gradienTv: Market-Based P2P Live Media Streaming on the Gradient Overlay

Amir H. Payberah, Jim Dowling, Fatemeh Rahimian, Seif Haridi

gradienTv: Market-based P2P live media streaming on the Gradient overlay

Amir H. Payberah[1,2], Jim Dowling[1], Fatemeh Rahimian[1,2], and Seif Haridi[1,2]

[1] Swedish Institute of Computer Science (SICS)
[2] KTH - Royal Institute of Technology

Abstract. This paper presents gradienTv, a distributed, market-based approach to live streaming. In gradienTv, multiple streaming trees are constructed using a market-based approach, such that nodes with increasing upload bandwidth are located closer to the media source at the roots of the trees. Market-based approaches, however, exhibit slow convergence properties on random overlay networks, so to facilitate the timely discovery of neighbours with similar upload bandwidth capacities (thus, enabling faster convergence of streaming trees), we use the gossip-generated Gradient overlay network. In the Gradient overlay, nodes are ordered by a gradient of node upload capacities and the media source is the highest point in the gradient. We compare gradienTv with state-of-the-art NewCoolstreaming in simulation, and the results show significantly improved bandwidth utilization, playback latency, playback continuity, and reduction in the average number of hops from the media source to nodes.

1 Introduction

Live streaming using overlay networks is a challenging problem. It requires distributed algorithms that, in a heterogeneous network environment, improve system performance by maximizing the nodes' upload bandwidth utilization, and improve user viewing experience by minimizing the playback latency, and maximizing the playback continuity of the stream at nodes.

In this paper, we improve on the state-of-the-art NewCoolstreaming system [7] for these requirements by building multiple media streaming overlay trees, where each tree delivers a part of the stream. The trees are constructed using distributed algorithms such that a node's depth in each tree is inversely proportional to its relative available upload bandwidth. That is, nodes with relatively higher upload bandwidth end up closer to the media source(s), at the root of each tree. This reduces load on the source, maximizes the utilization of available upload bandwidth at nodes, and builds lower height trees (reducing the number of hops from nodes to the source). Although we only consider upload bandwidth for constructing the Gradient overlay in this paper, the model can easily be extended to include other important node characteristics such as node uptime, load and reputation.

Our system, called *gradienTv*, uses a market-based approach to construct multiple streaming overlay trees. Firstly, the media source splits the stream into a set of sub-streams, called *stripes*, and divides each stripe into a number of *blocks*. Sub-streams allow more nodes to contribute bandwidth and enable more robust systems through

redundancy [4]. Nodes in the system compete to become *children* of nodes that are closer to the root (the media source), and *parents* prefer children nodes who offer to forward the highest number of copies of the stripes. A child node explicitly requests and *pulls* the first block it requires in a stripe from its parent. The parent then *pushes* to the child subsequent blocks in the stripe, as long as it remains the child's parent. Children can proactively switch parent when the market-modelled benefit of switching is greater than the cost of switching.

The challenge with implementing this market-based approach is to find the best possible matching between parents and children in a timely manner, while having as few parent switches as possible. In general, for a market-based system to work efficiently, information and prices need to be spread quickly between participants. Insufficient information at market participants results in inefficient markets. In a market implemented using an overlay network, where the nodes are market participants, the communication of information and prices between nodes is expensive. For example, finding the optimal parent for each node requires, in principle, flooding to communicate with all other nodes in the system. Flooding, however, is not scalable. Alternatively, an approach to find parents based on random walks or sampling from a random overlay produces slow convergence time for the market and results in excessive parent switching, as information only spreads slowly in the market. We present a fast, approximate solution to this problem based on the *Gradient overlay* [17]. The Gradient is a gossip-generated overlay network, built by sampling from a random overlay, 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. A node's neighbours in the Gradient have similar, or slightly higher upload bandwidth. The Gradient, therefore, efficiently acts as a *market maker* that matches up nodes with similar upload bandwidths, enabling the market mechanisms to quickly construct stable streaming overlay trees. As nodes with low relative upload bandwidths are rarely matched with nodes with high relative upload bandwidths (as can be the case in a random overlay), there is significantly less parent-switching before streaming overlay trees converge.

We evaluate gradienTv by comparison with NewCoolstreaming, a successful and widely used media streaming solution. We show in simulation that our market-based approach ensures that the system's upload bandwidth can be near maximally utilized, the playback continuity at clients is improved compared to NewCoolstreaming, the height of the media streaming trees constructed is much lower than in NewCoolstreaming, and, as a consequence, playback latency is less than NewCoolstreaming.

