# *GLive*: The Gradient overlay as a market maker for mesh-based P2P live streaming

Amir H. Payberah[†‡], Jim Dowling[†], Seif Haridi[†‡]
[†]*Swedish Institute of Computer Science (SICS)*
[‡]*KTH - Royal Institute of Technology*
*Email: {amir, jdowling, seif}@sics.se*

## Abstract

*Peer-to-Peer (P2P) live video streaming over the Internet is becoming increasingly popular, but it is still plagued by problems of high playback latency and intermittent playback streams. This paper presents* GLive*, a distributed market-based solution that builds a mesh overlay for P2P live streaming. The mesh overlay is constructed such that (i) nodes with increasing upload bandwidth are located closer to the media source, and (ii) nodes with similar upload bandwidth become neighbours. We introduce a market-based approach that matches nodes willing and able to share the stream with one another. However, market-based approaches converge slowly on random overlay networks, and we improve the rate of convergence by adapting our market-based algorithm to exploit the clustering of nodes with similar upload bandwidths in our mesh overlay. We address the problem of free-riding through nodes preferentially uploading more of the stream to the best uploaders. We compare GLive with our previous tree-based streaming protocol, Sepidar, and NewCoolstreaming in simulation, and our results show significantly improved playback continuity and playback latency.*

## 1 Introduction

Media streaming over Internet is becoming increasingly popular. Currently, most media is delivered using global content-delivery networks, providing a scalable and robust client-server model. For example, Youtube handle more than one billion hits per day[1]. However, content delivery infrastructures have very high cost, and an approach to reduce the cost of media delivery is to use peer-to-peer (P2P) overlay networks, where nodes share responsibility for delivering the media to one another.

[1]http://www.thetechherald.com/article.php/200942/4604/YouTube-s-daily-hit-rate-more-than-a-billion

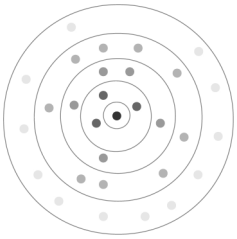


**Figure 1. The mesh overlay with different layers of the nodes (for legibility, the links between nodes are not shown). The nodes in each layer have similar upload bandwidth. The darker the node is, the higher upload bandwidth it has. The media source is located at the center of the overlay.**

Live media streaming using overlay networks is a challenging problem. Nodes should receive the stream with minimal delay over a best-effort network with varying bandwidth capacity, while adapting to other nodes joining, leaving and failing. From a system perspective, the overlay network should continuously optimize its structure to minimize the playback latency and maximize the timely delivery of the stream, by adapting to system and network dynamics. Furthermore, nodes should be intentivized to contribute and share their resources, through improved relative performance.

In this paper, we present *GLive*, a P2P streaming overlay network that uses both the Gradient overlay network and a distributed market mechanism to adaptively optimize its topology to minimize playback latency and maximize the timely delivery of the stream. The Gradient overlay network constructs a topology where (i) nodes with higher available upload bandwidth are positioned closer to the media source, and (ii) nodes with similar upload bandwidth become neighbours, producing logical *layers* (see Figure 1). As nodes

with relatively higher upload bandwidth can forward more copies of the stream to more nodes, positioning them closer to the media source will reduce the average number of hops from nodes to the media source, reducing both the probability of streaming disruptions and playback latency at nodes. Nodes are also incentivized to provide relatively more upload bandwidth, as nodes that contribute more upload bandwidth will have relatively higher playback continuity and lower latency than the nodes in lower layers.

In GLive, we divide the media stream into a sequence of *blocks*, and each node *pulls* the blocks of the stream from a set of nodes called *parents*. Nodes use a distributed market model, first introduced in [16], to choose parents from among the nodes in the system. A major problem with market-based approaches that select parents from random nodes is that they exhibit slow convergence properties. We improve the speed of convergence by nodes selecting from a small number of neighbouring nodes with similar upload bandwidth, i.e., a node either in its layer or in a layer closer to the media source (see Figure 1). The gossip-generated *Gradient overlay* network [19, 20] is used to enable nodes to sample neighbours with similar upload bandwidth, and, thus, it acts as a market-maker for our market model.

We evaluate GLive by comparing its performance with Sepidar [17] and the state-of-the-art NewCoolstreaming [8]. We show in simulation that GLive provides better playback continuity and lower playback latency than these systems under churn, flash-crowd, and massive-failure scenarios. We also evaluate the performance of GLive and Sepidar when a high percentage of nodes are free-riders. Finally, we evaluate the convergence of our market model when the node samples are taken from the Gradient overlay compared to a random overlay.

Our work is an extension of our previous work on multiple-tree live streaming [17, 16], and the contributions of this paper include:

- GLive, a distributed market-based solution to create a mesh overlay for P2P live media streaming,
- how mesh-based streaming outperforms multiple-tree streaming through comparing GLive and Sepidar [17],
- how the Gradient overlay can improve the convergence time of a mesh overlay in comparison with a random network,
- a *scoring* model to solve the free-rider problem in mesh overlays.

## 2 Related work

Many different overlay network topologies have been used for data delivery in P2P media streaming systems, but the two most widely used approaches are multiple-tree [16, 3, 13, 17] and mesh-based overlays [8, 6, 5]. Multiple-tree overlay networks use push-based content delivery over multiple tree-shaped overlays with the media source as a root of all trees. While multiple-tree overlay networks have the advantage of low latency data delivery, they are vulnerable to the failure of interior nodes. Rajaee et al. have shown in [10] that mesh overlays have consistently better performance than tree-based approaches for scenarios where there is node churn and packet loss.

