay. Unfortunately, this assumption is not realistic, since the time a user participates to a streaming session is usually unpredictable by the very same user.

Starting from the assumption that "the longer a node stays in the overlay, the longer it would stay in the future" [2], some authors [55, 60] tried to investigate the presence of the so-called "stable nodes" or "long-lived peers". From the point of view of users' behaviour, this phenomenon can be described as a combination of multiple metrics (channel popularity, session duration, online duration, arrival/departure), which are strongly related to environmental factors (day-of-week, time-of-day, channel/content type) and network performance parameters (delay, packet losses, bandwidth, discovery of partner peers, streaming quality, failure rate) [52].

Simulations described in [53] considered a hybrid overlay, where peers with the highest reputation (in terms of bandwidth, participation duration in the video session and locality) are connected close to the nearest landmark nodes, which are nodes assumed to be stable during the whole session, while low reputation peers are grouped into mesh clusters to achieve a better resilience. Results proved such overlay outperforms other hybrid approaches, such as TRMC (Tree, Ring and Mesh- Clusters), MTMC and mTreebone.

The cross layer design described in [39] involved scalable video codec, backup parents and hierarchical clusters. It adopted a hierarchical organization of backup parents to prevent loops in the tree and have higher bandwidth nodes in the upper layers of the tree.

Some multitree construction schemes [57] addressed flash crowd issues by putting near the root new peers with higher bandwidth and longer waiting time.

Several works focused on the construction of disjoint backup paths to minimize the possibility a single-link failure could affect more than one of them. In particular, a combination of capacity and diversity metrics was proposed in [1]. Such work highlighted also the importance of addressing the issue with a global scope rather than independently for each data stream. It analyzed the impact of single and multiple link failures in different scenarios by considering, in particular, the influence of the overlay size.

The study in [25] addressed the problems deriving from the simultaneous departure of several peers in tree-based overlays: simulation results proved the possibility of reducing the reconnection time by more than 400 ms by reserving a small capacity fraction when the parent is selected and keeping an ancestor list at each peer.

VMCast [14] was designed to improve the stability of a tree-based overlay. It exploits multicast virtual machines to distribute multicast data along a stable overlay tree and compensation virtual machines for a further enhancement based on dynamic streaming compensation.

Multiple routing path techniques, combined with video coding schemes such as MDC, are widely adopted to provide fault tolerance in video streaming over MANETs (Mobile Ad Hoc Networks), where frequent topology changes and limited bandwidth can seriously compromise QoS. Some simulations under *ns2* [41] compared the QoS performance of two multipath routing protocols, M-AODV and MDSDV, for MPEG-4 video transmission in MANET networks. However, such analysis does not focus on the performance of the swap process (i.e. the change of a parent node that is forwarding a video to another node) and evaluates only the effects of node mobility from a global point of view.

The Search for Quality (S4Q) algorithm [43] is based on the exchange of stable peer lists among overlay nodes. Each of these lists contains peers experimenting a better QoE, measured as a function of the number of missing video pieces called stress level. Other works described overlay construction strategies designed for QoS optimization.

The structured overlay described in [15] organizes the peers according to a "MinHeap algorithm" based on the round trip time (RTT).

TURINstream [29] tries to overcome the limitations of tree-based overlays: it organizes peers into clusters, which are small sets of fully connected collaborating nodes, to improve bandwidth utilization and resilience to peer churning. Clusters are connected in turn to form up a tree overlay. Playback continuity is guaranteed by the fact that clusters do not leave the overlay tree in case of peer departures and the stream propagation still goes on.

A multi-tree overlay based on MDC is the key of the Dagster system [34]: the authors focused on the construction scheme and on incentive rules that encourage nodes to share their outgoing bandwidth.

The hybrid architecture proposed in [30] exploits the IP network to enhance the quality of a base-layer video stream distributed through DVB-T2. IP multicast traffic is replicated among various P2P high level peers that are responsible for the distribution of the traffic to the peers in the same access network. Information on network resources and topology is periodically updated. By delimiting P2P traffic within small geographical areas, the system is able to mitigate the effects of peer churning, since also dynamical changes are self-contained to such areas.

Another work [13], based on a locality-aware topology-optimizer oracle hosted by the ISP, evaluated the possible benefits deriving from periodic topology optimizations in presence of peer churning.

Some authors [6–8] designed a dynamic QoS architecture for scalable layered streaming over OpenFlow software defined networks (SDNs). They employed network topology/state information and extended their framework to provide end-to-end QoS over multi-domain SDNs [5].

Two other examples of QoS-based dynamic overlay topologies addressed live streaming in VANET (Vehicular Ad Hoc Networks) scenarios [21] and 3D Video Collaborative systems [56]. The former [21] allowed parent switching to improve QoS in terms of packet losses and end-to-end delay. The latter [56] tried to optimize resource utilization and overlay stability under bandwidth constraints. However, none of the mentioned works analyzed in detail the swap process from a parent node to another one and the possible effects on QoS and playback continuity.

Most of the recent works proposed hybrid overlays, combining mesh and tree topologies. In the peer selection strategy described in [4], each node chooses its peers among the ones suggested by a tracker; then, during an adaptation phase, peers can change their positions in the overlay for optimization purposes. Such strategy aims at mitigating the effects of peer churning and takes into account propagation delay, upload capacity, buffering duration and buffering level. It outperforms two older methods presented in [37] and [16]. The FastMesh [37] overlay tried to minimize delay by maximizing power, defined as the ratio of throughput to delay. On the other hand, the Hybrid Live P2P Streaming Protocol (HLPSP) [16] only considered upload capacity for peer selection. The overlay described in [12] is based on redundant trees, where each node forwards to its siblings the video chunks received from its parent.

Our study can be seen as complementary to research work about backup/multiple paths and peer selection strategies. Firstly, it provides peers with a method to monitor their stream reception. Moreover, it analyzes the challenge of changing the sending peer (the relaying node) as quickly as possible to minimize QoS degradation. An efficient swap procedure is a key factor for the performance of an overlay for real-time streaming, regardless of the adopted topologies and peer organizing strategies.

3 Tree-based overlay for real-time streaming

Our work has been partially inspired by switch-trees protocols [18]: they introduced parent switching, which refers to the possibility of nodes to change their parents to reduce the source-node latency or the tree cost. In our design, we extend the necessity of parent switching to all cases of nodes experiencing a bad QoS.

The following special cases of switch-trees algorithms are known in literature [18]:

- switch-sibling, which allows a node to choose its new parent among its siblings (i.e. the nodes receiving from the same parent);
- switch-one-hop and switch-two-hop, which allow a node to choose its new parent among nodes within one hop and two hops from its current parent respectively;
- switch-any, that is the most general case, which allows a node to switch any non-descendant node.

Without loss of generality, our reference scenario focuses on the last case.

Our study about the performance of a swap procedure in tree-based overlays starts from the redesign of an architecture for real-time streaming we described in a previous work [48]. In this section we will focus on a single tree overlay where an entire stream is propagated without any MDC decomposition. We will explain how the proposed approach could be extended also to multi-tree overlays in Section 5.

We call "relaying" the process of propagating in real-time entire audio/video streams over the RTP protocol[1,2] through an overlay tree: some nodes, called relayers (R), recursively forward the streams they are receiving from a video source (S) to other nodes, called hosts (H), which can in turn act either as relayers or as leaf nodes.

In the following section we briefly summarize the working principles of such architecture. Then, we describe in detail our QoS monitoring algorithm and our design based on lock-free data structures, which are the new main contributions of this paper.

3.1 The CHARMS tree-based overlay

To support the development of tree-based overlays, in a previous work we designed ALRM (Application Layer Relaying Module) [46], a library for RTP relaying of multimedia streams over UDP. We used it to implement our own tree-based overlay for real-time streaming [48]. We called it "Cooperative Hybrid Architecture for Relaying Multimedia Satellite Streams" (CHARMS), because it was originally designed with the aim of propagating multimedia streams received via satellite to other sites that are not equipped with a proper receiving antenna. Later, the "satellite" term was dropped since the architecture proved

[1]RTP Topologies, RFC 5117, https://www.ietf.org/rfc/rfc5117.txt
[2]RTP Topologies, RFC 7667, https://www.ietf.org/rfc/rfc7667.txt

effective also with terrestrial Internet sources. In general, relayers are nodes with a large outgoing bandwidth and a good reception from the audio/video stream source, which are able to forward the received packets to a certain number of host nodes. In this sense, relayers could be also nodes receiving multimedia streams from a CDN service. In this way, the platform can be employed as an extension to enable the reception also on nodes that have no subscription to CDN services.

In the CHARMS architecture, we introduced some backup relayers, which we call *fallback* nodes, to allow a recovery of the stream transmission in case of node failures. Each overlay node has a TCP connection with a server, which assigns a relayer to each host and a fallback node that should replace the relayer in case of QoS problems. In a later phase, we decided to improve the efficacy of such mechanism. To better support a quick and smooth switch to a fallback node, we designed an enhanced version of ALRM, named MSRM (Multi-Source Relaying Module) [49]. It allows a host to hold for a short time a connection with two relayers (the fallback node and the relayer with the impaired link) and discards replicated packets. The connection with the old relayer is maintained until the reception from the new relayer (the ex-fallback node) properly starts to avoid any playback interruption.

MSRM is instantiated by the CHARMS overlay manager after the opening of five UDP channels between two peers by means of a NAT traversal procedure[3]: the first four channels are used by MSRM, while the fifth channel is used by the QoS monitoring described in the next sections. Each MSRM instance running on a client manages the relaying or the reception of one stream.

All the internal logic of MSRM is based on lock-free data structures. After this section summarizing our previous work, where we had just introduced lock-free structures [49], we will extend in the next sections their adoption in the whole overlay software architecture: we will describe the implementation of a global QoS monitoring algorithm (which goes beyond the features provided by MSRM, as explained in Section 3.2) and evaluate the achieved performance.

3.1.1 MSRM internal working

MSRM communicates with the CHARMS overlay manager through actions and responses: actions are commands given by the overlay manager to add or drop sources or destinations, while responses are feedbacks (mainly QoS feedbacks) delivered to the overlay manager.

MSRM QoS monitoring algorithm [46] combines audio/video packet loss ratio and jitter retrieved from RTCP packets into a QoS parameter that represents a warning level based on the recent QoS history. It computes the autocovariance $\gamma(x) = E\left[(x_i - \mu_i)(x_{i+1} - \mu_{i+1})\right]$ of n jitter and loss ratio samples x_i by using a Weighted Exponential Average in place of $E[\cdot]$. Then it normalizes such value by the sample variance to get an autocorrelation.

MSRM QoS responses use a three-level scale to express the connection quality of a receiving peer: GOOD CONNECTION, CONGESTED CONNECTION and BAD CONNECTION. The QoS is labeled as CONGESTED or BAD when the warning level exceeds the thresholds reported in Table 12 of Appendix A. The QoS warning level is decreased for low jitter and loss ratio values. It grows up to notify a congestion when the means and autocorrelations of jitter and loss ratio exceed the thresholds in Table 10. It assumes the highest

[3]NAT Behavior Discovery Using Session Traversal Utilities for NAT (STUN), RFC 5780, https://www.ietf.org/rfc/rfc5780.txt

values in case of an interruption in the RTCP packet exchange, which denotes a severe congestion or even a broken link. The ALERT status occurs when mean values for jitter and loss ratio are below the thresholds but the last samples are over the thresholds or when mean values are over the thresholds but not both the autocorrelations are over the thresholds. The warning rates, i.e. the values used to increment or decrement the QoS warning parameters, are reported in Table 11 of Appendix A.

The MSRM framework provides six different implementations for switch-to-fallback operations and Multiple Description Coding (MDC) scenarios in overlay networks for real-time streaming. Only the fifth and the sixth one are based on lock-free data structures (the former for switch-to-fallback and the latter for MDC), while the other ones still use traditional mutual exclusion locks. For this reason, we used the fifth MSRM implementation in our lock-free architecture. Figure 1 depicts a schema of its internal working. It exploits four lock-free queues, which form up the *Queue Buffer* in the figure, and four lock-free hash tables to manage a stream transmission of four substreams (RTP audio, RTCP audio, RTP video, RTCP video) on four UDP channels. In particular, it sets a flag for each received packet into hash tables to avoid collecting double packets during a switch-to-fallback node procedure, when a host is receiving from a new relayer without having yet closed the connection with the old relayer. It uses queues as packet buffers: while some threads store the received packets into queues, other threads concurrently read these queues and relay each packet toward a destination. A particular destination of the stream relaying is represented by the loopback interface of a client node: a multimedia player will be able to read packets arriving on it and reproduce the received stream by opening a SDP file from a local video player.

Also MSRM actions and responses are collected into lock-free queues. A concurrent lock-free read access to QoS responses while they are pushed into the queue is crucial to the efficient execution of the QoS monitoring algorithm. All the other internal structures, such as those representing threads and nodes (sources and destinations), are also based on lock-free queues.

