# Scalable Media Streaming in Large Peer-to-Peer Networks*

Duc A. Tran Kien A. Hua Tai T. Do
School of Electrical Engineering and Computer Science
University of Central Florida
Orlando, FL 32816
dtran,kienhua,tdo@cs.ucf.edu

## ABSTRACT

We design a peer-to-peer technique for single-source media streaming. This technique allows the media server to distribute content to many clients by organizing them into an appropriate tree rooted at the server. This application-layer multicast tree has a height $O(logN)$ where $N$ is the number of clients, and a node degree bounded by a constant. This helps reduce the number of processing hops on the delivery path to a client while avoiding network bottleneck. Consequently, the end-to-end delay is kept small. Although one could build a tree satisfying such properties easily, an efficient control protocol between the nodes must be in place to maintain the tree under the effects of network dynamics and unpredictable client behaviors. Our technique handles such situations gracefully requiring a constant amortized control overhead. Especially, failure recovery can be done regionally with little impact on the existing clients.

## Keywords

Application-Layer Multicast, Media Streaming, Peer to Peer

## 1. INTRODUCTION

We are interested in the problem of streaming live bandwidth-intensive media from a single source to a large quantity of receivers. This problem is challenging due to the lack of IP Multicast on the current Internet. The simplest solution dedicates an individual connection to stream the content to each receiver, however this would not be scalable. Therefore, we seek a solution that employs IP unicast only but offers significantly better efficiency than the dedicated-connection approach.

In the absence of extra resources, we opt to use the peer-to-peer (P2P) approach to tackle the problem. In a media streaming P2P architecture, the delivery tree is built rooted at the source and including all and only the receivers. A subset of receivers get the content directly from the source and the others get it from the receivers in the upstream. P2P consumes the source's bandwidth efficiently by capitalizing a receiver's bandwidth to provide services to other receivers. On the other hand, the following issues are important in designing an efficient P2P technique:

First, the end-to-end delay from the source to a receiver may be excessive because the content may have to go through a number of intermediate receivers. To shorten this delay (whereby, increasing the liveness of the media content), the tree height should be kept small and the join procedure should finish fast. The end-to-end delay may also be long due to an occurrence of bottleneck at a tree node. The worst bottleneck happens if the tree is a star rooted at the source. The bottleneck is most reduced if the tree is a chain, however in this case the leaf node experiences a long delay. Therefore, apart from enforcing the tree to be short, it is desirable to have the node degree bounded.

Second, the behavior of receivers is unpredictable; they are free to join and leave the service at any time, thus abandoning their descendant peers. To prevent service interruption, a robust technique has to provide a quick and graceful recovery should a failure occur.

Third, for efficient use of network resources and due to the resource limitation at each receiver, the control overhead at each receiver should be small. This is important to the scalability of a system with a large number of receivers.

We propose a technique that addresses all the issues above. Our multicast tree has a height $O(log_k N)$ where $N$ is the number of receivers and $k$ a constant, and a node degree $O(k^2)$. Furthermore, the effects of network dynamics and unpredictable receiver behaviors are handled gracefully requiring worst-case control overhead of $O(log_k N)$ for the worst receiver and $O(k)$ for an average receiver. Especially, failure recovery can be done regionally with only impact on a constant number of existing receivers and no burden on the source. In comparison, no previous solution [1, 2, 3, 4] to our problem can provide all the above features.

We present the protocol details of our scheme in the next section, and conclude this paper in Section 3 with a brief comparison to previous work.

