A Topologically-Aware Overlay Tree for Efficient and Low-Latency Media Streaming

Paris Carbone[1] and Vana Kalogeraki[1,2]

[1] Department of Informatics, Athens University of Economics and Business
[2] Department of Computer Science and Engineering, University of California-Riverside

Abstract. Streaming a live music concert over the Internet is a challenging task as it requires real-time, high-quality data delivery over a large number of geographically distributed nodes. In this paper we propose MusiCast, a real-time peer-to-peer multicast system for streaming midi events and compressed audio data. We present a scalable and distributed tree construction algorithm where nodes across the Internet self-organize into a low-latency tree. Our system is built ontop of the pastry DHT and takes advantage of the DHT's properties to construct an end-to-end low-latency dissemination tree using topology oriented information. The benefit of our scheme is that it is completely decentralized, allowing nodes to connect to each other using local information only, and achieves good performance by considering latency information when constructing the tree. Our experimental results illustrate the benefits of our approach.

Keywords: Overlay Networks, Multimedia Streaming.

1 Introduction

In the past few years, peer-to-peer multicast services have received a growing acceptance over traditional methods such as IP multicast that has been the de facto mechanism for delivering data streams to a large number of participants. Peer-to-peer systems offer a number of attractive characteristics including adaptivity, scalability and robustness, properties of increased importance with the growing popularity of the Internet today and the increasing interest for online multimedia content distribution.

However, multimedia streaming brings a number of challenges to the design of peer-to-peer multicast systems: First, multimedia data streams are produced in large volumes and high rates by a small set of sources to a large number of receivers. For these applications, low-latency delivery of the streaming data is of paramount importance. Second, such an application layer multicast system consists of a number of nodes that are geographically distributed, thus, the multicast system must consider not only the delay but also the topology of the nodes and their geographic proximity. Third, computational and communication resources are shared by multiple concurrent and competing streams; this poses certain restrictions on the number of connections of the peers.

A number of multicast streaming systems have been proposed in the literature. Most systems have made significant contributions on load balance and content distribution [3,5,6,9], while there are also a few examples [8] targeting latency requirements and low cost tree construction. End-to-end latency and cost measurements require relative topological approximations and frequently the combination of different optimal structures (min cost, min path trees, meshes etc.). There are also many challenges that need to be faced when dealing with delay metrics such as peer churn, node failures and dynamically changing application requirements.

In this paper we present MusiCast, a topologically-aware peer-to-peer multicast system. MusiCast builds upon the idea of making possible a live music concert, consisting of nodes with different roles (*i.e.* musicians, spectator nodes) to cooperate and transmit music over the Internet. The key challenge in the design of MusiCast is to construct a tree structure that distributes the load across all participating nodes and achieves this in a decentralized and scalable manner. We present a distributed tree construction algorithm where nodes organize into a topologically-aware, low-latency overlay tree. MusiCast is built ontop of the Pastry DHT [15] taking advantage of its properties. The advantage of our scheme is that it achieves high data delivery ratios and low end-to-end latencies. MusiCast offers robustness to node failures and disconnections; thus, the failure of a node does not affect the performance of the rest of the system. We have implemented MusiCast on a local area testbed and evaluated its performance on various metrics including end-to-end delay, jitter and bandwidth used. Our experimental results demonstrate the efficiency and performance of our approach.

The rest of the paper is organized as follows. Section 2 presents our system model and overview of our approach. In section 3 we describe the system components, the tree construction algorithm, the run-time operation of our algorithm and our approach to failure recovery. In section 4 we describe our performance evaluation. Section 5 presents related work and section 6 concludes the paper.

2 Problem Formulation

In this section we first present our system model and then we give an overview of our approach.

2.1 Our System Model

The overall system consists of a set of overlay nodes N , divided into two different categories. Each node category represents a different layer of quality requirements and constraints that need to be taken into account by the system.

– $M \subset N$: Musician nodes are the main data sources of the system and are responsible for streaming midi or audio data. The number of musician nodes is typically small ($\|M\| = [1, 10]$), following a typical music band size. The main requirement of the M nodes is to maintain low playback latency to synchronize among themselves in order to achieve continuous and smooth delivery of the data streams.

Table 1. Notations explanation

Notation	Meaning
V	A set of all participating spectator nodes in the system
J	A set of nodes currently joined in the multicast tree
K	Candidate parents set, $k_i \in K : k_i \in J$ and $S(k_i) > 0$
v_i	A node $v_i \in V$
$p(v_i)$	The parent node of v_i on the multicast tree
$C(v_i)$	The set of children of v_i on the multicast tree
$d(v_i, v_j)$	distance between nodes v_i and v_j
$l(v_i)$	Tree path distance v_i from the tree root c
B	Bandwidth required by stream
$r(v_i)$	outgoing bandwidth of v_i
$S(v_i)$	The number of available slots in v_i

– $S \subset N$: Spectator nodes receive, forward and playback data streams as they are generated by the sources. The number of Spectator nodes can range from a few hundred to a few thousand nodes. Spectators' demands are the most challenging ones due to the scale of the spectators' subsystem. Their goals include: low delay in tree construction, small latency, minimum jitter and load balancing.

  One of the Spectator nodes takes the role of the Coordinator c, which is responsible for musicians' synchronization, main sequence composition and tree construction initialization control. The coordinator also serves as the root of the multicast tree.

