Multimedia Systems (2006) 11(6): 497–512
DOI 10.1007/s00530-006-0030-4

REGULAR PAPER

Leslie S. Liu · Roger Zimmermann

Adaptive low-latency peer-to-peer streaming and its application

Published online: 12 April 2006

Abstract Peer-to-peer (P2P) streaming is emerging as a viable communications paradigm. Recent research has focused on building efficient and optimal overlay multicast trees at the application level. A few commercial products are being implemented to provide voice services through P2P streaming platforms. However, even though many P2P protocols from the research community claim to be able to support large scale low-latency streaming, none of them have been adopted by a commercial voice system so far. This gap between advanced research prototypes and commercial implementations shows that there is a lack of a practical and scalable P2P system design that can provide low-latency service in a real implementation. After analyzing existing P2P system designs, we found two important issues that could lead to improvements. First, many existing designs that aim to build a low-latency streaming platform often make the unreasonable assumption that the processing time involved at each node is zero. However in a real implementation, these delays can add up to a significant amount of time after just a few overlay hops and make interactive applications difficult. Second, scant attention has been paid to the fact that even in a conversation involving a large number of users, only a few of the users are actually actively speaking at a given time. We term these users, who have more critical demands for low-latency, active users. In this paper, we detail the design of a novel peer-to-peer streaming architecture called ACTIVE. We then present a complete commercial scale voice chat system called *AudioPeer* that is powered by the ACTIVE protocol. The ACTIVE system significantly reduces the end-to-end delay experienced among active users while at the same time being capable of providing streaming services to very large multicast groups. ACTIVE uses realistic processing assumptions at each node and dynamically optimizes the streaming structure while the group of active users changes over time. Consequently, it provides virtually all users with the low-latency service that before was only possible with a centralized approach. We present results from both simulations and our real implementation, which clearly show that our ACTIVE system is a feasible approach to scalable, low-latency P2P streaming.

Keywords Peer-to-peer streaming · Application level multicast · Tree optimization · Floor control

L. S. Liu (✉) · R. Zimmermann
Computer Science Department, University of Southern California, Los Angeles, California 90089, USA
E-mail: [lleslie, rzimmerm]@usc.edu

1 Introduction

The expanding capabilities of the Internet to handle digital media streams is enabling new applications and transforming existing applications in many areas. Currently peer-to-peer architectures (P2P) have become a popular platform for very scalable applications, many of which were only viable using a centralized server just few years ago. This advance is made possible with the great improvements both in terms of the computing power of the personal computers used by regular consumers and the fast growing availability of broadband connections. In the last few years many P2P protocols have been designed and implemented. Since we are focusing on issues about latency performance, we surveyed many P2P system designs and applications for low-latency streaming. One interesting finding is as follows: on one hand there are many proposed P2P designs that claim to be scalable and suitable for streaming, on the other hand the few successful P2P based commercial applications are not using these proposed designs, and most of them sacrifice scalability in exchange for low latency. We conclude that currently there is no practical P2P design that can satisfy both scalability and low-latency.

Large scale low-latency streaming is very useful in many areas such as emergency communications, battlefield monitoring and multiuser chat room used in distance education. There are two major challenges in these applications: low-latency and scalability. In this paper, we will present a novel design called adaptive core-based tree for interactive virtual

environments (ACTIVE)[1] [1] that fuses these two features under one umbrella. Furthermore, a commercial level audio chat room that is built on the ACTIVE technology is also presented to show the feasibility of our design. The reason we chose a multiparty audio chat room to test our ACTIVE protocol is: multi-user audio conference is very demanding in terms of both delay and scalability. Previous research has concluded that a round trip time (RTT) latency of more than 200 ms will make a voice conversation difficult. This amount of delay can easily be exceeded when using traditional multi-hop P2P architecture. Also, our audio system is used in distance education classroom, which each room could concurrently be used by hundreds of students from all around the country. To the best of our knowledge, current popular audio systems, such as iChat and Skype, cannot support audio groups of this size.

In order to provide low-latency streaming in a scalable P2P architecture, our ACTIVE protocol addresses two major issues that have not been seriously discussed before. First, most existing P2P solutions ignore the processing delay introduced at each intermediate node when they optimize the P2P streaming structure. However, in an overlay network the application-layer processing time is usually much larger than the processing time in the physical network routers. Since the processing delay is introduced at each intermediate node when propagating data through the overlay path, these relay latencies can add up to a significant amount after just few overlay hops. For example, we measured the processing delay at each overlay node in an application called AudioPeer [2] that uses ACTIVE for distance online education and we found that the average processing delay at each node is approximately 30 ms. Compared with the two to ten milliseconds physical network latency between the hosts we tested, this contributes a significant amount to the overall end-to-end delay. The average end-to-end delay in traditional P2P systems quickly exceeds the threshold for interactive scenarios when the group size grows. This explains why full-connected (i.e., mesh) designs dominate P2P systems which require low-latency performance. However, full-connected designs suffer from certain limitations. Since everybody is connected to everybody else, the system is not scalable. With today's advanced computer technology, this design may be capable of providing service to group sizes of tens, but not hundreds or thousands of participants. Second, even though the total number of users in P2P groups is often large, frequently only a small fraction of these users are interacting with other users at any given time. We call them *active users*. For example, in a large online classroom for distance education, the number of users joining a session could be in the hundreds, but the number of active users (e.g., the instructor plus the students who are asking questions), is limited to a few participants. This phenomenon provides us with an opportunity to optimize the P2P structure for better end-to-end delay among this relatively small group. To the best of our knowledge, no existing P2P architecture has been designed to take advantage of this opportunity.

[1] Adaptive Core-based Tree for Interactive Virtual Environments

Our ACTIVE protocol is capable of providing interactive streaming services to large groups. ACTIVE contributes the following new ideas to P2P architectures. First, it distinguishes active users from passive users so that an intelligent optimization can be performed on this subgroup instead of the whole group. Second, it dynamically adapts the P2P structure to maintain delay service quality for active users. Finally, it uses a dynamic floor control mechanism to allow a growing number of active users grows as the size of the multicast group increases. The floor control constraint can be adjusted while the system is running to dynamically allow more or less active participants. By dynamically optimizing the P2P structure, ACTIVE can significantly reduce the end-to-end delay among active users and at the same time, provide streaming service to very large multicast groups. In ACTIVE, virtually all users are provided with the low-latency service that before was only possible in a centralized or full-connected approach.

The rest of this paper is organized as follows: Sect. 2 lists some of the related work. Section 3 describes the design of ACTIVE in detail, which is followed by Sect. 4 on how we simulate and evaluate the performance of ACTIVE. Section 5 introduces a multiuser audio chat room using ACTIVE as the streaming protocol and the user feedback is discussed on Sect. 6. Finally we draw conclusions in Sect. 7 and show our gratitude in Sect. 7.

2 Related work

Many P2P architectures (e.g., Narada [3], Yoid [4], HMTP [5], NICE [6], OMNI [7], SCRIBE [8], CAN-Multicast [9], Zigzag [10], Skype [11] and Ostream [12]) have been proposed or adapted for streaming media services. However, due to the long end-to-end delay in overlay networks, these designs either make the impractical assumption that the processing delay on relay nodes is ignorable and hence failed to provide low latency service for real applications, or manage to provide low-latency streaming by using a full-connected architecture that is not very scalable.

Narada [3] is a mesh-based approach for many-to-many streaming. It constructs a tree whenever a sender wants to transmit a media stream to receivers. Due to the heavy control overhead, Narada does not scale well to large P2P groups. NICE [6] is designed to support a large streaming receiver set and its multi-layered design reduces the control overhead. A similar design is proposed in Banerjee et al. [7], which additionally tries to optimize the overall end-to-end delay among all streaming receivers. Zigzag [10] optimizes the performance of NICE by constraining the control overhead to a constant value. In these three designs, all peers are considered to have the same delay requirement and optimization is performed to reduce the delay among all peers, not among active users as in ACTIVE. The failure to distinguish between active and passive users makes it problematic for them to achieve low latency performance in an multi-hop architecture design. SCRIBE [8] and CAN-multicast [9] are

based on Distributed Hash Tables (DHT), which are used to generate node identifiers and then create a multicast tree. In these approaches, the multicast path between nodes is determined by the current topology of the multicast members, hence the delay between users cannot be reduced dynamically, as in ACTIVE, by optimizing the tree connections. Skype [11], which is also based on peer-to-peer technology, is emerging as a popular online audio conferencing application. Skype can provide low-latency audio service, but with a limitation on how many users can participate in the same room. Ostream [12] is designed to utilize the strong buffering capacities on the multicast overlay nodes and thus reduce the network bandwidth requirement for on-demand media distribution. Due to the delay introduced by the buffering at intermediate nodes, Ostream is not suitable for an interactive live streaming environment such as an audio chat room.