For a comparative performance evaluation of the lock-free design, we implemented also a lock-based version of the CHARMS platform. We based it on the third solution provided by MSRM, since experimental tests proved it is the most efficient among the four lock-based implementations [49]. In this solution, multiple receiving threads store packets arriving from different sources into a joint buffer (Fig. 2). The buffer is implemented as a hash table of packet lists: the hash key is the sequence number for RTP packets, the last sequence number

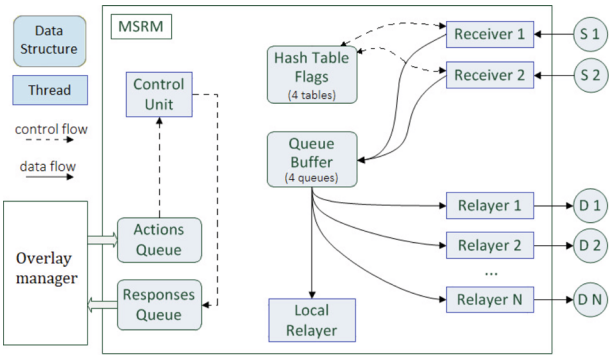

Fig. 1 Internal working of a lock-free MSRM instance handling a stream on a client

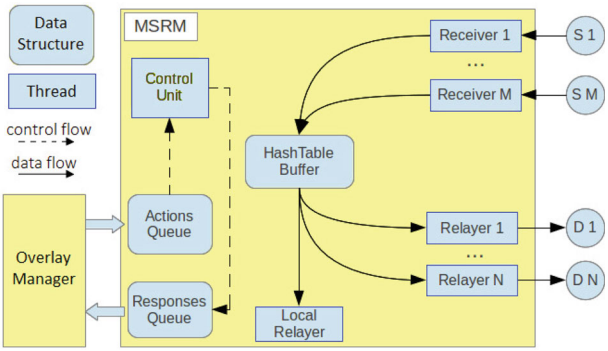

Fig. 2 Internal working of a lock-based MSRM instance handling a stream on a client

for RTCP Receiver Report (RTCP RR) packets and the last RTP timestamp for RTCP Sender Report (RTCP SR) packets. In this way, both read and write operations on the buffer can take place in a constant time.

3.2 Resilient tree overlays

The QoS parameter computed by MSRM gives only warnings about the overall QoS on a receiving node, without detecting which links of the overlay network are responsible for a QoS degradation. The present paper extends our previous work by describing a global QoS algorithm that monitors the entire overlay and takes the proper actions to restore an acceptable QoS in the stream propagation. In the last sections of the paper we will analyze the time spent to detect QoS degradations and perform the swap procedure and will evaluate the effects on video quality according to PSNR and SSIM metrics.

A typical scenario is depicted in Fig. 3, where R and H denote relayer and host nodes respectively. The design of a resilient tree overlay is based on a primary-backup approach: when a host node experiences any problem compromising QoS during a stream reception, it can eventually switch to the fallback node. In this way, the stream reception can go on without the user realizing what happened. Since a periodical check of each connection state performed by a central server would produce a sensible overhead on it, relayer and host

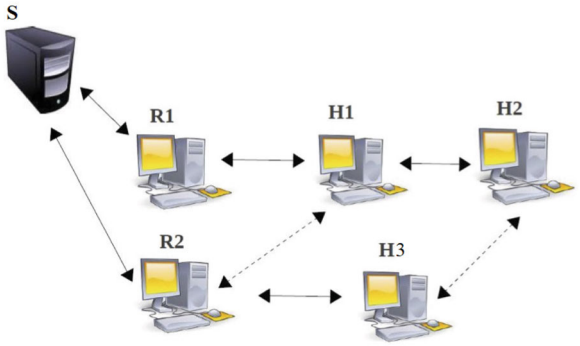

Fig. 3 A relaying scenario

nodes should autonomously monitor connections toward their peers and compute statistic analysis. In case of problems, a client node has the responsibility of informing the server about its intention of changing relayer.

However, a bad QoS could not be directly related to the *H-R* link because it could be also caused by a bad link between video source *S* and *R*. A careful analysis of overlay traffic parameters is required to distinguish between these two situations and choose the consequent actions to take.

Establishing a preliminary connection with the fallback node requires a periodical keepalive action on the five UDP channels, which consists of receiving and sending XML messages between two peers in order to keep the NAT mapping active on the edge routers of the peer local networks: this traffic can be used also to get the QoS information of a peer link based on the measurement of the Round Trip Time (RTT). On the other hand, during the stream relaying performed by MSRM on the first four UDP channels of a *H-R* link, an out of band fifth UDP channel is maintained to measure the RTT through periodical XML message exchanges. RTT data, together with QoS information based on packet losses and jitter, retrieved from RTCP packets analysis provided by MSRM, allow to attribute the responsibility of bad quality to a specific network relaying link and to take the consequent actions. In particular, the packet loss ratio can give insights on the average network congestion level, while a jitter analysis provides information about the transient states and can predict congestion problems before actual packet losses are experienced.

In the scenario depicted in Fig. 3, if H_2 is experiencing a bad transmission quality there are two possible causes:

1. a problem on H_2-H_1 link;
2. a problem on another link of the chain (R_1-H_1 or R_1-S).

These two situations can also coexist. When H_2 detects a bad transmission quality it has to perform one of the following actions:

– in the first case, H_2 must switch to H_3 fallback node after waiting a short time interval (which should assure the problem is not a transient one);
– in the second case, H_2 can wait the problem being solved by the node with a problematic link or switch to the fallback node (eventually waiting for a certain time to make sure the problem is not a temporary one or it has not been quickly solved by the node affected by link congestion).

In order to determine which is the case and thus to make an optimal choice, H_2 needs to directly estimate the link quality toward its relayer.

Accordingly, if *R* experiences a bad QoS it can denounce itself to the server, that will tell all the descendant hosts to switch to the fallback node. Furthermore, *R* can monitor the state of the links towards the host it is serving and inform the server about any performance degradation. Table 1 summarizes the described scenario. The status of *H-R* links is expressed by means of the same three-level scale used in MSRM QoS responses described in the previous section.

In the same way, a node can analyze the QoS of its fallback link and request a new fallback node when the link performance degrades. In order to detect the *H-R* link quality we use keepalive packets to estimate the link round trip time and compare this value with a threshold.

Each node is also able to get QoS information about video source transmission thanks to the RTCP SR packets generated by the source and forwarded to the various *R* and *H* nodes

Table 1 Relayer (*R*) and host (*H*) behaviour in presence of QoS problems

S-R link status	*R-H* link status	*R* action	*H* action
GOOD QUALITY	GOOD QUALITY	None	None
BAD QUALITY	GOOD QUALITY	Self-denounce to the server.	Wait for good quality restoring or switch to the fallback node.
GOOD QUALITY	BAD QUALITY	Server informed about *H*'s bad quality.	Switch to the fallback node.
BAD QUALITY	BAD QUALITY	Self-denounce to the server.	Switch to the fallback node.

along the chain. A node compares its own loss rate and jitter values with those measured by its peer: if values differ, the QoS variation should be attributed to the link connecting the two nodes.

3.3 Host-relayer interaction

The interaction between a host and a relayer could be asymmetric or symmetric. The two schemes are depicted in Fig. 4.

In asymmetric interaction the host takes care of retrieving and analyzing QoS information: it sends keepalive messages as RTT requests to which the relayer replies. Bidirectional

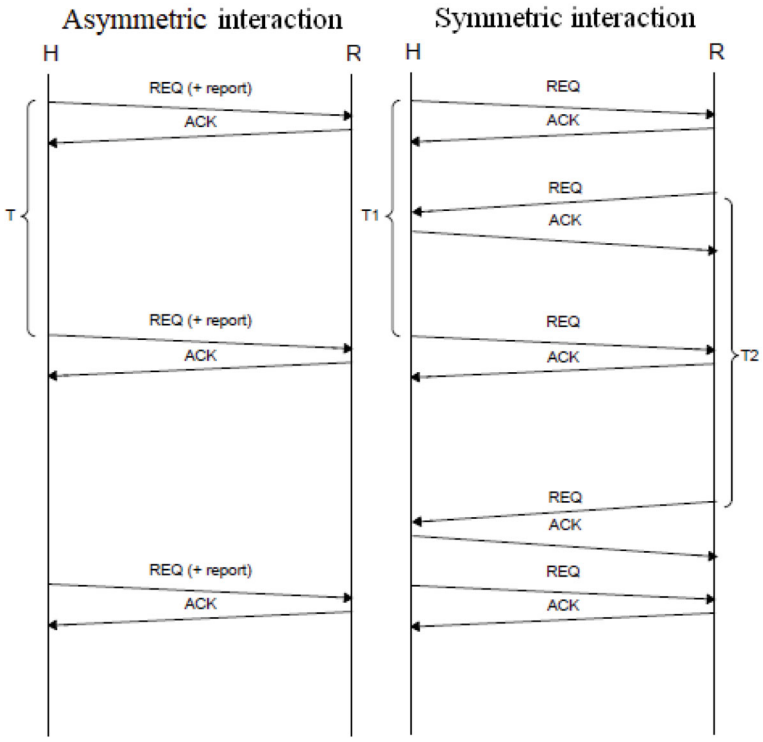

Fig. 4 Asymmetric and symmetric interactions between a host and a relayer

traffic allows the host to estimate the RTT from the timestamps measured for each request-replay messages pair. Besides, if there is an active RTP transmission, quality information about the received stream coming from RTCP RR packets can be added in the request and reply packets. In asymmetric interaction the relayer just acknowledges the received packets: therefore it is not able to compute the round trip time and thus to estimate QoS. In this scenario the host could inform the relayer about the detected QoS by means of reports sent through RTT request packets, but such information could become obsolete as the time interval between retransmissions could last more than a minute.

In symmetric interaction both relayers and hosts send request packets and acknowledge received packets; besides, both the nodes have the responsibility of autonomously estimating the link quality. Therefore, in this scenario nodes really act as peers because they have the same behaviour independently of their relayer or host role in the overlay. In this way the system can better tolerate occasional losses thanks to the fact that both nodes directly and independently estimate the link quality. The main drawback of this scenario is a greater bandwidth consumption, even though request packets have a greater size in asymmetric interactions (where, as said, they also carry report information from the relayer). Since a relayer has to test several different links (as many as the hosts it is serving), it could decrease the sending frequency to reduce the bandwidth consumption. We chose this kind of interaction because it offers more flexibility for future extensions. We adopted a XML format for the messages exchanged between relayers and hosts. The round trip time sample is measured by sending a probe packet to the host, which in turn replies with an acknowledgment packet. The interval between sending and receiving timestamps is used for channel round trip time estimation. The messages between a host and its fallback node contain an identifier that allows matching requests and acknowledgments. On the other hand, messages between a host and its primary relayer contain also four additional parameters about loss rate and jitter for audio and video.

While hosts and fallback nodes exchange messages on five UDP channels to keep all NAT entries active, hosts and relayers exchange messages only on the fifth out of the five channel set because the first four ones carry RTP/RTCP audio/video packets.

3.4 QoS assessment

The assessment of channel quality is different between a host-fallback link and a host-relayer link: on the former the analysis is only based on RTT measures, whereas on the latter it is also based on loss rate and jitter information. When a degradation of the channel quality is detected, a warning level is increased by a value reflecting the extent of the occurred problem. We call it *RTT status* on the host-fallback link and *QoS status* on the host-relayer link. When the warning level reaches a critic threshold, bad quality is signalled for the monitored channel.

In particular, when a relayer leaves the CHARMS overlay, the server (which helps hosts to get in touch with relayers) quickly detects a node departure (thanks to the TCP connection each node has with it, handled by means of I/O multiplexing system calls) and immediately informs the hosts that are receiving streams from it. Even though hosts can detect by themselves relayer departures through the QoS analysis we are describing, this measure allows to speed up the discovery. On the contrary, simple link degradations without relayer departures can be detected only by the QoS monitoring algorithm running on client nodes. However, relayer departures have more severe effects on QoS because they can cause complete interruptions in stream playing. For this reason, they need to be detected more quickly than link degradations.

3.4.1 QoS parameters of the fallback link

In this case, QoS analysis is based on the estimate of a mean RTT and the comparison with a threshold value, which is updated in an adaptive manner on the basis of the minimum detected round trip time. The mean round trip time is computed by updating its previous value through an exponentially weighted moving average:

$$RTT_{AVG} = (1 - \alpha)RTT_{AVG} + \alpha RTT_{SAMPLE} \tag{1}$$

This approach considers also the recent history of RTT values [22]: in this way, the RTT estimate increases mainly in case of congestion rather than for occasional delays caused by transient interferences. In particular we set $\alpha = 0.25$, but we can set a higher value if we want to give more importance to the last samples.

The adaptive threshold is estimated as $RTT_{THRESH} = kRTT_{MIN}$, where we set $k = 2.5$.

The warning level is increased by a certain value when the estimated mean RTT exceeds the threshold. However, also receiving a sample after some retransmissions or not receiving it at all could mean channel congestion: therefore in these cases the warning level has to be increased too.