2 Related work

There are two fundamental problems in building data delivery (media streaming) overlay networks: (i) what overlay topology is built for data dissemination, and (ii) how a node discovers 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 [14], ZigZag [18] and NICE [3]. The short latency of data delivery is the main advantage of

this approach [24]. 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 [4] improved this model by using multiple trees, where the stream is split into sub-streams and each tree delivers one sub-stream. Orchard [11], ChunkySpread [19] and CoopNet [12] are some other solutions in this class.

An alternative to tree structured overlays is mesh structure, in which the nodes are connected in a mesh-network [24], 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 [24]. SopCast [9], DONet/Coolstreaming [25], Chainsaw [13], BiToS [20] and PULSE [15] 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 [22], NewCoolStreaming [7], Prime [10] and [8]. GradienTv belongs to this class, where the mesh is the Gradient overlay.

The second fundamental problem is how nodes discover the other nodes that supply the stream. CoopNet [12] uses a centralized coordinator, GnuStream [6] uses controlled flooding requests, SplitStream [4] and [8] use DHTs, while NewCoolstreaming [7], DONet/Coolstreaming [25] and PULSE [15] use a gossip-generated random overlay network to search for the nodes.

NewCoolstreaming [7] has the most similarities with gradienTv. 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, gradienTv's use of the Gradient overlay to discover nodes to supply the stream contrasts with NewCoolStreaming that samples nodes from a random overlay (referred to as the partner-list). A second major difference is that NewCoolStreaming only reactively changes a parent when a sub-stream is identified as being slow, whereas gradienTv proactively changes parents to improve system performance.

3 Gradient overlay

The Gradient overlay is a class of P2P overlays that arrange 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 [16, 17]. As can be seen in Figure 1, the highest utility nodes (darkest colour) are found at the core of the Gradient, and nodes with decreasing utility values (lighter grays) are found at increasing distance from the centre.

The Gradient maintains two sets of neighbours using gossiping algorithms: a *similar-view* and a *random-view*. The similar-view of a node is a partial view of the nodes whose utility values are close to, but slightly higher than, the utility value of this node. Nodes periodically gossip with each other and exchange their similar-views. Upon receiving a similar-view, a node updates its own similar-view by replacing its entries with those nodes that have closer (but higher) utility value to its own utility value. In contrast, the random-view constitutes a random sample of nodes in the system, and it is used both to discover new nodes for the similar-view and to prevent partitioning of the similar-view.

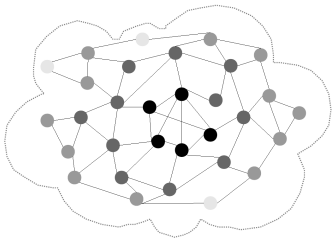


Fig. 1. Gradient overlay network

4 GradienTv system

In gradienTv, the media source splits the media into a number of *stripes* and divides each stripe into a sequence of *blocks*. GradienTv constructs a media streaming overlay tree for each stripe, where blocks are pushed from parents to children. Newly joined nodes discover stripe providers using the Gradient overlay and compete with each other to establish a parent-child relationship with providers. A node proactively changes its parent for a stripe, if it finds a lower depth parent for that stripe and if that parent either has a free *upload slot* or prefers this node to one of its existing children.

We use the term *download slot* to define a network connection at a node used to download a stripe. Likewise, an upload slot refers to a network connection at a node that is used to forward a stripe. If node p assigns its upload slot to node q's download slot, we say p is the *parent* of q and q is the *child* of p.

Our market model uses the following three properties, calculated at each node, to match nodes that can forward a stripe with nodes that want to download that stripe:

1. *Currency*: the total number of upload slots at a node, that is, the number of stripes a node is willing and able to forward simultaneously. A node uses its currency when requesting to connect to another node's upload slot.
2. *Connection cost*: the minimum currency that should be provided for establishing a connection to receive a stripe. The connection cost to a node that has an unused upload slot is zero, otherwise the node's connection cost 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 connection cost of p is 2.
3. *Depth*: the shortest path (number of hops) from a node to the root for a particular stripe. Since the media stream consists of several stripes, nodes may have different depths in different trees. The lower the depth a node has for a stripe, the more desirable a parent it is for that stripe. Nodes constantly try to reduce their depth over all their stripes by competing with other nodes for connections to lower depth nodes.

