Sepidar: Incentivized Market-Based P2P Live-Streaming on the Gradient Overlay Network

Amir H. Payberah
and Fatemeh Rahimian and Seif Haridi
Swedish Institute of Computer Science (SICS)
KTH - Royal Institute of Technology
Email: {amir, fatemeh, seif}@sics.se

Jim Dowling
Swedish Institute of Computer Science (SICS)
Email: jdowling@sics.se

***Abstract*—Live streaming of video content using overlay networks has gained widespread adoption on the Internet. This paper presents Sepidar, a distributed market-based model, that builds and maintains overlay network trees, which are approximately minimal height, for delivering live media as a number of substreams. A streaming tree is constructed for each substream such that nodes that contribute higher amounts of upload bandwidth are located increasingly closer to the media source at the root of the tree. While our distributed market model can be run against a random sample of nodes, we improve its convergence time to stabilize a tree by executing against a sample of nodes that contribute similar amounts of upload bandwidth. We use the Gradient overlay network to generate samples of such nodes. We address the problem of free-riding through parent nodes auditing the behaviour of their child nodes. We evaluate Sepidar by comparing it in simulation with state-of-the-art NewCoolstreaming. Our results show significantly improved playback latency and playback continuity under churn, flash-crowd, and catastrophic failure experiment scenarios. We also show that using the Gradient improves convergence time of our distributed market model compared to a random overlay network. Finally, we show that Sepidar punishes the performance of free-riders, and that nodes are incentivized to contribute more upload bandwidth by relatively improved performance.**

***Keywords*-P2P Overlay; Live Streaming; Gradient Overlay; Distributed Market Model;**

I. Introduction

Live streaming using overlay networks on the Internet requires distributed algorithms that strive to use the nodes' resources efficiently in order to ensure that the viewer quality is good. To improve user viewing experience, systems need to maximize the playback continuity of the stream at nodes, and minimize the playback latency between nodes and the media source. Nodes should be incentivised to contribute resources through improved relative performance, and nodes that attempt to freeride, by not contributing resources, should be detected and punished. In order to improve system performance in the presence of asymmetric bandwidth at nodes, it is also crucial that nodes can effectively utilize the extra resources provided by the "better" nodes.

In this paper, we meet these requirements by building multiple approximately minimal height streaming overlay trees, where the nodes with higher available upload bandwidth are positioned higher in the tree as they can support relatively more child nodes. Minimal height trees help reduce both the probability of streaming disruptions and the average playback latency at nodes [25]. The media stream is split into a set of sub-streams, called *stripe*, and each tree delivers one sub-stream. Multiple sub-streams allow more nodes to contribute bandwidth and enable trees to be more robust [5].

Our system, called *Sepidar*, models the problem of constructing and maintaining minimal height overlay trees as an assignment problem [31], where the stripes that can be uploaded by nodes (*upload slots*) are matched to the stripes that nodes attempt to download (*download slots*), such that the height of the tree (the cost function for all nodes) is minimized. We introduce a new market model, a distributed algorithm inspired by auction algorithms [4], where, for each stripe, nodes continuously compete to become *children* of nodes providing stripes that are closer to the root (the media source). *Parents* supplying stripes prefer children nodes who offer to forward the highest number of copies of the stripes. Children proactively switch parents, when the market-modelled benefit of switching is greater than the cost of switching, until the trees stabilize. Our market model works in the presence of freeriders by parents periodically auditing children. Children are audited by querying the children's children (*grandchildren*) to validate that the child is forwarding the copies of stripes it claims to forward.

To improve the speed of convergence of the trees, nodes execute the market model in parallel using samples taken from the *Gradient overlay* [24]. The Gradient is a gossip-generated overlay network where nodes organize into a gradient structure with the media source at the centre of the gradient and nodes with decreasing relative upload bandwidth found at increasing distance from the centre. When nodes sample from their neighbours in the Gradient, they receive nodes with similar upload bandwidths. In a converged minimal height streaming overlay tree, the sampled nodes will be located at similar depths in the tree. Although we only consider upload bandwidth for constructing the Gradient and overlay trees in this paper, the model can easily be extended to include other characteristics such as node uptime, load and reputation.

We evaluate Sepidar by comparison with NewCoolstreaming, a successful and widely used media streaming solution [10]. We show in simulation, under churn, flash-crowd, and massive-failure scenarios, that our market-based approach improves the playback continuity and decreases the average playback latency at clients compared to NewCoolstreaming. We also evaluate the performance of Sepidar when varying key system parameters such as block size, number of stripes, playback buffering time, and freerider detection sensitivity. Finally, we evaluate the performance improvement for the market model in sampling from the Gradient overlay compared to sampling from a random overlay.

We build on our previous work in [21] by providing a distributed market model that works in the presence of freeriders and dynamic upload bandwidths.

II. Related work

There are two fundamental problems in building the media streaming overlay networks: (i) how to disseminate data, and (ii) how to discover other nodes supplying the stream.