Since a sudden degradation of the channel may not be quickly detected if the estimated mean RTT is low, the warning level is increased also when the sample RTT is very high compared to the threshold, that is $RTT_{SAMPLE} > KRTT_{THRESH}$, where we chose $K = 4$.

The estimated mean RTT can be compared with the previous sample in order to check whether the mean value is increasing or not. This information can be used to give a lesser weight to cases where the estimated RTT is greater than the threshold but it is decreasing. If none of the alert causes occurs, the warning level is decremented in order to indicate a stabilization of the channel congestion level. Table 13 reports the increments and decrements of the warning level on the fallback link based on mean RTT and sample RTT.

As reported in Table 15, the channel is considered unreliable when the warning level exceeds a threshold, namely BAD_THRESHOLD, equal to 15. We chose also a lower threshold, namely CONGESTED_THRESHOLD, that we set to 9: this identifies a channel congestion level that does not yet require any countermeasure. Furthermore, an upper bound should be imposed on the warning level as soon as the threshold has been exceeded: this is necessary to allow a quick detection of any warning level reduction, which could indicate an improvement on the link state.

3.4.2 QoS parameters of the relayer link

The analysis on the relayer link is not only based on RTT samples but also on loss rate and jitter derived from the last RTCP RR packet sent to the stream source.

The jitter value is not an instantaneous measure, but it is already an estimate on a time interval: therefore it provides meaningful information about the channel quality.

On the contrary, the loss rate value, event though it refers to a time interval, does not consider losses before the last receiver report; furthermore it refers to a time interval that is probably different for each peer. For this reason, a moving average is computed for this value just as for RTT samples. The results are compared to the related values extracted from RTCP RR packets:

– if the difference between the two estimated mean loss rates is low, both the nodes are experiencing the same losses, so any loss is caused by other network links and not by the *H-R* link;
– if the difference between the jitter values is low, both the nodes are experiencing very similar delay variations, so the delay variation caused by the *H-R* link is low.

When these differences exceed the thresholds in Table 15, the warning level is increased as specified in Table 14.

We tried to use the same alert thresholds we set for the fallback node, that is 9 for congestion (i.e. poor QoS) and 15 for bad quality (i.e. very bad QoS) respectively: we noticed that in this way the threshold is reached more quickly because RTT, jitter and loss rate contributions are cumulated. This is a desired behaviour because the control frequency for the active link can be very lower and thus retrieving two samples may require also more than a minute. The alert threshold or the increase values must be chosen on the basis of the time interval between two transmissions. In particular, we considered a time interval between 20 and 30 s.

Furthermore for both links a higher warning level is defined: it can be accessed only when the channel age exceeds the maximum allowed value. This level means no acknowledgment to keepalive packets has arrived for such a long time that NAT mapping is no longer guaranteed. However, a new relayer will be requested before reaching this critical situation.

3.4.3 Retransmissions

The estimated mean RTT is used also to detect the retransmission timeout (*RTO*) as following:

$$RTT_{VAR} = (1 - \beta)RTT_{VAR} + \beta\,|RTT_{SAMPLE} - RTT_{AVG}| \tag{2}$$

$$RTO_{RTT} = RTT_{AVG} + 4RTT_{VAR} \tag{3}$$

$$RTO = max(RTO_{RTT}, RTO_{MIN}) \tag{4}$$

We set a lower bound on the *RTO* value to $RTO_{MIN} = 1.5$ sec to avoid too frequent retransmissions and $\beta = 0.25$.

We did not implement a back-off technique for the retransmission timer like the one in the TCP protocol because we do not need a reliable transmission but simply some information about channel quality. A missed reply after a scheduled timeout (which however is never less than 1.5 s) is enough to assign the channel a bad quality rate.

QoS information (represented by RTT status on the fallback link and by QoS status on the relayer link) is also used to limit the number of possible retransmissions when RTT request packets are not acknowledged to avoid further traffic on a heavily congestioned link. The maximum number of retransmissions is decreased if *rtt_status* is greater than some thresholds, as shown in Table 16.

3.4.4 Actions to be taken after QoS evaluation

When a relayer client gets a BAD QUALITY QoS response from its MSRM module, it reports the problem to the server, that consequently will label it as a bad relayer and will avoid to assign it to hosts requesting streams. When the number of congested and bad quality

links between a relayer and the hosts served by it exceeds a threshold, the relayer asks the central server to take load balancing policies for a better distribution of hosts among relayers. We set the load balancing threshold to 3 host links.

On the other hand, once a host client has the information about the links toward the relayer and the fallback nodes, it can combine these values with the QoS data about the RTP stream extracted from the MSRM module. When a host client experiences a bad QoS, it is able to attribute the responsibility to the relayer link or other network links: in this way it can take the best decision. Furthermore, it can compare the relayer link status with the fallback link status to detect whether it is worth performing a relayer swap. Besides, regardless of the experienced QoS, it can check the fallback link status and ask the server a new relayer when the link quality is not good. The actions taken by a host based on the status of the relayer link and of the fallback link are detailed in Appendix B.

3.5 QoS monitoring module

The QoS monitoring module is based on three types of threads running on each *R* and *H* client:

- a *peer keepalive* thread, which performs a RTT-based check of the link state on the five UDP channels connecting the peers; this thread runs before the actual stream relaying process starts;
- a *mono keepalive* thread, which monitors the link between the host and its primary relayer during the stream relaying by checking the QoS responses provided by MSRM and the RTT measures on the only fifth UDP channel (hence "mono");
- a *QoS check* thread, which collects the information on the link analysis provided by the other threads for a specific stream and contacts the server if necessary; it periodically updates the number of good, congested and bad links for a node relaying the stream and the relayer, fallback and MSRM status for a node receiving the stream.

As soon as a five channels UDP connection is established between a *R* and a *H* client, a *peer keepalive* thread is launched on both the peers. Then, if *R* is designed as the primary relayer for *H*, such thread is replaced by the stream relaying on the first four channels (RTP audio, RTCP audio, RTP video, RTCP video) and a *mono keepalive* thread on the fifth channel. On the contrary, if *R* has been chosen as a fallback node for *H*, the *peer keepalive* thread goes on with RTT message exchanges to keep the five channels UDP connection active and at the same time to monitor the link state between the peers.

The time line is splitted into multiple slots of T_c seconds within which a peer can wait for an acknowledgement to a keepalive packet even after trying some retransmissions.

$$T_c = \left\lceil \frac{2 * PACKET\_SIZE}{bandwidth} \right\rceil \tag{5}$$

By considering the different size of the two keepalive messages, we set $PACKET\_SIZE = 2280$ bits for the five channels handled by the *peer keepalive* thread and $PACKET\_SIZE = 2976$ bits for the single channel handled by the *mono keepalive* thread. We chose 300 bps as a value for the *bandwidth* limit.

We define the channel age as the number of slots passed since the last acknowledgement. We compute the maximum age as:

$$MAX\_AGE = \left\lfloor \frac{NAT\_TIME}{T_c} \right\rfloor - 1 \tag{6}$$

Since the RFC 4787[4] defines a timeout for a NAT entry should be greater than 2 min, we set $NAT\_TIME = 120$ s. This means a keepalive packet should be sent at least every 120 s on each channel. Moreover, a bidirectional packet exchange is necessary to maintain NAT entries also for those natting systems that distinguish between outgoing and ingoing traffic.

For the *peer keepalive* thread we use two FIFO queues to manage the scheduling of the slots assigned to the channels: the thread starts from a main queue containing the indices of all the five channels and an empty recovery queue, where channels receiving no acknowledgement are inserted. When an acknowledgement is received for a keepalive message, the channel is reinserted in the main queue, otherwise it is inserted in the recovery queue at the end of the time slot. If no acknowledgement has been received within the slots of the channels, when the maximum age is reached, a last attempt is done with the first element of the recovery queue. If the attempt succeeds, the extractions from the recovery queue go on, otherwise the NAT mapping is considered as not assured anymore. If some recovery succeeds before the first element in the recovery queue reaches the maximum age, the algorithm goes on until the recovery queue is empty. This algorithm is represented by pseudocode 1.

The *mono keepalive* thread executes a similar algorithm on a single UDP channel (thus without the need for queues to handle the channels). Moreover, the XML message sent by the *mono keepalive* thread to measure the RTT contains also the QoS parameters provided by MSRM (video loss rate, jitter loss rate, audio loss rate, audio jitter).

The diagrams in Figs. 5 and 6 show how the two keepalive threads work. The most important operations, in the highlighted boxes, are:

- **Check if peer is ready until recv ok**: a node (*R* or *H*) periodically sends messages to its peer and waits for a reply to detect when the peer is ready for reception;
- **Get RTT/QoS sample**: a RTT or QoS request packet is created and sent on a particular channel; then the process waits for an ack on that channel until a timeout expiration; if an out of order packet arrives the process discards it and returns to wait for the residual time;
- **Ack incoming RTT/QoS until timeout**: it listens on all the five channels of the fallback link and on the fifth channel of the relaying link and acknowledges all the arriving request packets;
- **Update RTT/QoS status**: an index of the link congestion level is updated according to the collected samples; pseudocode 2 describes the algorithm for updating the fallback *RTT status*, while pseudocode 3 describes the algorithm for updating the relayer *QoS status* (thresholds for loss rate and jitter are reported in Table 15, while warning increments for *RTT status* and *QoS status* are reported in Tables 13 and 14 of Appendix A respectively);
- **Update RTT/QoS link status**: the link status is evaluated according to the congestion index updated in the previous step; these data are made available to a thread that sends messages to the server according to the estimated QoS level.

[4]Network Address Translation (NAT) Behavioral Requirements for Unicast UDP, RFC 4787, https://www.ietf.org/rfc/rfc4787.txt

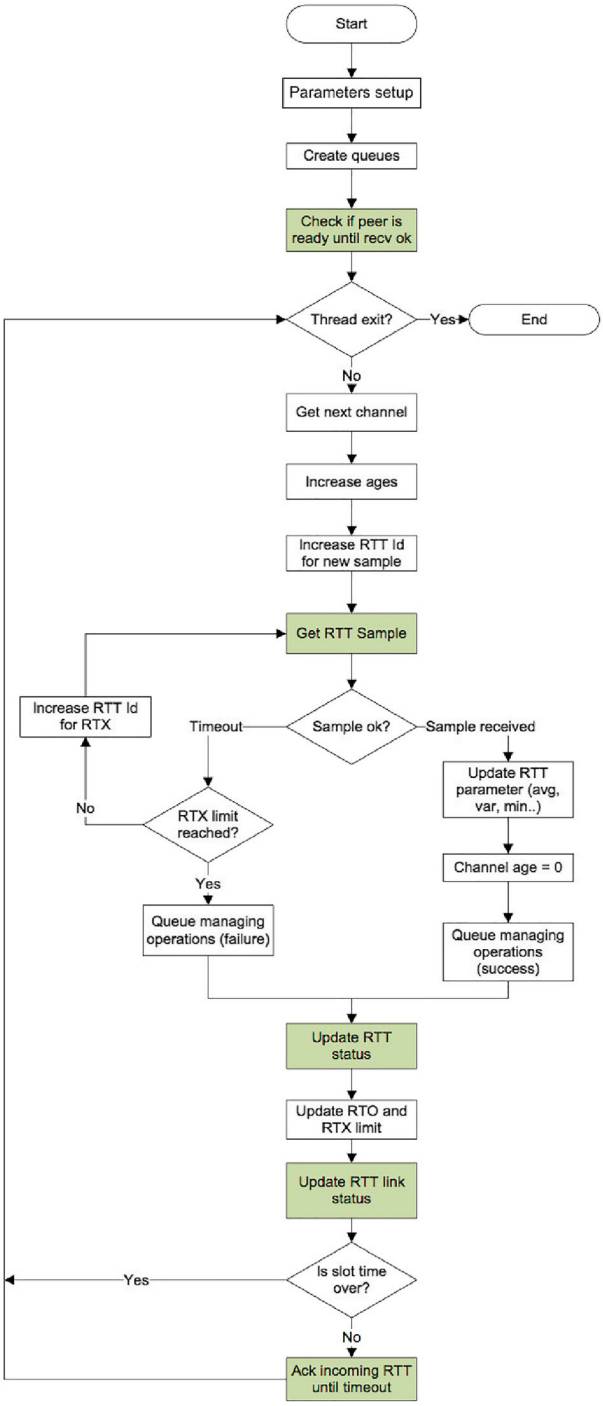

Fig. 5 Flowcharts for the threads on the fallback link (*peer keepalive* thread)