Mesh-based approaches use swarming content delivery over a typically random overlay network. In mesh-based overlays, unlike tree-based structures that data is pushed through the tree, nodes pull data from their neighbours in the mesh. The mesh structure is highly resilient to node failures, but it is subject to unpredictable latencies due to the frequent exchange of notifications and requests [26]. Gossip++ [6], NewCoolStreaming [8], Chainsaw [14], and PULSE [18] are the systems that use random overlay meshes for data dissemination. Recently, there has been work on using gossiping to build non-random mesh topologies, where the topology stores implicit information about node characteristics, such as upload bandwidth. In [5], Fortuna et al. attempt to organize nodes with decreasing upload bandwidth at increasing distance from the source. As such, these systems have similarities with how GLive uses the Gradient overlay to structure nodes. However, GLive also uses a market model to optimize its partners for media streaming.

The problem of reducing free-riding in P2P systems has been solved by many existing incentive mechanisms and reputation models [13, 21, 11]. Of particular relevance to GLive are Give-to-Get [12] and Sepidar [17] that use transitive dependencies to a child's children in order to audit children nodes. In contrast, GLive uses a scoring mechanism to identify free-riders.

Our market model is an example of a distributed auction algorithm with partial information. Our model differs from existing work, such as [27] amd [15], in that all nodes are decision makers, the set of tasks and resources are homogeneous and auctions are restartable. Finally, our block selection strategy is similar to BiTOS for video-on-demand [23].

## 3 Problem description

We assume a network of nodes that communicate through message passing. New nodes may join the network at any time to watch the video. Existing nodes may leave the system either voluntarily or by crashing. The video is divided into a set of $B$ blocks of equal size without any coding. Every block $b_i \in B$ has a sequence number to represent its playback order in the stream. Nodes can pull any

block independently from any other node that can supply it.

Each node has a *partner list*, a view of a small subset of nodes in the system. A node can create a bounded number of *download connections* to partners and accept a bounded number of *upload connections* from partners over which blocks are downloaded and uploaded, respectively. We define a node $q$ as the *parent* of the *child* node $p$, if an upload connection of $q$ is bound to a download connection of $p$. Children nodes continuously attempt to improve their download connections by changing to parents that are both closer to the media source and able to deliver blocks on time. Parents, who can accept or reject connection attempts, prefer children who have forwarded the most blocks within a recent time window. The result of these preference functions is that nodes who forward more blocks on time have shorter paths to the media source.

Nodes store a list of blocks that are available for download in a *buffer map*. Nodes periodically send their buffer map to their children (via their upload connections) to advertise their available blocks. Children can then *pull* any blocks they require from the node. As such, advertisements are not random, but rather are directed away from the source and down the gradient.

For each block, we now represent the problem of finding the best mapping of upload connections to download connections as an *assignment problem* [22]. We define the set of all download and upload connections as $D$ and $U$, respectively. In order to receive the block, a node requires one of its download connection needs to be assigned to an upload connection over which the block will be copied. We define an assignment or a *mapping* $m_{ijk}$, from a node $i$ to a node $j$ for block $b_k$, as a triplet containing one upload connection at $i$ and one download connection at $j$ for block $b_k$:

$$m_{ijk} = (u_i, d_j, b_k) : u \in U, d \in D, b \in B, i, j \in N, i \neq j \quad (1)$$

where $N$ is the set of all nodes, $b_k$ is block $k$ from the set of all blocks $B$, and the connection from $i$ to $j$ is between two different nodes. A *cost function* is defined for a mapping $m_{ijk}$ as the minimum distance from node $i$ to the media source in terms of numbers of hops, that is,

$$c(m_{ijk}) : m_{ijk} \rightarrow number\,of\,hops\,from\,i\,to\,source. \quad (2)$$

We define a *complete assignment* $A$ for a block $b$ as a set of mappings, where, there exists at least one download connection at every node that is assigned to an upload connection over which $b$ is downloaded. That is, for a block $b$, each node has a download connection over which it can pull the block before the block expires. The total cost of a complete assignment is calculated as follows:

$$c(A) = \sum_{m \in A} c\,(m) \quad (3)$$

The goal of our system is to minimize the cost function in equation 3 for every block $b \in B$, such that a shortest path tree is constructed over the set of available connections for every block.

If the set of nodes, connections, and the upload bandwidth of all nodes is static for all blocks $B$, then we can solve the same assignment problem $\mid B \mid$ times. However, P2P systems, typically have churn (nodes join and fail) and available bandwidth at nodes changes over time, so we have to solve a slightly different assignment problem every time a node join, exits or a node's bandwidth changes.

Centralized solutions, such as the auction algorithm [2], are possible in principle, where nodes bid to connect their download connections to better upload connections using the amount of blocks they forward as currency. However, nodes that offer upload connections may not deliver a block over a connection in time. As such, the problem can be viewed as a *restartable auction*, where the auction is restarted because a bidder did not have sufficient funds to complete the transaction. But, in general, it is not feasible to use centralized solutions in large and dynamic networks with real-time constraints. An alternative naive decentralized implementation of the auction algorithm that communicates will all nodes through flooding would not scale either. Approximate decentralized solutions, based on random walks or sampling from a random overlay, have slow convergence time, as we show in our evaluation.