Early data delivery overlays use a tree structure, where the media is pushed from the root to interior nodes to leave nodes. Examples of such systems include Climber [20], ZigZag [26], NICE [3], and [7]. The short latency of data delivery is the main advantage of this approach [33]. Disadvantages, however, include the fragility of the tree structure upon the failure of nodes close to the root and the fact that all the traffic is only forwarded by the interior nodes. SplitStream [5] improved this model by using multiple trees, where the stream is split into sub-streams and each tree delivers one sub-stream. Orchard [16], ChunkySpread [28] and CoopNet [18] are some other solutions in this class.

An alternative to tree structured overlays is the mesh structure, in which the nodes are connected in a mesh-network, and nodes request missing blocks of data explicitly. 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 [33]. SopCast [13], DONet/Coolstreaming [34], Chainsaw [19], and PULSE [22] are examples of mesh-based systems.

Another class of systems combine tree and mesh structures to construct a data delivery overlay. Example systems include CliqueStream [2], mTreebone [30], NewCoolStreaming [10], Prime [14] and [12].

The second fundamental problem is how nodes discover the other nodes that supply the stream. CoopNet [18] uses a centralized coordinator, GnuStream [8] uses controlled flooding requests, SplitStream [5] and [12] use DHTs, while NewCoolstreaming [10], DONet/Coolstreaming [34] and PULSE [22] use a gossip-generated random overlay network to search for the nodes. Sepidar uses the Gradient overlay for this purpose.

NewCoolstreaming [10] has the most similarities with Sepidar. Both systems have the same data dissemination model where a node subscribes to a sub-stream at a parent node, and the parent subsequently pushes the stream to the child. However, Sepidar's use of the Gradient overlay to discover nodes to supply the stream contrasts with NewCoolStreaming that samples nodes from a random overlay. A second major difference is that NewCoolStreaming only reactively changes a parent when a sub-stream is identified as being slow, whereas Sepidar proactively changes parents to improve system performance.

The problem of reducing freeriding in P2P systems has been addressed by many existing incentive mechanisms and reputation models [16], [25], [15]. Our solution for freerider identification is influenced by Give-to-Get [17], that first used transitive dependencies to a child's children in order to audit children nodes. In contrast to Sepidar, Give-to-Get is a video-on-demand protocol built on a mesh network, and based on BitTorrent.

Our market model is inspired by auction algorithms. The first widely-used auction algorithm was designed by Bertsekas [4], and has an equivalent representation as a weighted bipartite matching problem [31]. However, in contrast to auction algorithms, our market model does not assume that prices always rise - freeriders cause the price of an upload slot to be reset to zero. Also, our market model assumes local views of the system at nodes and that the discovery of nodes and price information is expensive.

Tan and Jarvis describe a payment-based approach to solving freeriding for live streaming [25]. Nodes run periodic auctions for their resources and earn points that can be used to access resources. Whereas we incentivize nodes to provide more resources to get better video performance, they incentivise nodes to remain in the system even when not viewing video to acquire an increased number of points. Similar to Sepidar, they also support a strategy for preferring the lowest depth parent resulting in the construction of a height-balanced tree. Another related approach to matching nodes for live streaming is based on finding maximal bipartite matchings using a flow algorithm by Li and Mahanti [11]. They transformed the traditional min-cost media flow dissemination problem into an auction problem.

III. Problem description

We assume the video is treated as a constant-rate bitstream that is divided into *blocks* of equal size without any coding, where every block has a sequence number to represent its playback order in the stream. The blocks are delivered to nodes over multiple sub-streams, called *stripes*, that each deliver an equal number of blocks per unit time. Nodes can retrieve any stripe independently from any other node that can supply the stripe. We define the number of copies of stripes that nodes are willing and able to forward as its number of *upload slots*. Nodes do not upload more stripes than they have upload slots. Each node has a number of upload slots, that is proportional to the amount of upload bandwidth capacity it contributes to the system. Every node has the same number of *download slots*, equal to the number of stripes. We assume all nodes have sufficient download bandwidth capacity to receive all stripes. A *parent* can forward a copy of any stripe over an upload slot, and a *child* node, that connects its download slot to an upload slot, requests a specific stripe for an upload slot. 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, although this can be addressed by a reputation management scheme [35].

The problem we address in this paper is how to deliver a video stream from a source as multiple stripes over multiple approximately minimal height trees. This problem can be represented as the *assignment problem* [27]. Centralized solutions to this problem are possible for small system sizes. For example, if all nodes send their number of upload slots to a central server, the server can use any number of algorithms that solve linear sum assignments, such as the auction algorithm [4], the Hungarian method [9], or more recent high-performance parallel algorithms [27].