4.1 Gradient overlay construction

Each node maintains two sets of neighbouring nodes: a random-view and a similar-view. Cyclon [21] is used to create and update the random-view and a modified version

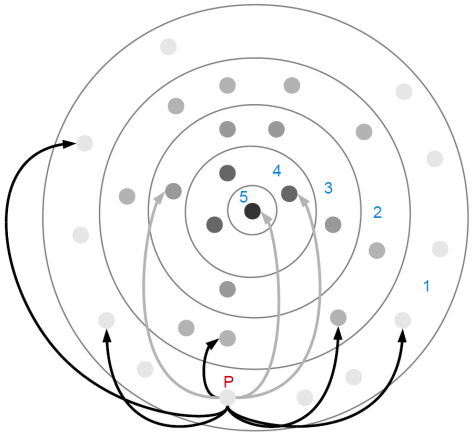


Fig. 2. Different market-levels of a system, the similar-view of node p and its fingers

of the Gradient protocol is used to build and update the similar-view. The node references stored in each view contain the *utility value* for the nodes. 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 that have the same utility value. For example, in figure 2, we define some 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, Fibre ($>$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, a node with 256 $Kbps$ upload bandwidth falls into slow DSL market-level, so its utility value is 2.

A node prefers to fill its similar-view with the 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). In Figure 2, 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.

Nodes bootstrap their similar-view using a bootstrap server, and, initially, the similar-view of a node is filled with random nodes that have equal or higher utility value. Algorithm 1 is executed periodically by the node p to maintain its similar-view. The algorithm describes how on every round, p increments the age of all the nodes in its similar-view. It removes the oldest node, q, from its similar-view and sends a subset of nodes in its similar-view to q (lines 3-6). Node q responds by sending back a subset of

Algorithm 1 Updating the similar-view

```
1:  procedure UpdateSimilarView ⟨this⟩
2:      this.similarView.updateAge()
3:      q ← oldest node from this.similarView
4:      this.similarView.remove(q)
5:      pView ← this.similarView.subset()                ▷ a random subset from p's similarView
6:      Send pView to q
7:      Recv qView from q                                ▷ qView is a subset of q's similarView
8:      for all node_i in qView do
9:          if U_p(node_i) = U(p) OR U_p(node_i) = U(p) + 1 then
10:             if this.similarView.contains(node_i) then
11:                 this.similarView.updateAge(node_i)
12:             else if this.similarView has free entries then
13:                 this.simialrView.add(node_i)
14:             else
15:                 node_j ← pView.poll()                ▷ get and remove one entry from pView
16:                 this.similarView.remove(node_j)
17:                 this.simialrView.add(node_i)
18:             end if
19:         end if
20:     end for
21:     for all node_a in this.randomView do
22:         if U_p(node_a) = U(p) OR U_p(node_a) = U(p) + 1 then
23:             if this.similarView has free entries then
24:                 this.simialrView.add(node_a)
25:             else
26:                 node_b ← (x ∈ this.similarView such that U_p(x) > U(p) + 1)
27:                 if (node_b ≠ null) then
28:                     this.similarView.remove(node_b)
29:                     this.simialrView.add(node_a)
30:                 end if
31:             end if
32:         end if
33:     end for
34: end procedure
```

Algorithm 2 Parent assignment

```
1:  procedure assignParent ⟨⟩
2:      for all stripe_i in stripes do
3:          candidates ← findParent(i)
4:          if candidates ≠ null then
5:              newParent ← a random node from candidates
6:              send ⟨ASSIGNREQUEST | i⟩ to newParent
7:          end if
8:      end for
9:  end procedure
```

Algorithm 3 Select candidate parent from the similar-view and the fingers

```
1:  procedure findParent ⟨i⟩
2:      candidates ← ∅
3:      if this.stripe_i.parnet = null then
4:          this.stripe_i.parnet.depth ← ∞
5:      end if
6:      for all node_j in (similarView ⋃ fingers) do
7:          if node_j.stripe_i.depth < this.stripe_i.parent.depth
8:           AND node_j.connectionCost < this.currency then
9:              candidates.add(node_j)
10:         end if
11:     end for
12:     return candidates
13: end procedure
```

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 (lines 8-20). 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 more than p's utility value plus one (lines 21-33).