*This research is partially supported by US National Science Foundation under grant ANI-0088026

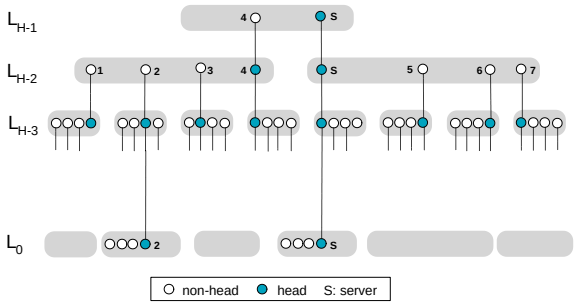


**Figure 1: Administrative organization of peers**

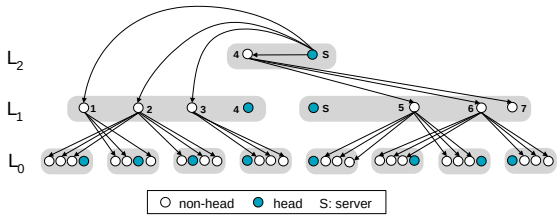


**Figure 2: The multicast tree of peers ($H = 3$, $k = 4$)**

## 2. PROPOSED SOLUTION

For the ease of exposition, we refer to the media source as the server and receivers as clients. They all are referred to as "peers". Firstly, we describe an administrative organization of the peers when the system is in the stable state. Secondly, we propose how the multicast tree is built based on this organization, and then the control protocol in which peers exchange state information. Finally, we propose policies to adjust the tree as well as the administrative organization upon a client join and departure.

### 2.1 Administrative Organization

An administrative organization is used to manage the peers currently in the system and illustrated in Fig. 1. Peers are organized in a multi-layer hierarchy of clusters recursively defined as follows (where $H$ is the number of layers, $k > 3$ is a constant):

(1) Layer 0 contains all peers.

(2) Peers in layer $j < H - 1$ are partitioned into clusters of sizes in $[k, 3k]$. Layer $H - 1$ has only one cluster which has a size in $[2, 3k]$.

(3) A peer in a cluster at layer $j < H$ is selected to be the head of that cluster. This head becomes a member of layer $j + 1$ if $j < H - 1$. The server $S$ is the head of any cluster it belongs to.

Initially, when the number of peers is small, the administrative organization has only one layer containing one cluster. As clients join or leave, this organization will be augmented or shrunk. The cluster size is upper bounded by $3k$ because we might have to split a cluster later when it becomes oversize. If the cluster size was upper bounded by $2k$ and the current size was $2k + 1$, after the split, the two new clusters would have sizes $k$ and $k + 1$ and be prone to be undersize as peers leave.

The above structure implies $H = \Theta(log_k N)$ where $N$ is the number of peers. Additionally, any peer at a layer $j > 0$ must be the head of the cluster it belongs to at every lower layer. We note that this hierarchy definition is not new. It was indeed presented in a similar form in [1]. How to map peers into the administrative organization, to build the multicast tree based on it, and to update these two structures under network dynamics are our main contribution.

We use the following terms for the rest of the paper: (1) Member: Non-head peers of a cluster headed by a peer $X$ are called "members" of $X$; (2) Sibling head: A non-head clustermate of a peer $X$ at layer $j > 0$ is called a "sibling head" of layer-($j$ - 1) members of $X$; (3) Sibling member: Layer-($j$-1) members of $X$ are called "sibling members" of any layer-$j$ clustermate of $X$; and (4) Sibling cluster: The layer-($j$-1) cluster of $X$ is called a "sibling cluster" any layer-$j$ clustermate of $X$.

### 2.2 Multicast Tree

Unlike in [1], the administrative organization in our approach does not infer a data delivery topology. For instance, we will see shortly that the head of a cluster at a layer $j < H - 1$ does not forward the content to any of its members as we might think of. In this section, we propose the rules to which the multicast tree must be confined and explain the motivation behind that. The join, departure, and optimization policies must follow these rules. The rules are listed below (demonstrated by Fig. 2):

(1) A peer, when not at its highest layer, cannot have any link to/from any other peer. E.g., peer 4 at layer 1 has neither outgoing nor incoming links.

(2) A peer, when at its highest layer, can only link to its sibling members. E.g., peer 4 at layer 2 only links to peers 5, 6, and 7 at layer 1, which are sibling members of 4. The only exception is the server; at the highest layer, the server links to each of its members.