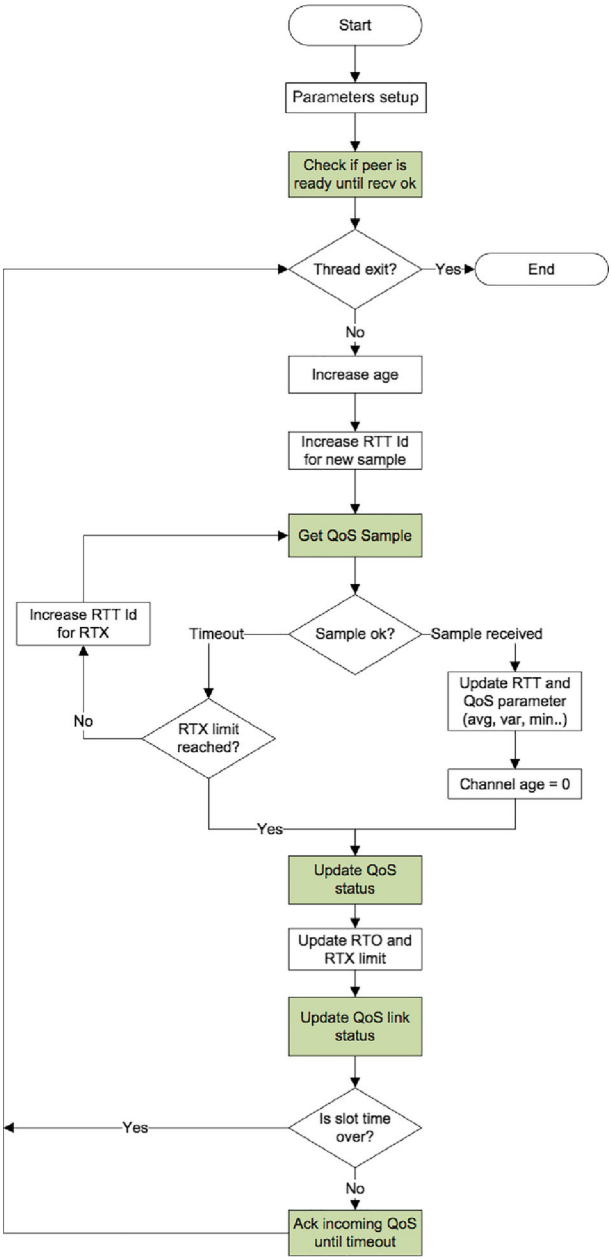

Fig. 6 Flowcharts for the threads on the relayer link (*mono keepalive* thread)

Algorithm 1 *Peer keepalive* thread.

```
1: begin
2:    main_queue = {0,1,2,3,4};
3:    pop_from_main_queue = TRUE;
4:    current_channel = 0;
5:    channel_ages[5] = {0,0,0,0,0};
6:    while (activeThread)
7:       if (pop_from_main_queue) then
8:          current_channel = FIFO_pop(main_queue);
9:       else
10:         current_channel = FIFO_oldest(recovery_queue);
11:      end if
12:      do
13:         sendPacketForRTTMeasurement(current_channel);
14:         sample_result = waitForResponse(current_channel);
15:      while (reception is ok && retransmissions are over);
16:      switch (sample_result)
17:         case SAMPLE_RECEIVED:
18:            updateRTT();
19:            channel_ages[current_channel] = 0;
20:            if (!pop_from_main_queue) then FIFO_pop(recovery_queue);
21:            FIFO_push(channel_index+current_channel, sizeof(int), main_queue);
22:         case SAMPLE_TIMEOUT:
23:            if (pop_from_main_queue) then FIFO_push(channel_index+current_channel,
   sizeof(int), recovery_queue);
24:            if (main_queue is empty) then pop_from_main_queue = FALSE;
25:            if (recovery_queue is not empty && chan-
   nel_ages[FIFO_oldest(recovery_queue)] >= max_age) then pop_from_main_queue =
   FALSE;
26:      end switch
27:      updateRTTstatus();
28:      updateRTO();
29:      updateMaxRTX();
30:      updateRTTlinkStatus();
31:      while (!slotTimeout)
32:         if (ackIncomingRTTUntilTimeout() == SAMPLE_TIMEOUT) then break;
33:      end while
34:   end while
35: end
```

3.6 Switch-to-fallback procedure

Each host node is able to perform switch-to-fallback operations atonomously. The server intervenes just to provide a new fallback node once the swap procedure has been completed. In particular, there are two feasible scenarios for the swap of a host node (H):

– session with the relayer (R) is definitively closed;
– relayer (R) and fallback node (F) are exchanged.

Pseudocode 5 represents the algorithm implemented by the main QoS monitoring thread. A separate instance of the thread is executed for each stream received and/or relayed to other nodes.

Algorithm 2 Update RTT status.

```
1:  begin
2:    RTT_THRESH = kRTT_MIN; //k = 2.5
3:    if (sampleReceivedWithinTimeout) then
4:      if (number_of_retransmissions > 0) then
5:        rtt_status += MEDIUM_WARNING;
6:      else
7:        if (RTT_AVG > RTT_THRESH) then
8:          if (RTT_AVG < RTT_PREV) then
9:            rtt_status += LOW_WARNING;
10:         else
11:           rtt_status += MEDIUM_WARNING;
12:         end if
13:       else if (RTT_SAMPLE > KRTT_THRESH) then
14:         rtt_status += LOW_WARNING;
15:       else
16:         rtt_status -= NO_WARNING;
17:         if (rtt_status < 0) then rtt_status = 0;
18:       end if
19:     end if
20:   else //timeout
21:     if (sampleAvailable) then
22:       rtt_status += MEDIUM_WARNING;
23:     else
24:       rtt_status += HIGH_WARNING;
25:     end if
26:   end if //sampleReceivedWithinTimeout
27:   if (rtt_status > RTT_MAX) then rtt_status = RTT_MAX;
28: end
```

The *switchToFallback()* call in the pseudocode represents the swap procedure with the complete interruption of the *R*-*H* session.

The swap procedure requires the following messages to be exchanged:

- SWAP is sent by *H* on all the five UDP channels to ask *F* to start the stream relaying. When *F* receives this message, it replies with a SWAP_ACK;
- TERMINATE_SESSION_FOR_SWAP is sent by *H* to *R* to close the UDP channels and stop receiving the stream from it. When *R* receives this message, it replies with a TERMINATE_SESSION_FOR_SWAP_ACK and stops the stream relaying towards *H*;
- SWAP_EXECUTED is sent by *H* to inform the server about the completed swapping process;
- REQUEST_NEW_FR is sent by *H* to inform the server that the stream reception from *R* has been stopped. When the server receives this message, it assigns a new fallback node to *H*.

Algorithm 3 Update QoS status.

```
1: begin
2:   RTT_THRESH = kRTT_MIN; //k = 2.5
3:   qos = 0
4:   if |myVideoLossRate − peerVideoLossRate|> LOSS_RATE_THRESHOLD
     then qos++;
5:   if |myVideoJitter − peerVideoJitter|> JITTER_THRESHOLD then qos++;
6:   if |myAudioLossRate − peerAudioLossRate|> LOSS_RATE_THRESHOLD
     then qos++;
7:   if |myAudioJitter − peerAudioJitter|> JITTER_THRESHOLD then qos++;
8:   switch (qos)
9:     case 1: qos_status + = LOW_WARNING;
10:    case 2: qos_status + = MEDIUM_WARNING;
11:    case 3: qos_status + = MEDIUM_WARNING;
12:    case 4: qos_status + = HIGH_WARNING;
13:  end switch
14:  if (sampleReceivedWithinTimeout) then
15:    if (number_of_retransmissions > 0) then
16:      qos_status + = HIGH_WARNING;
17:    else
18:      if (RTT_AVG > RTT_THRESH) then
19:        if (RTT_AVG < RTT_PREV) then qos_status + = LOW_WARNING; else
       qos_status + = MEDIUM_WARNING;
20:      //K = 4
21:      else if (RTT_SAMPLE > KRTT_THRESH) then
22:        qos_status + = LOW_WARNING;
23:      else if (qos == 0) then
24:        qos_status + = NO_WARNING;
25:        if (qos_status < 0) then qos_status = 0;
26:      end if
27:    end if
28:  else //timeout
29:    if (sampleAvailable) then
30:      qos_status + = HIGH_WARNING;
31:    else
32:      qos_status + = VERY_HIGH_WARNING;
33:    end if
34:  end if //sampleReceivedWithinTimeout
35:  if (qos_status > QOS_MAX) then qos_status = QOS_MAX;
36: end
```

As soon as *H* receives the SWAP_ACK message, it knows *F* is ready to relay the stream: it stops the *peer keepalive* thread and then it replaces the old *R* source with the new *F* source. Only at this stage it is worth sending the TERMINATE_SESSION_FOR_SWAP message to *R*, because stream reception is guaranteed: in this way the lag in the stream playing is reduced and the swapping process is very fast.

The inversion process between *R* and *F* is activated when the QoS estimated on *R* link is not bad but it is worse than the one estimated on *F* link. It is similar to the complete

interruption, with only a little difference: resources are not deallocated and *mono keepalive* thread is replaced by *peer keepalive* thread. In the same way, on *H*, once the acknowledgement from *R* has been received, *peer keepalive* thread is launched; then SWAP_EXECUTED and INVERSION_EXECUTED messages are sent to the server to notify the completion of the operations.

Algorithm 4 Update RTT/QoS link status.

```
LINK_CONGESTION = 1
LINK_BAD_QUALITY = 2
LINK_NAT_ALERT = 3
 1: begin
 2:   if (rtt_age <= max_age) then
 3:     if (rtt_status < CONGESTED_THRESHOLD) then link_status = LINK_OK;
 4:     else if (rtt_status < BAD_THRESHOLD) then link_status=
    LINK_CONGESTION;
 5:     else link_status = LINK_BAD_QUALITY;
 6:   else link_status = LINK_NAT_ALERT;
 7:   end if //rtt_age <= max_age
 8:   if (isRelayer && link_status ! = prev_link_status) then updateGoodCongested-
    BadLinks();
 9:   end if
10: end
```

4 Lock-free design and implementation

Both the MSRM library and the whole CHARMS application were developed in C with Unix system calls (SUSv3 specifications).[5] They rely on POSIX threads to perform multiple operations concurrently and on lock-free data structures.

Mutual exclusion locks (mutex) are widely used in concurrent programming to synchronize multiple threads concurrently accessing to some shared data. They preserve data integrity, consistency and coherence by serializing concurrent read/write and write/write operations. Despite such obvious benefits, possible races for lock acquisition among different threads may cause sensibile performance degradations. Moreover, such coarse-grained locks often serialize also non-conflicting operations. The consequence is a lower level of concurrency and also a poor scalability in presence of high number of locks and several concurrent threads [11]. For these reasons, we improved the concurrent multithreaded design of the QoS monitoring algorithm by adopting lock-free lists, queues and hash tables, which allow safe concurrent accesses to data with no need for synchronization. Lock-free data structures are based on built-in functions for atomic memory access: in our implementation we used built-in functions provided by the *gcc* compiler,[6] which "are intended to be compatible with those described in the Intel Itanium Processor-specific Application Binary Interface".[7]

[5]The Single UNIX Specification, Version 3. http://www.unix.org/version3/

[6]Built-in functions for atomic memory access. https://gcc.gnu.org/onlinedocs/gcc-4.4.3/gcc/Atomic-Builtins.html

[7]Intel Itanium Architecture Developer's Manual, Vol. 3. https://www.intel.com/content/www/us/en/processors/itanium/itanium-architecture-vol-3-manual.html

Algorithm 5 QoS monitoring thread running on a node for each received/relayed stream.

LINK_UNDEFINED = −1
LINK_OK = 0
LINK_CONGESTION = 1
LINK_BAD_QUALITY = 2
LINK_NAT_ALERT = 3
NODE_THRESHOLD = 3
MSRM_CONGESTED_MSG_CODE = 1
MSRM_BAD_MSG_CODE = 2

isRelayer = 1 if the node is relaying the received stream to ther nodes (0 otherwise)
isHost = 1 if the node is receving the stream from another node and not from the video source (0 otherwise)
R_rtt = round trip time on the *R-H* link
F_rtt = round trip time on the *F-H* link

```
1: begin
2:   while (ActiveQoSControlForTheStream)
3:     if (isRelayer) then
4:       if (msrm_quality == MSRM_BAD_MSG_CODE) then sendMessage-
   ToServer(SELF_DENOUNCE);
5:       if (congested_links + bad_links > NODE_THRESHOLD) then sendMessage-
   ToServer(LOAD_BALANCING);
6:     end if //isRelayer
7:     if (isHost) then
8:       if (msrm_quality == MSRM_CONGESTED_MSG_CODE) then
9:         if (F_link_quality == LINK_UNDEFINED) then
10:          sendMessageToServer(REQUEST_FALLBACK);
11:        else
12:          if (R_link_quality == LINK_OK) then
13:            //Congested S-R link, good R-H link
14:            if (F_rtt < R_rtt) then
15:              //Fallback link is better than relayer link
16:              swapToFallback();
17:            else if (F_link_quality > LINK_OK) then
18:              //Congested fallback link
19:              sendMessageToServer(REQUEST_FALLBACK);
20:            end if //F_rtt < R_rtt, F_link_quality > LINK_OK
21:            //if the fallback link is not better than the relayer link
22:            //(i.e. if F_rtt > R_rtt), no swap is executed since
23:            //in this case the congestion reported by MSRM library
24:            //is caused by reception problems of the R node
25:          else if (R_link_quality == LINK_CONGESTION) then
26:            if (F_rtt < R_rtt) then
27:              //The fallback link is better than the relayer link
28:              switchToFallback();
29:            else
30:              sendMessageToServer(REQUEST_FALLBACK);
31:            end if //F_rtt
32:          else if (R_link_quality < LINK_CONGESTION) then
33:            if (F_link_quality < LINK_BAD_QUALITY) then
34:              //The fallback link is better than the relayer link
35:              swapToFallback();
36:            else
37:              sendMessageToServer(REQUEST_RELAYER);
38:            end if //F_link_quality < LINK_BAD_QUALITY
39:          end if //R_link_quality
40:        end if //F_link_quality == LINK_UNDEFINED
41:      end if //msrm_quality
42:    end if //isHost
43:  end while
44: end
```

By exploiting a hardware native support provided by modern multiprocessor architectures, an atomic instruction handles read-modify-write operations through a sort of "implicit lock", which allows a finer grain synchronization and thus a higher level of concurrency [42]. Moreover, some empirical tests proved even in non-multiprogrammed environments lock-free hash tables performance is not worse than that of the most efficient traditional hash tables. The models described in [31, 42] and [51], which inspired the lock-free implementations adopted in our architecture, benefit from the livelock-freedom property, which assures that, if a thread is active, some thread (not necessarily the same one) will complete its operation in a finite number of steps. On the other hand, livelocks refer to situations when threads continue their operations forever without making any progress [45].