3 The design

The design of ACTIVE is comprised of five components: (a) classification of active and passive users, (b) the Credit Point system, (c) tree construction and maintenance, (d) tree optimization, and (e) the embedded floor control mechanism. We will now describe these five components in turn. Some of the frequently used terms and their definitions are listed in Table 1.

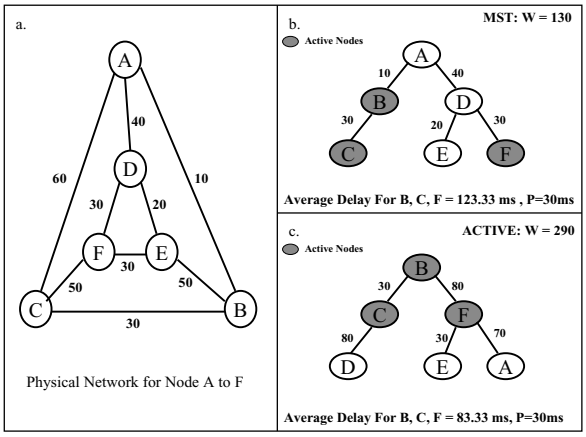
There are two major performance challenges for all P2P streaming systems: high scalability and low latency. High scalability refers to the ability to provide streaming services to large multicast groups without causing congestion on the physical network. Except for a few early designs [3], most current P2P architectures scale well to very large groups. Traditionally, P2P systems have been used for distributing stored content, where latency is not a critical design issue. However, when P2P technology is adapted for live streaming, low latency becomes an essential requirement. Tree-based approaches are very popular among systems designed for low-latency applications because the minimum spanning tree (MST) is proven to be able to provide the minimum end-to-end delay among all nodes. Since most multicast tree designs impose a degree limit K at each node, the MST problem becomes $\mathcal{NP}$-hard [13]. The performance of all existing tree-based solutions is bound by the performance of the optimal tree constructed with the MST algorithm.

Distinguishing active users is crucial to providing low-latency streaming service. Existing systems are building their overlay topologies to closely match an MST tree. However, as illustrated in Fig. 1a–c a MST tree linking all multicast nodes cannot guarantee the minimum delay among active users, where low latency is critical. Figure 1b shows that when the application layer processing delay is 30 ms at each node, the average overlay delay among active nodes is 123.33 ms. Shown in Fig. 1c, the delay in the ACTIVE generated tree is only 83.33 ms.

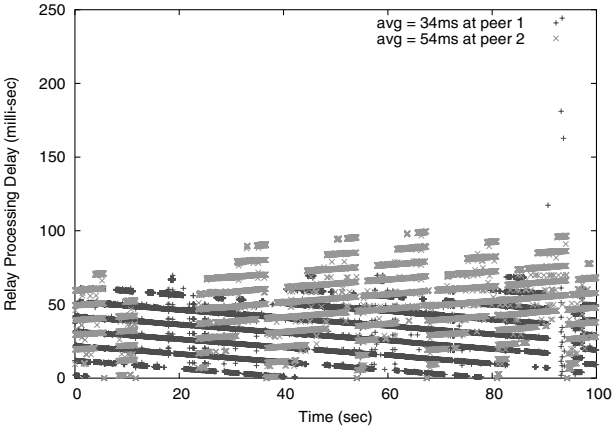
When the application processing delay P is added to the networking delay, the average latency in overlay P2P systems quickly rises above the threshold for a smooth interactive experience when the group size increases. This occurs even if an optimal tree is built with the MST algorithm. The application processing delay P is determined by various factors, e.g., the operating system, the CPU speed, and the available memory. Figure 1d shows the processing delay at two of the intermediate nodes of an audio conference application using ACTIVE.

Table 1 List of terms used in this paper and their respective definitions

Term	Definition	Units
V	Set of all users on current multicast group, $\|V\|$ is the total number of all users	
A	Set of all active users on current multicast group, $\|A\|$ is the total number of active users	
G	Application level complete graph containing all nodes in V	
E	Set of edges that connect all nodes in V, $E = V \times V$	
$\|E_{i,j}\|$	The physical network delay between host i and host j	
$N_{A'}$	Maximum number of active users allowed	
P	The application-layer processing delay. Also called relay delay.	ms
T	Current multicast tree that connects all nodes in V	
R	System optimization frequency control parameter	Hz
O_m	System wise control overhead for operation M	
I_i	Idle time at host i	s
K	Degree limit of T, maximum number of children allowed for each node in V	
F	System floor control parameter	
$D_{i,j}$	Overlay data delivery path from host i to host j	
$\|D_{i,j}\|$	Overlay end-to-end delay from host i to host j	ms
D_A	System wise average end-to-end delay among all active nodes in A	ms
$D_A(i)$	Average end-to-end delay from host i to all other active users	ms
D_V	System wise average end-to-end delay among all nodes in V	ms
$D_V(i)$	Average end-to-end delay from host i to all other host in V	ms
$HOP_{i,j}$	Number of intermediate relay nodes between host i and host j	
CP_i	Credit Point assigned to node i	
CPt	System wide CP threshold that allows a user to switch to active mode	
RDP-AU	Ratio of average overlay delay for active nodes compared to MST tree result	
RDP-ALL	Ratio of average overlay delay for all nodes compared to MST tree result	



(a)-(c) MST vs. ACTIVE



(d) Application-layer Processing Delay

Fig. 1 Examples

The goal of ACTIVE is to provide low latency streaming service to large multicast groups. ACTIVE extends the P2P service to the interactive domain where centralized approaches are still dominant. In this report, we present how ACTIVE brings together the best features of both centralized and distributed systems, namely low latency and high scalability.

To this end, ACTIVE builds a core-based tree T that includes all group members, and dynamically optimizes the tree to minimize the average delay among active users. Applications that leverage the ACTIVE streaming service are responsible to inform their ACTIVE module when the user switches between active and passive mode. Based on the provided information, ACTIVE will automatically construct, maintain and optimize the overlay topology to satisfy the performance requirements.

3.1 Problem formulation

3.1.1 Network model

The physical network consists of routers connected via links, and end-hosts that are connected to these routers through access connections. The delay between end-hosts and their accessing router is smaller than the average delay between routers. This models both the characteristics of a intra-domain and inter-domain network.

Subsequently, the overlay network is built from end-hosts and on top of the physical network topology. The overlay network can be modelled as a complete directed graph, denoted by $G = (V, E)$. V denotes the set of all end hosts and $E = V \times V$ refers to the set of connections. $E_{i,j}$ denotes the directed edge from host i to host j, while $|E_{i,j}|$ represents the physical network delay between host i and j. We assume symmetric links, i.e., $|E_{i,j}| = |E_{j,i}|$.

The multicast tree T is a subset of G and represents the P2P topology. The data delivery path $D_{i,j}$ consists of all the intermediate nodes on T from host i to host j and all the edges connecting them. $|D_{i,j}|$ represents the overlay end-to-end delay between host i and host j on T. It is calculated as shown in Eq. (1), where P_k denotes the processing delay at each of the intermediate node K on the path from host i to host j.

$$|D_{i,j}| = \sum_{k=i}^{j} P_k + \sum_{E_{m,n} \in D_{i,j}} |E_{m,n}| \tag{1}$$

For a distributed application, the processing delay P_k accounts for a large component of the overall delay. In turn, the processing delay is determined by a lot of factors, e.g., streaming type, compression algorithm, computing power of the peer machine, etc. We model the processing delay at a audio stream mixing node that use the GSM.610 audio codec in Eq. (2).

$$\begin{aligned} P_k = {} & T_{\text{recv\_buffer}} + T_{\text{decode\_buffer}} + T_{\text{mix\_processing}} \\ & + T_{\text{encode\_buffer}} + T_{\text{send\_buffer}} \end{aligned} \tag{2}$$

We can see that in a peer-to-peer network, P_k is usually not the same because of the varying computing power at each node. The decode/encode buffer delay is defined by the compression algorithm and the receive/send buffer is defined by the application's network module. For example, there is a 34 ms processing delay on average in one of our test nodes illustrated in Fig. 1d.

3.1.2 Solution objective

Recall that the objective of ACTIVE is to minimize the *average delay among active users*, D_A, in a given multicast group V. This problem can be formally stated as *degree-bound minimum spanning tree problem:* For a given graph G and user set V, find a tree T', where $T' \subseteq G$, so that the degree constraint is satisfied at each node and $\sum_{i,j \in A} |D_{i,j}|$ is minimized.

The degree-bound minimum spanning tree problem is $\mathcal{NP}$-hard, so we propose a heuristic solution which is described in detail in Sect. 3.3. As a basis of the proposed solution, we designed a distributed score computing algorithm termed Credit Point system, which is described next.