(3) At layer $j < H - 1$: since non-head members of a cluster cannot get the content from their head, they must get it somehow. In our multicast tree, they get the content directly from one and only one sibling head. E.g., non-head peers in layer-0 cluster of peer 1 have a link from their sibling head 2; peers 1, 2 and 3 have a link from their sibling head $S$.

It is trivial to prove the above rules guarantee a tree structure including all the peers. Hereafter, the terms "parent", "children", "descendant" are used with the same meanings as applied for conventional trees. The term "node" is used interchangeably with "peer" and "client".

Theorem 1: The worst-case node degree of the multicast tree is $O(k^2)$.

Theorem 2: The height of the multicast tree is $O(log_k N)$ where $N$ is the number of peers.

It now comes understandable why we do not use the head of a cluster to link to its members because doing so would increase the worst-case node degree to O($klog_kN$); especially, the bottleneck would occur very early in the delivery path. Our using a sibling head as the parent has another nice property. Indeed, when the parent peer fails, the head of its children is still working, thus helping reconnect the children to a new parent quickly and easily. We will discuss this in more detail shortly.

## 2.3 Control protocol

To maintain its position in the multicast tree and the administrative organization, each node $X$ in a layer-$j$ cluster periodically communicates with its layer-$j$ clustermates, its children and parent on the multicast tree. The theorem below tells that the control overhead for an average member is a constant. The worst node has to communicate with O($k \times log_kN$) other nodes, this is however acceptable since the information exchanged is just soft state refreshes.

Theorem 3: Although the worst-case control overhead of a node is O($k \times log_kN$), the amortized worst-case overhead is O($k$).

## 2.4 Client Join

The multicast tree is augmented whenever a new client joins. The new tree must not violate the rules specified in Section 2.2. We propose the join algorithm below.

A new client $P$ submits a request to the server. If the administrative organization currently has one layer, $P$ simply connects to the server. Otherwise, the join request is redirected along the multicast tree downward until finding a proper peer to join. The below steps are pursued by a peer $X$ on receipt of a join request (in this algorithm, $D(Y)$ denotes the currently end-to-end delay from the server observed by a peer $Y$, and $d(Y, P)$ is the delay from $Y$ to $P$ measured during the contact between $Y$ and $P$):

1. If $X$ is a leaf
1.1. Add $P$ to the only cluster of $X$
1.2. Make $P$ a new child of the parent of $X$
2. Else
2.1. If *Addable*($X$)
2.1.1. Select a child $Y$:
*Addable*($Y$) and $D(Y)+d(Y, P)$ is min
2.1.2. Forward the join request to $Y$
2.2. Else
2.2.1 Select a child $Y$:
*Reachable*($Y$) and $D(Y)+d(Y, P)$ is min
2.2.2. Forward the join request to $Y$

The goal of this procedure is to add $P$ to a layer-0 cluster $C$ and force $P$ to get the content from the parent of non-head members of $C$. The size of $C$ should be in [$k$, 3$k$) to avoid being oversize. The end-to-end delay is attempted to be better after each node contact.

Theorem 4: The join overhead is O($log_kN$) in terms of number of nodes to contact.

