ACTIVE: Adaptive Low-latency Peer-to-Peer Streaming

Roger Zimmermann and Leslie S. Liu
Computer Science Department
University of Southern California, Los Angeles, California 90089
[rzimmerm,lleslie]@usc.edu

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. However, scant attention has been paid to interactive scenarios where the end-to-end delay is crucial. Furthermore, even algorithms that construct an optimal minimum spanning tree often make the unreasonable assumption that the processing time involved at each node is zero. However, these delays can add up to a significant amount of time after just a few overlay hops and make interactive applications difficult. In this paper, we introduce a novel peer-to-peer streaming architecture called ACTIVE that is based on the following observation. Even in large group discussions only a fraction of the users are active at a given time. We term these users, who have more critical demands for low-latency, active users. 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 multicast tree 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 that show the feasibility and performance of our approach.

1. INTRODUCTION

Peer-to-peer architectures (P2P) have become a popular platform for very scalable applications. In the last few years many P2P protocols have been designed and implemented (e.g.[1–9]). Very recently, research has focused on adapting P2P architectures for media streaming. Especially in live streaming the challenge is to provide a solution that minimizes the average end-to-end delay among all the users. To the best of our knowledge, no system has been designed to support interactive streaming for large groups using a P2P architecture.

Interactive streaming applications — for example an multiuser audio chat room — are very demanding in terms of end-to-end delay among users. Previous research has concluded that a round trip time (RTT) latency of more than 200ms will make a voice conversation difficult. In P2P systems, the delay is more problematic because nodes are often not directly connected to each other, but instead communicate through some intermediate overlay nodes between them.

One challenge that has not been addressed in previous research is the following. Most existing P2P solutions ignore the processing delay introduced at each intermediate node when they optimize the P2P 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 one application that uses ACTIVE[1] for distance online education and we found that the average processing delay at each node is approximately 30ms. 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 centralized approaches currently dominate the design space for interactive applications.

However, centralized designs suffer from certain limitations. The server constitutes a single point of failure for the system and network congestion is observed near the central server. With today's advanced computer technology, massive hardware may be capable of providing service to several large multicast groups simultaneously, but it will eventually succumb if the number of session grows to hundreds or thousands.

The core idea and innovation of ACTIVE is based on the following observation. 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

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

participants. This phenomenon provides us with an opportunity to optimize the P2P structure for better end-to-end delay among this relatively small group. Unfortunately, no existing P2P architecture has been designed to take advantage of this opportunity.

In this paper, we introduce a novel peer-to-peer streaming system called ACTIVE, which 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 the maximum number of active users grows with the size of the multicast group. The floor control constraint can be changed 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 approach.

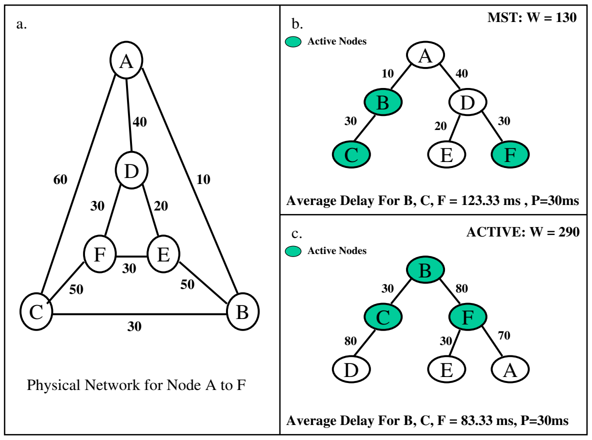
The rest of this paper is organized as follows: Section 2 describes the design of ACTIVE in detail. Section 3 briefly introduces an application using ACTIVE for distance education. Section 4 describes how we simulate and evaluate the performance of ACTIVE. Finally, Section 5 lists some related work and Section 6 draws the conclusions.

2. 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,[1] 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 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.[10] 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. 1 a MST tree linking all multicast nodes cannot guarantee the minimum delay among active users, where the low-latency is critical. Fig. 1(b) 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. 1(c), the delay in the ACTIVE generated tree is only 83.33 ms.


a.
Physical Network for Node A to F
b.
MST: W = 130
Active Nodes
Average Delay For B, C, F = 123.33 ms , P=30ms
c.
ACTIVE: W = 290
Active Nodes
Average Delay For B, C, F = 83.33 ms, P=30ms
Figure 1. Example: MST vs. ACTIVE

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 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. Fig. 2 shows the processing delay at two of the intermediate nodes of an audio conference application using ACTIVE.