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.

4.2 Streaming tree overlay construction

Algorithm 2 is called periodically by nodes to build and maintain a streaming overlay tree for each stripe. For each stripe i, a node p checks if it has a node in its similar-view or finger list that has (i) a lower depth than its current parent, and (ii) a connection cost less than p's currency. If such a node is found, it is a added to a list of candidate parents for stripe i (Algorithm 3). Next, we use a random policy to select a node from the candidate parents, as it fairly balances connection requests over nodes in the system. In contrast, if we select the candidate parent with the minimum depth, then for even low variance in currency of nodes, it causes excessive connection requests to those nodes with high upload bandwidth.

Algorithm 4 Handling the assign request

```
1:  upon event ⟨AssignRequest | i⟩ from p
2:      if has free uploadSlot then
3:          assign an uploadSlot to p
4:          send ⟨assignAccepted | i⟩ to p
5:      else
6:          worstChild ← lowest currency child
7:          if worstChild.currency ≥ p.currency then
8:              send ⟨assignNotAccepted | i⟩ to p
9:          else
10:             assign an uploadSlot to p
11:             send ⟨release | i⟩ to worstChild
12:             send ⟨assignAccepted | i⟩ to p
13:         end if
14:     end if
15: end event
```

Algorithm 4 is called whenever a receiver node q receives a connection request from node p. If q has a free upload slot, it accepts the request, otherwise if p's currency is greater than the connection cost of q, q abandons one of its children with the lowest currency and accepts p as a new child. In this case, the abandoned node has to find a new parent. If q's connection cost is greater than p's currency, q declines the request.

5 Experiments and evaluation

In this section, we compare the performance of gradienTv with NewCoolstreaming under simulation. In summary, we define three different experiment scenarios: join-only, flash-crowds, and catastrophic failure, and, we show that gradienTv outperforms NewCoolstreaming in all of these scenarios for the following metrics: playback continuity, bandwidth utilization, playback latency, and path length. [1]

Experiment setup
We have implemented both gradienTV and NewCoolstreaming using the Kompics platform [1]. Kompics provides a framework for building P2P protocols, and simulation support using a discrete event simulator. Our implementation of NewCoolstreaming is based on the system description in [7, 23]. We have validated our implementation of NewCoolstreaming by replicating, in simulation, the results from [7].

In our experimental setup, we set the streaming rate to 512 $Kbps$ and unless stated otherwise, experiments involve 1000 nodes. The stream is split into 4 stripes and each stripe is divided into a sequence of 128 KB blocks. The media source is a single node with 40 upload slots. Nodes start playing the media after buffering it for 30 seconds. This is comparable with the most widely deployed P2P live streaming system, SopCast's that has average startup time of 30-45 seconds [9]. The size of a node's partial view (the similar-view in gradienTv, the partner list in NewCoolstreaming) is 15 nodes.

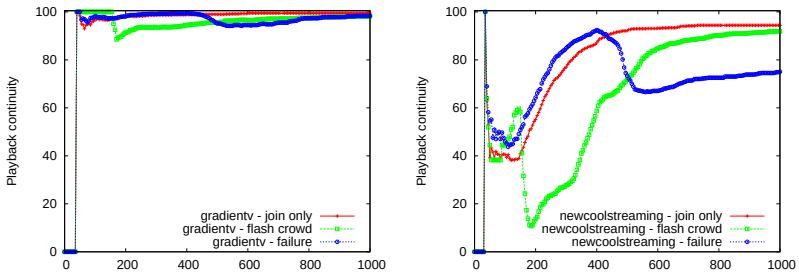
The number of upload slots for the non-root nodes is picked randomly from 1 to 10, which corresponds to upload bandwidths from 128 $Kbps$ to 1.25 $Mbps$. As the average upload bandwidth of 704 $Kbps$ is not much higher than the streaming rate of 512 $Kbps$, nodes have to find good matches as parents in order for good streaming performance. We assume all the nodes have enough download bandwidth to receive all the stripes simultaneously. In gradienTv, 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 one upload slot (128 $Kbps$) are the members of the first market-level, nodes with two upload slots (256 $Kbps$) are located in the second market-level, and the media source with 40 upload slots ($>$5 $Mbps$) is the only member of the 11th market-level.