In the next section, we introduce our market model that finds approximate solutions to the assignment problem using partial views sampled from the Gradient overlay (to improve convergence time compared to a random overlay). Nodes are not assumed to be cooperative; nodes may execute protocols that attempt to download the stream without forwarding it to other nodes. We do not, however, address the problem of nodes colluding to receive the video stream.

## 4 GLive system

We now present our distributed market-model, a modified version of the distributed auction algorithm with partial information introduced for tree-based live streaming in [17]. The following properties are used by the model and calculated locally at each node:

1. *Money*: the total number of blocks uploaded to children during the last 10 seconds. A node uses its money to bid for a binding to a partner's upload connection.

2. *Price*: the minimum amount of money that should be bid when binding to an upload connection. The

price of a node that has an unbound upload connection is zero, otherwise the node's price equals the lowest amount of money at its existing children. For example, if node $p$ has three upload connections and three children with monies 2, 3 and 4, the price of $p$ is 2.

3. *Cost*: the cost of a node is the distance from that node to the media source via its shortest path. The shorter the path length (i.e., the lower its cost), the more desirable a parent it is.

Our market-model is based on minimizing costs (the path length of nodes to the media source) through nodes iteratively bidding for upload connections. Each node periodically sends its money, cost and price to all its partners. The partners of a node include all the nodes in its *similar-view* and *finger-list* in the Gradient overlay, see subsection 4.2. For each of its download connections, a child node $p$ sends a bid request to nodes that: (i) have lower cost than one of the existing parents assigned to download connections in $p$, and (ii) the price of a connection is less than $p$'s money. Nodes bid with their entire money (although the money is not used up, it can be reused for other bids for other connections).

A parent node who receives a bid request accepts it, if: (i) it has a free upload connection (its cost is zero), or (ii) it has assigned an upload connection to another node with a lower amount of money. If the parent re-assigns a connection to a node with more money, it abandons the old child who must then bid for a new upload connection. When a child node receives the acceptance message from another node, it assigns one of its download connections to the upload connection of the parent. Since a node may send more connection requests than its has download connections, it might receive more acceptance messages than it needs. In this case, if all its download connections are already assigned, it checks the cost of all its assigned parents and finds the one with the highest cost. If the cost of that parent is higher than the new received acceptance message, it releases the connection to that parent and accepts the new one, otherwise it ignores the received message.

Although there is no guarantee that the parent will forward all blocks over its connection to a child, parents who forward a relatively lower number of blocks will be removed as children of their parents. Nodes that claim that they have forwarded more blocks than they actually have forwarded are removed as children, and, an auction is restarted for the removed child's connection. Nodes are incentivized to increase the upper bound on the number of their upload connections, as it will help increase their upload rate and, hence, their attractiveness as children for parents closer to the root.

### 4.1 Auction restarting - free-rider detection and punishment

Whenever a node assigns a download connection to the upload connection of another node, it sends the address of its current children to its parent. It subsequently informs its parents of any changes in its children. Thus, a parent node knows about its childrens' children, or *grandchildren* for short.

*Free-riders* are nodes that forward a much lower number of blocks than they claimed they forward when connecting to a parent. We implment a *scoring* mechanism to detect free-riders, and thus motivate nodes to forward blocks. Each child assigns a score to each of its parents, which is initially set to zero, for a time window covering the last 10 seconds. When a child requests and receives a non-duplicate block from a parent within the last 10 seconds, it increments the score of that parent. Thus, the more blocks a parent node sends to its children, the higher score it has among its children. We chose 10 seconds as it is the same as the choking period in BitTorrent [4] and does not unneccessarily punish nodes because of variance in the rate of block forwarding.

Each node periodically sends a score request to its grandchildren, and the grandchildren nodes send back a score response containing the scores of the original node's children. The node sums up the received scores for each child. Free-rider nodes forward a lower number of blocks, and hence they have lower scores compared to others.

When a node with no free upload connection receives a connection request, it sorts its children based on their latest scores. If an existing child has a score less than a threshold $s$, then the child is identified as a free-rider. The parent node abandons the free-rider nodes and accepts the new node as its child. If there is more than one child whose score is less than $s$, then the lowest score is selected. If all children have a score higher than $s$, then the parent accepts the connection if the connecting node has offers more money than the lowest money of its existing children. When the parent accepts such a connection, it then abandons (removes the connection to) the child with the lowest money. The abandonned child then has to search for and bid for a new connection to a new parent.

A crucial difference between our market-model and the classical auction algorithm is that our solution is decentralized; nodes have only a partial (changing) view of a small number of nodes in the system with whom they can bid for upload connections. We use the Gradient overlay to provide nodes with a constantly changing partial view of other nodes that have a similar number of upload connections. Thus, rather than have nodes explore the whole system for better parent nodes, the Gradient enables us to limit exploration to the set of nodes with a similar number of upload

connections.

### 4.2 Gradient overlay construction

Nodes search for parents by sampling partners from the Gradient overlay. The Gradient overlay is a gossip-generated overlay, where nodes are arranged according to their local utility function, such that the highest utility nodes are located topologically in the centre of the overlay, while lower utility nodes are located at increasing distance from the centre [19, 20].