The problem with a decentralized implementation of the auction algorithm is the communication overhead in nodes discovering the node with the upload slot of highest net value. The auction algorithm assumes that the cost of communicating with all nodes is close to zero. In a decentralized system, however, communicating with all nodes requires flooding, which is not scalable. An alternative approach to compute an approximate solution is to find good upload slots based on random walks or sampling from a random overlay. However, such solutions typically have slow convergence time, as we show in section V. In the next section, we introduce our market model that finds approximate solutions using the partial views sampled from the Gradient overlay.

IV. Sepidar system

Our distributed market model uses the following three properties, calculated at each node, to build trees:

1) *Currency*: the total number of upload slots at a node. A node uses its currency to bid for a connection to another node's upload slot for each stripe.
2) *Price*: the minimum currency that should be bid when establishing a connection to an upload slot. The price of a node that has an unused upload slot is zero, otherwise the node's price equals the lowest currency of its already connected children. For example, if node p has three upload slots and three children with currencies 2, 3 and 4, the price of p is 2.
3) *Cost*: the cost of an upload slot at a node for a particular stripe is the distance from that node to the root for that stripe. Since the media stream consists of several stripes, nodes may have different costs for different stripes. The lower the depth a node has for a stripe (the lower its cost), the more desirable a parent it is for that stripe.

Our market model is based on minimizing costs through nodes iteratively bidding for upload slots. This model could be best described as an approximate auction algorithm, where there is a *continuous auction* and *no reserve price*. For each stripe, child nodes place bids of their entire currency for upload slots at the parent nodes with lowest cost (depth). Child nodes always bid with their entire currency to avoid the complexity of price-setting. A parent node sets a price of zero for an upload slot when at least one of its upload slots is unassigned or when it has a free-riding child. Thus, the first bid for an upload slot will always win (no reserve price), enabling children to immediately connect to available upload slots. When all of a parent's upload slots are assigned, it sets the price for an upload slot to the currency of its child with the lowest number of upload slots. If a child with more currency than the current price for an upload slot bids for an upload slot, it will win the upload slot and the parent will replace its child with the lowest currency with the new child. A child that has lost an upload slot has to discover new nodes and bid for their upload slots. In contrast to the auction algorithm, there are no bidding and assignment phases, thus, we call it a continuous auction.

In contrast to the auction algorithm, the price of upload slots does not always increase - it can be reset to zero if a child node is detected as a freerider, that is, if the node is not correctly forwarding all the stripes it promises to supply. As such, it is a *restartable auction*, where the auction is restarted because a bidder did not have sufficient funds to complete the transaction. Another crucial difference with the auction algorithm is that our market model 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 slots. We use the Gradient overlay to provide nodes with a constantly changing partial view of other nodes that have a similar number of upload slots. 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 slots.

A. Gradient overlay construction

The Gradient overlay is an overlay network that arranges nodes using a local utility function at each node, such that nodes are ordered in descending utility values away from a core of the highest utility nodes [23], [24]. The highest utility nodes are found at the centre of the Gradient topology, while nodes with decreasing utility values are found at increasing distance from the centre.

The Gradient is built by both gossiping and sampling from a random overlay network (we use Cyclon [29]). Each node maintains a set of neighbours called a *similar-view* containing a small number of nodes whose utility values are close to, but slightly higher than, the utility value of the node. Nodes periodically gossip to exchange and update their similar-views. Node references stored in the similar view contain the *utility value* for the neighbours. In Sepidar, the utility value of a node is calculated using two factors: a node's upload bandwidth and 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 1, 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 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,

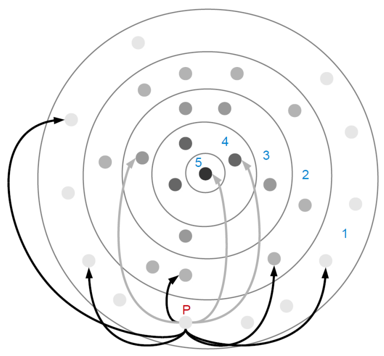


Fig. 1. Different market-levels of a system, and the similar-view and fingers of p.

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. A feature of this preference function is that low-bandwidth nodes only have connections to one another. However, low bandwidth nodes often do not have enough upload bandwidth to simultaneously deliver all stripes in a stream. Therefore, in order to enable low bandwidth nodes to utilize the spare slots of higher bandwidth nodes, nodes maintain a *finger list*, where each *finger* points to a node in a higher market-level (if one is available). We illustrate the market levels and fingers in figure 1. Each ring represents a market-level, the black links show the links within the similar-view and the gray links are the fingers to nodes in higher market-levels.

In order for nodes to be able to explore to find new nodes with which to execute our market model, a node constantly updates its neighbours within its market level. Each node p periodically increments the age of all the nodes in its similar-view, removes the oldest node, q, from its similar-view and sends a subset of nodes in its similar-view to q. Node q responds by sending back a subset of its own similar-view to p. Node p then merges the view received from q with its existing similar-view by iterating through the received list of nodes, and preferentially selecting those nodes in the same market-level as p or at most one level higher. If the similar-view is not full, it adds the node, and if a reference to the node to be merged already exists in p's similar-view, p just refreshes the age of its reference. If the similar-view is full, p replaces one of the nodes it had sent to q with the selected node. What is more, p also merges its similar-view with its own local random-view, in the same way described above. Upon merging, when the similar-view is full, p replaces a node whose utility value is higher than p's utility value plus one.