3.2 Credit point system

As an integral part of the design of ACTIVE, we introduce a credit point (CP) system. From the observation of the dynamics of active users in a large online audio conference system, we conjecture that the number of active users grows sub-linearly with the size of the group. In our CP system, the total number of credit points grows logarithmically with the group size. As we will see soon, the credit points control the total number of active users in the system and how ACTIVE optimizes the tree. The CP system provides some key features of ACTIVE and works as follows.

Each node is assigned a CP value when it joins the multicast tree T, with a value ranging from zero to one. If the degree limit of T is denoted K, then the formula for CP assignments is as follows:

$$\begin{aligned} \mathrm{CP}_{\mathrm{root}} &= 1 \\ \mathrm{CP}_{\mathrm{Child}} &= \mathrm{CP}_{\mathrm{Parent}}/K, \quad K \geq 2 \end{aligned} \tag{3}$$

Once a node i is assigned its CP_i, it will keep the value until there is a topology change. As mentioned earlier, ACTIVE distinguishes the nodes in V as active users A and passive users $V - A$. A user can switch from active to passive status, or vice versa. CP_i is used to validate these transitions in a distributed manner. CPt denotes the system-wide threshold for switch transitions, and the validation condition is shown in Eq. (4).

$$\mathrm{CP}_i \geq \mathrm{CP}t, \quad i \in V \tag{4}$$

If Eq. (4) is satisfied, node i can switch from passive to active mode. Since nodes in active mode usually generate more data for delivery, Eq. (4) is also called the *Floor Control Equation*. In ACTIVE, there is no constraint for switching from active to passive mode. The total number of active nodes $|A|$ is bound by CPt and degree K. If we denote $N_{A'}$ as the maximum number of active nodes allowed, then the following equation illustrates how $N_{A'}$ is constrained by CPt and K:

$$|A| \leq N_{A'} = \frac{K - \mathrm{CP}t}{\mathrm{CP}t \times (K-1)}, \quad K \geq 2, \;\; 0 < \mathrm{CP}t < 1 \tag{5}$$

The degree limit K is usually stable for a multicast configuration, but the threshold CPt should change according to the size of the group. If we denote F as the dynamic floor control parameter with a value ranging from 0 to 1, ACTIVE calculates CPt based on following equation:

$$\mathrm{CP}t = \frac{F}{\log_k |V|}, \qquad 0 < F < 1 \tag{6}$$

By substituting Eq. (6) into Eq. (5), we arrive at an equation to calculate $N_{A'}$ from V directly: $N_{A'} = \frac{K \times \log_k |V| - F}{F \times (K-1)}$. This shows that the maximum number of active users $N_{A'}$ is a logarithmic function of the total number of users $|V|$.

The complexity of the CP system is low and, since the active user set A is usually only a small subset of V, ACTIVE is able to achieve close to optimal performance in most cases (described in Sect. 4.2).

3.3 Heuristic solution

Finding an optimal solution for degree-bound minimum spanning tree is $\mathcal{NP}$-hard, therefore we propose a heuristic solution which can be computed in a fast and distributed fashion. The basic idea is as follows: Since the number of active users is relatively small even in a large group, forming a cluster among active users in which every active user is *directly* connected to another active user will effectively reduce the number of intermediate nodes, thus reducing the processing delay introduced by these nodes. This solution is formally stated as follows.

3.3.1 Heuristic solution to solve degree-bound minimum spanning tree problem

For $\forall i, j \in A$, $HOP' \leftarrow \max(|HOP_{i,j}|)$. Find a subtree T' ($T' \subseteq T$, $A \subset T'$), so that the following condition is satisfied:

$$\begin{cases} HOP' \leq \log_k \dfrac{1}{\mathrm{CP}t^2} - 1 \\ \forall i \in A, \quad \exists j : |HOP_{i,j}| = 0, \quad j \in A, \quad j \neq i \end{cases} \tag{7}$$

The above algorithm is designed to be efficient and minimize the control overhead by avoiding an exhaustive search for an optimal result. An efficient solutions is desirable for the following reasons. First, while a complex heuristic algorithm may generate better results in a static topology, it must restart the compute process whenever there is a topology change. P2P systems are highly dynamic environments where the overlay topology is frequently changing, and a complex algorithm may need a long time to reach its result. Second, the novel design of distinguishing active users from passive users makes ACTIVE able to establish very low-latency service among active users, thus reducing the need to compute an optimal result. In Sect. 4 we show that ACTIVE can even generate better performance among active users compared with the MST solution.

Note that even though the goal of the optimization process is to reduce the delay among active users, in our proposed heuristic solution we do not need to measure the delay among active users in order to achieve our performance goal, thus reducing the control overhead in our system.

3.4 Tree construction

The construction of an ACTIVE system is formally the process of building a multicast tree. Tree maintenance is required to repair the tree when there is an error in the tree topology. In this section, we focus on the construction and maintenance procedures of the multicast tree T. Tree optimization in ACTIVE is discussed in Sect. 3.5.

3.4.1 Join

During the join process, ACTIVE uses a heuristic Shortest Path Tree (SPT) algorithm (Algorithm 1) to construct the tree. A new node starts the join process by contacting a rendez-vous point (RP) node. Such a bootstrapping technique is widely used in peer-to-peer applications. The RP node could be a dedicated, well-known service or just any node that participates currently in the P2P multicast group, as long as the new node knows the RP's network address. After joining the P2P system, a node runs independently of the RP node.

Algorithm 1 *JOIN*

Require: RP is online
1: $L \leftarrow$ candidate nodes from RP
2: **while** $L \neq \emptyset$ **do**
3: $C \leftarrow$ nearest node in L
4: **if** C is ok to join **then**
5: setup connection with C
6: $parent \leftarrow C$
7: break while loop
8: **else**
9: $L \leftarrow$ add new candidate nodes referred by C
10: remove C from L
11: **end if**
12: **end while**

A new node obtains a list of candidate parents L during the join process. The new node finds its nearest neighbor by sending out request messages simultaneously to all candidate nodes in list L. A candidate node will immediately reply once it receives the request message. The new node recodes L in the order in which it receives the response messages and the first responding node C in L will be considered the nearest node. Node C will be chosen as the candidate parent and a setup process immediately follows. If an error is causing a setup failure (e.g., C exceeds its degree limit or suddenly goes offline), the new node will remove this candidate from L, choose the next node C' at the head of list L and starts the above process again.

3.4.2 Leave

A leaving node must perform a few steps, as shown in Algorithm 2, to ensure that after its departure the multicast tree T is loop-free and all nodes are reachable through T. Failure to finish these steps is considered an error in ACTIVE and its impact is discussed in Sect. 3.4.3. The only exception occurs when the current node has no child nodes. In this case, the node can simply disconnect from the service.

Algorithm 2 *LEAVE*

1: inform all neighbor nodes that N is leaving
2: **if** N is the core
3: $C' \leftarrow$ nearest neighbor node
4: setup C' as the new core
5: inform all other neighbor nodes to set C' as parent
6: **else**
7: **if** N has child node **then**
8: $N' \leftarrow N$'s parent
9: inform all child nodes to set N' as parent
10: **end if**
11: **end if**
12: disconnect from the service

3.4.3 Error detection and recovery

In a tree-based architecture there are two types of errors in the topology: loops and tree splits. ACTIVE uses different mechanisms to detect these two errors.

A tree split can be easily detected at the nodes which lost connections to their parents or children. To avoid redundant message exchanges, we use an asymmetric scheme: a lost connection to the parent is considered an error at the children nodes but a lost connection to a child node is not an error at the parent node. This rule makes it a child node's responsibility to repair the tree and find a new parent when the connection is broken. Detecting a connection error is simple in ACTIVE because it utilizes the TCP protocol in order to stream to nodes behind NAT devices, and TCP will automatically detect and report disconnection errors. The process of finding a new parent is different from the join process and can be done without help from the RP server. As mentioned in Sect. 3.4.1, each node saves a list of usable candidate parent nodes L during the join procedure. When a node loses the connection to its parent, L will be used to find a new parent, following the process described in Algorithm 3. Note that the rejoin process is different from the join process. First, when initially joining, a node tries to find the closest node in L, while with rejoin a node is attached to the first usable node in L. Second, the rejoin process is independent of the RP server, which is required as a starting point in the initial join process. These differences are designed to make the rejoin process in ACTIVE fast and scalable.

Whenever a node i changes its parent, we consider this to be a topology change in the multicast tree. Node i will send out update messages to its direct child nodes, which will update their local information and in turn forward the update message to their child nodes. If this change at node i forms a loop, the update message from node i will eventually be forwarded back to itself, and a loop error is

detected. Once node i detects a loop, it will disconnect from the parent and start the rejoin process. Note that in the rejoin process, the new parent must have an equal or bigger CP value than CP_i. This will avoid node i to form another loop by joining the nodes in the subtree rooted at itself, where every node has a smaller CP value.

