

Cross-Tree Adjustment for Spatialized Audio Streaming over Networked Virtual Environments

Ke Liang
School of Computing
National University of Singapore
Singapore 117417
liangke@comp.nus.edu.sg

Roger Zimmermann
School of Computing
National University of Singapore
Singapore 117417
rogerz@comp.nus.edu.sg

ABSTRACT

In recent years, integrated spatialized voice services have become an appealing application for networked virtual environments (NVE), e.g., Second Life. With a spatialized voice service, people can identify who is talking if there are several participants in the vicinity. The key challenge in a spatialized audio streaming application is how to disseminate audio streams while observing bandwidth limits of end-user computers and tight latency constraints, and it can be modeled as NP-complete problem. In this paper, we propose a heuristic algorithm called CTA for spatialized audio streaming over NVEs in a peer-to-peer manner. The proposed algorithm was applied to real avatar mobility traces collected from Second Life, and the simulation results demonstrate that (a) CTA can achieve a high ratio of successful to candidate receivers and (b) CTA enables most of the successful receivers to enjoy minimum latency.

Categories and Subject Descriptors

H.4.3 [Information Systems Applications]: Communications Applications—*Computer conferencing, teleconferencing, and videoconferencing*

General Terms

Algorithms, Measurement, Performance

Keywords

Proximity audio, spatial audio, peer-to-peer streaming, application level multicast

1. INTRODUCTION

Networked virtual environments (NVE) are large-scale interactive applications in which users can move their avatars in a shared virtual world while interacting with the environment and other online users. One of the most well-known and successful NVE's is Second Life, which is an online 3D virtual world built by its residents.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

NOSSDAV'09, June 3-5, 2009, Williamsburg, Virginia, USA.
Copyright 2009 ACM 978-1-60558-433-1/09/06 ...\$5.00.

The goal of voice services provided in a NVE is to enable users to speak to each other naturally and seamlessly. Some existing systems (e.g., Skype and AudioPeer [9]) provide monophonic voice services that are separate from any NVE, while others aim to provide a more integrated approach. The most important disadvantage of existing monophonic audio systems is that they are not designed to utilize the spatial information (e.g., location, distance and directionality) between avatars. However, with a spatialized voice service, users can more easily identify who is speaking if several people surround them. By matching users' aural environment to their visual perception more immediacy and dynamism is lent to their interaction with others, providing an overall more immersive experience.

Spatialized 3D audio rendering is now available on PCs with the help of software libraries (e.g., OpenAL) that can take advantage of different hardware configurations. 3D audio can be rendered via two or more loudspeakers, or via stereo headphones. Although there are some algorithms that regenerate 3D audio from only one or two audio streams, most 3D audio processing modules require access to each spatially positioned source and hence multiple audio streams should be kept separate until rendered.

Boustead and Safaei [1] have examined different delivery architectures for audio transmission in large-scale virtual environments, including client-server and peer-to-peer infrastructures. In a client-server topology for 3D audio transmissions, the central server is responsible for receiving all audio streams directly from all sources (speakers) and forwarding pre-rendered 3D streams to corresponding receivers. 3D audio rendering for all audio streams is accomplished by the central server based on the avatars' spatial information and then encapsulated into stereo streams that are sent back to each receiver. In this approach, the central server requires significant computational resources and a large amount of bandwidth to process and disseminate audio streams. To remove the bottleneck at the central server, some proxy-based solutions [7] have been proposed to balance the bandwidth and processing load [3]. The 3D voice provider of Second Life, Vivox, employs a client-server infrastructure.

By utilizing a peer-to-peer based approach – which can alleviate the scalability problems caused by the client-server design – we disseminate audio streams using application-level multicast trees built on overlay networks that take into consideration heterogeneous bandwidth limits of users and tight latency constraints. Because the upload and down-

load bandwidths¹ of each node in a peer-to-peer network are limited and heterogeneous, the task to effectively utilize the network resources when disseminating several separate audio streams while keeping the overlay latency bounded becomes very challenging. Some receivers may not be able to receive the audio streams that they are supposed to (we call those users unsuccessful receivers), and some successful receivers may have a higher latency due to a non-optimal dissemination path of the audio stream. Hence, our research aims to (1) maximize the number of successful receivers of corresponding audio streams and (2) minimize the latency for those successful receivers. The task can be modeled as a well known \mathcal{NP} -complete problem [2]. Recent work exists on heuristic algorithms to construct a degree-bounded minimum average-latency spanning tree for one node [1, 5] or all nodes [4].