Latencies between nodes are modelled using a latency map based on the King dataset [5]. 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. *Bandwidth utilization*: the ratio of the total number of utilized upload slots to the total number of requested download slots;
3. *Playback latency*: the difference in seconds between the playback point of a node and the playback point at the media source;
4. *Path length*: the minimum distance in number of hops between the media source and a node for a stripe.

We compare our system with NewCoolstreaming using the following scenarios:

[1] The source code and the results are available at: http://www.sics.se/∼amir/gradientv



(a) gradienTv. (b) NewCoolstreaming.

Fig. 3. Playback continuity in percent (Y-axis), against time in seconds (X-axis).

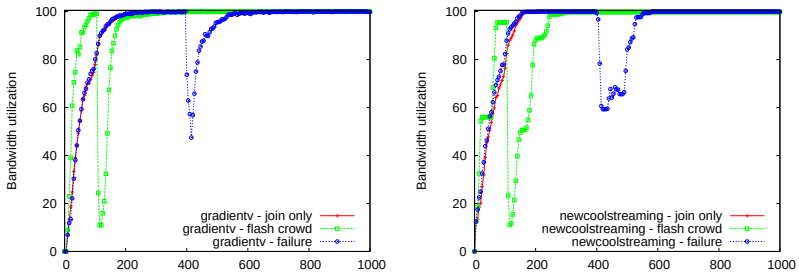
1. *Join-only*: 1000 nodes join the system following a Poisson distribution with an average inter-arrival time of 100 milliseconds;
2. *Flash crowd*: first, 100 nodes join the system following a Poisson distribution with an average inter-arrival time of 100 milliseconds. Then, 1000 nodes join following the same distribution with a shortened average inter-arrival time of 10 milliseconds;
3. *Catastrophic failure*: as in the join-only scenario, 1000 nodes join the system following a Poisson distribution with an average inter-arrival time of 100 milliseconds. Then, 400 existing nodes fail following a Poisson distribution with an average inter-arrival time 10 milliseconds. The system then continues its operation with only 600 nodes.

In addition to these scenarios, we also evaluate the behaviour of gradienTv when varying two key parameters: (i) the playback buffering time and (ii) the number of nodes.

Playback Continuity

In this section, we compare the playback continuity of gradienTv and NewCoolstreaming in three different scenarios: join-only, flash crowd and catastrophic failure. In figures 3(a) and 3(b), the X-axis shows the time in seconds, while the Y-axis shows the percentage of the nodes in the overlay that have a playback continuity more than 90%. We can see that gradienTv significantly outperforms NewCoolstreaming for the whole duration of the experiment in all scenarios. Moreover, after the system stabilizes, we observe a full playback continuity in gradienTv. This out-performance is due to the faster convergence of the streaming overlay trees in gradienTv, where 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. Another reason for out-performance is the difference in policies used by a child to pull the first block from a new parent. In gradienTv, whenever a 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 head of the partners. This causes NewCoolstreaming to miss blocks when switching parent.

Bandwidth Utilization
Our second experiment compares the bandwidth utilization of gradienTv (figure 4(a)) and NewCoolstreaming (figure 4(b)). We observe that when the system has no churn, as in the join-only scenario, both systems equally utilized the bandwidth. In the flash crowd and catastrophic failure scenarios, the performance of the both systems drops significantly. However, gradienTv recovers faster, as nodes are able to find parents more quickly using the Gradient overlay.



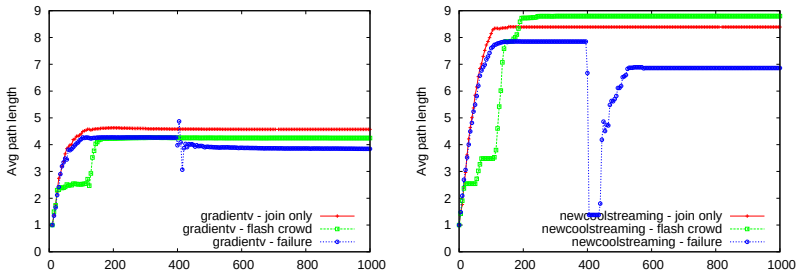
(a) gradienTv. (b) NewCoolstreaming.

Fig. 4. Bandwidth utilization in percent (Y-axis), against time in seconds (X-axis).