Algorithm 3 *REJOIN*

```
1:  L ← the candidate node list built in JOIN process
2:  while L ≠ ∅ do
3:     C ← first node in L that has bigger or equal CP
4:     if C is ok to join then
5:     setup connection with C
6:        parent← C
7:        break while loop
8:     else
9:        L ← add new candidate nodes referred by C
10:       remove C from L
11:    end if
12: end while
```

Note that ACTIVE is an event-driven protocol and there is no message flooding at any time during tree construction, tree maintenance or tree optimization.

3.5 Tree optimization

The delay among active users is gradually decreased by executing a tree optimization algorithm. The optimization function is run at each node and triggered by the users' requests to become active. For example, in an audio conferencing application, speakers are active users, and passive users can become active either manually, e.g., by pressing a button, or automatically when there is a voice input detected. The local node will determine whether the request can be granted based on Eq. (4). If the request can not be granted, the optimization process is triggered.

3.5.1 Clustering active users

The clustering is achieved by moving the active nodes gradually towards the root of the tree until Eq. 4 is satisfied. This move is accomplished by exchanging an active child node with its non-active parent or another higher-level node (Algorithm 4). It is worth mentioning that even though the complete optimization may take as long as a few seconds, each step only requires a few milliseconds to setup the new streaming connections. The loss of data during these steps is so small that it does not affect the playback quality. In fact, in the audio conferencing application that runs on the ACTIVE protocol, we cannot detect any audible glitches during the optimization.

Algorithm 4 *OPTIMIZE*

```
1:  AGAIN:
2:  P ← parent of local host i
3:  I_P ← idle time at P
4:  if I_P ≤ R^{-1} then
5:     return
6:  end if
7:  //First Phase
8:  while P is not active do
9:     if P is core then
10:       ask P to set local host i as parent
11:       setup local host i as core
12:    else
13:       P′ ← the parent of P
14:       ask P to set local host i as its parent
15:       ask P′ to set local host as new child
16:       P ← P′
17:       setup connection with P′
18:    end if
19: end while
20:
21: //Second Phase
22: IF CP_i ≥ CPt then
23:    send update message to all children nodes
24:    return
25: else
26:    L′ ← all passive node immediate connected
             to A
27:    remove all host j from L′ if CP_j ≤ CP_i or
             CP_j = 1
28:    if L′ = ∅ then
29:       send update message to all children nodes
30:       return
31: else
32:       C ← first node in L′
33:       P′ ← the parent of C
34:       ask C to set P as its parent
35:       ask P′ to set local host as new child
36:       P ← P′
37:       setup connection with P′
38:    end if
39: end if
40: goto AGAIN:
```

The optimization algorithm first observes the idle time I_P at the parent node to see if it has been idle for long enough. The system optimization frequency control parameter R determines how frequently a reconstruction can be performed on a parent node. If $I_P \leq R^{-1}$, the optimization is canceled. For example, in a system with $R = 0.1$, a node would need to continuously remain idle for 10 s before another node can exchange position with it. As shown in Algorithm 4, the rest of the optimization procedure is performed in two phases. In the first phase, the active node i gradually moves towards the root until its parent is also an active node. In the second phase, ACTIVE condenses the cluster of active users further if necessary.

If the optimization function returns without being able to find a better position for node i, it means there are currently too many active users in this multicast group, and node i will be denied to switch to active status. This also functions as part of the floor control mechanism in ACTIVE, which is discussed next.

3.6 Floor control

A floor control mechanism is of practical importance for large scale streaming systems because too many active users may saturate system resources or degrade the overall streaming quality. For example, if too many people are talking simultaneously in an audio chat room, the conversation will become incomprehensible.

ACTIVE implements a dynamic floor control function (Eq. (5)) to control the total number of active users. The maximum number of active users $N_{A'}$ gradually increases along with the size of the group $|V|$. In Eq. (6), F is called the floor control parameter. A system administrator can dynamically change F to lower or raise the threshold CPt, and thus control the total number of active users in the system. We reformat Eq. (5) as follows:

$$N_{A'} = \frac{K \times \log_k |V| - F}{F \times (K-1)} \tag{8}$$

If the system requires that all users can be active at the same time, the floor control mechanism can be conveniently disabled by setting F to the value of F' calculated in Eq. (9). It is not hard to prove that in this case Eq. (4) is always satisfied.

$$F' = \frac{K \times \log_k |V|}{|V| \times (K-1) + 1} \tag{9}$$

3.7 Control overhead analysis

We define the control overhead O for an operation M as the total number of messages received at all involved end-hosts multiplied by the frequency of this operation. Since floor control is performed along with tree optimization, there is no floor control overhead added to the ACTIVE system.

Many existing P2P protocols depend on a refreshment based mechanism to maintain their service. For example, in the NICE protocol, each node i needs to send out a *Heart-Beat* message periodically to all other nodes in the same cluster. This process continues to consume network bandwidth as long as the node is in the system. If we denote the refreshment period as h, and N' as the average number of nodes receiving the refreshment message, the control overhead O_r for all refreshment based protocols is:

$$O_r = \frac{|V| \times N'}{h} \tag{10}$$

ACTIVE uses the CP system to distribute the computation for tree maintenance and optimization. Because ACTIVE is an event-driven system, in the worst case messages are required to be sent to all nodes in V. If the operation happens with the same frequency as in the refreshment-based approach, the control overhead in ACTIVE is:

$$O_A = \frac{|V|}{h} \tag{11}$$

Equation (11) shows that given the same event frequency, ACTIVE achieves a much smaller control overhead compared with refreshment based designs.

4 Performance evaluation

We evaluated our ACTIVE design with simulations in an NS-2 environment [14]. The ACTIVE code used in the simulation is the same as the one used in the audio chat room application running on Windows. Only minor changes were made to compile the code in a Linux environment. A module to collect the actual delay among nodes was also added. In the simulation we compared ACTIVE's performance to trees generated by Prim's minimum spanning tree algorithm [15] with the same physical network topologies. The generated MST trees are the optimal solution for all existing tree-based designs (e.g., [3–6]) which assume no application-layer processing delay at each node. Note that these MST trees are not the optimal solutions for ACTIVE.

4.1 Performance metrics

In our performance evaluation, the MST tree is used as the performance baseline. Here we introduce two terms to describe the performance: RDP-AU and RDP-ALL. RDP denotes the *relative delay penalty*, which is the ratio of the average overlay delay in ACTIVE to the average overlay delay in MST. RDP-AU is the RDP calculated only for all active nodes and RDP-ALL is calculated for all multicast nodes. Smaller RDP values indicate better performance and an ACTIVE tree outperforms the MST tree if its RDP-AU is smaller than 1. Note that RDP-ALL is never less than 1.

In the NS2 environment, each node i can easily calculate the overlay delay to another node j by comparing the global time-stamp of a packet received from node j to the current system global time. The average overlay delay at node i to all active nodes can be calculated as $D_A(i) = \frac{\sum_{j \in A} |D_{i,j}|}{|A|-1}$ or $D_V(i) = \frac{\sum_{j \in V} |D_{i,j}|}{|V|-1}$ to all nodes in V . All nodes report their $D_V(i)$ and $D_A(i)$ to the RP server, where the overall average delay is calculated by using the following equations:

$$D_A = \frac{\sum_{i \in A} |D_A(i)|}{|A|}, \quad i \in A \tag{12}$$

$$D_V = \frac{\sum_{i \in V} |D_V(i)|}{|V|}, \quad i \in V \tag{13}$$

To measure the performance of ACTIVE, we need to measure the actual delay in our performance evaluation experiments. An delay monitoring module is added to ACTIVE to collect such data. Each active node sends out update messages to other nodes to calculate the overlay delay when overlay topology changes. In the NS-2 environment, there exists a global system timer that every node can access. This makes the delay measurements easy and accurate. When a node i sends out an update message, it stamps the message packet with its sending time. The update packet is forwarded at each node that receives it, after adding the processing delay P. Each node, e.g., host j, upon receiving an update message, can simply subtract the sending time from current system time and obtain $|D_{i,j}|$, the overlay delay between host i and host j. As each node knows if its remote nodes are active or passive by using the information contained in the update messages, it can calculate $D_A(i)$ and $D_V(i)$ as follows.

$$D_A(i) = \frac{\sum_{j \in A} |D_{i,j}|}{|A| - 1}, \quad i \neq j \tag{14}$$

$$D_V(i) = \frac{\sum_{j \in V} |D_{i,j}|}{|V| - 1}, \quad i \neq j \tag{15}$$

Each node will send its $D_A(i)$ and $D_V(i)$ to RP server node, where the system wide D_A and D_V are calculated:

$$D_A = \frac{\sum_{i \in A} |D_A(i)|}{|A|}, \quad i \in A \tag{16}$$

$$D_V = \frac{\sum_{i \in V} |D_V(i)|}{|V|}, \quad i \in V \tag{17}$$

For a given network topology, $|E_{i,j}|$ is fixed between any host i and host j. Also, for a given multicast tree T, the hop distance between any two hosts is also a constant. When we take a snapshot of the system topology at a particular time, D_A and D_V are linear function of P. Our experimental results show this relationship in Fig. 2.