Besides lock-free data structures, non-blocking programming can be based on wait-free data structures [27]. Herlihy et al. [20] highlighted the difference between a lock-free and a wait-free data structure: while the former ensures *some* process makes progress in a finite number of steps, the latter ensures *each* process makes progress in a finite number of steps.

Due to the higher complexity of wait-free data structures, which exploit sophisticated progress assurance algorithms [10], we chose lock-free data structures for our application. In particular, we used the lock-free hash table described in [42] and lock-free lists and queues provided by the RIG lock-free open source library,[8] which exploits hazard pointers to reclaim memory for arbitrary reuse [32].

QoSLinkStateManager instances, which are handles containing data of QoS check threads, and MSRM instances are stored into lock-free hash tables, where the keys indexing them are the identifiers of the streams they deal with.

A keepalive thread can be univocally identified by a relayed/received stream and by the peer that is receiving from/relaying to the node. For this reason, each *QoSLinkStateManager* instance contains four hash tables, where *peer keepalive* and *mono keepalive* data handles are indexed by the identifiers of the peers that are in contact with the node. In particular, keepalive handles related to stream relaying and stream receiving activities are stored into separate tables, namely *peer_keepalive_table_R/mono_keepalive_table_R* and *peer_keepalive_table_H/mono_keepalive_table_H*.

Communications between two threads are based on some shared variables (flags): a thread updates a flag value to trigger some action within another thread that periodically checks the same flag. For instance, when the main thread of a *H* client sets a particular flag inside a *peer keepalive* data handle, the corresponding *peer keepalive* thread breaks the cycle that manages RTT monitoring toward a fallback node *F*. Then it starts the switch-to-fallback procedure, by sending the proper message to *F* on the five UDP channels, and terminates. We used the built-in functions for atomic memory access provided by the *gcc* compiler to update and read flag values. In this way, we can avoid race conditions between reads and updates performed by concurrent threads without the need of mutex locks. In a similar way, all the other shared variables are accessed by more concurrent threads by means of built-in atomic functions.

We used atomic variables also in the conditions of the loops performed by keepalive and QoS check threads: in a lock-based design the need to acquire a lock before modifying the value of such shared variables can cause a delay in thread termination, which could have a performance impact during a swap procedure.

[8] RIG lock-free library. https://github.com/llongi/rig

In particular, values of shared variables are set by means of the *__sync_val_compare_and_swap* function, which atomically compares and swaps the current value of a variable with a new one.

Increments of shared variables are computed safely by means of the *__sync_add_and_fetch* function, which atomically sums a value to a variable and sets the new computed value into that variable. The same atomic function can be used also to implement lock-free read operations safely: by passing 0 as a value to sum, the function becomes a simple read of the current value of a variable.

5 From single tree to multi-tree overlay

The approach described in Section 3.2 can be generalized to multi-tree overlays, where the stream is decomposed into multiple descriptions or substreams and each node receives a certain number of substreams from various other nodes. In this case, each substream propagation would be monitored independently by a different instance of the same QoS/RTT monitoring algorithm. Even though a node can still receive a stream in presence of interruptions or QoS degradations affecting only some substreams, a quick switch-to-fallback procedure could avoid a sensible QoE degradation caused by a reduced number of received descriptions.

6 Experimental tests

To have an idea of the system performance in a real-world scenario, we carried out the following preliminary test: we delivered a video stream from a source in the GARR network via terrestrial unicast to a *R* node in the GARR network, which in turn relays it to a *H* node in a Telecom ADSL network. Table 2 reports the minimum and maximum delay and jitter measured for *R* and *H*.

However, the main goal of this study is the assessment of the efficiency of the swap procedure. To this end, we considered a wired local network with a star topology, where we disconnected overlay nodes to emulate peer churning and used the *NetEm* emulator to emulate network problems that can rise in a real-world scenario.

We measured some temporal parameters to evaluate the effectiveness of the implemented method. They consist in the time spent by the system:

- to detect a node has left the overlay, namely T_{Rleft};
- to detect a QoS degradation on a *R*-*H* link (we focused on bandwidth narrowing), namely T_{QoS};
- to perform the actual switch-to-fallback operation, namely T_{swap}, once one of the above problems has been detected;

Table 2 Delay and jitter in a real-world scenario

	Network	Delay	Jitter
Video source	GARR	–	–
R	GARR	0.012–0.179	0.0–0.127
H	Telecom ADSL	0.056–0.345	−0.001–0.214

– to activate the stream reception from the new source, namely $T_{restore}$ (which is just a part of the whole swap procedure time).

T_{Rleft} does not match exactly the time during which the stream reproduction is stopped, since the buffer of 50 packets used for local playing on the H nodes partially mitigates the effect of packet losses. Moreover, we should point out the exchange of SWAP and SWAP_ACK messages between the host and the fallback nodes theoretically depends on network conditions. However, since our algorithm triggers the swap procedure only when the host-fallback link is in a good state, we can assume the time spent for such small message exchange to be almost negligible.

From the times defined above we derived the two most important temporal parameters for QoS:

– $T_{blank} := T_{Rleft} + T_{restore}$, which represents the time during which the stream reception is stopped because R has left the overlay but reception from F has not started yet;
– $T_{weak} := T_{QoS} + T_{restore}$, which represents the time during which the stream reception is affected by packet losses due to a bandwidth limitation on the R-H link.

We tested these performance parameters on different multi-core Linux machines equipped with Linux kernel 4.12.14 (on the openSuSE Leap 15.1 distribution) and *gcc* compiler 7.4.0 respectively: they both provide native support to the *compare-and-swap* operation, which is the basis of the lock-free data structures we used in the procedure implementation.

Table 3 summarizes the hardware features of the machines involved in our testbed.

Table 3 Machines employed in the two tests

	Processor	Cores/ threads	Cache	Memory
1	AMD Sempron 64 3200+ 1.8 GHz	1/1	128 KB L1 128 KB L2	2 GB
2	Intel Core i5-430M 2.26-2.53 GHz	2/4	3 MB SmartCache	8 GB
3	Intel Core 2 Quad Q6600 2.4 GHz	4/4	8 MB L2	8 GB
4	AMD A10-9600P 2.3-3.2 GHz	4/4	2 MB L2	8 GB
5	Intel Core i7-4960X 3.6-4.0 GHz	6/12	15 MB SmartCache	64 GB
6	Intel Core i7-3610QM 2.3-3.3 GHz	4/8	6 MB SmartCache	16 GB
7	Intel Core i5-3210M 2.5-3.1 GHz	2/4	3 MB SmartCache	8 GB

1: server

2: streaming source

3, 4: R clients

5, 6, 7: H clients

We launched a client R_1 in R mode and three clients in H mode. We disabled for the H nodes the possibility to forward to other nodes the stream they would receive: in this way, we were sure all the three H nodes would be attached to R_1, which would perform a unicast relaying of the stream towards each H node.

Then we launched another client R_2 in R mode as a fallback node for the H nodes. We synchronized the clocks of all the overlay clients through the NTP protocol.

After this first overlay construction, we started a multicast video session involving R_1 and R_2 by means of a modified version of *EvalVid*'s *mp4trace*[9] tool [24], where we added the delivery of RTCP SR packets. We used a MP4/H.264 video file of 20 min and 50 s encoded with the *x264* utility at a constant bitrate of 500 kbps, a frame rate of 24 fps, a key frame every 24 frames and a resolution of 704x480. We set the size of RTP Maximum Transmission Unit (MTU) to 1024 bytes to keep the overall size of the link MTU (including UDP and IPv4 headers) below 1500 bytes and avoid packet fragmentation. We considered only I frames and P frames in the encoding of our video, since B frame losses usually have a negligible impact on the perceived quality [44]. We set the video buffer size of R and H nodes to a maximum of 50 RTP video packets.

We performed two tests to assess the effectiveness of the swap procedure in the case of nodes leaving the overlay and in the case of bandwidth narrowing respectively.

In the former, we alternately disconnected one node (R_1 or R_2) during the streaming session to make all the H clients perform a switch to the other R node. After each swap process we restarted the disconnected R, which was reassigned as a fallback node. In this way, we studied the swap process of the H nodes from R_1 to R_2 and from R_2 to R_1.

In the latter, we limited with *NetEm* [19, 23, 38, 50] the outgoing bandwidth of one node between R_1 and R_2 alternately to produce a QoS degradation and a consequent switch of the H nodes to the other R node. *NetEm* is a Linux network emulator based on a queuing discipline implemented as a kernel module between the protocol output and the network device. It can reproduce network characteristics such as bandwidth constraints, packet losses, packet reordering, delay and jitter. In a previous work [50], we used *NetEm* to emulate isolated packet losses to study the effects on some QoE video metrics. On the contrary, in this test scenario we chose to emulate bandwidth bottlenecks, which cause various impairments (delay, packet loss, jitter) on a longer term, because our QoS monitoring algorithm considers the recent QoS history and not isolated phenomena. Indeed, some experiments [38] proved *NetEm* accuracy in emulating sequences of contiguous and correlated losses (burst losses) is poor. Moreover, bandwidth bottlenecks can encompass a combination of multiple QoS impairment factors, such as packet losses, delay and jitter.

The aim of this experiment was to measure the time spent by the H nodes to detect the bad quality on the R-H links. For a constant bitrate B, each R node should have at least an outgoing bandwidth equal to NB to provide an efficient stream relaying towards N H nodes. For this reason, we alternately limited the outgoing bandwidth of R_1 and R_2 to $\frac{NB}{2}$ to produce a sensible QoS degradation. We executed the two tests for $N = 3$ H clients.

The *etmp4* utility, included in the *EvalVid* framework [24], allowed us to compute frame losses [50] starting from data collected through the *tcpdump* command on the streaming source and on the H receiving nodes. Moreover, it reconstructed video traces as they were seen on the host nodes, affected by artifacts caused by packet losses.