Path Length
In the third experiment, we compare the average path length of both streaming overlays. Before looking at the experiment results, we calculate the minimum depth of a k-ary tree with n nodes using $log_k(n)$. In our experiments, there are on average 5 upload slots per node (as upload slots are uniformly distributed from 1 to 10), and the minimum depth of the trees is expected to be $log_5(1000) \approx 4.29$. Figures 5(a) and 5(b) show tree depth of the system for gradienTv and NewCoolstreaming. We observe that gradienTv constructs trees with an average height of 4.3, which is very close to the minimum height. The figures also show that the depth of the trees in gradienTv are half the depth of the trees in NewCoolstreaming. Shorter trees enable lower playback latency.

What is more, we observe that the average depth of the trees is independent of the inter-arrival time of the joining nodes. This can be seen in figures 5(a) and 5(b), where the depth of the trees, after the system stabilizes, is the same. More interestingly, in the catastrophic failure scenario, we can see a sharp drop in NewCoolstreaming tree depth, as a result of the drop in the number of nodes remaining in the system and the fact that many remaining nodes do not have any path to the media source. The same behaviour is observed in gradienTv, but since the nodes can find appropriate nodes to connect to more quickly, the fluctuation in the average depth of trees is less than in NewCoolstreaming.

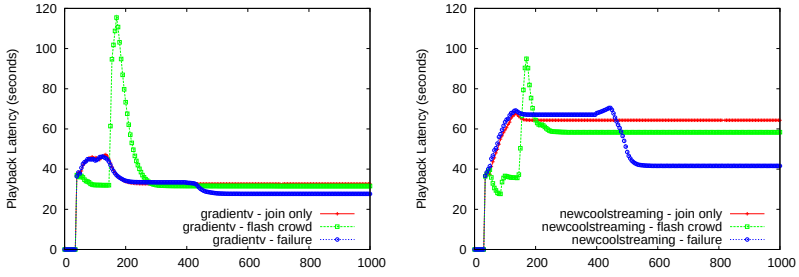
Playback Latency
This experiment shows how the average playback latency of nodes changes over time



(a) gradienTv. (b) NewCoolstreaming.

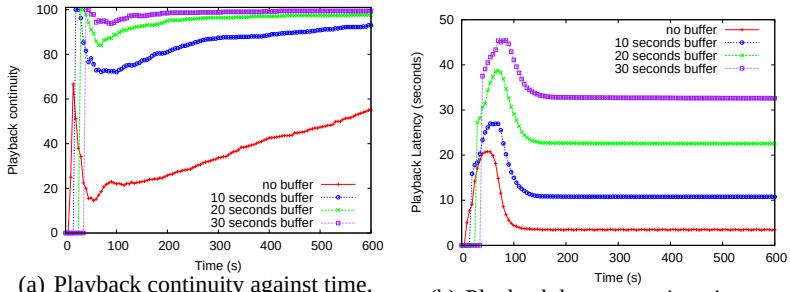
Fig. 5. Average path length in number of hops (Y-axis), against time in seconds (X-axis).

in our three scenarios (figures 6(a) and 6(b)). In the join-only scenario, we can see that 200 seconds after starting the simulation, the playback latency in gradienTv converges to just over 30 seconds, close to the initial buffering time, set at 30 seconds. For the join-only scenario, gradienTv exhibits lower average playback latency than NewCoolstreaming. This is because its streaming trees have lower depth, and, therefore, nodes receive blocks earlier than in NewCoolstreaming. This is also the case for the two other experiment scenarios, flash crowd and catastrophic failure. Here, we can see an increase in the average playback latency for both systems. This is due to the increased demand for parents by new nodes and nodes with failed parents. While the nodes are competing for parents, they may fail to receive the media blocks in time for playback. Therefore, they have to pause until a parent is found and the streaming is resumed. This results in higher playback latency. Nevertheless, when both systems stabilize, nodes will ignore the missing blocks and fast forward to the play from the block where the streaming from the new parent is resumed. Hence, the playback latency will improve after the system has settled down.



(a) gradienTv. (b) NewCoolstreaming.

Fig. 6. Average playback latency in seconds (Y-axis), against time in seconds (X-axis).



(a) Playback continuity against time.

(b) Playback latency against time.

Fig. 7. The behaviour of gradienTv for different playback buffer lengths (in seconds).

There is a significant difference between the behaviour of gradienTv and NewCoolstreaming upon an increase in the playback latency. In gradienTv, 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) causing it to miss blocks, thus it negatively affects playback continuity.