4.2 Simulation setup

The router-level physical network is generated according to the Transit-Stub graph model, using the Georgia Tech Internetwork Topology Models (GT-ITM). Delay between routers is randomly distributed from 5 to 35 ms. Each end-host node is randomly attached to one of the routers with an access delay of 0.5 ms. Figure 3 shows one of the topologies we generated with 400 participants and 20 active users. In addition, based on the measurement we conducted with the audio application (Fig. 1d), we used the average value $P = 30$ ms as the processing delay at each node to simplify the simulation. However, using this specific value does not affect the final conclusion of our simulation results.

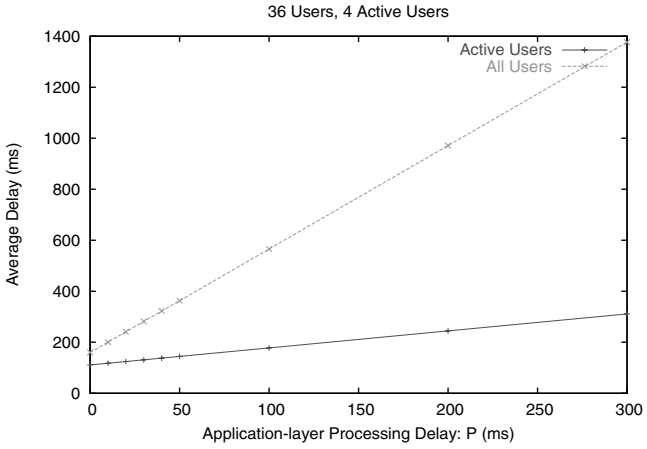


Fig. 2 Example: D_A and D_V vs. P

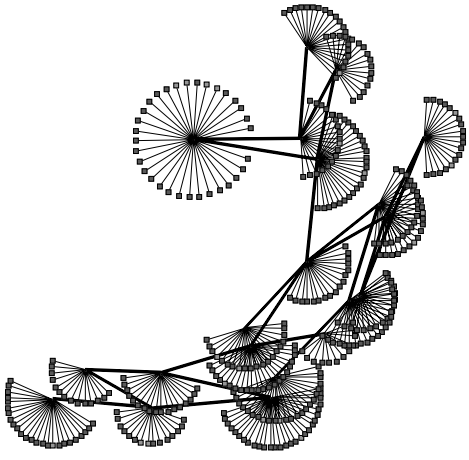
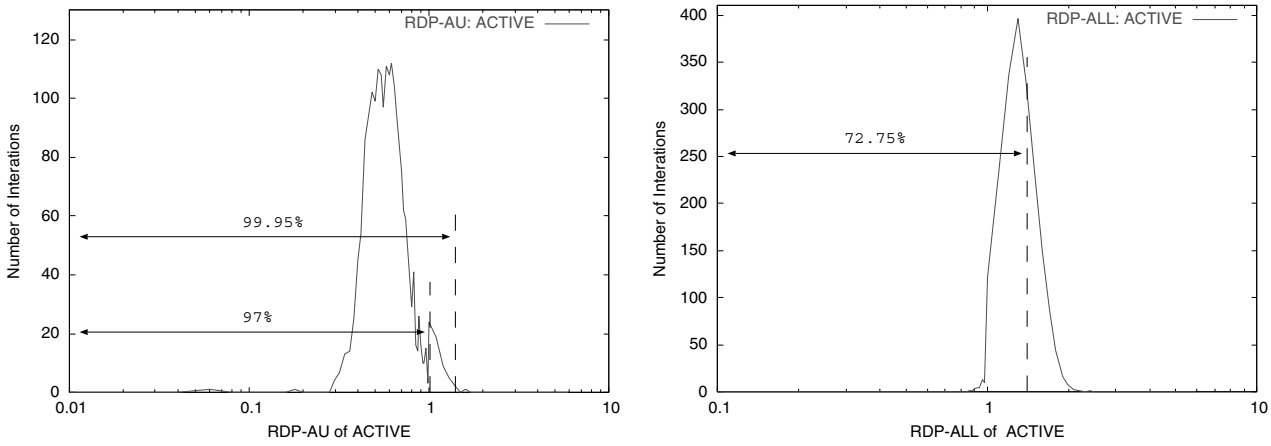


Fig. 3 Network topology (400 users)

4.3 Simulation results

ACTIVE achieved dramatic performance improvements over other existing systems in our simulation. We ran 2000 iterations of the simulation with uniformly distributed user groups ranging in size from 4 to 400 and active user groups ranging in size from 4 to 20. Figure 4a shows that compared with the MST solution, ACTIVE achieves a smaller delay among active users in 97% of the 2000 simulations we conducted. If we consider $RDP\text{-}AU = 1.5$ as an evaluation threshold for good overlay delay performance, then in an astounding 99.95% of all cases ACTIVE delivered good performance.

For reference purposes, we also calculated the average delay among all users. As illustrated in Fig. 4b, ACTIVE delivered good performance in 72.75% of all cases. It is interesting to see that in 1.90% of the cases, ACTIVE can, surprisingly, provide better performance than the MST tree. After investigating the cause, we found that these counter-intuitive results are not incorrect. For all leaf nodes, which



(a) for active users (b) for all users

Fig. 4 Delay performance of ACTIVE

will not perform any forwarding job, the application processing delay will not be counted for the overall delay. This means that the more leaf nodes are in the overlay tree, the less overall delay we can achieve. The MST algorithm only guarantees to generate a tree with minimum tree weight, not the minimum number of leaf nodes. Hence, in some rare cases, ACTIVE can outperform MST.

4.3.1 Experimental scenario

To show how tree optimization in ACTIVE reduces the delay among active users over time, we divided our simulation iterations into three phases: A, B and C. In phase A, the ACTIVE protocol did not identify any active users, and no optimization was performed; in phase B, active users started to emerge at random times, and optimization was dynamically invoked to optimize the tree. In phase C, all active users were identified and optimization completed, hence the multicast tree became stable.

We chose three iterations from the 2000 simulations as examples to illustrate how ACTIVE optimizes the delay performance among active users over the time (Fig. 5a–c). In all these three iterations, ACTIVE has a smaller application level delay D_A than the MST when the simulation finishes phase B and reaches phase C. It is very important to recognize that by having a close performance compared to MST, as shown in Fig. 5a and c, ACTIVE out-performs other existing algorithms which consider MST as the optimal solution. Also illustrated in these figures, the average overall end-to-end delay had not been significantly changed during the optimization process. Our experimental results show that while D_A is reduced by 50% or more, D_V remains almost the same in most cases, or is even reduced in some cases.

It is also worth noting that while ACTIVE has been deployed for practical use, the MST algorithm is computationally too complex to be used in any real-time system. The computation complexity of Prim's algorithm is $O(|E| + |V| \log |V|)$.

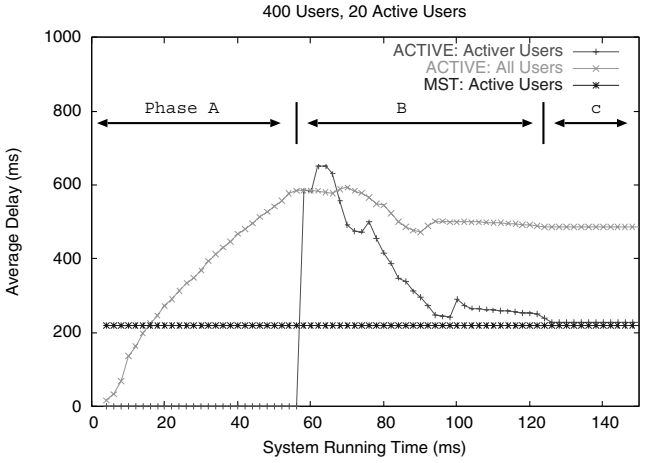
5 Application using ACTIVE

Educational tools that utilize the Internet to reach off-campus students are becoming more popular and many educational institutions are exploring their use. Here we describe a multiuser audio chat system called *AudioPeer*. AudioPeer is built on the ACTIVE protocol, it is designed to foster collaboration and interactive learning between students, teaching assistants and instructors. The AudioPeer system provides an interactive platform where groups of students can discuss assignments, teaching assistants can conduct lab sessions and professors can answer questions from students during lectures. The decentralized nature of the AudioPeer system avoids bottlenecks and allows it to scale to large groups of participants.