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.

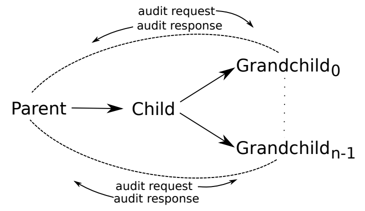


Fig. 2. Transitive auditing by parents querying grandchildren about the performance of children.

B. Streaming tree overlay construction

Nodes periodically send their currency, cost, price, number of children and *buffer map* to their similar-view nodes. The buffer map shows the last blocks that a node has in its buffer. For each stripe i, a node p periodically checks if it has a node in its similar-view and finger list that has (i) a lower cost (depth) than its current parent, (ii) a price less than its currency and (iii) blocks ahead of its block in stripe i. If such a node is found, it is added to a list of candidate parents for stripe i. Next, the node sorts the candidates by the term $S = \frac{numOfChildren}{currency}$, and selects the node with smallest S. That is, it biases selection towards nodes with fewer children and higher currency. If two nodes have the same S, it selects the one with higher currency.

If a node q receives a connection request from node p for stripe i, has a free upload slot, it accepts the request, otherwise if p's currency is greater than the price of q, q abandons its child that has the lowest currency, and accepts p as a new child. If q has a freeriding child (see section IV-C), it abandons that node as the child with the lowest currency. The disconnected node has to find a new parent. If q's price is greater than or equal to p's currency, q declines the request.

C. Freerider detection and punishment

Freeriders are nodes that supply less upload bandwidth than claimed. To detect freeriders, we introduce the *freerider detector* component with *strong completeness* property. By strong completeness property, we mean that, if a *non-freerider* node does not have free upload slots, eventually it detects all its freeriding children.

Nodes identify freeriders through transitive auditing using their children's children (Figure 2). To do this, a non-freerider parent p periodically sends an *audit request*, about its child q, to q's claimed children. Whenever a grandchild receives a message from p, it checks if q is its parent, and has properly forwarded the stripe(s) it has promised to supply. The grandchild, then, sends back either a positive or negative *audit response* to p that shows whether these conditions are satisfied or not.

We now show how strong completeness property is satisfied for the freerider detector. Assume a node q claims it has k upload slots, such that m of them are assigned to other nodes and n of them are free upload slots, $k = m + n$. Its parent p periodically sends audit requests to q's m claimed children. Before the next iteration of sending audit requests, p calculates

F as the sum of (i) the number of audit responses not received before a timeout, (ii) the number of negative audit responses, and (iii) the n free upload slots. If F is more than $M\%$ of k, q is suspected as a freerider. If q becomes suspected in N consecutive iterations, it is detected as a freerider. For example, if N equals 2, a node is detected as a freerider if it is suspected on two consecutive iterations of the freerider detector. The higher the value of N, the more accurate but slower the detection is.

In a converged tree, for nodes not in the two bottom levels (market-levels one and two), we expect that at least $M\%$ of their upload slots are meeting their contracted obligation to correctly supply a substream over that upload slot. M is a threshold for freerider suspicion. For example, if M is 90%, then node q is suspected as a freerider, if 10% or more of its upload slots are either not connected to child nodes or connected to child nodes but do not supply the stream at the requested rate.

After detecting a node as a freerider, the parent node p, decreases its own price (p's price) to zero and as a *punishment* considers the freerider node q as its child with the lowest currency. On the next bid from another node, p replaces the freerider node with the new node. So, if a node claims it has more upload bandwidth than it actually supplies, it will be detected and punished. In a converged tree, many members of the market-level one and two may have no children, because they are the leaves of the trees. So, the nodes in these two market-levels are not suspected as freeriders. Freeriders can use the extra resources in the system without any punishment if they just join as a member of market-level one or two.

V. Experiments and evaluation

In this section, we establish the performance of Sepidar for different system parameter settings, and then compare the performance of Sepidar with NewCoolstreaming under simulation.

A. Experiment setup

We have implemented both Sepidar and NewCoolstreaming using the Kompics platform [1]. Kompics provides a framework for building P2P protocols and a discrete event simulator for simulating them using different bandwidth, latency and churn models. Our implementation of NewCoolstreaming is based on the system description in [10], [32]. We have validated our implementation of NewCoolstreaming by replicating, in simulation, the results from [10].