We assume that the overlay network of the spectators is represented as a graph $G = (V, E)$, where V is the set of all the spectator nodes, including the coordinator and $E = VxV$ is the set of the edges between the nodes. The weight of each edge $< u_i, u_j >$ represents the distance $d(u_i, u_j)$ between the two nodes. We will assume here that $d(u_i, u_j)$ denotes the actual unicast delay between u_i and u_j as the distance metric but it could also be replaced by other metrics. We assume that each node u_i has a maximum number of connections, also reffered to as slots of u_i, $S(u_i)$. The maximum number of slots each node can handle is $\frac{r(u_i)}{B}$, where $r(u_i)$ describes the maximum outgoing bandwidth of u_i and B specifies the bitrate of the overall streaming sequence. Every node should offer at least one slot in order to join the system, so $S(u_i) > 0$. In order to disseminate a live data stream to all spectator nodes effectively, it is required to construct a spanning tree on G in which: (a) the degree constraints are satisfied and (b) the maximum end-to-end delay from the root to each node is minimized.

One important question is how to measure the distance between the nodes. We use a distributed distance measurement scheme based on the binning scheme proposed in [1]. The disadvantage of using other approaches where the distance between each pair of nodes is obtained using active end-to-end measurements, is that, the system would not be scalable due to the increased number of measurements needed ($O(n^2)$) and the lack of network bandwidth required for such consumable operations. We note, however, that this is an NP-hard problem, however, our approach manages to calculate the distance between the nodes efficiently, as we will explain later in the paper.

2.2 Approach Overview

Our approach is two-fold: (a) First, at an initialization phase, our goal is, given a number of musician and spectator nodes, to construct a low-latency tree structure that offers small latency and a good load balance while respecting the network constraints at the nodes. The advantage of our approach compared to past techniques such as the ones proposed in Bullet, MeshTree and Coolstreaming [2,3,8], is that they often start by building random trees and then using an increasing amount of messages in a greedy order, aim to transform it into a low-delay tree. (b) Second, at the run-time phase, we employ optimization techniques in order to reduce the average delay to respond to dynamic changes to peer churn and resource availability. This will enable us to reduce unwanted jitter caused by joining or departing nodes that can affect the playback quality throughout the entire tree.

3 System Overview

In this section we discuss the operation of our system. MusiCast consists of musician and spectator nodes. We first describe the operation of the musician and spectator subsystems. Then we discuss our distributed distance measurement scheme and how topological awareness is accomplished using content stored at the peers, followed by the discription of the tree construction algorithm used, our run-time optimizations and how we deal with failures. The architecture of our system is illustrated in Figure 1.

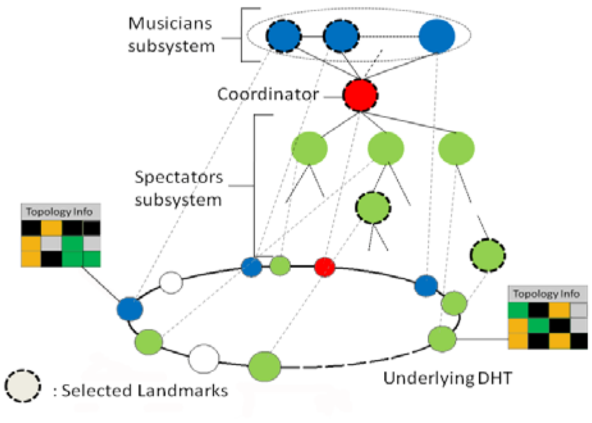

Fig. 1. Architecture of our System

3.1 The Musicians Subsystem

Each musician is responsible for streaming on a specific midi or audio channel while reproducing the sequences received from other musicians in real time. There are strict latency constraints that need to be taken into account. Event messages generated by each musician need to get to all other musicians (and to the coordinator) as soon as possible, without any extra delays, in order to achieve a responsive and effective playback. That is why we chose to establish a broadcasting scheme between them using the many-to-many unicast technique. Our goal is to keep low latencies, as large latencies could cause confusion to musicians, thus lowering the quality of their performance. Also, the number of musicians is ordinarily small and so as the bit rate of compressed audio and midi packets, so this technique is suitable for this specific setting. All midi packets received on each musician are instantly directed for playback without further buffering while compressed audio packets are buffered to the lowest degree. Control messages and basic synchronization between musicians are handled by the coordinator node. By synchronization we mean the maintenance of a global accurate timing, rate and channel distribution between them. The coordinator gives to each musician its global timing offset at startup with the use of the NTPv4 protocol which is shown to achieve an accuracy of 1-2ms in a LAN infrastructure or 8-10ms in a WAN. It is also the coordinator which instructs the musicians about the exact global time each one will start streaming on its specifically given channel. Each sequence stream from the musicians is also forwarded to the coordinator node which synthesizes the main sequence by aggregating all packets on top of the NTP timing protocol. This main sequence is being disseminated to the spectators' multicast tree from there.

3.2 The Spectators Subsystem

The spectators' subsystem consists of the spectator nodes and the coordinator which serves as the main source. Our goal is to organize the spectator nodes into a low latency multicast tree and achieve minimum overhead for control and optimizations.