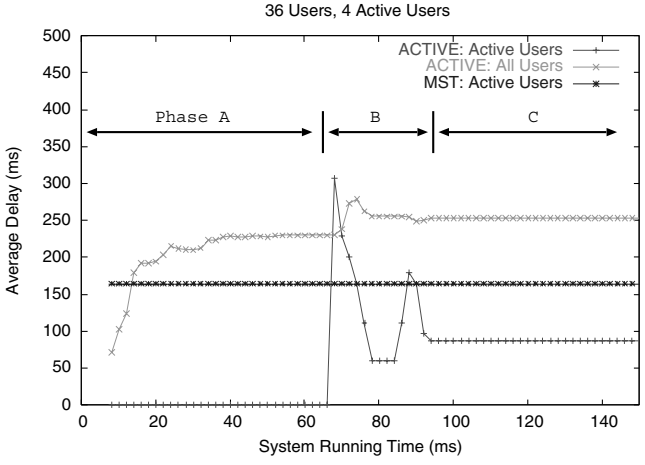
A multiuser audio chat system involves numerous technical challenges that need to be addressed to build such an application. The number of participants in a chat session may be several dozens, with each student needing to hear and possibly talk to any other person in the session. Additionally, the end-to-end audio latency needs to be kept sufficiently low such that natural interaction is possible. Hence, we aim for our system to be scalable, practical (e.g., work with different types of network connections), integratable with other distance education components, and extensible with new features (e.g., speaker recognition). In the following subsections, we will describe the system architecture of AudioPeer, its multiple components and how it integrates with the existing distance education infrastructure called DEN [16] at our university.

5.1 System architecture

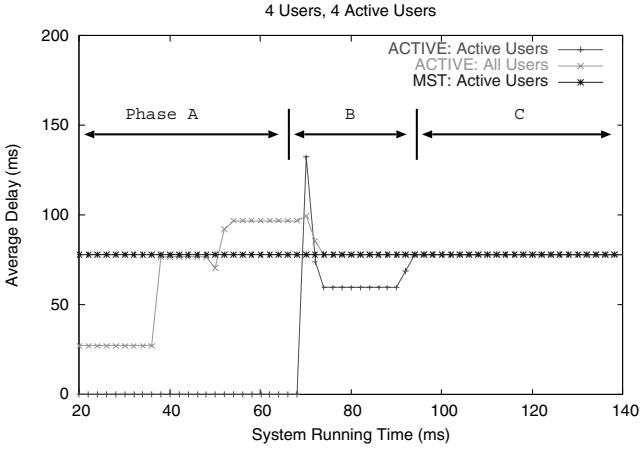
AudioPeer is a P2P based audio chat system applicable to, for example, distance education. As illustrated in Fig. 6, users are connected to each other in a decentralized



(a) Large group.



(b) Medium group.



(c) Small group.

Fig. 5 Examples of average delays for three different groups

fashion. Every user is regarded as a connecting node in the streaming topology and all nodes are connected to form a shared-multicast tree. Voice streams are merged at each node to maintain a fixed bandwidth requirement regardless of the number of users in the system. As most P2P designs do, AudioPeer also contains centralized components, namely the website and the rendezvous point server. However, these centralized parts serve principally as the bootstrap point and do not affect the scalability of the overall streaming.

5.1.1 Website

AudioPeer is currently designed as a web-based system. This enables us to deploy and upgrade the system in a fast and easy way. Also, since most of DEN's functions are also web-based, this approach makes AudioPeer an integrated component of the DEN website.

Figure 7 shows a screen shot of the current AudioPeer homepage. It provides simple but detailed step-by-step instructions on how to setup the AudioPeer plug-in for new

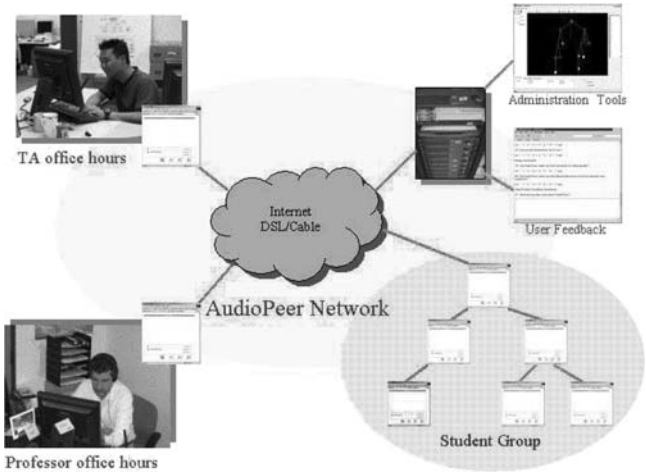


Fig. 6 AudioPeer architecture

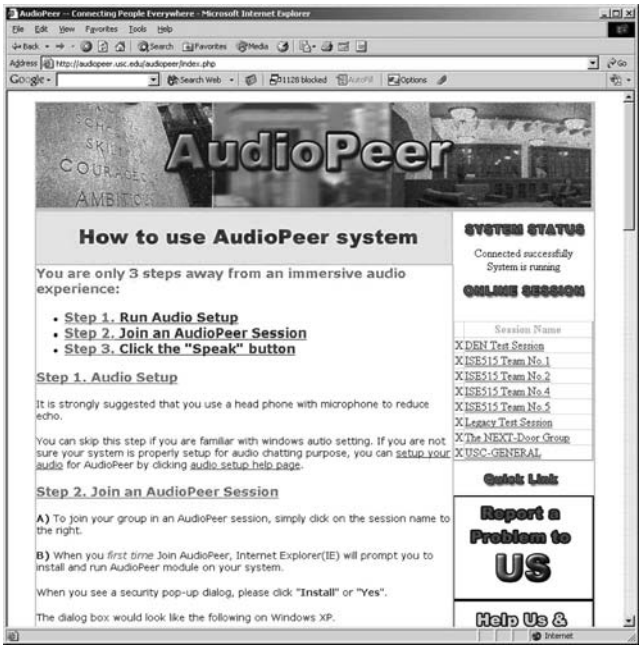


Fig. 7 AudioPeer official website

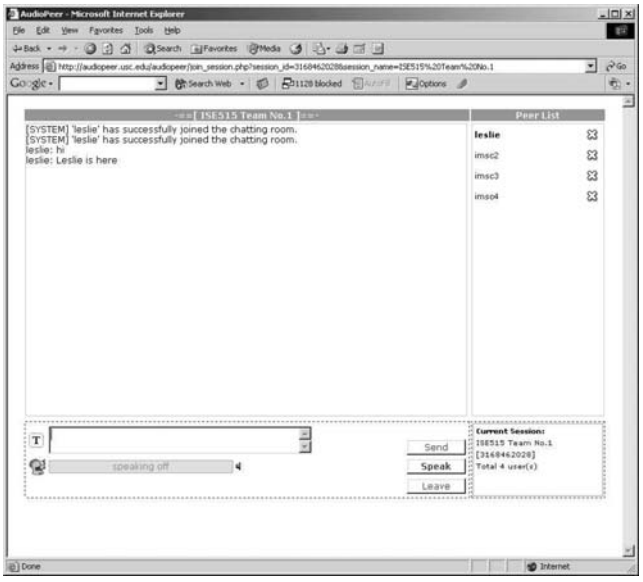


Fig. 8 AudioPeer client interface

users. On the right side of the homepage, we list the currently available chat sessions. Return users can simply click on these links to join an on-going session. This website serves as the bootstrap point that we can find in most P2P system designs. Additionally the website provides a link to a questionnaire where users can provide us with feedback. We will discuss the user feedback data later.

5.1.2 Client software

The core component of the AudioPeer system is the client program. Currently the implementation is based on Microsoft ActiveX technology, which enables us to integrate our client software seamlessly with almost any website and run it on any web browser that support ActiveX technology, such as Internet Explorer and the newly popular FireFox browser.

We wrap our ActiveX control with a HTML based interface (as shown on Fig. 8) and use Javascript to connect the web interface with our ActiveX control. The majority area of the interface is taken up by a text chatting window which allows users to communicate with text in addition to voice. On the right side is a panel which shows the list of users currently in this chat room. The icons to the right of their names indicate the current status of the users (speaking or mute). Users can start talking with a click of a button at the bottom of the window. We also implemented our own silence detection algorithm, which proved to be very effective. If a user remains silent for more than three minutes, he/she will be automatically muted by the system so that other users who are speaking can get a chance to move to a better position with lower latency with respect to other speakers.

Another important function provided by each client is audio stream merging. As mentioned before, AudioPeer is designed to support large size audio chat group without linear growth in the network bandwidth. This great feature is achieved through audio merging at every node on the peer-to-peer network. The details of audio processing in AudioPeer is discussed in Sect. 5.2.

5.1.3 Rendezvous Point Server

Most P2P systems provide peers with a bootstrap server to start the service. This server is usually called Rendezvous Point Server (RP server). In the AudioPeer system design, the functions of the RP server is minimized to just store session information and record user activities for administrative and research purposes.