In our experimental setup, we set the streaming rate to $512Kbps$. The stream is split into 8 stripes and each stripe is divided into a sequence of $16Kb$ blocks. Nodes start playing the media after buffering it for 15 seconds. The size of a node's partial view (the similar-view in Sepidar and the partner list in NewCoolstreaming) is 15 nodes. The number of upload slots for the non-root nodes equals $2i$, where i is picked randomly from the range 1 to 10. Considering that the size of an upload slot equals $64Kbps$, this number corresponds to an 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 80 upload slots. We assume all the nodes have enough download bandwidth to receive all the stripes simultaneously. Here, we define 11 market-levels, such that the nodes with the the same number of upload slots are located at the same market-level. For example, nodes with two upload slot ($128Kbps$) are the members of the first market-level, nodes with four upload slots ($256Kbps$) are located in the second market-level, and the media source with 80 upload slots ($> 5Mbps$) is the only member of the 11th market-level. Latencies between nodes are modelled using a latency map based on the King data-set [6]. The failure detector settings are $N = 2$ and $M = 50\%$.

In the experiments, we measure the following metrics:

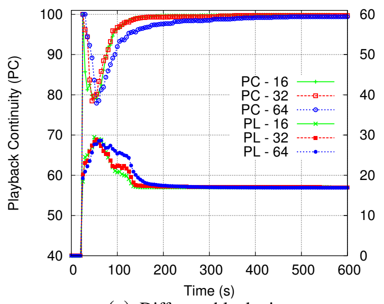
1) *Playback continuity*: the percentage of blocks that a node received before their playback time. In our experiments to measure playback quality, we count the number of nodes that have a playback continuity of greater than 90%;
2) *Playback latency*: the difference in seconds between the playback point of a node and the playback point at the media source.

We use the following scenarios in the experiments:

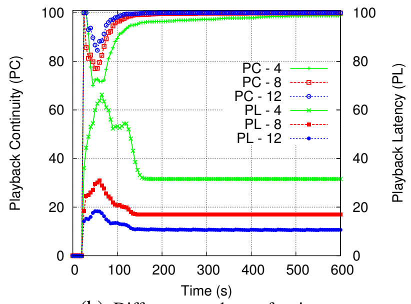
1) *Join only*: 1000 nodes join the system following a Poisson distribution with an average inter-arrival time of 100 milliseconds;
2) *Flash crowd*: first, 100 nodes join the system following a Poisson distribution with an average inter-arrival time of 100 milliseconds. Then, 1000 nodes join following the same distribution with a shortened average inter-arrival time of 10 milliseconds;
3) *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. The system then continues its operation with only 500 nodes;
4) *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;
5) *Freerider*: 1000 nodes join the system following a Poisson distribution with an average inter-arrival time of 100 milliseconds, such that 20% of the nodes are freeriders.

B. Establishing parameters for good system performance

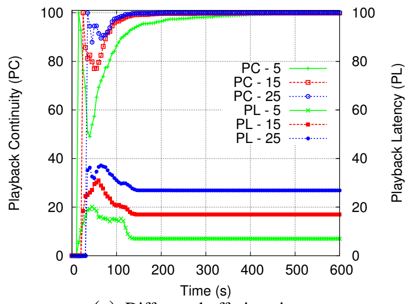
Here, we evaluate the performance of the system for different system settings. These experiments are based on the join only scenario. In the first experiment, we measure the performance of the system for varying block sizes: $16Kb$, $32Kb$ and $64Kb$. Figure 3(a) shows better playback continuity



(a) Different block sizes.



(b) Different numbers of stripes.



(c) Different buffering times.

Fig. 3. System performance for different system settings.

and less playback latency for smaller block sizes. The same result is shown in the NewCoolstreaming paper [10], and our subsequent experiments comparing Sepidar with Newcoolstreaming are based on a block size of $16Kb$.

Another system parameter is the number of stripes. As can be seen in figure 3(b), as more stripes are used, playback continuity increases and playback latency reduces. For a media stream split into K stripes, a node that receives the whole stream should assign its K download slots to K upload slots. If a node misses M of its parent connections, its misses $\frac{M}{K}$ of the stream. So, as K increases, nodes lose less of the stream for a single failed parent connection.

Figure 3(c) shows the behaviour of Sepidar for different initial playback buffering times. We compare three different settings: 5, 15 and 25 seconds of initial buffering time. Buffering 5 seconds of blocks in advance results in playback interruptions when nodes change their parents, but better playback continuity is achieved for 15 and 25 seconds of buffering. We can also see that playback latency increases when the buffering time is increased. Thus, the initial buffering time is a parameter that trades off better playback continuity against worse playback latency.

C. *Freerider detector settings*

Here, we measure the playback continuity of nodes for different freerider detector settings. This experiment is based on the join only scenario. We consider the case where 30% of all nodes are freeriders and 20% are *weak* nodes, such that the ratio of the number of upload slots to download slots is less than one. Weak nodes are members of the market-level one or two, that is, nodes who only have enough upload bandwidth to forward at most half of the media stream. Nodes that are neither weak nor freerider nodes are called *non-freeriders*. Our experiments vary the freerider detector parameter N, while we measure the playback continuity of the different nodes.