Term	Definition	Units
V	Set of all users on current multicast group, $\lvert V\rvert$ is the total number of all users	
A	Set of all active users on current multicast group, $\lvert A\rvert$ is the total number of active users	
G	Application level complete graph containing all nodes in V	
E	Set of edges that connecting all nodes in V, $E = V \times V$	
$\lvert E_{i,j}\rvert$	The physical network delay between host i and host j	
$N_{A\prime}$	Maximum number of active users allowed	
P	The application-layer processing delay. Also called relay delay.	ms
T	Current multicast tree that connecting 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	sec
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 deliver path from host i to host j	
$\lvert D_{i,j}\rvert$	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 allow a user to switch to active mode	
$RDP\text{-}AU$	Ratio of average overlay delay for active nodes compared to MST tree result	
$RDP\text{-}ALL$	Ratio of average overlay delay for all nodes compared to MST tree result	

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

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.

2.1. Problem Formulation

2.1.1. 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 Equation 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 streaming mixing node that use the GSM.610 audio codec in Equation 2.

$$P_k = T_{recv\_buffer} + T_{decode\_buffer} + T_{mix\_processing} + T_{encode\_buffer} + T_{send\_buffer} \quad (2)$$

We can see that in a peer-to-peer network, P_k is usually not the same because of the different 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 30ms processing delay on average in one of our test nodes Fig. 2.

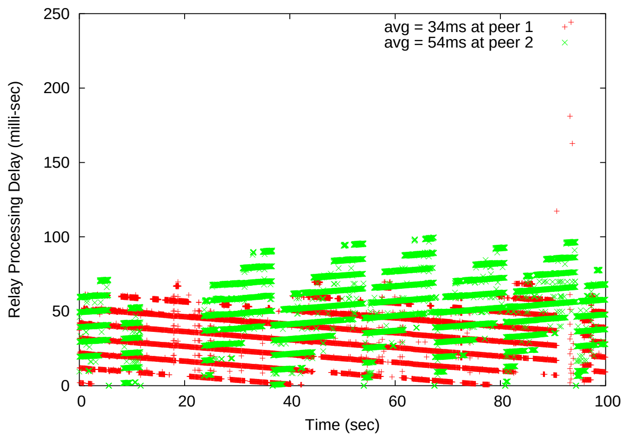


Figure 2. Application-layer Processing Delay

2.1.2. 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 $T\prime$, where $T\prime \subseteq G$, 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 Section 2.3. As a basis of the proposed solution, we designed a distributed score computing algorithm called *Credit Point System*, which is described next.

2.2. 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:

$$\begin{aligned} CP_{root} &= 1 \\ CP_{Child} &= CP_{Parent}/K \ , \quad K \geq 2 \end{aligned} \quad (3)$$

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 Equation 4.

$$CP_i \geq CPt \ , \ i \in V \quad (4)$$

If Equation 4 is satisfied, node i can switch from passive to active mode. Since nodes in active mode usually generate more data for delivery, Equation 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 $N_{A\prime}$ as the maximum number of active nodes allowed, then the following equation illustrates how $N_{A\prime}$ is constrained by CPt and K:

$$|A| \leq N_{A\prime} = \frac{K - CPt}{CPt \times (K-1)} \ , \ K \geq 2, \ 0 < CPt < 1 \quad (5)$$

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:

$$CPt = \frac{F}{log_k|V|} \ , \ 0 < F < 1 \tag{6}$$

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

The complexity of the Credit Point 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 Section 4.2).

2.3. 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.

Heuristic solution to solve degree-bound minimum spanning tree problem: For $\forall i, j \in A$, $HOP\prime \leftarrow max(|HOP_{i,j}|)$. Find a subtree $T\prime$ ($T\prime \subseteq T$, $A \subset T\prime$), so that the following condition is satisfied:

$$\begin{cases} HOP\prime \leq log_k \dfrac{1}{CPt^2} - 1 \\ \\ \forall i \in A, \exists j : |HOP_{i,j}| = 0 \ , \ j \in A, j \neq i \end{cases} \tag{7}$$

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 Section 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 don't need to measure the delay among active users in order to achieve our performance goal, thus reduces the control overhead in our system.

2.4. 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 Section 2.5.

2.4.1. Join

During the join process, ACTIVE uses a heuristic 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.

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 the order it receives the response messages and the first respond node C in L will be considered to be 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 the degree limit or suddenly goes offline), the new node will remove this candidate from L, chose the next node $C\prime$ remaining in L and starts the above process again.