[9] EvalVid with GPAC - Usage. http://www2.tkn.tu-berlin.de/research/evalvid/EvalVid/docevalvid.html

7 Results

During the streaming test conducted with the settings described in the previous section, we measured the time T_{Rleft} to detect a relayer departure, the time $T_{restore}$ to restore the stream reception, the time T_{blank} during which the stream reception is stopped and the time T_{swap} to perform the switch-to-fallback procedure. Times measured for each switch-to-fallback operation and the percentages of I and P lost frames on the three hosts are reported in Table 4. Times measured during a run of a lock-based version of the application are reported in Table 5: this version is based on the use of mutual exclusion (mutex) locks of POSIX threads to synhcronize the access to traditional data structures. By comparing the two tables we can notice a significant difference in T_{Rleft}, $T_{restore}$, T_{blank} and T_{swap} values. The worse performance of the lock-based version is caused by the contention for the acquisition of locks between two different threads. In particular, the time for restoring the stream reception is the parameter that highly benefits from the lock-free design, since it is reduced even by three orders of magnitude compared to the value in the lock-based version. On the other hand, the total time T_{swap} spent to perform the switch-to-fallback procedure is the least influenced by the lock-free design. However, the time $T_{blank} = T_{Rleft} + T_{restore}$ during

Table 4 Times, expressed in seconds, elapsed during the six switch-to-fallback node operations and loss percentages for *I* and *P* frames

	Relayer disconnection scenario (lock-free version)				
Host	T_{Rleft} (s)	$T_{restore}$ (s)	T_{blank} (s)	T_{swap} (s)	Lost frames (%)
i5-3210M	2.824	0.008	2.832	17.378	I: 4.71
	2.584	0.009	2.593	17.642	P: 3.48
	1.123	0.008	1.131	19.100	
	2.680	0.009	2.689	17.503	
	4.592	0.009	4.601	15.607	
	0.968	0.009	0.977	19.350	
i7-3610QM	0.660	0.009	0.669	2.470	I: 5.58
	0.513	0.009	0.522	19.816	P: 4.57
	1.224	0.010	1.234	19.098	
	2.780	0.009	2.789	17.523	
	4.715	0.010	4.725	15.609	
	1.092	0.010	1.102	19.326	
i7-4960X	1.204	0.009	1.213	19.702	I: 7.25
	3.407	0.009	3.416	17.034	P: 5.88
	1.339	0.009	1.348	19.079	
	2.878	0.009	2.887	17.535	
	4.827	0.009	4.836	15.591	
	1.187	0.009	1.196	19.327	

T_{Rleft}:= time to detect a relayer departure

$T_{restore}$:= time to restore the stream reception once the relayer departure has been detected

T_{blank}:= time during which the stream reception is stopped

T_{swap}:= time to perform the switch-to-fallback procedure once the relayer departure has been detected

Table 5 Times, expressed in seconds, elapsed during the six switch-to-fallback node operations in a lock-based version

	Relayer disconnection scenario (lock-based version)			
Host	T_{Rleft} (s)	$T_{restore}$ (s)	T_{blank} (s)	T_{swap} (s)
i5-3210M	8.775	20.001	28.776	20.002
	15.086	10.001	25.087	24.669
	15.458	5.000	20.458	15.103
	15.964	5.000	20.964	23.712
	10.310	5.000	15.310	15.120
	16.536	15.001	31.537	15.521
i7-3610QM	22.416	10.001	32.417	17.379
	6.167	10.000	16.167	14.471
	11.341	15.000	26.341	15.038
	11.781	15.001	26.782	22.924
	5.339	20.001	25.340	20.002
	11.447	5.000	16.447	15.633
i7-4960X	11.577	10.000	21.577	17.144
	10.029	10.001	20.030	20.160
	5.889	5.000	10.889	19.586
	10.877	5.000	15.877	15.152
	6.662	15.001	21.663	19.003
	6.770	15.000	21.770	18.163

T_{Rleft}:= time to detect a relayer departure

$T_{restore}$:= time to restore the stream reception once the relayer departure has been detected

T_{blank}:= time during which the stream reception is stopped

T_{swap}:= time to perform the switch-to-fallback procedure once the relayer departure has been detected

which the video playing is stopped is significantly reduced by the lock-free design: this is the parameter with the heaviest effects on the perceived QoE, whereas the time T_{swap} covers the whole procedure represented in Fig. 7 of Section 3.6 and includes also the time spent for the activation of the new thread for the *R-H* link monitoring.

Times for switch-to-fallback operations in the narrow bandwidth test and the percentages of lost frames are reported in Table 6: it seems there is no significant correlation between the T_{Rleft}, T_{QoS}, T_{swap} and $T_{restore}$ times and the computing capabilities of each host.

Charts in Figs. 8 and 9 represent the ECDFs (Empirical Cumulative Distribution Functions) of the end-to-end delay of the received frames in the relayer disconnection scenario and in the narrow bandwidth scenario respectively.

By comparing the two charts, we can notice better performance in the relayer disconnection scenario than in the narrow bandwidth scenario. In the former the end-to-end delay pratically never exceeds 0.5 s on the three hosts, while in the latter there are small probabilities (about 5%) it could exceed 1 second. However, also in the narrow bandwidth scenario the delay is lower than 4 s with very high probability. Especially in such scenario, the 6-core machine (Core i7-4960x) exhibits slightly better performance than the other two hosts.

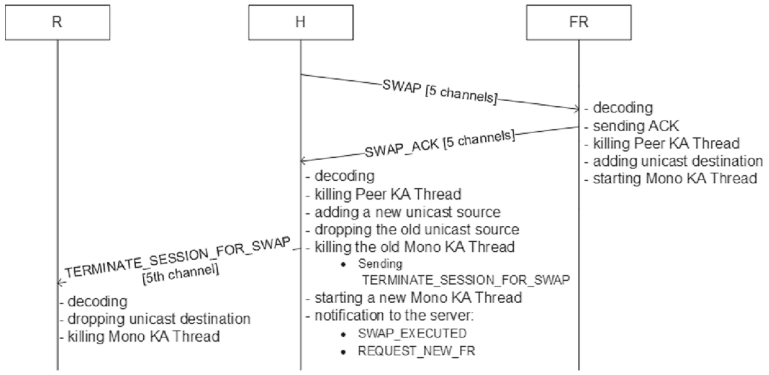

Fig. 7 Interaction between peers for a definitive close of the *R-H* link

Table 6 Times, expressed in seconds, elapsed during the six switch-to-fallback node operations and loss percentages for *I* and *P* frames

	Narrow bandwidth scenario (lock-free version)				
Host	T_{QoS}	$T_{restore}$	T_{weak}	T_{swap}	Lost frames (%)
i5-3210M	31.37	0.009	31.383	3.119	I: 11.81
	29.211	0.009	29.220	2.697	P: 9.22
	28.549	0.009	28.558	1.264	
	19.918	0.010	19.928	0.797	
	29.810	0.010	29.820	2.936	
	27.285	0.009	27.294	2.230	
i7-3610QM	32.384	0.009	32.393	2.761	I: 13.99
	24.863	0.010	24.873	3.492	P: 12.30
	29.998	0.009	30.007	3.289	
	23.519	0.010	23.529	2.842	
	30.465	0.010	30.475	3.084	
	23.088	0.009	23.097	3.840	
i7-4960X	27.215	0.008	27.223	2.649	I: 13.91
	24.660	0.090	24.750	3.687	P: 11.83
	29.987	0.009	29.996	0.730	
	20.809	0.009	20.818	1.461	
	31.372	0.008	31.380	2.971	
	23.879	0.009	23.888	2.790	

T_{QoS}:= time to detect a QoS degradation on the R-H link

$T_{restore}$:= time to restore the stream reception

T_{weak}:= time during which the stream reception is affected by packet losses

T_{swap}:= time to perform the switch-to-fallback node procedure once the QoS degradation has been detected

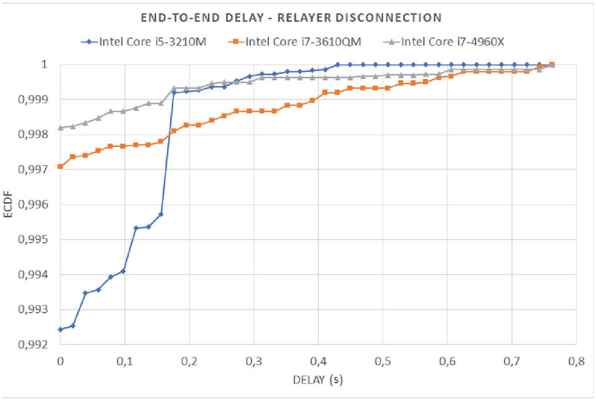

Fig. 8 ECDFs of the end-to-end delay of the frames received by the three hosts in the relayer disconnection scenario

We evaluated also video quality in terms of PSNR and SSIM, which are two common QoE metrics [50]: they are classified as Full-Reference metrics, since they compare the received video, affected by the noise generated by compression artifacts and packet losses, and the original uncompressed video. However, we should consider that frame losses and video impairments partially depend also on video encoding instantaneous parameters, which are related to specific features of video scenes. For each video frame we define $\Delta PSNR$ and $\Delta SSIM$ QoE distortions as the differences between the metrics computed for the reference compressed video and the metrics computed for the received video:

$$\Delta PSNR := PSNR_{reference} - PSNR_{received} \tag{7}$$

$$\Delta SSIM := SSIM_{reference} - SSIM_{received} \tag{8}$$

Both reference and received video QoE values were computed by comparing the video to its original uncompressed form. In this way, the two aforementioned differences express only the effects of network impairments on QoE degradation. Charts representing the ECDFs of $\Delta PSNR$ and $\Delta SSIM$ for the two test scenarios are shown in Figs. 10 and 11.

It can be pointed out that the relayer disconnection scenario exhibits better performance than the narrow bandwidth scenario. In particular, for all the three hosts there is a slightly

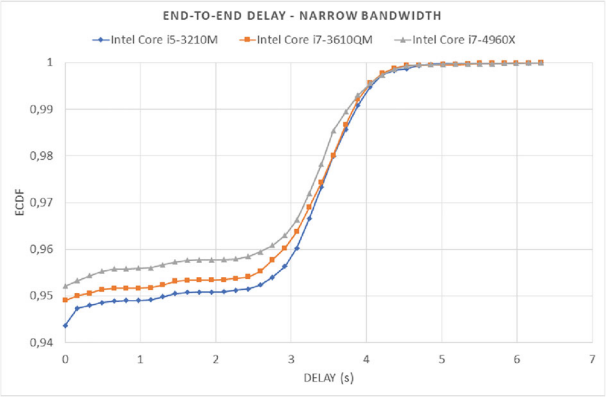

Fig. 9 ECDFs of the end-to-end delay of the frames received by the three hosts in the narrow bandwidth scenario

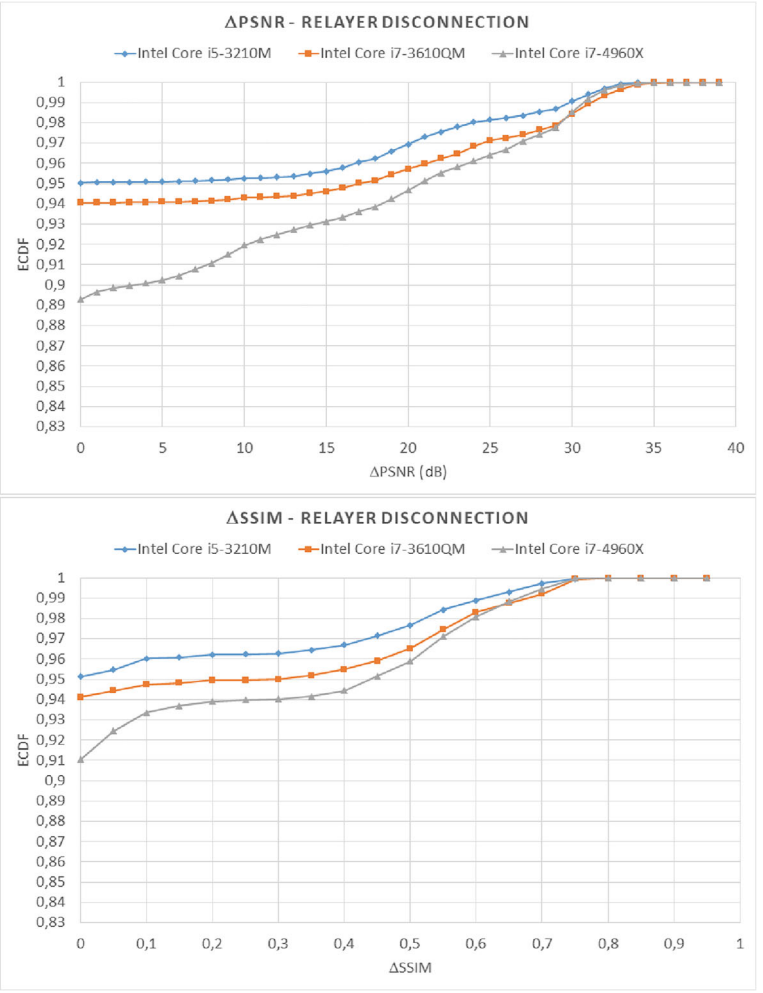

Fig. 10 ECDFs of the PSNR and SSIM degradations measured on the three hosts in the relayer disconnection scenario

higher probability (about +3%) to have a PSNR degradation lesser than 20 db. This performance gap is larger for SSIM metric. Surprisingly, the 6-core machine results in the worst performance, while the 2-core machine (Core i5-3210M) has the highest probability to have lower PSNR and SSIM degradations in most cases. For both PSNR and SSIM, this performance gap is larger in the relayer disconnection scenario. Since times measured on the 6-core machine are not worse than those measured on the 2-core machine, but frame losses are slightly higher, we think the reason of the performance gap involves the local relaying threads. Indeed, we collected the traffic traces on the *H* nodes through *tcpdump* by listening on the loopback network interface, which is the one used by MSRM to deliver RTP packets to the local video player (as we explained in Section 3.1.1). Thus, we can conclude that on more powerful machines packets collected into the buffer are consumed faster by local relaying threads, which therefore need a larger buffer or a buffer to be filled more quickly.