The initial tree structure is an important decision, because despite optimizations, a reliable system is ought to guarantee high quality services from the point it begins operating. Many popular recent end-to-end multicast systems such as Bullet and MeshTree often start by creating a random structure [2,3,6,11]. Random tree structures offer some benefits such as resilience and costless initialization, though this could result in unbalanced situations with high average latency and jitter. Another disadvantage concerning these techniques is the over increasing overhead caused during runtime due to required optimizations which can affect playback quality greatly. A low cost initial tree on the other hand could guarantee a low average jitter while getting transformed easier into a more balanced one by following simple transformations as we will explain later.

Next, we will introduce some good attributes a dissemination tree is reasonable to have for high quality streaming. During a top down distributed construction of

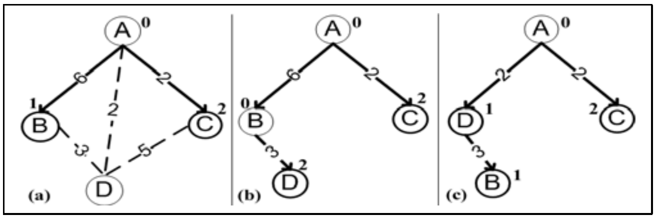

Fig. 2. Example: Operation of our System

the multicast tree there is a set of nodes J , $J \subset V$, that have already joined the overlay tree. When a node finds an available parent and establishes a connection it is considered as joined. Due to variable bandwidth availability, as it has been already mentioned in section 2.2, each node u_i is capable of supporting a certain fixed number of children (or outgoing connections) $S(u_i)$, known as slots. That signifies that there's only a set of candidate parents K at any time, specified as tree members having free slots.

Definition 1. *Candidate parents K, $K \subseteq J, u_i \in K$ iff $u_i \in J$ and $S(u_i) > 0$.*

Figure 2 shows an example with four nodes A, B, C and D for which the edges between them illustrate their distance (unicast latency in our example). Nodes A, B and C have already joined the multicast tree, so the set of joined nodes is $J = \{A, B, C\}$. Assume that their slots availability is as follows: $S(A) = 0, S(B) = 1$, and $S(C) = 2$. That denotes that if D considers joining the tree, it has to take into account only the set of candidate parents $K = \{B, C\}$ once A does not have any available slots left. Nodes B and C are children of A in the multicast tree and this set is defined as $C(A) = \{B, C\}$. Also the parent of a node u_i is defined as $p(u_i)$, so $p(B) = p(C) = A$. The problem here is which candidate D should choose as a parent. If D choose B as a parent then $d(D, B) = 3$ and $l(D) = 9$, where $l(u_i)$ is the tree path distance of a node u_i. Alternatively, by joining C, $d(D, C) = 5$ and $l(D) = 6$, so the choice that should be made here is non-trivial. Although both per-hop and end-to-end latency attributes can be considered for high quality streaming, we chose to use the per-hop distance as priority constraint. In the example that means that $p(D)$ should connect to B despite high end-to-end latency. Formally, the parent of each node following that pattern should be found using the following min-cost rule:

Definition 2. *$p(u_i) = u_j$ iff $d(u_i, u_j) == min\{d(u_i, u_k)\}, \forall u_k \in K$.*

A tree constructed following the min-cost join metric defined above is essential for a starting point on multicast streaming and offers the minimum possible per-hop latency which is important to maintain low jitter levels at startup. However, this is not sufficient for scalability and overall quality. Another key metric that should be taken into account is the average end-to-end latency from the root c to each tree node, defined as $\sum_{i=0}^{\|J\|} \frac{l(u_i)}{\|J\|}$. It is quite simple to transform a min cost tree into a low latency one by applying the following rule to each node until it gets to the highest possible level in the tree:

Definition 3. *if* $p(u_i) == u_j$ *and* $d(u_i, p(u_j)) < d(u_j, p(u_j))$ *then* $p(u_i) \rightarrow p(u_j)$ *and* $p(u_j) \rightarrow u_i$, $\forall u_i \in J$ *where* $S(u_i) > 0$

A node occasionally checks whether it is closer to its grandparent than its parent and if so it can be swapped with its parent. This is needed because of high churn: in a dynamic environment a peer can connect and disconnect from the tree at random times and without a priori notification. This can leave the tree unbalanced. Each node having that property moves closer to the root and after a period of convergence the resulting tree has nodes relatively close to the root occupying the top levels of the tree while nodes further from the root occupying the bottom levels. That is an effective way for minimizing the average end-to-end delay. In the previous example, D should swap with its parent B once $d(A, D) < d(B, D)$ resulting in $d(A, D) = l(D) = 3$ and a decreased average latency of 3. In our system, the low cost tree is being created during a pilot initialization phase, in which topology information is being collected and then the optimization process takes place at run-time.

3.3 Accomplishing Topology Awareness

One important challenge in our setting is whether it is possible to gather topological information in a manner that is both practical and scalable and if so, how could this information be effectively incorporated into the design of distributed systems such as overlay networks and content distribution systems. An effective solution was proposed in [1] called *binning scheme*, in which technique nearby nodes cluster themselves into groups, called 'bins', such that nodes that fall within a given bin are relatively close to one another in terms of network latency.