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
```

2.4.2. 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 Section 2.4.3. The only exception occurs when the current node has no child nodes. In this case, the node can simply disconnect from the service.

2.4.3. 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 Section 2.4.1, 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 2 $LEAVE$

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

Algorithm 3 $REJOIN$

```
1:  $L \leftarrow$ the candidate node list built in JOIN process
2:  while $L \neq \emptyset$ do
3:      $C \leftarrow$ first node in $L$ that has bigger or equal $CP$
4:      if $C$ is ok to join then
5:          setup connection with $C$
6:          $parent \leftarrow C$
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
```

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.

2.5. 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 Equation 4. If the request can not be granted, the optimization process is triggered.

2.5.1. Clustering Active Users

The clustering is achieved by moving the active nodes gradually towards the root of the tree until Equation 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 glitches during the optimization.

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 seconds 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.

2.6. 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 $N_{A\prime}$ gradually increases along with the size of the group $|V|$. In Equation 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 Equation 5 as follows:

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

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 $F\prime$ calculated in Equation 9. It is not hard to prove that in this case Equation 4 is always satisfied.

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

Algorithm 4 $OPTIMIZE$

```
1:  AGAIN:
2:  $P \leftarrow$ parent of local host i
3:  $I_P \leftarrow$ idle time at $P$
4:  **if** $I_P \leq R^{-1}$ **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:         $P\prime \leftarrow$ the parent of $P$
14:         ask $P$ to set local host i as its parent
15:         ask $P\prime$ to set local host as new child
16:         $P \leftarrow P\prime$
17:         setup connection with $P\prime$
18:     **end if**
19: **end while**
20:
21: //*Second Phase*
22: **if** $CP_i \geq CPt$ **then**
23:     send update message to all children nodes
24:     return
25: **else**
26:     $L\prime \leftarrow$ all passive node immediate connected to A
27:     remove all host j from $L\prime$ if $CP_j \leq CP_i$ or $CP_j = 1$
28:     **if** $L\prime = \emptyset$ **then**
29:         send update message to all children nodes
30:         return
31:     **else**
32:         $C \leftarrow$ first node in $L\prime$
33:         $P\prime \leftarrow$ the parent of $C$
34:         ask $C$ to set $P$ as its parent
35:         ask $P\prime$ to set local host as new child
36:         $P \leftarrow P\prime$
37:         setup connection with $P\prime$
38:     **end if**
39: **end if**
40: goto AGAIN:
```

2.7. 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 *HeartBeat* 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 $N\prime$ as the average number of nodes receiving the refreshment message, the control overhead O_r for all refreshment based protocols is:

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

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.

3. EXAMPLE APPLICATION USING ACTIVE

ACTIVE is used as the streaming platform for an audio conference application for distance education.[11] The program is written in C++ and runs on Windows 2000 and XP. The audio module is implemented with the Windows MME library and the GUI is coded with MFC. The ACTIVE protocol code itself is mostly written in pure C++ and is easy to port to another operating system. This design allowed us to quickly port ACTIVE to a NS-2/Linux environment for performance evaluation and system simulation.

4. PERFORMANCE EVALUATION

We evaluated our ACTIVE design with simulations in an NS-2 environment.[12] 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[13] with the same physical network topologies. The generated MST trees are the optimal solution for all existing tree-based designs (e.g., [1–4]) 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.

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:

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

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

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. Fig. 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. 2), we used the average value $P = 30$ ms as the processing delay at each node to simplify the simulation, but using this specific value does not affect the final conclusion of our simulation result.

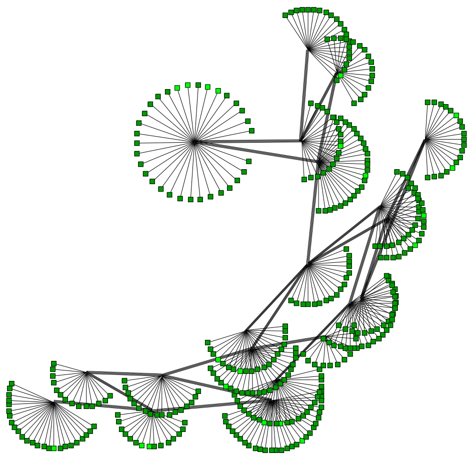
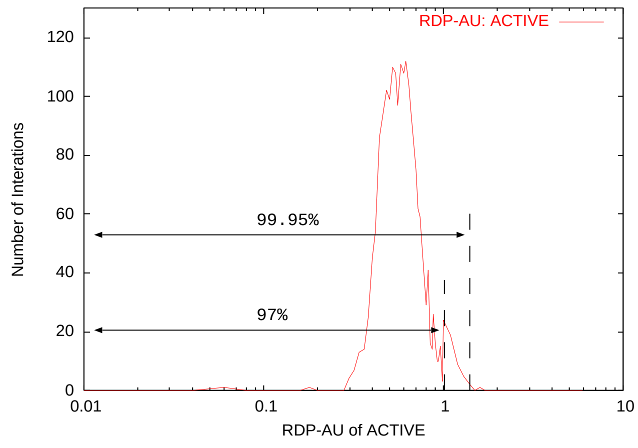


Figure 3. Network Topology (400 Users)

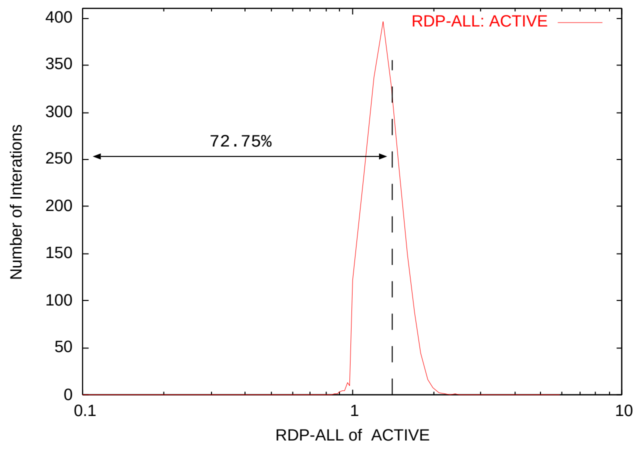
4.3. Simulation Results

ACTIVE achieved dramatic performance improvements over other existing systems in our simulation. We ran 2000 iterations of simulation for uniformly distributed user groups ranging in size from 4 to 400 and active user groups ranging in size from 4 to 20. Fig. 4(a) 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. 4(b), 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 investigated the cause, we found that these counter-intuitive results are not incorrect. For all leaf nodes, which 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.



(a) for active users



(b) for all users

Figure 4. Delay performance of ACTIVE

4.3.1. Experimental Scenario

To show how tree optimization in ACTIVE reduces 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 (Figs 5, 6 and 7). 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. 5 and Fig. 7, 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 $O(|E| + |V|log|V|)$.

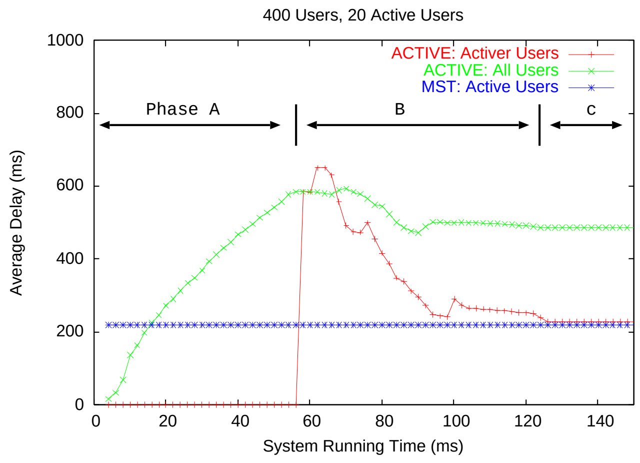


Figure 5. Example: Large Group

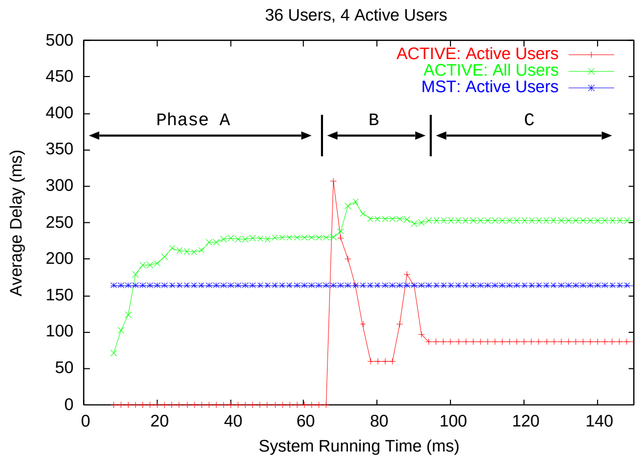


Figure 6. Example: Medium Group

5. RELATED WORK

Many P2P architectures (e.g. Narada,[1] Yoid,[2] HMTP,[3] NICE,[4] OMNI,[5] Scribe,[6] CAN-Multicast,[7] Zigzag[8] , Skype[9] and oStream[14]) have been proposed or adapted for streaming media services. However, due to the long end-to-end delay in overlay networks, none of these approaches are capable of handling (or are designed for) the low-latency demands of interactive environments.

Narada [1] 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 [4] 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.[5] , which additionally tries to optimize the overall end-to-end delay among all streaming receivers. Zigzag[8] optimized 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 them unsuitable to achieve the delay constraints required for large scale interactive applications.