The session information is stored in a MySQL database and made available through website links. As we have discussed before, users can simply click a link on the website to start the chatting experience. After clicking a link for one chat room, the information about the session is retrieved from the database and inserted into the AudioPeer ActiveX control, which will use these information to start the join process, mostly without further help from the RP server.

The RP database also persistently stores a variety of system-wide information including: debugging and history information, user and session management data, demographic user and course information, stored media descriptions for recorded sessions, and interactive user information such as user ratings of the system. These collected data are very useful not only for research, but also for administrative purposes. In the following section, we will discuss the use of the data from the RP database to visualize the ongoing system architecture.

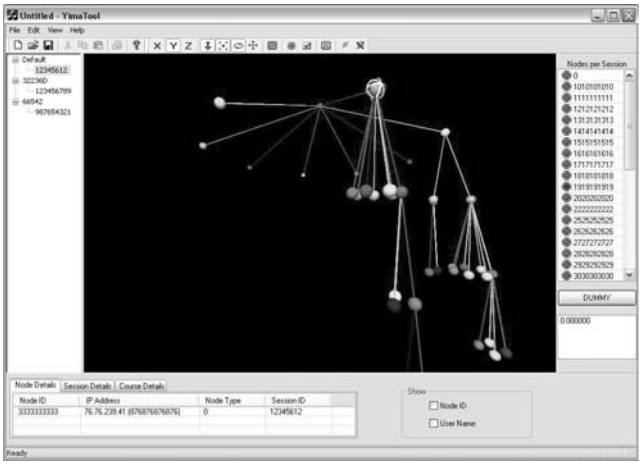


Fig. 9 Administration control interface (ACI)

5.1.4 Administration control interface

We have implemented an administration control interface (ACI) to efficiently visualize the peer-to-peer overlay network in real-time for administrative purposes (Fig. 9). This tool is implemented with Visual C++ with OpenGL libraries running on the Windows operating system.

Many useful system parameters are displayed on-the-fly along with the 3D representation of the current streaming tree topology. ACI is an information visualizer that also allows active management. Besides functionalities such as displaying system-wide information about individual nodes, sessions and courses, ACI can also issue direct system commands to individual nodes and sessions to govern the system at large. For example, ACI can force a node to leave, mute, relocate to another chat room if necessary, or merge two chat rooms into one.

ACI provides a 3D control interface for the AudioPeer system, in which there could be many concurrent chat sessions. We concluded that by using a 3D visualization, system administrators can gain a much better perception of the system structure compared with a 2D version. Also a 3D representation allows us to display more information in the same area, which becomes an important feature for a large scale system.

5.2 Audio mixing

The audio mixing algorithm focuses on minimizing the network utilization for our audio conferencing application. Unlike in video conferencing, it is possible to aggregate the uncompressed audio sources through simple arithmetic calculations, preserving the original audio bandwidth.

Each peer node is equipped with an audio mixing module that relays the incoming audio from remote nodes to the outgoing connections. We use a software-based audio mixing algorithm called *decode-mix-encode* [17]. A linear mixing algorithm requires all the input audio bitstreams to be uncompressed for simple arithmetic additions and subtractions. Thus, all incoming encoded bitstreams are decoded into their uncompressed form, and the resulting uncompressed bitstreams are merged into a mixed bitstream. This stream is later used when constructing the outgoing streams for each respective remote node.

Table 2 lists the currently supported audio media types and their characteristics: high-quality, low latency (PCM stereo); medium-quality, low latency (PCM mono); low-quality, low latency (GSM.610); and high-quality, high latency (MPEG-1 Layer 3).

5.3 Implementation of ACTIVE protocol

ACTIVE is an application-level multicast protocol designed to serve as a reliable audio streaming platform that provides minimal overall end-to-end delay among active nodes. Aiming at high scalability and low latency, the ACTIVE protocol dynamically maintains a shared multicast tree among all peer nodes. For space reasons we restrict our discussion of the ACTIVE protocol implementation to those issues that have not been discussed before in Sect. 3 and also in relatively high-level description.

We implemented the ACTIVE protocol using C/C++. At any given moment, a user can be in either of following states: Idle, Joining, Select, Setup, Joined, Leaving, Recover and Core (Fig. 10). We use TCP instead of UDP as streaming transportation protocol. The reason is that a lot of remote users now connect from behind a network address translation (NAT) device such as a home DSL modems to

Table 2 Audio types supported

	Supported audio media types			
Compression type	Uncompressed		Compressed	
Audio format	PCM	PCM	GSM.610	MPEG Layer3
Bits per sample	8	16	8	16
Channels	Mono	Stereo	Mono	Stereo
Sampling rate (kHz)	8/16	48	8	48
Delivery rate (kbps)	64/128	1.536	13	56
Usage method	LAN	LAN	Dial-up modem	Cable modem, DSL
Latency requirement	Low	Low	Low	High
Audio quality	Medium	High	Low	High

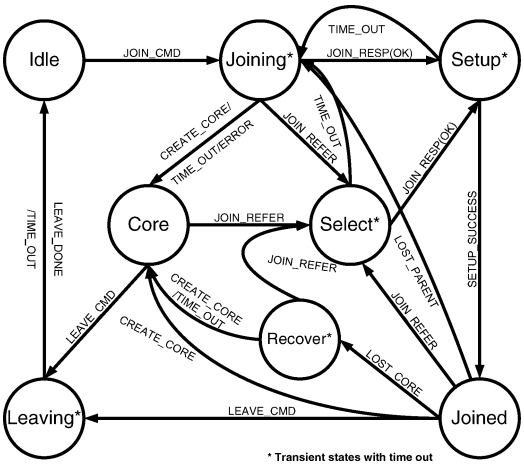


Fig. 10 AudioPeer state diagram

the Internet, which complicates the building of a peer-to-peer network using UDP without a centralized server. We also include network time protocl (NTP) functionality in our client software to synchronize our system in a distributed fashion.

6 User feedback and measured delay

In this section we present some feedback we obtained from the users who used our AudioPeer system for various purposes. We also present some preliminary data measuring the actual end-to-end delay in our system in Sect. 6.2.

6.1 User feedback

We deployed our AudioPeer system to diverse user groups for testing in 2004 and early 2005. We collected feedback from users via online questionnaires. In Tables 3 and 4 we present the results of one questionnaire we handed out to a class of about 160 students in the Fall of 2004. Table 3 shows the statistics of the user group and the detailed question and feedback are listed on Table 4. As we can see, the average rating of AudioPeer system is about 4–5 on a scale of 7. Many users found Audiopeer very easy to use, but we realized as a work in progress, we need to put more effort on the audio quality, which is part of our future work.

6.2 Measured end-to-end delay

We present some preliminary results from experiments measuring the end-to-end audio delay between two AudioPeer clients in a LAN/WAN environment and identify the primary causes of any delays. We performed our experiments on a Windows platform. Note that all results are subject to improvements due to changes in audio I/O drivers, operating system support, and specialized hardware.

The end-to-end audio delay is modeled as the summation of the capture interval, the play-out delay, and any network latency. With our experiments, we identified the two primary components of the end-to-end audio delay: the capture interval and minimum play-out delay. The *capture interval*, in milli-seconds (ms), is the time period between successive callback functions to collect the audio samples from the audio driver. If the capture interval is 20 ms, a callback function is invoked every 20 ms to transmit the captured audio samples from the driver to the application. The *minimum*

Table 3 Statistic of user groups

No.	Question	Options	Feedback (%)
1.	Gender	Male	73.33
		Female	26.76
2.	Position	Student	100
		Others	0
3.	Age range	18–22	40
		23–27	60
4.	How many hours per week do you use computers?	0–40	33.3
		41–80	53.3
		81+	13.4
5.	How many hours per week do you use email?	0–10	40
		11–20	46.7
		21+	13.4
6	How many hours per week do you use online chat systems (e.g., instant messenger)	0–5	46.7
		6–10	26.7
		11–20	13.3
		21+	13.3
7.	What is the purpose of your use of Audiopeer?	Chat with friends	26.7
		Educational use	60
		Others	13.3
8.	How many people did you chat with using audiopeer?	One other person	20
		Two or more	80
9.	Did you talk to two or more people at the same time?	Yes	46.7
		No	53.3

Table 4 User feedback

No.	Question	Feedback (scale 1–7)
1.	Once you had learned how to operate AudioPeer, did you find it easy to use?	5.06
2.	Did you find AudioPeer a useful way to communicate?	4.4
3.	Was AudioPeer helpful in your communications?	4.53
4.	Do you find AudioPeer an efficient way to communicate?	4.2
5.	Were you frustrated by any delays caused by communicating with AudioPeer?	5.2
6.	Did you like communicating using AudioPeer?	4.2
7.	Did you enjoy communicating using AudioPeer?	4.13
8.	Did you find AudioPeer fun to use?	4.13
9.	Did AudioPeer make you feel connected to other people?	4.4
10.	Did AudioPeer make you feel that people were at hand to answer your questions?	4.4

play-out delay, also represented in milli-seconds, is the time used to pre-load the audio samples for smooth audio play-out and to compensate for network jitters and slightly irregular capture intervals.