Previously we have proposed a local tree adjusting (LTA) algorithm [8] designed to maximize the number of successful receivers while keeping the latency for those receivers low. LTA can fully utilize the available network resources for each speaker to construct a multicast tree. However, it lacks fairness to new audio sources since existing ones may have already exhausted network resources. In this paper, we propose a heuristic cross-tree adjustment (CTA) algorithm to solve this fairness issue by constructing and adjusting multiple multicast trees using a greedy approach. The proposed algorithm differs from previous work in that (1) we only construct a multicast tree for a node when it starts speaking and (2) we adjust the new multicast tree with respect to other existing multicast trees who have common nodes with the new multicast tree. The simulation results demonstrate that CTA can outperform LTA both in terms of the number of successful receivers and the ratio of successful receivers that experience minimal latency.

The rest of this paper is organized as follows. Section 2 describes the system model for spatialized audio streaming over NVEs. The problem we are aiming to solve is formulated in Section 3. The proposed algorithm, CTA, is described in detail in Section 4. Section 5 describes the evaluation of our proposed algorithm. Finally, Section 6 draws conclusions of the whole paper.

2. SYSTEM MODEL

The users (avatars) in an NVE can move in the system at will and any two users can communicate with each other when their distance is less than r , the radius of a node's area of interest (AoI). The AoI is defined as the space around the user where interaction (e.g., voice communication) is likely to happen and it corresponds to the hearing range of each user in our audio streaming scenario. We assume that location and AoI information of each node are maintained at the game server. Existing algorithms and methods exist for this purpose and we do not provide any details here. Every user can speak anytime and at any location. Each user j has a limited upload bandwidth u_j^{max} . The upload capacity is measured in *units*, where 1 unit is defined as the required bandwidth for a single audio stream and measured in bits/s. We will use the terms user, node and avatar interchangeably throughout this paper.

We model the system as a digraph $G = (V, E)$. V is the set of nodes representing users in the system. E is the set

¹We only consider upload bandwidth limits in this paper.

of edges. An edge $e(i, j)$ indicates that node n_i is willing to send (if n_i is a speaker) or forward an audio stream to n_j . The nodes within the AoI of node n_i are called *neighbor nodes* of n_i . Every user periodically exchanges information (i.e., bandwidth usage and end-to-end latency) with its neighbor nodes. When a node starts speaking, all its neighbor nodes are *candidate receivers* for the audio stream sent from the speaker. Due to the bandwidth limits of the neighbor nodes and the speaker, only a fraction of the candidate receivers, called *successful receivers*, may actually be able to receive the stream.

Transmission of any audio stream is restricted within the AoI of the speaker who sent the stream. That is, all nodes outside of the hearing range of the speaker will not receive or forward the audio stream. When a node n_i starts to speak, it will construct a temporal multicast tree T_0^i rooted at n_i . All candidate receivers are members of T_0^i , that is, T_0^i is initially constructed without considering the bandwidth limits of nodes. T_0^i is a shortest path tree (SPT) which is a combination of shortest paths from the speaker to all nodes within its AoI.

If some nodes in T_{tmp}^i have conflicts² (since one node may be an intermediate node in other existing multicast trees), T_0^i should be tailored to T_{spt}^i by cutting corresponding edges incident with those *conflict nodes*. This operation will result in some *orphan nodes* which are candidate receivers but cannot obtain the audio stream from the speaker. The orphan nodes will be attached to nodes in T_{spt}^i that still have upload bandwidth available, and then the final multicast tree is formed as M^i . The reason we chose SPT as the basis of the multicast tree is two-folded. First, members of an SPT will benefit from its built-in characteristics, i.e., the latencies from the source to members of SPT are minimal. Second and more importantly, compared to other spanning tree techniques, e.g., a minimum spanning tree or Hamiltonian path, an SPT has less depth and a larger fan-out. Therefore, edge cutting operations will result in less orphan nodes.

Our goal is (in the presence of bandwidth limits of nodes) to maximize the total number of successful receivers while at the same time minimizing the average latency from speakers to their successful receivers. The most critical challenge is that constructing an optimal multicast trees is highly computationally complex as the task is known to be \mathcal{NP} -complete. Next we will formally state the problem and then study its solutions with heuristics.