The binning scheme is fully distributed and it requires only a set of k well-known machines $l_1, l_2, .., l_k$ to be set as landmarks in a specific ordering. The technique works as follows: a node measures its distance, i.e. round-trip time, to this set of well known landmarks and independently falls into a particular bin based on these measurements. This is performed by all the nodes in the network. We chose to use this technique because it offers great advantages: (a) it is simple and cost-effective, there are only $O(nk)$ operations required, where k is the number of landmarks and n the number of all nodes, instead of $O(n^2)$ and (b) it requires very little support from the infrastructure. The only infrastructure required is a small number (depending on the overlay size, usually 4-6 suffice) of relatively stable landmark machines which they only need to echo 'ping' messages. These landmarks could in fact be unsuspecting participants in the binning scheme. Landmarks do not actively initiate measurements nor gather or disseminate measurement information. Another advantage of the binning scheme is that it is scalable because nodes independently discover their bins without communicating or coordinating with other application nodes. Finally, this technique is robust to the failure of one or more landmark nodes as described in section 3.7.

In our approach we have extended the binning scheme as follows: The bin a node v_i belongs to is represented as a vector of values in specific ordering $< q_1, q_2, q_3, q_4 >$,where q_i is a certain level of latency between landmark l_i and v_i. Levels of latency are usually between 3 to 5 and are computed by the landmarks based on ping measurements gathered at the system's initialization period, set in a way to dissociate nodes most effectively. We have modified the original binning scheme by setting the order of the participant landmarks due to their distance from the tree root and having as first landmark the actual root of the tree. The distance metric we used to approximate the distance $d(B_i, B_j)$ between two different bins, B_i and B_j, is the following:

$$d(B_i, B_j) = \sum_{l=0}^{k} [\|B_{il} - B_{jl}\| * (k - l + 1)] \tag{1}$$

For example the distance between the bins $B1 = \{2, 2, 1, 0\}$ and $B2 = \{1, 2, 2, 2\}$ is $(4 + 0 + 2 + 2) = 8$ based on the metric mentioned above. As the distance metric dictates, landmarks that are closer to the root are more important than landmarks further from it so along with relative distance, a bin also reflects the actual distance from the tree root in relation to other bins.

3.4 Content Management

Topological information in our system is stored on specific responsible nodes. There are two types of topological information. First we have the *bin data* which specifies close nodes in the overlay due to the unicast latency proximity metric. Second, we have the *zone* oriented content. A zone is one extra layer of topology measure which is derived directly from the bins. Each zone contains all bins starting with this zone which is simply the first value of a bin vector (always referring to the system's root). For example zone '0' contains all bins starting with '0' (eg $< 0, 2, 2, 3, 2 >$, $< 0, 1, 2, 0, 3 >$). By using zones, we can reduce the amount and size of topology oriented messages by requesting bins on a specific zone to apply operations and not involving the remaining bins in the system. We can also guarantee some scalability during tree construction by starting the top-down building algorithm from member nodes of bins belonging to zone 0 and then continuing to the next zones.

We have built our system on top of the Pastry DHT [15]. The Pastry DHT is mainly used for storing the content management information. The advantage of Pastry is that the content storage is well-balanced across the system, and retrievals of it can be achieved only with a small amount of messages. Responsible content nodes, maintain certain states of the bins and zones and update them based on any changes that occur in the tree. Bin data state for example specifies whether all bin members have joined the tree. When all nodes on a bin have joined the multicast tree this is reflected at the zone state too so that everyone knows whether all zone bins have joined the tree when asking for zone specific information. When a content state changes, the responsible node for the specific content (having the numerically closest ID hash number to the content's hash

Algorithm 1. Tree Construction Join Algorithm()

```
if ($SLOTS > 0$) then
```

```
$SLOTS \Leftarrow node.availableSlots$;
$currentZone \Leftarrow node.zone$;
if ($SLOTS > 0$) then
  obtain bin information;
  if (unjoined node on same bin > 0) then
    add unjoined nodes in $C(node)$
  end if
  while ($SLOTS > 0$)AND($currentZone \leq MAXzone$) do
    // there are still slots available
    $bins[] \Leftarrow zone[currentZone++].bins$;
    sort(bins[]); //due to the distance from current node's bin
    for all (bins in bins[]) do
      add the closest not joined node in $C(node)$
      if ($SLOTS == 0$) then
        return;
      end if
    end for
  end while
end if
return;
```

number), is being notified to update its content. Content can also be replicated to multiple responsible nodes thus making the system more robust during node failures. Information retrieval is pretty straightforward, by using the lookup(ID) function on the DHT to get the information needed by having only a given ID. Note that all DHT operations in Pastry require $O(logn)$ messages.

3.5 Initial Tree Construction

During the initialization phase all musician nodes join the overlay apart from the spectator nodes, the binning information is being computed and then stored into the DHT and the low cost tree is formed for the streaming process to begin. At first, the coordinator and then the musicians join the system and a phase begins in which the number of spectators join the system and compute their distances to binning landmarks which have been predefined by the root. The first landmark is always the root and the next landmarks are usually occupied by the musicians considering their stability in addition to more, possibly random landmarks, all ordered due to their distance to the root. After enough latency measurements have been made on all landmarks, the bin levels are computed on each landmark and broadcasted to the whole system using Scribe [5], an event-based notification system built on top of Pastry. The latency range of each level is chosen based on the variance, the mean latency value and the number of latency levels(in our experiments we used 4 zones). For example, using four zones, latencies are normalized and dissociated into ranges by the z values of the latency distribution measured $z_i = (\mu + i * \sigma)$ as such $< [0, z_{-1}), [z_{-1}, z_0), [z_0, z_1), [z_1, +\infty) >$

thus granting the maximum separability possible with the use of bins. When all bins are computed and content updates finish this phase is over and the initial tree construction phase begins. The initial tree construction is initiated by the coordinator root which runs first the initial low-cost tree building algorithm (Algorithm-1). The resulted tree is also end-to-end-latency aware due to zone priorities used in the join process.