Each node in the Gradient overlay maintains two sets of neighbours: *similar-view* and *random-view*. Similar-view is a partial list of the nodes in the system whose *utility values* are close to, but slightly higher than the utility value of the node. However, the nodes in the random-view are sampled from a random overlay network. We use Cyclon [24] to create and update the random-view. Nodes periodically gossip with each other and exchange their views. Upon receiving the views from a neighbour, a node merges it with its own similar-view and retains those entries that have closer (but higher) utility to its own utility value. The connections to the random nodes in random-view allow nodes to explore the network in order to discover other potentially similar neighbours.

In GLive, the utility value of a node is calculated using two factors: (i) a node's upload bandwidth, and (ii) a disjoint set of discrete utility values that we call *market-levels*. A market-level is defined as a range of network upload bandwidths. For example, in figure 2, we define 5 example market-levels: mobile broadband (64-127 $Kbps$) with utility value 1, slow DSL (128-511 $Kbps$) with utility value 2, DSL (512-1023 $Kbps$) with utility value 3, fiber ($>$1024 $Kbps$) with utility value 4, and the media source with utility value 5. A node measures its available upload bandwidth (e.g., using a server or trusted neighbour) and calculates its utility value as the market-level that its upload bandwidth falls into. For instance, a node with 256 $Kbps$ upload bandwidth falls into slow DSL market-level, so its utility value is 2. Nodes may also choose to contribute less upload bandwidth than they have available, causing them to join a lower market level.

A node prefers to fill its similar-view with nodes from the same market-level or one level higher. As a result, the nodes with similar utility value (almost the same upload bandwidth) become the neghibours of each other. In addition to similar-view and random-view, nodes maintain *finger-list* that contains at most one node from higher market levels (if one is available). Finger list reduces the probability of the overlay partitioning due to excessive clustering. Moreover, low bandwidth nodes often do not have enough upload bandwidth to simultaneously deliver all the stream. Therefore, in order to enable low bandwidth nodes to utilize

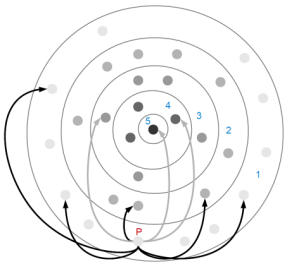



**Figure 2. Different market-levels of a system, and the similar-view and fingers of p.**

the spare connections of higher bandwidth nodes, nodes can use the connections in finger-list (figure 2).

To update the similar-view, each node $p$ periodically chooses one random node $q$ from its similar-view, and sends it a random subset of the nodes from its similar-view. Upon receiving the list of nodes, $q$ sends back a random subset of the nodes from its similar-view. When node $p$ receives the $q$'s view, first merges the received view with its existing similar-view by iterating through the received list of nodes, and preferentially selecting those nodes in the same market-level or at most one level higher. If its similar-view is not full, it adds the node, otherwise, it replaces one of the nodes it had sent to $q$ with the selected node. Moreover, to allow nodes to find other potentially similar neighbours, $p$ repeat the same procedure by merging its similar-view with its own local random-view.

The fingers to higher market-levels are also updated periodically. Node $p$ goes through its random-view, and for each higher market-level, picks a node from that market-level if there exists such a node in the random-view. If there is not, $p$ keeps the old finger. For more details, you are kindly referred to our work in [16].

### 4.3 Data dissemination

Each parent node periodically sends its *buffer map* and its *load* to all its assigned children. The buffer map shows the blocks that a node has in its buffer, and the load shows the ratio of the number of blocks that a node has forwarded to the number of its upload connections.

A child node, uses the information received from its parents to schedule and pull the required blocks in different iteration. We define a *sliding window* that shows the number of blocks that a child node can request in each iteration. If the playback point of a node is $t$, and the sliding window size is $n$, the node can request the blocks from $t$ to $t + n$ in each iteration.

One important question in pulling blocks is the order of requests. The main constraint in data dissemination in live

media streaming is that the blocks should be received before their playback time. Therefore, a node should pull the missing block with the closest playback time first, that is, blocks should be pulled in-order. Another potential strategy, as used by BitTorrent, is to pull the rarest blocks in the system, as this is known to increase aggregrate network throughput [23].

We have designed a download policy that attempts to marry the benefits for playback latency of in-order downloading with the improved network throughput of rarest-block policy. We divide the sliding window into two sets: an *in-order set* and a *rare set*. The first $m$ blocks in the sliding window are the blocks in the in-order set and the rest of the blocks of the sliding window are the rare set blocks. As the names of these sets imply, blocks from the in-order set are requested in order and the least popular block (from among the node's partners) is chosen from the rare set. A node selects a block from the in-order set with probability $h\%$ and from the rare set with $(100 - h)\%$, where $h$ is a system parameter. If multiple parents can provide a block, the child node chooses the parent that has the lowest load.

## 5 Experiments and evaluation

In this section, we compare the performance of GLive with P2P live-streaming systems Sepidar [17] and NewCoolstreaming [8] under simulation. Sepidar has a multiple-tree architecture and NewCoolstreaming has a random mesh-based architecture.

### 5.1 Experiment setup

We have used Kompics [1] to implement GLive, Sepidar and NewCoolstreaming. Kompics is a framework for building P2P protocols and it provides a discrete event simulator for simulating them using different bandwidth, latency and churn models. We have implemented Sepidar and NewCoolstreaming based on the system descriptions from [17] and [25].