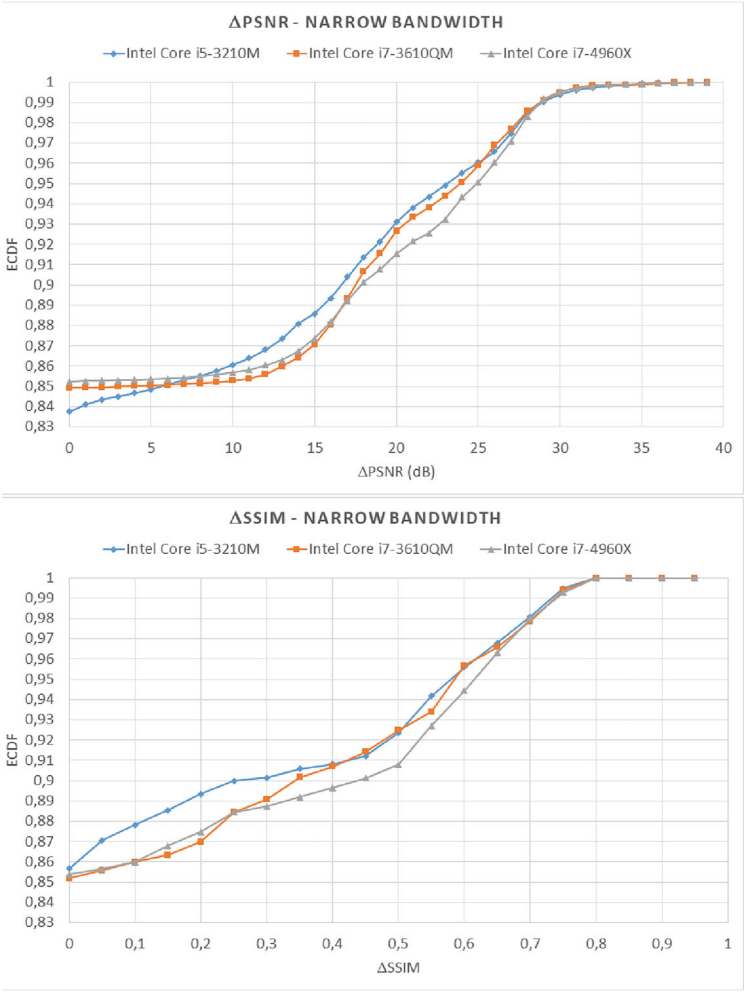

Fig. 11 ECDFs of the PSNR and SSIM degradations measured on the three hosts in the narrow bandwidth scenario

For this reason, further performance improvements should address a dynamic buffer dimensioning and a thread priority balancing tailored to suit the specific computational capabilities of the target machine.

7.1 Signalling overhead

Table 7 reports the signalling overhead on a *R* node during the relaying of a stream to a *H* node. Table 8 reports the signalling overhead on a *H* node while it is receiving a stream from a *R* node and handling a session with a fallback node. We have distinguished between the traffic related to the exchange of standard RTCP SR/RR packets and the additional traffic generated by our QoS monitoring algorithm, which is necessary to assess the state of the *R*-*H* link and of the fallback link.

Table 7 Signalling overhead on a *R* node

RTCP SR	Additional signalling
500 bit/s	40 bit/s

7.2 System scalability

In the previous sections we have evaluated the benefits achieved through the adoption of lock-free data structures in a small pilot study. However, we can expect substantial advantages even for larger overlay sizes, involving a higher number of nodes, links and streams. When the number of threads rises, due to a higher number of relaying links in the overlay tree, a lock-free design could offer even more evident benefits compared to a lock-based design, where synchronization among threads could accumulate delays having a significant impact on the system performance.

Compared to the extensible lock-based hash table inspired by the Java *ConcurrentHashMap*, the lock-free hash table employed in our architecture achieves the most important performance improvements for more than 10 concurrent threads and reaches its peak performance for 44 threads, when it is almost three times faster [42]. Compared to generic lock-based queues, the lock-free queue we adopted achieves the most important performance improvements for more than 10 concurrent threads, when it becomes at least four times faster [51].

Here is a summary of the threads instantiated for each stream received/relayed by a node:

- A *mono keepalive* thread for the stream reception on the relayer link;
- A *peer keepalive* thread for the fallback link;
- A *mono keepalive* thread for each stream relaying (i.e. for each host receiving the stream from the node);
- A *peer keepalive* thread for each host for which the node is acting as a fallback;
- A QoS monitoring thread;
- In MSRM, a receiving thread for each active source, a relaying thread for each destination.

Also the adoption of a MDC schema would produce a higher number of threads. In particular, it would require:

- A *mono keepalive* thread for each relayer link used for receiving one or more descriptions;
- A *peer keepalive* thread for each fallback link that can be used to swap the reception of one or more descriptions;
- A *mono keepalive* thread for each destination of the relayed descriptions;

Table 8 Signalling overhead on a *H* node

	RTCP RR	Additional signalling
RTCP RR from *H* to *R*	500 bit/s	
Ack messages from *H* to *R* on the two RTP subsessions		54 bit/s
Signalling on the 5th UDP channel with a relayer node		34 bit/s
Signalling with a fallback node		200 bit/s
Total signalling overhead	500 bit/s	288 bit/s

- A *peer keepalive* thread for each host for which the node is acting as a fallback;
- A QoS monitoring thread for each description;
- In MSRM, a receiving thread for each received description, a relaying thread for each relayed description.

In the case of multiple descriptions relayed through a multi-tree overlay, a lock-free design can bring significant improvements also in the merging process, which is responsible for the reconstruction of the high quality original video.

Some future extensions could take even better advantage of the performance scalability offered by lock-free data structures. A possible scenario could enable a simultaneous reception even from more than two source nodes that could last for all the streaming sessions and not only for the transient phase of the swapping procedure. Moreover, a lock-free design could be applied to the reduntant trees proposed in [12], where each node forwards to its siblings the video chunks received from its parent. These solutions, which would exploit the same internal MSRM architecture, would take advantage of several receiving threads writing into the same buffer queue and the same hash table.

8 Conclusions and future work

In this paper we have presented a QoS monitoring algorithm for real-time streaming overlays. The adoption of lock-free data structures allowed us to implement a fast switch-to-fallback mechanism: as soon as QoS problems rise, a receiving node can quickly restore the stream reception from a backup node (the so-called "fallback node relayer"). This contributes in enhancing playback continuity in presence of peer churning and limited bandwidth, two typical problems affecting overlay networks.

Experimental tests proved the good responsiveness of the implemented algorithm to QoS degradations. Nevertheless, they also highlighted the possibility of further improvements through a better exploitation of the most powerful machines, where the local relaying threads run faster and tend to consume the buffer content before new packets arrive after a switch-to-fallback procedure. To this aim, future work will focus on a dynamic fine tuning of the buffer size to further reduce the probability of blank periods in the stream playing. Of course an optimal buffer size should achieve a good trade-off between a smooth stream reproduction and a reduced playback lag with the root video source. For this reason, the scheduling of real-time threads with different priorities seems a more promising solution to define an optimal configuration for playback continuity.

We will also evaluate a fine tuning of the empirically detected parameters of the monitoring algorithm in our future work. Once we have compared the performance between the lock-based and lock-free approaches, it could be interesting to evaluate how the performance gap can change with different models and parametrizations. We will conduct more accurate tests in a multitree scenario, where the source stream is splitted into substreams (MDC descriptions) that can be propagated along different trees.

Furthermore, we will try to implement a real-time QoE monitoring algorithm based on some no-reference and reduced reference QoE metrics [26, 36], to activate the switch-to-fallback procedure in presence of video quality degradations. At the same time, we will employ some metrics to assess audio playback continuity and understanding. In this way, the monitoring algorithm will have a higher adherence to the actual human perception of the stream quality.

Appendix A: Parameters for MSRM monitoring algorithm

Tables 9 and 10 report the parameters and thresholds used in the QoS monitoring algorithm included in the MSRM library, which is based on the analysis of RTCP packets.

Table 9 Parameters for MSRM QoS monitoring algorithm

Parameters	Values
WEA factor (w)	0.125
Stream state buffer size (n)	5
Monitoring interval	5 s
Broken link threshold	10 s

Table 10 Thresholds for MSRM QoS monitoring algorithm

Thresholds	Values
LOSS_MEAN_THRESHOLD	15
JITTER_MEAN_THRESHOLD	450 s
LOSS_AUTOCORRELATION_THRESHOLD	0.5
JITTER_AUTOCORRELATION_THRESHOLD	0.5

Table 11 reports the warning rates, i.e. the increment and decrement steps, used by MSRM to update the QoS warning level according to the loss rate and jitter retrieved from RTCP packets.

Table 11 Warning rates used to increment/decrement the QoS warning level in MSRM

Warning rates	Values
GOOD_WARNING_RATE	1
ALERT_WARNING_RATE	2
CONGESTED_WARNING_RATE	3
BAD_WARNING_RATE	6

Table 12 Intervals used to classify the QoS warning level assessed by MSRM

QoS states	Intervals
GOOD	$qos \leq 3$
CONGESTED	$3 < qos \leq 6$
BAD	$qos > 6$

Appendix B: Parameters for QoS monitoring algorithm

Tables 13 and 14 summarize the scenarios, addressed by algorithms 2 and 3, leading to an increment or a decrement of the warning level on the fallback link and on the relayer link respectively. While for the fallback link the variations of the warning level depend only on RTT, for the relayer link they depend also on loss rate and jitter. Table 15 reports the thresholds for loss rate and jitter variations (used by algorithm 3) and for congestion level (used by Algorithm 4). Table 16 reports the maximum number of retransmissions based on RTT/QoS status.

Table 13 Increment and decrement of the alert level on the fallback link

Situation	Importance	Increment/ decrement value
Sample available without retransmissions, estimated mean RTT and sample RTT lower than the threshold	NO_WARNING	−1
Sample available without retransmissions, estimated mean RTT lower than the threshold, sample RTT greater than the threshold	LOW_WARNING	+2
Sample available without retransmissions, estimated mean RTT lower than the threshold, sample RTT greater than the threshold	LOW_WARNING	+2
Sample available without retransmissions, estimated mean RTT greater than the threshold and lower than the last measurement	MEDIUM_WARNING	+3
Sample available after retransmissions	MEDIUM_WARNING	+3
Sample not available	HIGH_WARNING	+5

Table 14 Increment and decrement of the alert level on the relayer link

Situation	Importance	Increment/ decrement value
Sample available without retransmissions, estimated mean RTT, sample RTT, loss rate/jitter variations lower than the threshold	NO_WARNING	−1
Sample not available	VERY_HIGH_WARNING	+8
RTT problem		
Sample available without retransmissions, estimated mean RTT lower than the threshold, sample RTT greater than the threshold	LOW_WARNING	+2

Table 14 (continued)

Situation	Importance	Increment/ decrement value
Sample available without retransmissions, estimated mean RTT lower than the threshold, sample RTT greater than the threshold	LOW_WARNING	+2
Sample available without retransmissions, estimated mean RTT greater than the threshold and lower than the last measurement	MEDIUM_WARNING	+3
Sample available after retransmissions	MEDIUM_WARNING	+3
Loss rate/jitter problem		
One of the loss rate/jitter variations higher than the threshold	LOW_WARNING	+2
Two or three of the loss rate/jitter variations higher than the threshold	MEDIUM_WARNING	+3
All the loss rate/jitter variations higher than the threshold	HIGH_WARNING	+5

Table 15 Thresholds for updating the congestion level and the link status in the overlay manager

Thresholds	Values
LOSS_RATE_THRESHOLD	LOSS_MEAN_THRESHOLD·0.7 = 11
JITTER_THRESHOLD	JITTER_MEAN_THRESHOLD·0.7 s = 315 s
QOS_MAX, RTT_MAX	16
CONGESTED_THRESHOLD	9
BAD_THRESHOLD	15

Table 16 Maximum number of retransmission based on RTT/QoS status

RTT/QoS alert level	Maximum number of retransmissions
$RTT/QoS\ status < 6$	Unchanged
$6 \leq RTT/QoS\ status < 9$	3
$9 \leq RTT/QoS\ status < 12$	2
$12 \leq RTT/QoS\ status < 15$	1
$RTT/QoS\ status 15$	0

Appendix C: Actions on a host node

Table 17 summarizes the actions taken by a host according to the status of its links.

Table 17 Actions taken by a host node

MSRM QoS	$H - R_{primary}$ link status	$H - R_{fallback}$ link status	Consequent actions
GOOD QUALITY	–	–	No one
CONGESTED	GOOD QUALITY	GOOD QUALITY	Bad QoS in *R* reception. A relayer change should be considered.
CONGESTED	GOOD QUALITY	CONGESTED or BAD QUALITY	Bad QoS in *R* reception. The fallback link is worse than the relayer link, so it is not worth changing *R*.
CONGESTED	CONGESTED	GOOD QUALITY	Bad QoS probably on *H*-*R* link. Swap should be performed.
CONGESTED	CONGESTED	CONGESTED or BAD QUALITY	Bad QoS probably on *H*-*R* link. The fallback link is worse. Swap should not be performed.
CONGESTED	BAD QUALITY	GOOD QUALITY or CONGESTED	Bad QoS on *H*-*R* link. Swap should be performed.
CONGESTED	BAD QUALITY	BAD QUALITY	Bad QoS on *H*-*R* link. The fallback link is not reliable. A new relayer should be requested.
BAD QUALITY	GOOD QUALITY	GOOD QUALITY	Bad QoS in *R* reception. A relayer change should be considered.
CONGESTED	GOOD QUALITY	CONGESTED or BAD QUALITY	Bad QoS in *R* reception. The fallback link is worse than the relayer link, so it is not worth swapping.
BAD QUALITY	CONGESTED	GOOD QUALITY	Bad QoS probably on *H*-*R* link. Swap should be performed.
BAD QUALITY	CONGESTED	CONGESTED or BAD QUALITY	Bad QoS probably on *H*-*R* link. The fallback link is worse than the relayer one, so it is not worth swapping.
BAD QUALITY	BAD QUALITY	GOOD QUALITY or CONGESTED	Bad QoS on *H*-*R* link. Swap should be performed.