The algorithm works as follows: when a parent finds an appropriate node to attach to its children, the new child starts the same process and searches for appropriate children. It begins by getting its own bin's content information and then checking whether there are any unregistered nodes and if so, it asks them to become its children. If there are more slots available it continues the search for child nodes by asking the DHT for its own zone information. If there are bins containing unregistered nodes it asks those nodes to become its children in a specific order, relevant to the distance to those bins. If the zone does not contain any bins with unregistered nodes it asks the DHT for the next zone information and the algorithm continues until there are no available slots at this node or when all nodes have been registered during the initialization phase. Note that after all nodes in the overlay have joined the multicast tree, the coordinator orders the musicians to start streaming their sequences on a specific global time point and the stream is then being disseminated in the multicast tree, setting the outset of the run-time phase.

We will demonstrate the usage of Algorithm-1 by giving an example. In figure 3(a) we visualize a part of the multicast tree during the initialization phase (top down initial construction). Nodes A and B have already joined the tree while C, D and E wait to get placed on appropriate positions. Next to each node we've attached the actual bin ID for its corresponding bin. In (a) node B searches for one more node to add to its children by following Algorithm-1. Unregistered nodes (C, D, E) have been grouped by their zone below the graph. Node B first asks for zone 2 information once it belongs to zone 2 (and there are no unregistered nodes in its own bin) and gets only the bin $< 2122 >$ in which contains the node C. Then, B asks C to join the tree as one of its children and stops there because it has no slots available. Node C now (figure 3 (b)) finds no unregistered nodes in its own bin and also gets informed that zone 2 bin nodes have all been registered. At that point node C can take the zone 3 list of unregistered bins, which are $< 3101 >$ and $< 3231 >$ which contain nodes D and E respectively. Distances due to the relative bin metric proposed in section 3.3 of this paper, are d($< 2122 >$, $< 3101 >$) = 9 and d($< 2122 >$,$< 3231 >$) = 10 so C first adds node D to its children. Assuming that C maintains one more available slots, it adds node E too as its second child and stops there (figure 3(c)).

3.6 Run-Time Optimizations

After the initial phase, during run-time, the tree is being transformed into a more balanced one with the use of tree transformations. We followed the optimization rule (3) stated in section 3.2 (Definition 3). Occasionally each registered node in the multicast tree (or a newly joined node) checks whether it is closer to its

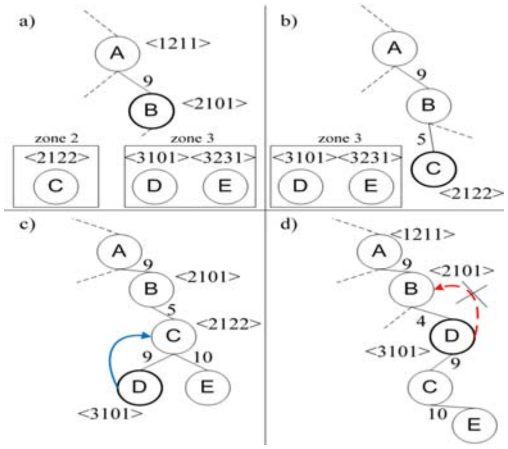

Fig. 3. Examples of the initialization and optimization phases

grandparent than its parent is, due to the relative distance. If that is true and the node has at least one available slot, its parent becomes its child and it takes the place of its parent in the tree by setting its grandparent as its parent. Then, it checks again for a new swapping with its new parent. By following this rule, nodes that are closer to the root tend to ascent the tree and thus granting a low average end-to-end latency for the system.

In our previous example (figure 3 (c)), node D checks whether it is closer to B than C is, by calculating the relative distances d(D,B)=d($< 3101 >$,$< 2101 >$)=4 and d(C,B) = d($< 2122 >$,$< 2101 >$)=5. It is obvious that it can be swapped with its parent (assuming that C maintains an additional free slot) and so its new parent now is B (figure 4 (d)). However, $d(D, A) = 13$ while $d(B, A) = 9$ so D cannot proceed on further optimizations for the time being.