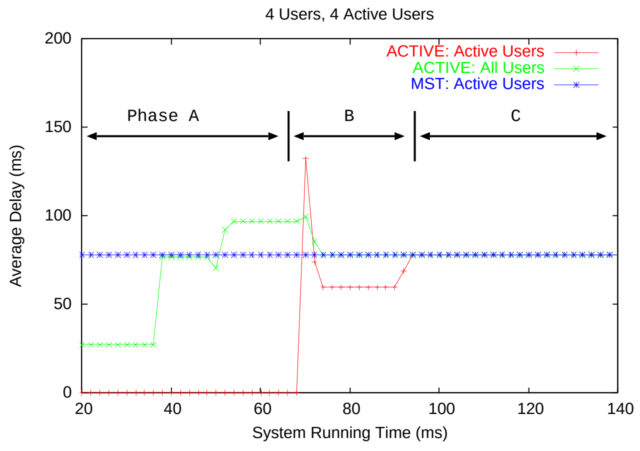


Figure 7. Example: Small Group

Scribe[6] and CAN-Multicast[7] 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[9] , which is also based on peer-to-peer technology, is emerging as a popular online audio conferencing application. It is not designed for large groups, but rather for many concurrent groups with small number of users in each group.

oStream[14] 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 interactive live streaming environment such as an audio chat room.

6. 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 and providing better QoS in a interactive streaming environment compared even with the optimal solution generated by the minimum spanning tree algorithm. This performance improvement is achieved without significantly increasing the delay among all other multicast members. We have implemented ACTIVE as a substrate for an 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.

REFERENCES

1. Y. hua Chu, S. G. Rao, S. Seshan, and H. Zhang, "Enabling Conferencing Applications on the Internet Using an Overlay Multicast Architecture," in *ACM SIGCOMM 2001*, ACM, (San Diago, CA), Aug. 2001.
2. P. Francis, "Yoid: Your Own Internet Distribution," *Available online at http://www.aciri.org/yoid/* , April 2000.
3. B. Zhang, S. Jamin, and L. Zhang, "Host Multicast: A Framework for Delivering Multicast to End Users," in *Proceedings of IEEE Infocom*, (New York), June 23-27, 2002.
4. S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "Scalable Application Layer Multicast," *Technical report, UMIACS TR-2002* , 2002.
5. S. Banerjee, C. Kommareddy, K. Kar, B. Battacharjee, and S. Khuller, "Construction of an Efficient Overlay Multicast Infrastructure for Real-time Applications," in *IEEE INFOCOM 2003*, 2003.
6. M. Castro, P. Druschel, A. Kermarrec, and A. Rowstron, "SCRIBE: A Large-scale and Decentralized Application-level Multicast Infrastructure," *IEEE Journal on Selected Areas in communications (JSAC)* , 2002.
7. S. Ratnasamy, M. Handley, R. Karp, and S. Shenker, "Application-level Multicast Using Content-Addressable Networks," in *Third International Workshop Networked Group Comm.*, 2001.
8. D. A. Tran, K. A. Hua, and T. T. Do, "A Peer-to-Peer Architecture for Media Streaming," *Journal on Selected Areas in Communications(JSAC), Special Issue on Advances in Service Overlay Networks* , 2003.
9. SkyPe, *http://www.skype.com*.
10. M. Blum, P. Chalasani, D. Coppersmith, B. Pulleyblank, P. Raghavan, and M. Sudan, "The minimum latency problem," in *ACM Symposium on Theory of Computing*, 1994.
11. R. Zimmermann, B. Seo, L. S. Liu, R. S.Hampole, and B. Nash, "Audiopeer: A Collaborative Distributed Audio Chat System," *Distributed Multimedia Systems, San Jose, CA* , 2004.
12. "NS, the Network Simulator." Information about NS is availabale at http://www.isi.edu/nsnam/ns/.
13. T. Cormen, C. Leiserson, and R. Rivest, "Introduction to algorithms," *MIT Press* , 1997.
14. Y. Cui, B. Li, and K. Nahrstedt, "ostream: Asynchronous Streaming Multicast in Application-layer Overlay Networks," *IEEE Journal on Selected Rreas in Communications (JSAC)* **22**(1), pp. 191–196, 2004.