In our experimental setup, we set the streaming rate to $512Kbps$, which is divided into blocks of $16Kb$. Nodes start playing the media after buffering it for 15 seconds, which compares favourably to the 60 seconds of buffering used by state-of-the-art (proprietary) SopCast [9]. The size of similar-view in GLive and Sepidar and the partner list in NewCoolstreaming is 15 nodes. We assume all the nodes have the same number of download connections, which is set to 8. To model upload bandwidth, we assume that each upload connection has available bandwidth of $64Kbps$ and that the number of upload connections for nodes is set to $2i$, where $i$ is picked randomly from the range 1 to 10. This means that nodes have upload bandwidth between $128Kbps$ and $1.25Mbps$. As the average upload bandwidth of $704Kbps$ is not much higher than the streaming rate of $512Kbps$, nodes have to find good matches as parents in order for good streaming performance. The media source is a single node with 40 upload connections, providing five times the upload bandwidth of the stream rate. This setting is based on SopCast's requirement that the source has at least five times the upload capacity of the stream rate [9]. In our simulations we assume 11 market-levels, such that the nodes with the the same number of upload connections are located at the same market-level. For example, nodes with two upload connection ($128Kbps$) are the members of the first market-level, nodes with four upload connections ($256Kbps$) are located in the second market-level, and the media source with 40 upload connections ($2.5Mbps$) is the only member of the 11th market-level. Latencies between nodes are modeled using a latency map based on the King data-set [7].

We assume the size of sliding window for downloading is 32 blocks, such that the first 16 blocks are considered as the in-order set and the next 16 blocks are the blocks in the rare set. A block is chosen for download from the in-order set with $90\%$ probability, and from the rare set with $10\%$ probability. In the failure detector settings, we set the threshold of the score, $s$, to zero. The window used for our scoring mechanism is set to 10 seconds.

In the experiments, we measure the following metrics:

1. *Playback continuity*: the percentage of blocks that a node received before their playback time. We consider two metrics related to playback continuity: where nodes have a playback continuity of (i) greater than 90% and (ii) greater than 99%;

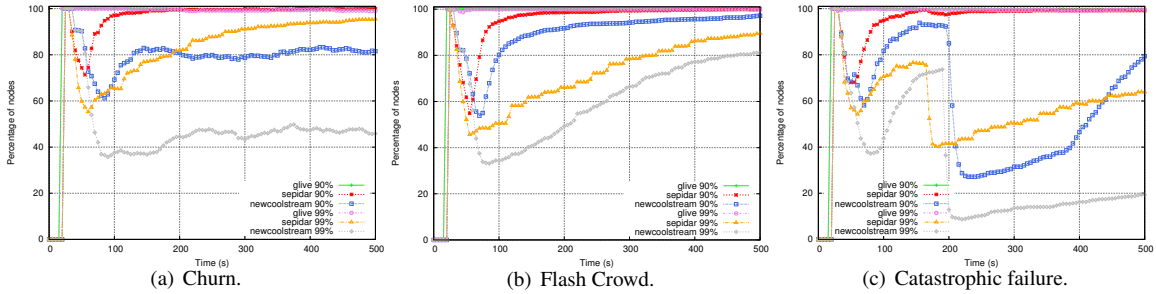
2. *Playback latency*: the difference in seconds between the playback point of a node and the playback point at the media source.

### 5.2 GLive vs. Sepidar vs. NewCoolstreaming

In this section, we compare the playback continuity and playback latency of GLive with Sepidar and NewCoolstreaming in the following scenarios:

1. *Flash crowd*: first, 100 nodes join the system following a Poisson distribution with an average inter-arrival time of 100 milliseconds. Then, 1000 nodes join following the same distribution with a shortened average inter-arrival time of 10 milliseconds;

2. *Catastrophic failure*: 1000 nodes join the system following a Poisson distribution with an average inter-arrival time of 100 milliseconds. Then, 500 existing nodes fail following a Poisson distribution with an average inter-arrival time 10 milliseconds;




(a) Churn. (b) Flash Crowd. (c) Catastrophic failure.

**Figure 3. Playback continuity of the systems in different scenarios.**

3. *Churn*: 500 nodes join the system following a Poisson distribution with an average inter-arrival time of 100 milliseconds, and then till the end of the simulations nodes join and fail continuously following the same distribution with an average inter-arrival time of 1000 milliseconds;

Figures 3 shows the percentage of the nodes that have playback continuity of at least 90% and 99%. We see that all the nodes in GLive receive at least 90% of all the blocks very quickly in all scenarios, while it takes more time in Sepidar. That is because in Sepidar, at the beginning, nodes spend time constructing the trees, while in GLive the nodes pull blocks quickly as soon as at least one of their download connections is assigned. As we see in figure 3, both GLive and Sepidar outperform NewCoolstreaming in playback continuity for the whole duration of the experiment in all scenarios. GLive and Sepidar use the Gradient overlay for node discovery. The Gradient overlay arranges nodes based on their number upload bandwidth capacity, and so the neighbours of a node are those with the same upload bandwidth capacity, or slightly higher. This helps the high capacity nodes to quickly discover the media source. In contrast, NewCoolstreaming uses a random overlay, and it takes more time for nodes to find appropriate parents. The result is a higher number of changes in parent connections, causing lower playback continuity in NewCoolstreaming compared to GLive and Sepidar.