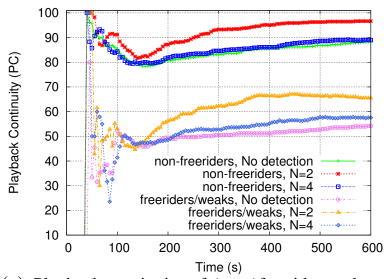
Figure 4(a) shows the playback continuity of nodes for three values of N: $N = 0$, that is, no freerider detection, $N = 2$, and $N = 4$. We set M to 50% in all the simulations to take into account delayed replies by children and to decrease the false positive detection threshold for freeriders. We measured the playback continuity for other values of N, but to aid the readability of the plots we left them out. Although higher values of N increase the accuracy of the detector, the late detection of freerider decreases the playback continuity of nodes. Figure 4(a) confirms our conclusions as we see that the playback continuity of nodes when $N = 0$ and $N = 4$ are almost the same. The figure shows that $N = 2$ provides better playback continuity for the all nodes. Another important result here is the lower playback continuity of freeriders/weak nodes compared to non-freeriders. If a node detects one of its children as a freerider, it selects the freerider node as its child with the lowest currency, and replaces it with other nodes as soon as it receives a request. Losing a parent decreases the playback continuity of freeriders.

In figure 4(b), we measure the total number of suspected nodes and the nodes that are correctly detected. As we see here, when N has lower values, the fraction of nodes that are correctly detected as freeriders decreases.

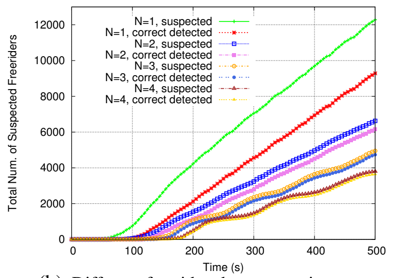
D. *Sepidar vs. NewCoolstreaming*

In this section, we compare the playback continuity and playback latency of Sepidar and NewCoolstreaming in the churn (figures 4(c)), catastrophic failure (figures 4(d)), flash crowd (figures 4(e)), and freerider (figures 4(f)) scenarios. In these figures, the Y1-axis (PC) shows the percentage of the nodes in the overlay that have a playback continuity higher than 90%, and the Y2-axis (PL) shows the average playback latency.

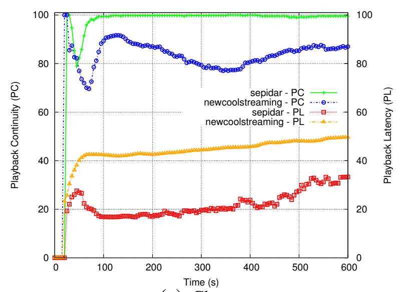
We see that Sepidar significantly outperforms NewCoolstreaming in playback continuity for the whole duration of the experiment in all scenarios. This outperformance is due to quicker discovery of appropriate parents and faster construction of overlay trees in Sepidar. In Sepidar, high capacity nodes can quickly discover and connect to the source using the similar-view, while in NewCoolstreaming nodes take longer to find parents as they search by updating their random view through gossiping. In addition, nodes in NewCoolstreaming do not consider the available upload bandwidth at the parent node when selecting a new parent, so nodes change their parents more often. This is the reason for the slow convergence of playback continuity in NewCoolstreaming. Another reason for outperformance is the difference in policies used by a child to pull the first block from a new parent. In Sepidar, whenever a



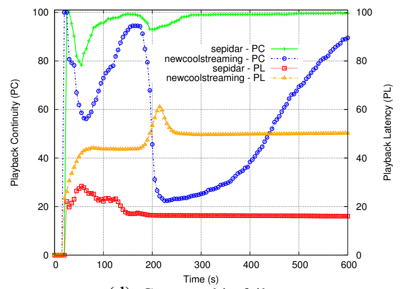
(a) Playback continuity of (non-)freerider nodes.



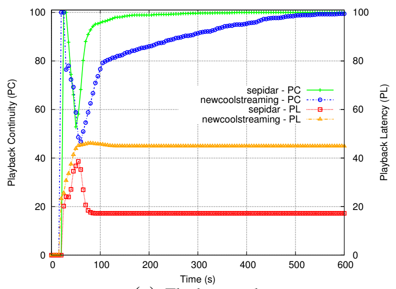
(b) Different freerider detector settings.



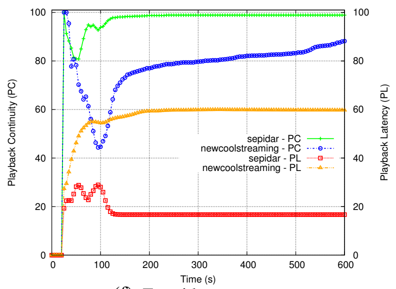
(c) Churn.