3.7 Failure Recovery

There is a number of possible failures that can happen during the system's operation: (a) spectator node failures, (b) failure of a landmark node, and (c) failure of musician nodes. The easiest to deal with is spectator failures. If a spectator leaves the system without warning, its child in the multicast tree will diagnose its parent's loss and will instantly ask it's grandparent to add it to its children. If that cannot happen the child asks the next closest node from the same bin to add it until it finds an available parent. The second type of failure which is more serious, is the failure of a landmark node. In this case, considering that only a small number of landmark nodes can concurrently fail, the best solution is for each node to drop the landmark identifier from its bin vector and new responsible landmark nodes need to be found by Pastry. This will require only $O(logn)$ extra messages for each bin to find the new responsible node via Pastry. Finally, the failure of a musician node could result in the loss

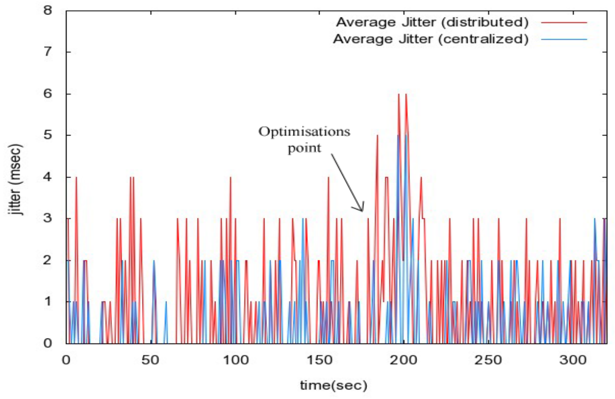

Fig. 4. Average Jitter Measurements experienced by the Spectator nodes

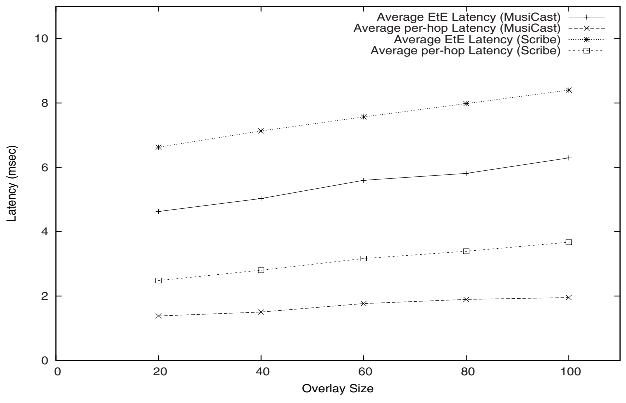

Fig. 5. Latency as a function of the Overlay Size

of one channel but not the failure of the whole system. Even if the root failed, the closest spectator or musician could be instantly act as the new tree root by broadcasting a control message of its address to the entire overlay indentifying itself as the new root.

4 Performance Evaluation

We have performed a series of experiments to evaluate the efficiency and performance of our system, using up to 100 peers deployed in a network of 20 local x86 machines consisted of a 3.0GHz single core CPU and 1024MB RAM which were connected via 100Mbps Fast Ethernet.

Three of the nodes acted as musicians of the system (sources) and the coordinator which was running at a well known address acted as a bootstrap for the

Pastry DHT. As a typical streaming content we chose a combination of midi sequences among a 64kbps audio sequence, taken from typical big band tracks. We tried to select music having occasional variations in rate in order to observe our system's response in rapid rate peaks. For delay oriented measurements, we have managed to achieve microsecond accuracy to comply with the LAN's typical latency scale having a 8-10 microseconds possible amount of error. Also, all sequences where streamed via UDP for the minimum possible delay.

To evaluate the performance of our system, we have compared it with a centralized version of it. This centralized version uses the same algorithm (Algorithm-1) introduced for top-down tree construction during the initialization phase, with the difference that a central process manages the whole tree construction and orders all nodes based on their distance from the root before connecting each of them to the appropriate low cost parent. That results in a more balanced low cost tree which maintains a low end-to-end latency and better load balance from startup. After the initialization phase, during run-time, the centralized system continues to operate normally as in the distributed scheme. We have also evaluated our system in comparison with an implementation of Scribe, which is another popular multicast system on top of Pastry [5]. Scribe creates a multicast tree at runtime by using reverse routing paths to a specific node (known as rendezvous point or group creator). In this implementation the coordinator is the actual scribe multicast group's creator and we've also included an extra check during initialization of each node on bandwidth availability as follows: when a Scribe node tries to establish a route to the multicast tree root, it checks whether the next hop (parent) following this route has available slots, if not it rejoins the DHT using a different random node ID, thus connecting to the group from a possibly different location. This process continues until the node finds a parent with available slots.

In the first set of experiments we measured the average jitter experienced by the Spectator nodes. The average jitter level among the spectators is an important consideration because it affects playback quality directly, especially if it outruns a predicted buffering delay. Figure 4 shows the average latency of the Spectator nodes. During our experiments we noticed slight increases in jitter levels (typically 0-4 milliseconds in a LAN) on specific parts of a music act. These jitter peaks can be noticed in Figure 4 among the distributed and the centralized version of MusiCast. Even though the peaks are unnoticeable due to their microsecond level in a WAN infrastructure, jitter could cause increased variations in length resulting to decreased playback quality. There are two types of jitter peaks that can be noticed in Figure 4 in both versions: (a) some casual small peaks that appear every 65 seconds, starting from second 48 and (b) a larger peak at second 200 having increased spanning. Small casual peeks appear due to increased bit rate at specific parts of a music act and that explains their repetitive nature. The bigger peak on the other hand is caused by the increased amount of tree operations during the optimization period. It is clear that the centralized scheme has lower average jitter before the optimizations occur and the same jitter levels after them. That implies that low end-to-end latency