3. PROBLEM FORMULATION

Our objective is to enable many-to-many audio streaming, which requires the construction of a forest that consists of multicast trees rooted at all the speakers in the presence of bandwidth limits of nodes and latency constraints. We define the problem as constructing a *forest of degree-bounded minimum latency trees*:

$$\text{Maximize} : |R^S| = \sum_{i \in S} |R_i| \quad (1)$$

$$\text{Minimize} : \frac{1}{|R^S|} \sum_{j \in S} \sum_{k \in R_j} L(j, k) \quad (2)$$

$$\text{S.t} : u_x \leq u_x^m, m \in V$$

²In case of the upload bandwidth required for the node exceeds its maximum capacity.

where R_i is defined as the set of successful receivers of speaker n_i , and S refers to the set of speakers. $L(j, k)$ denotes the latency of the path from node n_j to node n_k . The first optimization objective is to maximize the number of successful receivers, subject to nodes' bandwidth limits. Once the first goal is satisfied, the average latency of those successful receivers achieved by Eqn. 1 is to be minimized. Due to node dynamics, system churn and avatars' free will to speak, it is very challenging to find an optimal solution to the multi-objective problem stated above.

Term	Definition
M^i	Multicast tree rooted at speaker i
T_0^i	Temporal multicast tree root at speaker i
T_x^i	SPT deduced by removing leaf nodes from T_{x-1}^i
T_{spt}^i	SPT root at speaker i , $T_{spt}^i = M^i \cap T_0^i$
$V(T)$	Set of nodes in the tree T
V_c^i	Set of all nodes in T^i that have conflicts
V_x^i	Set of leaf nodes of T_x^i
D_j^i	Set of descendants of node j in T^i
W_j^i	Local weight set of node j in tree T^i
W_j^g	Global weight set of node j
$f(j)$	Forward node for node j
u_j^m	Maximum upload units of node j
u_j^s	Used upload units for nodes in SPTs at node j
c_j^i	Number of children of node j in T^i

Table 1: List of terms used.

Instead of achieving the two objectives sequentially, we try to achieve a compromise between the two concurrently. In our approach, the multicast tree M^i for each speaker n_i consists of a SPT (T_{spt}^i) and further orphan nodes attached to it. That is, T_{spt}^i is a subgraph of M^i ($T_{spt}^i \subseteq M^i$). The temporal multicast tree for speaker n_i is T_0^i , which is also a SPT. The heuristic is that the larger $|M^i \cap T_0^i|$ is, the more likely the multicast tree will have both, a high successful receiving ratio and low average latency. Therefore, our proposed heuristic algorithm is to maximize $|M^i \cap T_0^i|$ for all multicast trees in the system:

$$\begin{aligned} \text{Maximize} & : |T_{spt}^S| = \sum_{i \in S} |T_{spt}^i| \\ \text{s.t} & : u_x \leq u_x^m, m \in V(T_{spt}^S) \end{aligned} \quad (3)$$

The key challenge of our approach is how to allocate the upload bandwidth of bandwidth-deficient nodes for the multiple SPTs that they are associated with.

4. APPROACH

Douceur [4] proposed a approximation algorithm to maximize the total upload capacity by constructing a depth-2 multicast tree for each node in the system. However, in voice communication applications it is a waste of network resources to maintain the multicast trees that are not in use. Therefore, distribution trees for voice dissemination should only be constructed once speakers start to talk, and be destroyed after the conversation ends.

In our previous approach [8], a multicast tree was constructed independently for each speaker, and a local tree adjusting algorithm (LTA) was applied to modify the multicast tree with respect to the remaining available bandwidth of neighboring nodes. This straightforward approach can be

quickly employed at each speaker, but it is unfair to new speakers as the network resources may have been exhausted by existing speakers.

Here we introduce a heuristic method to allocate network resources for all speakers, termed cross-tree adjusting (CTA), which outperforms LTA both in terms of fairness and the total number of successful receivers. In addition, the proposed algorithm can increase the ratio of successful receivers with minimum latency. CTA constructs the multicast trees incrementally using a greedy approach. If a new multicast tree and other existing multicast trees have overlapping nodes then all involved trees will be adjusted to observe the bandwidth limits of all nodes. Furthermore, any multicast tree will be destroyed if the corresponding speaker has kept silent for t_m seconds.

A new multicast tree is constructed in two phases. The first step, which is similar to our earlier work [8], involves the constructing of a shortest paths tree (SPT) rooted at the speaker without considering the nodes' upload bandwidth limits. The second step is to re-allocate the upload bandwidth for nodes that have bandwidth conflicts. The proposed algorithm is applied in the second step.