(d) Catastrophic failure.



(e) Flash crowd.



(f) Freeriders present.

Fig. 4. Figures 4(a) and 4(b) show the behaviour of Sepidar in different settings of the freerider detector, and the other figures show the performance of Sepidar and Newcoolstreaming in different scenarios.

node p selects a new parent q, p informs q of the last block it has in its buffer, and q sends subsequent blocks to p, while in NewCoolstreaming, the requested block is determined by looking at the buffer head of the partners [10]. This causes NewCoolstreaming to miss blocks when switching parent.

As we see in all the scenarios, NewCoolstreaming keeps its playback latency constant, which is because of reactively changing parents when nodes playback latency is greater than the predefined threshold. There is a trade-off between playback continuity and playback latency in NewCoolstreaming, such that lower playback latency results in lower playback continuity [10]. In Sepidar the nodes have higher playback latency in the beginning, but they decrease it very soon when they finds appropriate parents, by ignoring the missed blocks and fast forwarding the stream to play from the block where streaming from the new parent is resumed.

An important point of difference between the two systems is the behaviour of Sepidar and NewCoolstreaming upon an increase in the playback latency. In 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 [10].

E. Incentivizing nodes to contribute upload bandwidth

Here, we investigate the level of incentives for nodes to contribute more upload bandwidth by measuring the performance of the top 10% of upload bandwidth nodes (*strong* nodes) and the bottom 10% of upload bandwidth nodes (*weak* nodes). We use the churn scenario explained in section V-A. Since, weak nodes have lower upload bandwidth (and lower currency) compared to strong nodes, it takes longer for them to find an appropriate parent, and as a consequence their playback continuity decreases and their playback latency increases. Figure 5 compares the playback continuity and playback latency of strong nodes and weak nodes. As we can see, the strong nodes receive the stream with higher playback continuity and lower playback latency compared to weak nodes. Moreover, while there is churn in the system, we see less fluctuation in the playback continuity of strong nodes. As such, nodes are strongly incentivized to contribute more upload bandwidth through receiving improved relative performance.

F. Comparing the Gradient with random neighbor selection

In the last experiment, we measure the convergence speed of our market model, in terms of number of parent switches in the Gradient overlay and a random network. Again, we compare them using the churn scenario. Our market model is run using (i) samples taken from the Gradient overlay, where the sampled nodes have similar upload bandwidth or currency,

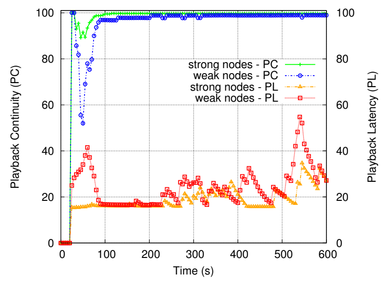


Fig. 5. Playback continuity and playback latency of strong nodes vs. weak nodes.

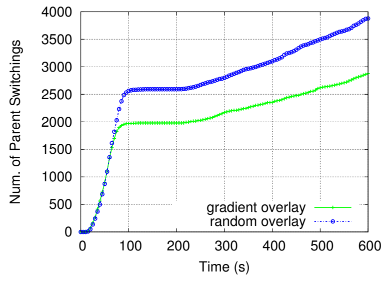


Fig. 6. CDF of number of parent switches.

and (ii) samples taken from a random network, where the sampled nodes have random amounts of currency. Since in the Gradient overlay, nodes receive bids from a set of nodes with almost the same currency, the difference between received bids is less than the expected difference for the random network. Figure 6 shows the CDF of number of parent switches for both overlays against time, and we can see that the Gradient overlay has a substantially lower number of parent switches.

VI. Conclusions

In this paper, we presented Sepidar, a P2P live streaming system that uses both the Gradient overlay and a distributed market-based approach to build multiple minimal height trees, where nodes with higher available upload bandwidth are positioned higher in the tree. Sepidar addresses the problem of free-riding through parent nodes auditing the behaviour of their children nodes by querying their grandchildren. We showed how the Gradient overlay helped nodes efficiently find good neighbours for building these streaming trees. Our simulations showed that, compared to NewCoolstreaming, Sepidar has higher playback continuity and lower playback latency.

References

[1] Cosmin Arad, Jim Dowling, and Seif 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] S. Asaduzzaman, Y. Qiao, and G. Bochmann. CliqueStream: an efficient and fault-resilient live streaming network on a clustered peer-to-peer overlay. In *Proceedings of the 2008 Eighth International Conference on Peer-to-Peer Computing*, pages 269–278. IEEE Computer Society, 2008.

[3] Suman Banerjee, Bobby Bhattacharjee, and Christopher Kommareddy. Scalable application layer multicast. In *SIGCOMM '02: Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 205–217, New York, NY, USA, 2002. ACM.

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

[5] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, Animesh Nandi, Antony Rowstron, and Atul 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.