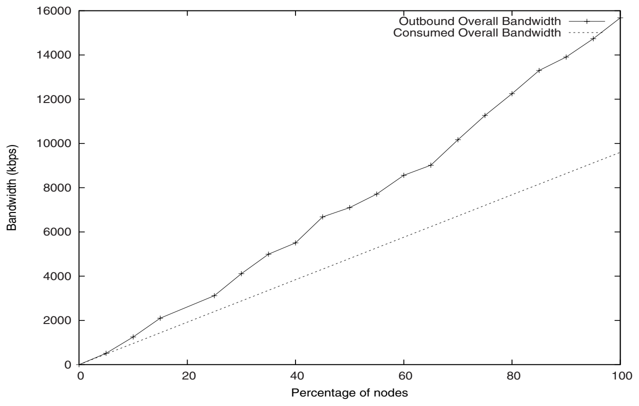

Fig. 6. Overall Bandwidth

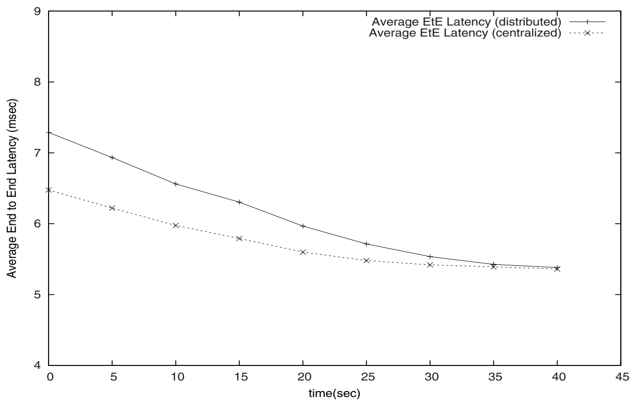

Fig. 7. End-to-end delay during optimizations

decreases the average jitter of the system and that is reasonable concerning shorter paths to the root and possibly fewer connections.

Our system's low latency performance can be seen in Figure 5. The latency is shown as a function of the overlay size in comparison with the one of the special version of Scribe described above. Our algorithm achieves 27% improvement on the average end-to-end latency while showing similarly low increase rate to Scribe. Per-hop latency is also lower in our system due to the initial tree values that have been maintained.

Another metric that needs to be considered in every multicast system is the overall offered and consumed bandwidth. There must always be enough remaining bandwidth for new nodes to join the system at any time. In Figure 6 it is clear that the available bandwidth is over-increasing during the joining of new

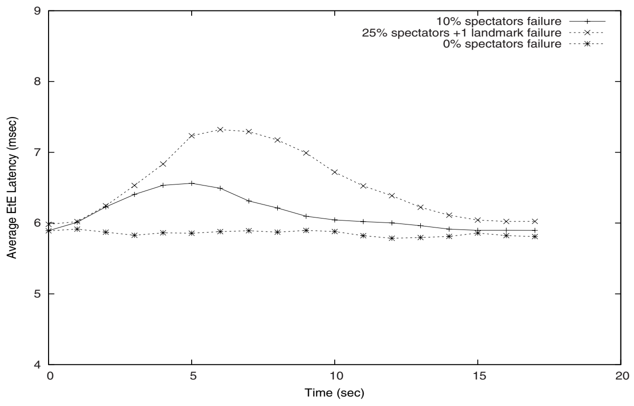

Fig. 8. Latency during failures

nodes. The available bandwidth cannot reach zero because once we've set the lower limit of one slot to each node, the actual increase of the offered bandwidth can be at least equal to the one of the consumed bandwidth. Also, the joining process is flexible enough, allowing a new node to join the closest available neighbor easily without imposing it like scribe does when sticking only to a specific route ordered by Pastry.

System's convergence during optimizations lasted for about 35 seconds on the distributed version and 25 seconds on the centralized scheme because the tree was already balanced in some degree. In Figure 7 the average end-to-end latency decrease is visualized during the optimization period. Both schemes seem to converge nearly on the same optimal average end to end latency level. We can conclude here that at first the centralized scheme is having a better structure, though after the optimization period the distributed scheme managed to achieve the same streaming quality level with the centralized one.

We have finally tested our system's recovery performance during different failure rates. Latency measurements took place throughout the system from the moment a number of randomly selected spectators had failed. During this period the recovery procedure took place which was described briefly in section 3.7 . As the figure shows, the system managed to achieve a complete recovery when 10% of the spectators had failed, in about 13 seconds. However, recovery was a more difficult task when 25% of the spectators left the system including one landmark. As shown in figure 8, in this case, the system managed to converge on a little higher point of average latency after 15 seconds and that's because the binning scheme was a little less accurate than before, having one less landmark in each bin. In special cases when more landmarks could fail, new landmarks need to be chosen again resulting in a small period of reconstruction ending in a complete recovery of the system, otherwise, continuing with the reduced bins is preferable and cost effective.

5 Related Work

Several recent projects make use of application-level multicast and overlays to achieve media streaming [2,3,5,6,8,9,11,12]. Some of them such as Scribe, ChunkySpread and SplitStream have adopted tree or multi-tree structures achieving low overhead solutions although they have failed on maintaining low end-to-end latency or link stress and effective failure recovery [5,6,9]. Bullet, MeshTree, mTreeBone and Coolstreaming offer a different approach by including an initialization using a random structure followed by a series of optimizations during runtime. These solutions have many benefits when streaming encoded pre-buffered content or other content not sensible by high jitter. They also offer resilience and simplicity during initialization. However these techniques are not suitable for streaming content sensible by jitter such as midi events and that's due to their: a) initial high delay structure and b) the increased overhead caused during runtime to achieve all the required optimizations.