The join procedure terminates at step 1.2 at some leaf $X$,

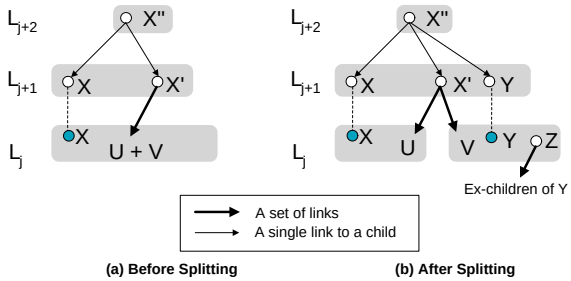


**Figure 3: Split Algorithm**

which will tell $P$ about other members of the cluster. $P$ then follows the control protocol as discussed earlier. If the new size of the joined cluster is still in [$k$, 3$k$], no further work is needed. Otherwise, this cluster has to be split so that the newly created clusters must have sizes in [$k$, 3$k$]. To avoid the overhead of splitting, we propose to do so periodically, not right after a cluster size becomes 3$k$+1. Suppose we decide to split a layer-$j$ ($j \in$ [1, H-2]) cluster[1] with a head $X$ and non-head peers $X_1$, .., $X_n$. The non-head currently get the content from a peer $X'$ and $X$ currently gets the content from $X''$. Let $x_{il}$ be the number of peers that are both children of $X_i$ and layer-($j$-1) members of $X_l$. Clearly, $x_{ii} = 0$ for all $i$ because of the rules described in Section 2.2. The split takes several steps (illustrated in Fig. 3):

(1) Partition $\{X, X_1, .., X_n\}$ into two sets $U$ and $V$ such that the condition $|U|, |V| \in [k, 3k]$ is satisfied first, and then $\Sigma_{X_i \in U, X_l \in V}(x_{il}+x_{li})$ is minimized. This condition is to effortfully reduce the number of peer reconnections affected by the split. Suppose $X \in U$.

(2) For each node $X_i \in U$ and each node $X_l \in V$ such that $x_{il} > 0$, remove all the links from $X_i$ to layer-($j$-1) members of $X_l$, and select a random peer in $V$ other than $X_l$ to be the new parent for these members. Inversely, for each node $X_i \in V$ and for each node $X_l \in U$ such that $x_{il} > 0$, a similar procedure takes place except that the new parent must not be peer $X$.

(3) Now we need to elect a new head $Y$ for cluster $V$. $Y$ is chosen to be a peer in $V$ with the minimum degree because we want this change to affect a smallest number of child peers. $Y$ becomes a new member of the cluster at layer-($j$+1) which also contains $X$. Consequently, the children of $Y$ (after Step 2) now cannot get data from $Y$ anymore (due to the rules in Section 2.2). For each child cluster (i.e., cluster whose non-head members used to be children of $Y$), we select a peer $Z \neq Y$ in $V$ having the minimum degree to be the new parent; $Z$ must not be the head of this cluster. Furthermore, the highest layer of $Y$ is not layer $j$ anymore, but layer $j$+1. Therefore, we remove the current link from $X'$ to $Y$ and add a link from $X''$ to $Y$. $Y$ will happen to have no children at this moment. This still does not violate the rules enforcing our multicast tree.

It might happen that the cluster on layer $j$+1 becomes over-

[1]The cases where $j = 0$ or $j = H$-1 are even easier and can be handled similarly.

size due to admitting $Y$. This would have to wait until the next period when the split algorithm will be called. The split algorithm is run locally by the head of the cluster to be split. The results will be sent to all peers that need to change their connections. Since the number of peers involved in the algorithm is a constant, the computational time to get out the results is not a major issue. The main overhead is the number of peers that need to reconnect. However, the theorem below tells that the overhead is indeed very small.

Theorem 5: The worst-case split overhead is $O(k^2)$.

## 2.5 Client Departure

The new tree after a client departs must not violate the rules specified in Section 2.2. We propose the algorithm to handle a client departure below.

Consider a peer $X$ who departs either purposely or accidentally due to failure. As a result of the control protocol described in Section 2.3, the parent peer of $X$, all members of $X$ (if any), and all children of $X$ (if any) are aware of this departure. The parent of $X$ needs to delete the link to $X$. If $X$'s highest layer is layer 0, no further overhead emerges.

Suppose that $X$'s highest layer is $j > 0$. For each layer-($j$-1) cluster whose non-head members are children of $X$, the head $Y$ of the cluster is responsible for finding a new parent for them. $Y$ just selects $Z$, a layer-$j$ non-head clustermate, that has the minimum degree, and asks it to forward data to $Y$'s members at layer $j$-1.