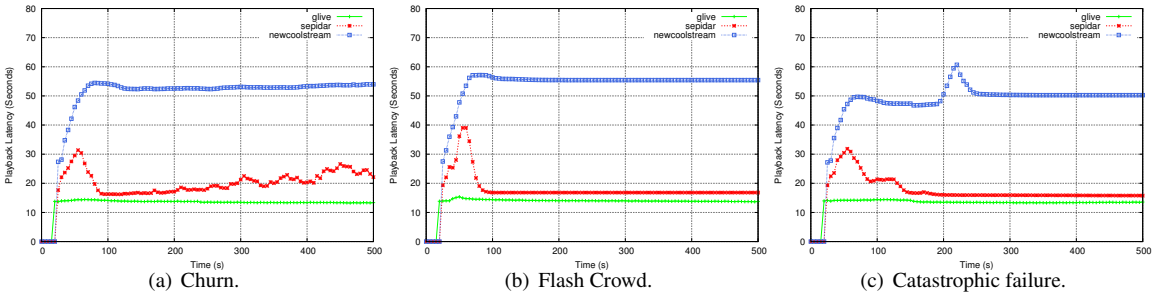
As we see in figure 3, the difference between GLive and Sepidar increases, when we measured the percentage of the nodes that receive 99% of the blocks in time. Again, the tree structure used in Sepidar causes this difference. Although, Sepidar has a multiple-tree structure, which is resilient to the failures, it has a lower playback continuity than GLive when nodes crash. In a multiple-tree structure, a stream is split into a number of sub-streams, and a node receives each sub-stream from a parent. Although, a node typically receives the blocks of each sub-stream independently, if the parent providing a sub-stream fails, then it loses the block from that sub-stream. While the node is trying to find a new parent for that sub-stream, it will miss the blocks for that sub-stream. However, this problem does not apply to the mesh overlay, because the nodes pull the blocks independently of each other. Therefore, if a node loses one of its parents, it can pull the required blocks from other parents.

Figure 7 shows the playback latency of the systems in different scenarios. As we can see, GLive keeps its playback latency relatively constant, close to 15 seconds, which is the initial buffering time. The playback latency of Sepidar also converges to 15 seconds, but it takes longer to converge than GLive. The reason for this delay is, again, the time needed to construct the trees. The playback latency of GLive and Sepidar, are both less than NewCoolstreaming. In NewCoolstreaming, the higher playback latency is a result of nodes only reactively changing parents when their playback latency is greater than a predefined threshold.

Another difference between GLive, Sepidar and NewCoolstreaming is the behavior of the systems when playback latency increases. In GLive and Sepidar, if playback latency exceeds the initial buffering time and enough blocks are available in the buffer, nodes are given a choice to fast forward the stream and decrease the playback latency. In contrast, NewCoolstreaming jumps ahead in playback by switching parent(s) even it misses several blocks, thus negatively affecting playback continuity [8].

### 5.3 Free-rider detector settings

Here, we compare the playback continuity of GLive and Sepidar in the *free-rider scenario*. In this scenario, 1000 nodes join the system following a Poisson distribution with an average inter-arrival time of 100 milliseconds, such that 30% of the nodes are free-riders, and the total amount of upload bandwidth in the system is less than total amount of download bandwidth required by nodes. Figure 5 shows the percentage of the nodes that receive 99% of the blocks before their playback time. It shows this value for all the nodes in the system, including the *strong nodes* (top 10% of upload bandwidth nodes), the free-riders, and the *weak nodes* (the bottom 10% of upload bandwidth nodes).




(a) Churn. (b) Flash Crowd. (c) Catastrophic failure.

**Figure 4. Playback latency of the systems in different scenarios.**

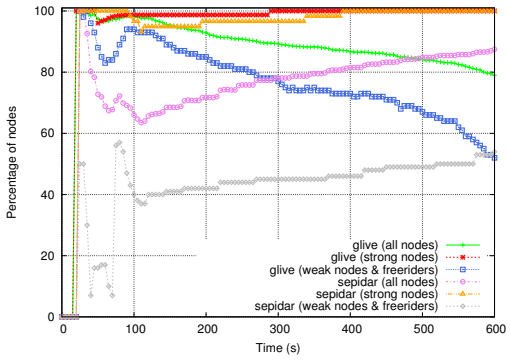



**Figure 5. Playback continuity in the free-rider scenario.**

Figure 5 shows that all the strong nodes in both systems receive all the blocks in time, however, GLive converges faster than Sepidar. In GLive, we are using the scoring mechanism to find the nodes who contribute less bandwidth than they claim when bidding for connections, while Sepidar uses a free-rider detector module that identifies nodes that do not meet their contractual requirement to forward the stream to their child nodes [17]. In GLive, at the beginning, a high percentage of weak nodes and free-riders receive all the blocks in time, which shows that free-riders have not been detected yet. That is because nodes need time to update and validate the scores of their parents, and, thus, identify freeriders. Meanwhile, the free-riders use the resources of the system. However, after enough time has passed and the nodes' scores have been updated, the free-riders are detected. Thus, after about 100 seconds the percentage of the free-riders who have a high playback continuity decreases. As figure 5 shows, after about 600 seconds from the beginning of the experiment, in both GLive and Sepidar the free-riders and weak nodes receive roughly the same quality of stream, that is, they have the same percentage of playback continuity. As the playback continuity of the weak nodes and free-riders keeps decreasing in GLive, we can also see that the playback continuity decreases for all nodes in GLive. After 500 seconds, playback continuity even decreases below Sepidar.