Mesh structures are more preferable to trees in projects such as Bullet and CoolStreaming and that's because meshes offer more robustness and ease of locating and maintaining low latency links between peers while achieving a good load balance [2,3,12]. However, meshes can potentially incur high network or CPU overheads due to the demand of extra amount of control messages for mesh maintenance. In our system meshes were not suitable because control messages could make an impact to jitter values and as a result the playback of midi messages could be affected. There is also a hybrid approach, combining tree and mesh structures such as the one on MeshTree and mTreebone where trees and meshes' favorable properties have been merged achieving an impressive result [8,11]. Hybrid solutions are essential for high-volume, bandwidth-demanding data streaming yet are not preferable for medium-volume data such as midi events or compressed live audio types on which the extra overhead caused by the meshes' control messages used could even outmatch the actual data size and cause undesirable jitter.

Finally none of the systems mentioned has achieved topology awareness in the degree our system did. We have managed to replace any actual delay computation between overlay nodes, with the distance metric of the binning scheme [1] while achieving the minimum possible overhead to establish such an accurate delay evaluation scheme without even measuring corresponding delays by taking advantage of the content storing properties of a DHT such as Pastry.

6 Conclusions

In this paper we have presented the design principles and mechanism behind MusiCast, a peer-to-peer multicast system specifically oriented to low and medium size content streaming, such as midi events and compressed audio. Our system manages to synchronize musician nodes that produce midi events or audio streams and the overall stream is then being passed to a large number of connected spectators in a highly scalable way by building a topology awareness scheme on top of the Pastry DHT, while keeping low latency and low average jitter levels. Our results extracted from various local experiments certify the effectiveness of our approach in dealing with diverse latencies and node capacities.

Acknowledgements

This research has been supported by NSF Award 0627191 and a Marie-Curie fellowship. We would like to thank A. Rowstron and P. Druschel for their open source implementations of Pastry (Free Pastry).

References

1. Ratnasamy, S., Handley, M., Karp, R., Shenker, S.: Topologically-Aware Overlay Construction and Server Selection. In: IEEE INFOCOM (2002)
2. Kostic, D., Rodriguez, A., Albrecht, J., Vahdat, A.: Bullet: High Bandwidth Data Dissemination Using an Overlay Mesh. In: SOSP 2003 (2003)
3. Zhang, X., Liuy, J., Liz, B., Yum, T.-S.P.: CoolStreaming/DONet: A Data-Driven Overlay Network for Efficient Live Media Streaming. In: IEEE INFOCOM, Miami (March 2005)
4. Magharei, N., Rejaie, R., Guo, Y.: Mesh or Multiple-Tree: A Comparative Study of Live P2P Streaming Approaches. In: IEEE INFOCOM (2007)
5. Castro, M., Druschel, P., Kermarrec, A.-M., Rowstron, A.: Scribe: A large-scale and decentralized application-level multicast infrastructure. IEEE Journal on Selected Areas in Communications 20(8) (October 2002)
6. Venkataraman, V., Francisy, P., Calandrino, J.: Chunkyspread: Multi-tree Unstructured Peer-to-Peer Multicast IPTPS, Santa Barbara, CA (2006)
7. Jin, X., Xia, Q., Gary Chan, S.-H.: A Cost-based Evaluation of End-to-End Network Measurements in Overlay Multicast. In: IEEE INFOCOM (2007)
8. Tan, S.-W., Waters, G., Crawford, J.: MeshTree: A Delay-optimised Overlay Multicast Tree Building Protocol. University of Kent, Technical Report 5-05
9. Castro, M., Druschel, P., Kermarrec, A.-M., Nandi, A., Rowstron, A., Singh, A.: SplitStream: High-Bandwidth Multicast in Cooperative Environments. In: ACM SIGOPS Operating Systems Review (2003)
10. Chu, Y., Rao, S.G., Seshan, S., Zhang, H.: A Case for End System Multicast. In: ACM Sigmetrics, Marina de Rel, CA (2002)
11. Wang, F., Xiong, Y., Liu, J.: mTreebone: A Hybrid Tree/Mesh Overlay for Application-Layer Live Video Multicast. In: ICDCS, Ontario, Canada (2007)
12. Li, B., Xie, S., Qu, Y., Keung, G.Y., Lin, C., Liu, J., Zhang, X.: Inside the New Coolstreaming: Principles, Measurements and Performance Implications. In: IEEE INFOCOM, Phoenix, AZ (2008)
13. Banerjee, S., Lee, S., Bhattacharjee, B., Srinivasan, A., Zhang, X.: Resilient Multicast using Overlays. In: ACM SIGMETRICS (2003)
14. Banerjee, S., Kommareddy, C., Kar, K., Bhattacharjee, B., Khuller, S.: Construction of an Efficient Overlay Multicast Infrastructure for Real-time Applications. In: IEEE INFOCOM, San Fransisco, CA (2003)
15. Rowstron, A., Druschel, P.: Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. In: IFIP/ACM International Conference on Distributed Systems, Heidelberg, Germany (2001)