4.1 Temporal Multicast Tree Construction

Every node in an AoI periodically exchanges information with its neighbor nodes. Therefore, every node is aware of bandwidth limits/usage of every neighbor node and the end-to-end latency between any two neighbor nodes. Once node n_i starts speaking, it can quickly construct a temporal SPT (T_0^i) for the speaker without considering upload bandwidth limits of its neighbor nodes. The complexity to construct a SPT is $O(|V(T_0^i)| \log |V(T_0^i)|)$ when a Fibonacci heap is utilized.

After the temporal SPT is constructed, a *post-order tree traversal* with time complexity $O(|V(T_0^i)|)$ is applied to T_0^i , marking conflicting nodes and partitioning all nodes into different node sets ($V_j^i, 0 \leq j \leq m$, where m is the length of the longest path in T_0^i). Let T_1^i denote a SPT which is derived by removing all leaf nodes from T_0^i . Through the same reasoning, T_{x+1}^i is obtained by removing all leaf nodes from T_x^i , while at the same time those leaf nodes are grouped into V_x^i , $V_x^i = V(T_x^i) \setminus V(T_{x+1}^i)$. After the post-order tree traversal on T_0^i , all nodes in the AoI of speaker i will be partitioned into node sets $V_0^i, V_1^i, \dots, V_m^i$, as shown in Figure 1.

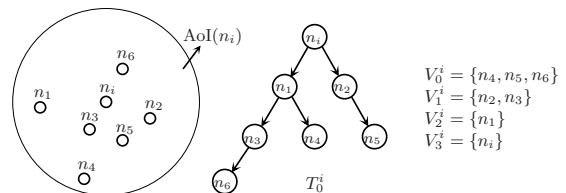


Figure 1: Temporal multicast tree construction and node partitions.

4.2 Cross-Tree Adjusting (CTA)

The second step involves the handling of conflicts caused by T_0^i . Some edges in the new SPT T_0^i – or other existing multicast trees that have overlapping intermediate nodes with T_0^i – will be removed in this step. The idea is to al-

locate the upload bandwidth of conflict nodes to maximize $|T_{spt}^i|$. Differing from the previous algorithm LTA, which allocates upload bandwidth of nodes in the local multicast tree, the proposed algorithm CTA involves all multicast trees that have overlapping conflict nodes with the local multicast tree.

There are two processes for CTA: (1) a voting process, that decides which potential edges need to be removed in the local and other existing multicast trees, and (2) an allocation process, which allocates upload bandwidth for all nodes and modifies the topologies of multicast trees according to the voting results from the first process. We will describe the two processes in detail.

4.2.1 Voting Process

Each node in the multicast tree only has one parent node which is responsible for forwarding the audio stream to the node. Once the link between the node and its parent is removed, the node and all its descendant nodes will no longer receive the stream. We define the *weight* of an edge, $w(e)$, as the *maximum* number of downstream nodes (descendants) who incur the minimum latency. For example, as shown in Figure 1, $w(e(n_3, n_6)) = 1$ and $w(e(n_i, n_1)) = 4$ since there is only one downstream node which edge $e(n_3, n_6)$ can support while $e(n_i, n_1)$ can support four. It should be noted that $w(e(n_i, n_j))$ depends on the weight of the outgoing edges and the upload bandwidth limits of n_j . For instance, if $u_j^m = 1$, then $w(e(n_i, n_1)) = 3$, hence $e(n_1, n_4)$ should be removed from the multicast tree.

Algorithm 1 $Voting(T_0^i)$

```

1:  $M^i \leftarrow T_0^i$ 
2: for each  $n_j \in V_0^i$  do
3:    $W_{f(j)}^l[j] \leftarrow 1$ 
4: end for
5: for each  $n_j \in V_x^i, 0 < x < m$  do
6:   if  $n_j \notin V_c^i$  then
7:      $w_j \leftarrow \sum_{k \in W_j^l} W_j^l[k]$ 
8:      $W_{f(j)}^l[j] \leftarrow w_j$ 
9:   else
10:     $u_j^a \leftarrow u_j^m - u_j^s$ 
11:     $w_j \leftarrow u_j^a > 0 ? \sum_{k=0}^{u_j^a-1} W_j^l[k] : 0$ 
12:    if  $B_j \neq \emptyset$  then
13:       $x \leftarrow UnitReallocate(B_j, w_j, u_j^a, c_j^i)$ 
14:    end if
15:    if  $u_j^s \neq 0$  or  $W_j^g[0] < W_j^l[x]$  and  $x < c_j^i$  then
16:       $UnitSchedule(w_j, x, W_j^l, W_j^g)$ 
17:    end if
18:  end if
19:   $W_{f(j)}^l[j] \leftarrow w_j + 1$ 
20:   $Update(M^i)$ 
21: end for