Importantly, as we can see in figure 5, the existing free-riders in the system have a very low effect on the playback continuity of the strong nodes in GLive. Strong nodes have consistently higher playback continuity than weak nodes and free-riders. This is due to the fact that weak nodes have a lower amount of money compared to strong nodes, which makes them take longer to find good parents. Also, the punishment of free-riders negatively affects their playback continuity. As such, nodes are strongly incentivized to contribute more upload bandwidth through receiving improved relative performance.

### 5.4 Comparing the Gradient with random neighbour selection

In this experiment, we compare the convergence speed of our market model for the Gradient overlay and a random overlay. We use the churn scenario in this experiment, as this is the most typical environment for P2P streaming systems on the Internet. Our market model is run using (i) samples taken from the Gradient overlay, where the sam-

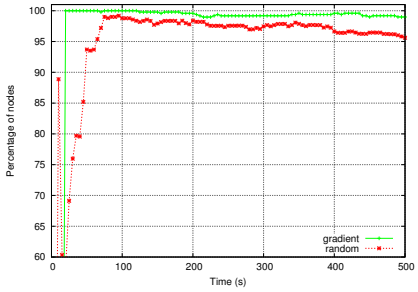


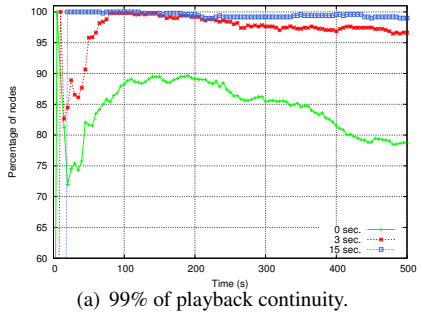

**Figure 6. 99% of playback continuity of the GLive in the Gradient overlay and the random overlay.**

pled nodes have similar upload bandwidth or money, and (ii) samples taken from a random network, where the sampled nodes have random amounts of money.

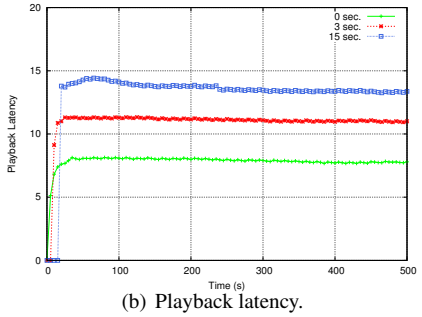
As nodes in the Gradient overlay receive bids from a set of nodes with almost the same money, the difference between received bids is less than the expected difference for the random network. Figure 6 shows that in the case of using the Gradient overlay, more nodes can quickly receive high playback continuity. As such, the Gradient overlay can be said to be a more efficient market maker for our distributed market model than a random overlay.

### 5.5 Varying buffering time

Finally, we compare the performance of GLive for different buffering times. We compare three different settings: 0, 3 and 15 seconds of buffering time in the churn scenario. Buffering 0 seconds of blocks, means nodes start playing the media as soon as they receive the first block. As we see in figure 7(a), the higher the buffering time, the higher the percentage of the nodes who receive blocks in time. However, higher initial buffering times increase the playback latency (figure 7(b)). As such, there is a trade-off between increasing playback continuity and decreasing playback latency.




(a) 99% of playback continuity.




(b) Playback latency.

**Figure 7. The performance of the system in different initial buffering time.**

## 6 Conclusions

In this paper, we presented GLive, a P2P live streaming system that uses a distributed market-model to construct a mesh overlay with two properties: (i) nodes with increasing upload bandwidth are located closer to the media source, and (ii) nodes with similar upload bandwidth are the neighbours of each other. Our distributed market-model leverages the structure of the Gradient overlay to efficiently assign suitable connections to other nodes. We addresse the problem of free-riding in GLive through parent nodes auditing the behaviour of their children nodes by querying their grandchildren. We showed in simulation that the mesh-based implemention of our market-model has better performance in different scenarios compared to both a multiple-tree implementation of the system in Sepidar and NewCoolstreaming.

## References


[1] C. Arad, J. Dowling, and S. Haridi. Developing, simulating, and deploying peer-to-peer systems using the kompics component model. In *COMSWARE '09: Proceedings of the Fourth International ICST Conference on COMmunication System softWAre and middlewaRE*, pages 1–9, New York, NY, USA, 2009. ACM.

[2] D. P. Bertsekas. The auction algorithm: a distributed relaxation method for the assignment problem. *Ann. Oper. Res.*, 14(1-4):105–123, 1988.

[3] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. Splitstream: high-bandwidth multicast in cooperative environments. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 298–313, New York, NY, USA, 2003. ACM Press.

[4] B. Cohen. Incentives build robustness in bittorrent. In *In Proceedings of the 1st Workshop on Economics of Peer-to-Peer Systems*, 2003.

[5] R. Fortuna, E. Leonardi, M. Mellia, M. Meo, and S. Traverso. QoE in Pull Based P2P-TV Systems: Overlay Topology Design Tradeoffs. In *Peer-to-Peer Computing (P2P), 2010 IEEE Tenth International Conference on*, pages 1–10. IEEE, 2010.