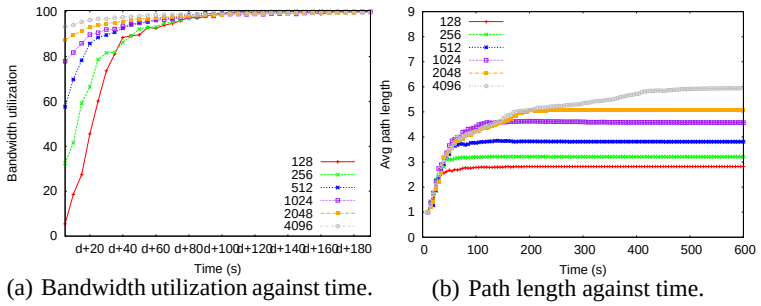
Buffering Time

We now evaluate the behaviour of gradienTv for different initial playback buffering times. We compare four different settings: 0, 10, 20 and 30 seconds of initial buffering time. Two metrics that are affected by changing the initial buffering time are playback continuity and playback latency. Figure 7(a) shows that when there is no initial buffering, the playback continuity drops to under 20% after 50 seconds of playback, but as the system stabilizes the playback continuity increases. Buffering 10 seconds of blocks in advance results in less playback interruptions when nodes change their parents, but better playback continuity is achieved for 20 and 30 seconds of buffering. Figure 7(b) shows how 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.

Number of Nodes

In this experiment, we evaluate the performance of the system for different system sizes. We simulate systems with 128, 256, 512, 1024, 2048, and 4096 nodes, where nodes join the system following a Poisson distribution with an average inter-arrival time of 100 milliseconds. In figure 8(a), we show the bandwidth utilization after all the nodes have joined (for the different system sizes). We define d as the time when all nodes have joined for a particular size. This means that for the system with 128 nodes, d is 13 seconds, while for the system with 4096 nodes d is 410 seconds. This experiment shows that, regardless of system size, nodes successfully utilize the upload slots at other nodes. This implies that convergence in terms of matching upload slots to download slots, appears to be independent of the number of nodes in the system. A

necessary condition, of course, is that there is enough available upload and download bandwidth to deliver the stream to all nodes.



(a) Bandwidth utilization against time. (b) Path length against time.

Fig. 8. Bandwidth utilization and path length for varying numbers of nodes

In the second experiment, we measure the tree depth while varying system sizes. We can see in figure 8(b) that the depth of the trees are very close to the theoretical minimum depth in each scenario. For example, the average depth of the trees with 1024 nodes is 4.34, which is very close to $log_5(1024) \approx 4.30$.

6 Conclusions

In this paper, we presented gradienTv, a P2P live streaming system that uses both the Gradient overlay and a market-based approach to build multiple streaming trees. The constructed streaming trees had the property that the higher a node's upload capacity, the closer that node is to the root of the tree. We showed how the Gradient overlay helped nodes efficiently find good neighbours for building these streaming trees. Our simulations showed that, compared to NewCoolstreaming, gradienTv has higher playback continuity, builds lower-depth streaming trees, has better bandwidth utilization performance, 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. 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.
5. Krishna P. Gummadi, Stefan Saroiu, and Steven D. Gribble. King: Estimating latency between arbitrary internet end hosts. In *SIGCOMM Internet Measurement Workshop*, 2002.
6. 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.
7. 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.
8. 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.
9. 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.
10. Nazanin Magharei and Reza Rejaie. Prime: Peer-to-peer receiver-driven mesh-based streaming. In *INFOCOM*, 2007.
11. 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.
12. 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.
13. 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.
14. 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.
15. Fabio Pianese, Joaqun Keller, and Ernst W. Biersack. Pulse, a flexible p2p live streaming system. In *INFOCOM*. IEEE, 2006.
16. 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*.
17. 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.
18. Duc A. Tran, Kien A. Hua, and Tai T. Do. Zigzag: An efficient peer-to-peer scheme for media streaming. In *INFOCOM*, 2003.
19. 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.
20. Aggelos Vlavianos, Marios Iliofotou, and Michalis Faloutsos. Bitos: enhancing bittorrent for supporting streaming applications. In *In IEEE Global Internet*, pages 1–6, 2006.
21. 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.
22. 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.
23. S. Xie, B. Li, G.Y. Keung, and X. Zhang. Coolstreaming: Design, Theory and Practice. *IEEE Transactions on Multimedia*, 9(8):1661, 2007.
24. 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.
25. 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.