```

The weight of every edge in T_0^i is computed at its end vertex, since every node has only one parent in the tree structure. Furthermore, weights of all edges are computed in a *bottom-up* manner, starting from nodes in V_0^i to V_{m-1}^i . Every node n_j in the system will have a *global weight set* (denoted by W_j^g) which stores the outgoing edges and their weights in existing SPTs. The size of W_j^g is determined by its outgoing bandwidth limits. Elements in W_j^g are sorted

from left to right in non-increasing order. Every node n_j in new SPT (T_0^i) will have a temporal *local weight set* (denoted by W_j^l) which stores the same type of information as W_j^g but is merged into W_j^g in the second process. W_j^g is initialized when the node joins the system and W_j^l is initialized by the post-order tree traversal in the temporal SPT tree construction step.

The voting process starts from the leaf node set V_0^i , as shown in Algorithm 1 (lines 2–4). In the non-leaf node sets, nodes without conflicts will be processed first. If a non-leaf node n_j has no conflicts (Algorithm 1, lines 6–9), it will calculate w_j ($w_j = w(e(f(j), n_j))$) and propagate this weight to its parent’s local weight set, $W_{f(i)}^l$. For a non-leaf conflict node n_j , all the remaining upload units of n_j will be allocated to edges in T_0^i , as shown in lines 10–11 of Algorithm 1. Furthermore, *UnitReallocate* will re-allocate some units indicated in B_j and occupied by other SPTs to T_0^i . B_j , which is defined as the *backup unit set* of n_j will be explained later. If all upload units of the conflict node n_j are allocated to nodes in SPTs, w_j is calculated and propagated to $f(j)$. In the meantime, M^i is updated (Algorithm 1, line 19) by removing the remaining edges in W_j^l and their downstream edges and nodes.

When some remaining local edges have larger weights than edges in W_j^g , the upload unit scheduling algorithm, as shown in Algorithm 2, will be applied to calculate w_j and propagate it to the parent’s local weight set, $W_{f(j)}^l$.

Algorithm 2 $UnitSchedule(w_j, x, W_j^l, W_j^g)$

```

1:  $k \leftarrow 0, E \leftarrow \emptyset$ 
2: while  $x < c_j^i$  and  $k < |W_j^g|$  do
3:   if  $W_j^g[k] < W_j^l[x]$  then
4:      $n_s \leftarrow Root(W_j^g[k])$ 
5:      $n_z \leftarrow D_j^s \cap V_c^i$ 
6:      $w_j \leftarrow w_j + W_j^l[x]$ 
7:      $E \leftarrow (n_j, W_j^g[k])$ 
8:     if  $n_z \neq \emptyset$  or  $n_z \notin Leaf(T_{spt}^s)$  then
9:       if  $seq(n_z) < seq(n_j)$  and  $n_z \in V(M^i)$  then
10:         $Propagate(\Delta_z, z, j)$ 
11:       else if  $n_z \notin V(M^i)$  then
12:         $Propagate(\Delta_z, z, y)$ 
13:         $Propagate(\Delta_z, y, j)$ 
14:       else if  $seq(n_z) > seq(n_j)$  then
15:         $w_j \leftarrow w_j + \Delta_z$ 
16:         $B_z \leftarrow B_z + \{(n_j, s)\}$ 
17:       end if
18:     end if.
19:   end if
20:    $Update(V_c^i), Update(D_j^s)$ 
21:    $k \leftarrow k + 1, x \leftarrow x + 1$ 
22: end while

```

Let n_s denote the speaker that has an overlapped conflict node n_j with T^i . Let n_z refer to a descendant node of n_j , both in T_0^i and T_{spt}^s , and n_z be a conflict node in T_0^i . If n_z does not exist, or n_z is a leaf node in T_{spt}^s , the upload unit allocated for the edge $e(W_j^g[k])$ will be re-allocated to the edge $e(W_j^l[x])$ (Algorithm 2, line 6) since the re-allocation may bring in more receivers that have minimum latency. As a result, a tuple $(n_j, e(W_j^g[k]))$ will be inserted into E , the