[6] D. Frey, R. Guerraoui, A. Kermarrec, and M. Monod. Boosting Gossip for Live Streaming. In *Peer-to-Peer Computing (P2P), 2010 IEEE Tenth International Conference on*, pages 1–10. IEEE, 2010.

[7] K. P. Gummadi, S. Saroiu, and S. D. Gribble. King: Estimating latency between arbitrary internet end hosts. In *SIGCOMM Internet Measurement Workshop*, 2002.

[8] B. Li, Y. Qu, Y. Keung, S. Xie, C. Lin, J. Liu, and X. Zhang. Inside the new coolstreaming: Principles, measurements and performance implications. In *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, pages 1031–1039, 2008.

[9] Y. Lu, B. Fallica, F. Kuipers, R. Kooij, and P. V. Mieghem. Assessing the quality of experience of sopcast. *Journal of Internet Protocol Technology*, 4(1):11–23, 2009.

[10] N. Magharei, R. Rejaie, and Y. Guo. Mesh or multiple-tree: A comparative study of live p2p streaming approaches. In *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, pages 1424–1432. Ieee, 2007.

[11] M. Meulpolder, J. A. Pouwelse, D. H. J. Epema, and H. J. Sips. Bartercast: A practical approach to prevent lazy freeriding in p2p networks. In *IEEE International Symposium on Parallel&Distributed Processing*, pages 1–8, Washington, DC, USA, 2009. IEEE Computer Society.

[12] J. Mol, J. Pouwelse, M. Meulpolder, D. Epema, and H. Sips. Give-to-get: Free-riding-resilient video-on-demand in p2p systems. In *Multimedia Computing and Networking 2008*, volume 6818. SPIE Vol. 6818, January 2008.

[13] J. J. D. Mol, D. H. J. Epema, and H. J. Sips. The orchard algorithm: P2p multicasting without free-riding. In *P2P '06: Proceedings of the Sixth IEEE International Conference on Peer-to-Peer Computing*, pages 275–282, Washington, DC, USA, 2006. IEEE Computer Society.

[14] V. Pai, K. Kumar, K. Tamilmani, V. Sambamurthy, A. E. Mohr, and E. E. Mohr. Chainsaw: Eliminating trees from overlay multicast. In *Workshop on Peer-to-Peer Systems (IPTPS)*, pages 127–140, 2005.

[15] C. Park, W. An, K. R. Pattipati, and D. L. Kleinman. Distributed Auction Algorithms for the Assignment Problem with Partial Information. In *International Command and Control Research and Technology Symposium*, June 2010.

[16] A. H. Payberah, J. Dowling, F. Rahimian, and S. Haridi. gradienTv: Market-based P2P Live Media Streaming on the Gradient Overlay. In *Lecture Notes in Computer Science (DAIS 2010)*, pages 212–225. Springer Berlin / Heidelberg, Jan 2010.

[17] A. H. Payberah, J. Dowling, F. Rahimian, and S. Haridi. Sepidar: Incentivized market-based p2p live-streaming on the gradient overlay network. *International Symposium on Multimedia*, 0:1–8, 2010.

[18] F. Pianese, J. Keller, and E. W. Biersack. Pulse, a flexible p2p live streaming system. In *INFOCOM*. IEEE, 2006.

[19] J. Sacha, B. Biskupski, D. Dahlem, R. Cunningham, R. Meier, J. Dowling, and M. Haahr. Decentralising a service-oriented architecture. *Peer-to-Peer Networking and Applications*, 3(4):323–350, 2010.

[20] J. Sacha, J. Dowling, R. Cunningham, and R. Meier. Discovery of stable peers in a self-organising peer-to-peer gradient topology. In F. Eliassen and A. Montresor, editors, *6th IFIP WG 6.1 International Conference Distributed Applications and Interoperable Systems (DAIS)*, volume 4025, pages 70–83, Bologna, June 2006.

[21] G. Tan and S. A. Jarvis. A payment-based incentive and service differentiation scheme for peer-to-peer streaming broadcast. *IEEE Trans. Parallel Distrib. Syst.*, 19(7):940–953, 2008.

[22] C. N. Vasconcelos and B. Rosenhahn. Bipartite graph matching computation on gpu. In *7th International Conference on Energy Minimization Methods in Computer Vision and Pattern Recognition*, pages 42–55, Berlin, Heidelberg, 2009. Springer-Verlag.

[23] A. Vlavianos, M. Iliofotou, and M. Faloutsos. Bitos: enhancing bittorrent for supporting streaming applications. In *In IEEE Global Internet*, pages 1–6, 2006.

[24] S. Voulgaris, D. Gavidia, and M. van Steen. CYCLON: Inexpensive Membership Management for Unstructured P2P Overlays. *Journal of Network and Systems Management*, 13(2):197–217, 2005.

[25] S. Xie, B. Li, G. Keung, and X. Zhang. Coolstreaming: Design, Theory and Practice. *IEEE Transactions on Multimedia*, 9(8):1661, 2007.

[26] W. P. K. Yiu, X. Jin, and S. H. G. Chan. Challenges and approaches in large-scale p2p media streaming. *IEEE MultiMedia*, 14(2):50–59, 2007.

[27] M. M. Zavlanos, L. Spesivtsev, and G. J. Pappas. A distributed auction algorithm for the assignment problem. In *2008 47th IEEE Conference on Decision and Control*, pages 1212–1217. IEEE, December 2008.