Table 17 (continued)

MSRM QoS	$H - R_{primary}$ link status	$H - R_{fallback}$ link status	Consequent actions
BAD QUALITY	BAD QUALITY	BAD QUALITY	Bad QoS on *H-R* link. The fallback link is not reliable. A new relayer should be requested.
–	–	BAD QUALITY	A new fallback node should be requested.

References

1. Backhaus M, Schafer G (2017) Backup paths for multiple demands in overlay networks. In: 2016 Global information infrastructure and networking symposium, GIIS 2016
2. Bishop M, Rao S, Sripanidkulchai K (2006) Considering priority in overlay multicast protocols under heterogeneous environments. In: Proceedings IEEE INFOCOM 2006. 25th IEEE international conference on computer communications, pp 1–13
3. Bista BB (2009) A proactive fault resilient overlay multicast for media streaming. In: 2009 International conference on network-based information systems, pp 17–23
4. Budhkar S, Tamarapalli V (2017) Delay management in mesh-based P2P live streaming using a three-stage peer selection strategy. J Netw Syst Manag 26(2):401–425
5. Egilmez HE, Tekalp AM (2014) Distributed QoS architectures for multimedia streaming over software defined networks. IEEE Trans Multimed 16(6):1597–1609
6. Egilmez HE, Gorkemli B, Tekalp AM, Civanlar S (2011) Scalable video streaming over OpenFlow networks: an optimization framework for QoS routing. In: 2011 18th IEEE international conference on image processing, pp 2241–2244
7. Egilmez HE, Dane ST, Bagci KT, Tekalp AM (2012) OpenQoS: an OpenFlow controller design for multimedia delivery with end-to-end quality of service over software-defined networks. In: Proceedings of the 2012 Asia Pacific signal and information processing association annual summit and conference, pp 1–8
8. Egilmez HE, Civanlar S, Tekalp AM (2013) An optimization framework for QoS-enabled adaptive video streaming over OpenFlow networks. IEEE Trans Multimed 15(3):710–715
9. Espina F, Morato D, Izal M, Magaña E (2014) Analytical model for MPEG video frame loss rates and playback interruptions on packet networks. Multimed Tools Appl 72(1):361–383
10. Feldman S, LaBorde P, Dechev D (2015) A wait-free multi-word compare-and-swap operation. Int J Parallel Program 43(4):572–596
11. Fraser K (2004) Practical lock-freedom. Tech. Rep. UCAM-CL-TR-579, University of Cambridge, Computer Laboratory. http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-579.pdf
12. Fujita S (2019) Resilient tree-based video streaming with a guaranteed latency. J Interconnect Netw 19(4):1950009. https://doi.org/10.1142/S0219265919500099
13. Garroppo RG, Giordano S, Spagna S, Niccolini S, Seedorf J (2012) Topology control strategies on P2P live video streaming service with peer churning. Comput Commun 35(6):759–770
14. Gu W, Zhang X, Gong B, Zhang W, Wang L (2015) VMCast: a VM-assisted stability enhancing solution for tree-based overlay multicast. PLoS ONE 10(11):e0142888. https://doi.org/10.1371/journal.pone.0142888
15. Gupta AK, Singh M (2016) Structured p2p overlay networks for multimedia traffic. In: 2016 International conference on innovation and challenges in cyber security (ICICCS-INBUSH), pp 80–85
16. Hammami C, Jemili I, Gazdar A, Belghith A, Mosbah M (2014) Hybrid live P2P streaming protocol. Procedia Comput Sci 32(Supplement C):158–165. The 5th international conference on ambient systems, networks and technologies (ANT-2014), the 4th international conference on sustainable energy information technology (SEIT-2014)
17. 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
18. Helder D, Jamin S (2002) End-Host multicast communication using switch-trees protocols. In: 2nd IEEE/ACM international symposium on cluster computing and the grid, 2002, pp 419–419

19. Hemminger S (2005) Network emulation with NetEm. In: Pool M (ed) LCA 2005, Australia's 6th national Linux conference (linux.conf.au). Linux Australia. Linux Australia, Sydney
20. Herlihy MP, Wing JM (1990) Linearizability: a correctness condition for concurrent objects. ACM Trans Program Lang Syst 12(3):463–492
21. Hsieh YL, Wang K (2012) Dynamic overlay multicast for live multimedia streaming in urban VANETs. Comput Netw 56(16):3609–3628
22. Jeyasekar A, Kasmir Raja SV, Annie Uthra R (2017) Congestion avoidance algorithm using ARIMA(2,1,1) model-based RTT estimation and RSS in heterogeneous wired-wireless networks. J Netw Comput Appl. https://doi.org/10.1016/j.jnca.2017.05.008
23. Jurgelionis A, Laulajainen JP, Hirvonen M, Wang AI (2011) An empirical study of NetEm network emulation functionalities. In: 2011 Proceedings of 20th international conference on computer communications and networks (ICCCN), pp 1–6
24. Klaue J, Rathke B, Wolisz A (2003) EvalVid—a framework for video transmission and quality evaluation. In: Kemper P, Sanders WH (eds) Computer performance evaluation. Modelling techniques and tools. Springer, Berlin, pp 255–272
25. Kouchi T, Fujita S (2015) Maintaining tree-structured P2P overlay being resilient to simultaneous leave of several peers. IEICE Trans Inf Syst E98.D(9):1667–1674. https://doi.org/10.1587/transinf.2015EDP7021
26. Kwon JC, Jang SH, Chin Y, Oh SJ (2010) A novel video quality impairment monitoring scheme over an IPTV service with packet loss. In: 2010 second international workshop on quality of multimedia experience (QoMEX), pp 224–229
27. Laborde P, Feldman S, Dechev D (2017) A wait-free hash map. Int J Parallel Program 45(3):421–448
28. Magharei N, Rejaie R, Guo Y (2007) Mesh or multiple-tree: a comparative study of live P2P streaming approaches. In: IEEE INFOCOM 2007—26th IEEE international conference on computer communications, pp 1424–1432
29. Magnetto A, Gaeta R, Grangetto M, Sereno M (2010) Turinstream: a totally push, robust, and efficient p2p video streaming architecture. IEEE Trans Multimed 12(8):901–914
30. Marques H, Silva H, Logota E, Rodriguez J, Vahid S, Tafazolli R (2017) Multiview real-time media distribution for next generation networks. https://doi.org/10.1016/j.comnet.2017.03.002
31. Michael MM (2002) High performance dynamic lock-free hash tables and list-based sets. In: Annual ACM symposium on parallel algorithms and architectures. https://doi.org/10.1145/564879.564881, pp 73–82
32. Michael MM (2004) Hazard pointers: safe memory reclamation for lock-free objects. IEEE Trans Parallel Distrib Syst 15(6):491–504
33. Mwela JS, Adebomi OE (2010) Comparison of algorithms for concealing packet losses in the transmission of compressed video
34. Ooi WT (2005) Dagster: contributor-aware end-host multicast for media streaming in heterogeneous environment. In: Multimedia computing and networking 2005, vol 5680. International Society for Optics and Photonics, pp 77–90. https://doi.org/10.1117/12.592088
35. Ramzan N, Park H, Izquierdo E (2012) Video streaming over P2P networks: challenges and opportunities. Signal Process: Image Commun 27(5):401–411
36. Reibman A, Vaishampayan V, Sermadevi Y (2004) Quality monitoring of video over a packet network. IEEE Trans Multimed 6(2):327–334
37. Ren D, Li YTH, Chan SHG (2009) Fast-Mesh: a low-delay high-bandwidth mesh for peer-to-peer live streaming. IEEE Trans Multimed 11(8):1446–1456
38. Salsano S, Ludovici F, Ordine A, Giannuzzi D (2012) Definition of a general and intuitive loss model for packet networks and its implementation in the Netem module in the NetEm Module in the Linux Kernel
39. Sayit M, Demirci S, Kaymak Y, Tunali ET (2016) Adaptive, incentive and scalable dynamic tree overlay for P2P live video streaming. Peer-to-Peer Netw Appl 9(6):1074–1088. https://doi.org/10.1007/s12083-015-0390-7
40. Schwarz H, Marpe D, Wiegand T (2007) Overview of the scalable video coding extension of the H.264/AVC standard. IEEE Trans Cir Sys Video Technol 17(9):1103–1120
41. Sedrati M, Benyahia A (2017) Multipath routing to improve quality of service for video streaming over mobile ad hoc networks. Wirel Pers Commun 99(2):999–1013
42. Shalev O, Shavit N (2003) Split-ordered lists: lock-free extensible hash tables. In: Proceedings of the twenty-second annual symposium on principles of distributed computing, PODC '03. ACM, New York, pp 102–111
43. Sousa P, Rocha AA, De Lucena S, Diniz MC, Menasche DS (2016) S4Q: searching for QoE in P2P streaming neighborhoods. In: 2016 11th international conference on digital information management, ICDIM 2016

44. Staelens N, Moens S, Van den Broeck W, Mariën I, Vermeulen B, Lambert P, Van de Walle R, Demeester P (2010) Assessing quality of experience of IPTV and video on demand services in real-life environments. IEEE Trans Broadcast 56(4):458–466. https://doi.org/10.1109/TBC.2010.2067710
45. Taubenfeld G (2017) Contention-sensitive data structures and algorithms. Theor Comput Sci 677:41–55. https://doi.org/10.1016/j.tcs.2017.03.017
46. Tommasi F, De Luca V, Melle C (2013) A library for RTP relaying and QoS monitoring in Application Layer Multicast. In: 2013 Fifth international conference on ubiquitous and future networks (ICUFN), pp 418–423
47. Tommasi F, De Luca V, Melle C (2014) Are P2P streaming systems ready for interactive e-learning? In: 2014 International conference on education technologies and computers (ICETC), pp 49–54
48. Tommasi F, Melle C, De Luca V (2014) OpenSatRelaying: a hybrid approach to real-time audio-video distribution over the internet. J Commun 9(3):248–261
49. Tommasi F, De Luca V, Melle C (2015) Efficient multi-source RTP stream relaying in overlay networks. In: 2015 2nd world symposium on web applications and networking (WSWAN), pp 1–7
50. Tommasi F, De Luca V, Melle C (2015) Packet losses and objective video quality metrics in H.264 video streaming. J Vis Commun Image Represent 27:7–27
51. Tsigas P, Zhang Y (2001) A simple, fast and scalable non-blocking concurrent FIFO queue for shared memory multiprocessor systems. In: Annual ACM symposium on parallel algorithms and architectures, pp 134–143. https://doi.org/10.1145/378580.378611
52. Ullah I, Doyen G, Bonnet G, Gaiti D (2012) A survey and synthesis of user behavior measurements in p2p streaming systems. IEEE Commun Surv Tutor 14(3):734–749
53. Uma Maheswari B, Ramesh TK (2018) Location-aware resilient hybrid overlay structures for peer-to-peer video streaming. In: Proceedings of the 4th international conference on applied and theoretical computing and communication technology, iCATccT 2018, pp 255–260. https://doi.org/10.1109/iCATccT44854.2018.9001285
54. Wang Y, Reibman A, Lin S (2005) Multiple description coding for video delivery. Proc IEEE 93(1):57–70
55. Wang F, Liu J, Xiong Y (2008) Stable peers: existence, importance, and application in peer-to-peer live video streaming. In: IEEE INFOCOM 2008—the 27th conference on computer communications
56. Wu W, Yang Z, Nahrstedt K (2009) Dynamic overlay multicast in 3D video collaborative systems. In: Proceedings of the 18th international workshop on network and operating systems support for digital audio and video, NOSSDAV '09. ACM, New York, pp 1–6
57. Wu H, Xu K, Zhou M, Wong AK, Li J, Li Z (2013) Multiple-tree topology construction scheme for p2p live streaming systems under flash crowds. In: 2013 IEEE wireless communications and networking conference (WCNC), pp 4576–4581
58. Yong Goh C, Shyong Yeo H, Lim H, Kuan Hoong P, Lim JW, Tan IK (2013) A comparative study of tree-based and mesh-based overlay P2P media streaming. Int J Multimed Ubiquitous Eng 8(4):97–105
59. Zhang M, Zhang Q, Sun L, Yang S (2007) Understanding the power of pull-based streaming protocol: can we do better? IEEE J Sel Areas Commun 25(9):1678–1694
60. Zheng Q, Long Y, Qin T, Yang L (2011) Lifetime characteristics measurement of a p2p streaming system: focusing on snapshots of the overlay. In: 2011 9th World Congress on Intelligent Control and Automation, pp 805–810