Furthermore, since $X$ used to be the head of $j$ clusters at layers 0, 1, .., $j$-1, they must have a new head. This is handled easily. Let $X'$ be a random member of $X$ at layer 0. $X'$ will replace $X$ as the new head for each of those clusters. $X'$ also appears at layer $j$ and gets a link from the existing parent of $X$. No other change is required. The overhead of failure recovery is consequently stated as follows:

Theorem 6: In the worst case, the number of peers that need to reconnect due to a failure is $O(k^2)$.

As the result of many client departures, a cluster might become undersize. In this case, it is merged with another cluster of the same layer. Suppose that $U$ is an undersize cluster at layer $j$ to be merged with another cluster $V$. The simplest way to find $V$ is to find a cluster having the smallest size. Then, the following steps are taken to do the mergence: (1) The new head of $U$+$V$ is chosen between the head $X$ of $U$ and the head $Y$ of $V$. If $Y$ (or $X$) is the head of $X$ (or $Y$) at the next layer, $Y$ (or $X$) will be the new head. In the other cases, the new head is the one having a larger degree to reduce the number of children to reconnect (since the children of the non-chosen must reconnect). Supposing $X$ is the new head, $Y$ will no longer appear at layer $j$+1.

(2) The new parent of non-head members in $U$+$V$ is chosen to be a layer-($j$+1) non-head clustermate of $X$ and $Y$. This new parent should currently have the minimum degree.

(3) If the existing children of $Y$ happen to be $U$, or that of $X$ happen to be $V$, no more work is needed since Step (2) already handles this case. Otherwise, two possibilities can happen:

- $X$ is the head at layer $j$+1: For each child cluster of $Y$, a sibling head $Z \neq Y$ that has the minimum degree will be the new parent; $Z$ must not be the head of this cluster.
- $X$ is not the head at layer $j$+1: The new parent for the existing children of $Y$ will be $X$.

Similar to the split procedure, the merge procedure is called periodically to reduce overhead. It runs centrally at the head of $U$ with assistance from the head of $V$. Since the number of peers involves is a constant, the computational complexity should be small. In terms of number of reconnections, the worst-case overhead is resulted from the theorem below.

Theorem 7: The worst-case merge overhead is $O(k^2)$.

# 3. CONCLUSIONS

We presented in this short paper a technique for streaming media in a large P2P network. Our goal was to reduce the worst-case values for important performance metrics such as end-to-end delay, bandwidth bottleneck, failure recovery overhead, control overhead, and re-configuration overhead. No previous solution [1, 2, 3, 4] can achieve all the results as provided by our technique. Indeed, [2] has to get the server involved whenever a failure occurs. [4] also puts a heavy burden on the server since it assumes that the server has full knowledge of all distribution trees. [3] incurs severe peer bottleneck because the tree height is forced to be at most 2. The most recent work [1] takes advantage of the multi-layer hierarchical clustering idea as we do, but incurs a high bottleneck of $O(log_k N)$. Though an extension could be done to reduce this bottleneck to a constant, the tree height could become $O(log_k N \times log_k N)$. Our technique, no worst than theirs in terms of the other metrics, has a worst-case delay of $O(logN)$ while keeping the bottleneck bounded by a constant. Furthermore, the failure recovery overhead in our technique is bounded by a constant while [1] requires $O(log_k N)$. All these are a significant improvement for bandwidth-intensive applications such as media streaming.

# 4. REFERENCES


[1] S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable application layer multicast. In *ACM SIGCOMM*, Pittsburgh, PA, 2002.

[2] H. Deshpande, M. Bawa, and H. Garcia-Molina. Streaming live media over a peer-to-peer network. In *Work at CS-Stanford. Submitted for publication*, 2002.

[3] C. Guo, G. Shen, S. Li, and Y. Zhong. Pasa: Peer-assisted architecture and protocol for scalable multimedia streaming. Joined Work at Microsoft Research and Tsinghua. Unpublished, 2002.

[4] V. N. Padmanabhan, H. J. Wang, P. A. Chou, and K. Sripanidkulchai. Distributing streaming media content using cooperative networking. In *ACM/IEEE NOSSDAV*, Miami, FL, USA, May 12-14 2002.