6.2.1 Experimental setup

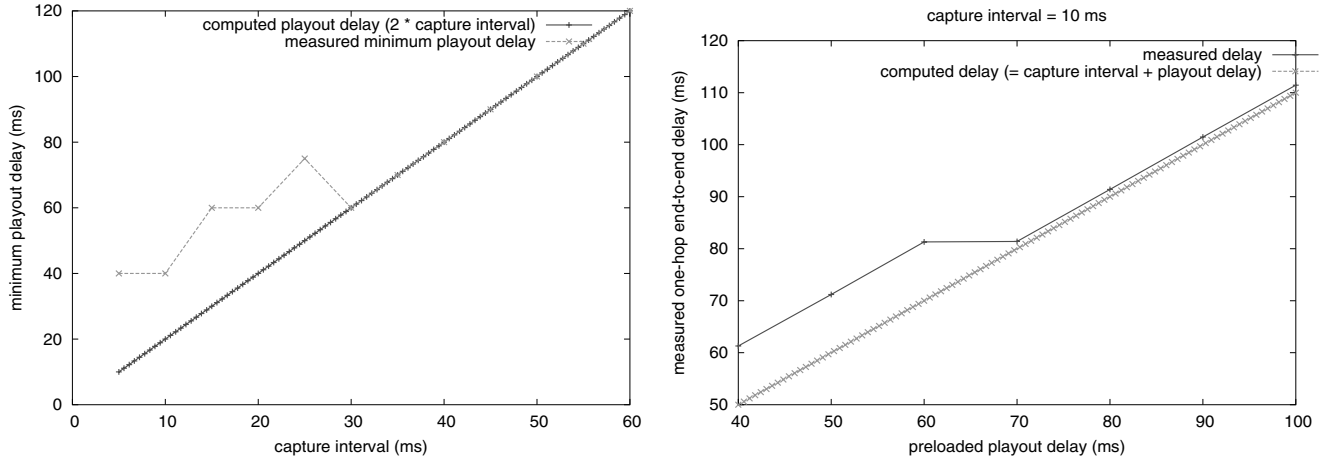
The Windows multiMedia extension (MME) API was used for waveform capture and playback. To precisely measure the end-to-end audio delay, we used an audio split cable. The two inputs of the cable were connected to the original audio source and the receiving AudioPeer. The output was recorded on another machine and the maximum delay offset between the two inputs was computed using cross-correlation in MATLAB. We repeated this experiment ten times with the same configuration to reduce the statistical variations.

Figure 11a shows the minimally required play-out delay as a function of the capture interval with no audio dropouts. We set the play-out delay as a multiple of the capture interval. We observe that the minimum play-out delay increases as the capture interval increases and also the CPU is more heavily loaded as the capture interval decreases. If the capture interval rises above 30 ms, the play-out delay linearly increases by a factor of two. However, the minimum play-out delay levels off at 40 ms.

Figure 11b shows the measured end-to-end latency when the capture interval is 10 ms with varying play-out delays. In a LAN environment the network transmission delay was less than 1 ms and the capture interval and play-out delay would be expected to dominate the end-to-end delay. Our expectation is confirmed in Fig. 11a where the play-out delay is greater than or equal to 70 ms. The small distance between the two curves is caused by the network delay and system overhead. For play-out delays in the range of 40–60 ms an additional 10 ms delay is introduced whose cause is still under investigation.

From these two figures we find that the optimal capture interval and the play-out delay are 10 and 40 ms, respectively. Thus, the minimum one-hop audio delay is 60 ms in a LAN environment. Accordingly, any multi-hop end-to-end audio delay can be roughly represented as *minimum one-hop audio delay (=60 ms) + end-to-end network delay*. Note that placing mixing modules at intermediate relay nodes may add additional processing delays.

Next, we also measured the end-to-end audio delay in a WAN environment between the USC campus in Los Angeles and Information Sciences Institute in Arlington, Virginia. We placed a loop-back module at the east coast



(a) minimum play-out delay

(b) one-hop end-to-end delay

Fig. 11 Measured delay in AudioPeer

site to receive audio packets and reflect them back to the sender. The average round trip time (RTT) was measured at 70 ms with a small variance. The end-to-end audio delay in the WAN environment exactly matches that of the LAN environment plus the one-way network transmission delay.

These encouraging results show the feasibility of a medium-sized application-level audio chat service in a commodity Windows environment. Similar results have also been confirmed by other research groups [18, 19].

7 Conclusions

We proposed a novel P2P streaming architecture called ACTIVE with the innovative feature of distinguishing active users from other users in a multicast group. Our analysis and experiments show that this approach achieves the scalability of P2P topologies while at the same time significantly reducing the delay among active users in an interactive streaming environment. This performance improvement is achieved without significantly increasing the delay among all other multicast members. We also tested our ACTIVE protocol in a multiuser audio conferencing application and therefore its feasibility has also been demonstrated. We plan to implement and deploy ACTIVE with other applications, for example video streaming, in the future.

Acknowledgements We wish to express our appreciation to the many wonderful people who helped us design, implement and improve our system. We acknowledge (in no particular order) the help of Beomjoo Seo, Kamel Oral Cansizlar, Yuli Huang, and Margaret McLaughlin. Many of their thoughts and comments have been incorporated into this work. The ACTIVE and AudioPeer research was made possible by NSF grants MRI-0321377 and Cooperative Agreement No. EEC-9529152, and by an unrestricted cash gift by the Lord Foundation.

References

1. Zimmermann, R., Liu, S.: Active: adaptive low-latency peer-to-peer streaming. Multimedia Computing and Networking (MMCN). San Jose, CA (2005)
2. Zimmermann, R., Seo, B., Liu, L.S., Hampole, R.S., Nash, B.: Audiopeer: a collaborative distributed audio chat system. Distributed Multimedia Systems, San Jose, CA (2004)
3. Chu, Y. hua, Rao, S.G., Seshan, S., Zhang, H.: Enabling conferencing applications on the internet using an overlay multicast architecture. In: ACM SIGCOMM 2001. ACM, San Diago, CA, (2001)
4. Francis, P.: Yoid: Your own Internet distribution. Available online at http://www.aciri.org/yoid/ (2000)
5. Zhang, B., Jamin, S., Zhang, L.: Host multicast: a framework for delivering multicast to end users. In: Proceedings of IEEE Infocom. New York (2002)
6. Banerjee, S., Bhattacharjee, B., Kommareddy, C.: Scalable Application Layer Multicast, Technical report, UMI-ACS TR-2002 (2002)
7. Banerjee, S., Kommareddy, C., Kar, K., Battacharjee, B., Khuller, S.: Construction of an efficient overlay multicast infrastructure for real-time applications. In: IEEE INFOCOM 2003 (2003)
8. Castro, M., Druschel, P., Kermarrec, A., Rowstron, A.: SCRIBE: a large-scale and decentralized application-level multicast infrastructure. IEEE J. Select. Areas Commun. (JSAC) (2002)
9. Ratnasamy, S., Handley, M., Karp, R., Shenker, S.: Application-level multicast using content-addressable net-works. In: the 3rd International Workshop on Networked Group Communication (2001)
10. Tran, D.A., Hua, K.A., Do, T.T.: A peer-to-peer architecture for media streaming. J. Select. Areas Commun.(JSAC) (Special Issue on Advances in Service Overlay Networks) (2003)
11. SkyPe, http://www.skype.com
12. Cui, Y., Li, B., Nahrstedt, K.: Ostream: asynchronous streaming multicast in application-layer overlay net-works. IEEE J. Select. Areas Commun. (JSAC) **22**(1), 191–196 (2004)
13. Blum, M., Chalasani, P., Coppersmith, D., Pulleyblank, B., Raghavan, P., Sudan, M.: The minimum latency prob-lem. In: ACM Symposium on Theory of Computing (1994)
14. NS, the Network Simulator.: Information about NS is availabale at http://www.isi.edu/nsnam/ns/
15. Cormen, T., Leiserson, C., Rivest, R.: Introduction to Algorithms. MIT Press, Cambridge, MA (1997)
16. USC Distance Education Network.: Information about DEN is availabale at http://den.usc.edu/
17. Singh, K., Nair, G., Schulzrinne, H.: Centralized conferencing using SIP. In: Internet Telephony Workshop (2001)
18. Koguchi, K., Jiang, W., Schulzrinne, H.: QoS measurement of VoIP end-points. In: IEICE Group meeting on Network Systems (2002)
19. MacMillan, K., Droettboom, M., Fujinaga, I.: Audio latency measurements of desktop operating systems. In: International Computer Music Conference (ICMCŠ01) (2001)