[6] Krishna P. Gummadi, Stefan Saroiu, and Steven D. Gribble. King: Estimating latency between arbitrary internet end hosts. In *SIGCOMM Internet Measurement Workshop*, 2002.

[7] S.A. Jarvis, G. Tan, D.P. Spooner, and G.R. Nudd. Constructing Reliable and Efficient Overlays for P2P Live Media Streaming. In *21 st UK Performance Engineering Workshop*, page 31. Citeseer, 2005.

[8] Xuxian Jiang, Yu Dong, Dongyan Xu, and B. Bhargava. Gnustream: a p2p media streaming system prototype. In *ICME '03: Proceedings of the 2003 International Conference on Multimedia and Expo*, pages 325–328, Washington, DC, USA, 2003. IEEE Computer Society.

[9] H. W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistic Quarterly*, 2:83–97, 1955.

[10] 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.

[11] Zongpeng Li and Anirban Mahanti. A progressive flow auction approach for low-cost on-demand p2p media streaming. In *International conference on Quality of service in heterogeneous wired/wireless networks*, page 42, New York, NY, USA, 2006. ACM.

[12] Thomas Locher, Remo Meier, Stefan Schmid, and Roger Wattenhofer. Push-to-Pull Peer-to-Peer Live Streaming. In *21st International Symposium on Distributed Computing (DISC), Lemesos, Cyprus, Springer LNCS 4731*, September 2007.

[13] Yue1 Lu, Benny Fallica, Fernando Kuipers, Robert Kooij, and Piet Van Mieghem. Assessing the quality of experience of sopcast. *Journal of Internet Protocol Technology*, 4(1):11–23, 2009.

[14] Nazanin Magharei and Reza Rejaie. Prime: Peer-to-peer receiver-driven mesh-based streaming. In *INFOCOM*, 2007.

[15] 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.

[16] 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.

[17] J.J.D. Mol, J.A. Pouwelse, M. Meulpolder, D.H.J. Epema, and H.J. 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.

[18] Venkata N. Padmanabhan, Helen J. Wang, Philip A. Chou, and Kunwadee Sripanidkulchai. Distributing streaming media content using cooperative networking. In *NOSSDAV '02: Proceedings of the 12th international workshop on Network and operating systems support for digital audio and video*, pages 177–186, New York, NY, USA, 2002. ACM.

[19] Vinay Pai, Kapil Kumar, Karthik Tamilmani, Vinay Sambamurthy, Alexander E. Mohr, and Er E. Mohr. Chainsaw: Eliminating trees from overlay multicast. In *Workshop on Peer-to-Peer Systems (IPTPS)*, pages 127–140, 2005.

[20] K. Park, S. Pack, and T. Kwon. Climber: An incentive-based resilient peer-to-peer system for live streaming services. In *Workshop on Peer-to-Peer Systems (IPTPS)*, 2008.

[21] Amir H. Payberah, Jim Dowling, Fatemeh Rahimian, and Seif 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.

[22] Fabio Pianese, Joaqun Keller, and Ernst W. Biersack. Pulse, a flexible p2p live streaming system. In *INFOCOM*. IEEE, 2006.

[23] Jan Sacha, Bartosz Biskupski, Dominik Dahlem, Raymond Cunningham, René Meier, Jim Dowling, and Mads Haahr. Decentralising a service-oriented architecture. *Accepted for publication in Peer-to-Peer Networking and Applications*.

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

[25] Guang Tan and Stephen 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.

[26] Duc A. Tran, Kien A. Hua, and Tai T. Do. Zigzag: An efficient peer-to-peer scheme for media streaming. In *INFOCOM*, 2003.

[27] Cristina Nader Vasconcelos and Bodo 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.

[28] Vidhyashankar Venkataraman, Kaouru Yoshida, and Paul Francis. Chunkyspread: Heterogeneous unstructured tree-based peer-to-peer multicast. In *ICNP '06: Proceedings of the Proceedings of the 2006 IEEE International Conference on Network Protocols*, pages 2–11, Washington, DC, USA, 2006. IEEE Computer Society.

[29] 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.

[30] Feng Wang, Yongqiang Xiong, and Jiangchuan Liu. mtreebone: A hybrid tree/mesh overlay for application-layer live video multicast. In *ICDCS '07: Proceedings of the 27th International Conference on Distributed Computing Systems*, page 49, 2007.

[31] Douglas B. West. *Introduction to Graph Theory (2nd Edition)*. Prentice Hall, August 2000.

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

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

[34] Xinyan Zhang, Jiangchuan Liu, Bo Li, and Tak shing Peter Yum. Coolstreaming/donet: A data-driven overlay network for peer-to-peer live media streaming. In *IEEE Infocom*, 2005.

[35] Runfang Zhou, Kai Hwang, and Min Cai. Gossiptrust for fast reputation aggregation in peer-to-peer networks. *IEEE Trans. on Knowl. and Data Eng.*, 20(9):1282–1295, 2008.