A Peer-to-Peer Architecture for Media Streaming

Duc A. Tran, *Member, IEEE,* Kien A. Hua, *Senior Member, IEEE,* and Tai T. Do

***Abstract*— We have witnessed the success of peer-to-peer (P2P) applications in both commercial and research fields. However, a practical area has received little attention in the context of P2P to date: media streaming. Given the fact that the current Internet does not widely support IP Multicast while content-distribution-networks technologies are costly, P2P could be a promising start for enabling large-scale streaming systems. In our so-called Zigzag approach, we propose a method for clustering peers into a hierarchy called the administrative organization for easy management, and a method for building the multicast tree atop this hierarchy for efficient content transmission. In Zigzag, the multicast tree has a height logarithmic with 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. Zigzag handles such situations gracefully requiring a constant amortized worst-case control overhead. Especially, failure recovery is done regionally with impact on at most a constant number of existing clients and with mostly no burden on the server.**

***Index Terms*— Application-Layer Multicast, Media Streaming, Peer to Peer.**

I. Introduction

Peer-to-peer (P2P) computing has been of interest for quite long and numerous file-sharing systems based on its concepts have been developed [1]. In this paper, we investigate the applicability of P2P to the problem of streaming *live* media. In the P2P streaming 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. Consequently, this paradigm promises to address many critical problems in large-scale streaming systems: (1) the network bandwidth bottleneck at the media source; (2) the cost of deploying extra servers, which would be incurred in content distribution networks; and (3) the infeasibility of IP Multicast on the current Internet [2]. Building an efficient P2P streaming scheme, however, is truly a challenge due to several issues, including the following:

1) 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.
2) 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.
3) Receivers may have to store some local data structures and exchange state information with each other to maintain the connectivity and improve the efficiency of the P2P network. The control overhead at each receiver for fulfilling such purposes should be small to avoid excessive use of network resources and to overcome the resource limitation at each receiver. This is important to the scalability of a system with a large number of receivers.

There are other concerns that make P2P streaming hard to implement, such as how to deal with security and client heterogeneity. We will investigate such problems as part of our future work. In this paper, we only address the three issues listed above. Specifically, we propose a solution called Zigzag. Zigzag organizes receivers into a hierarchy of clusters and builds the multicast tree atop this hierarchy according to a set of rules called C-rules. A cluster has a head and an associate-head, the head responsible for monitoring the memberships of the cluster and the associate-head responsible for transmitting the content to cluster members. Therefore, the failure of the head does not affect the service continuity of other members, or in case the associate-head departs, the head is still working and can designate a new associate-head quickly. In summary, Zigzag provides the following desirable features (here, N is the number of the receivers):

- No matter how N can grow, the node degree in the multicast tree is always bounded by a constant.
- The multicast tree's height is bounded by O($logN$).
- Failure recovery can be done regionally with only impact on at most a constant number of existing receivers and mostly no burden on the source. This is an important benefit because the source is usually overwhelmed by huge requests from the network.
- The protocol control overhead is low. A receiver needs to exchange control information to O($logN$) other receivers in the worst case. On average, it communicates with at most a constant number of other receivers.
- The join procedure is fast, and the maintenance overhead

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

The authors are with the Department of Computer Science, School of Electrical Engineering and Computer Science, University of Central Florida, Orlando, FL 32816. Email: {dtran,kienhua,tdo}@cs.ucf.edu

for the cluster structures is small and independent of N.

Several previous solutions [3], [4], [5] to our problem can provide a subset of the above features, but none can achieve all. The promising performance of Zigzag is substantiated by both theoretical and simulation analyses which also include a comparison with a recent method [3]. The results indicate that Zigzag is indeed a scalable, robust, and efficient solution, especially for large-scale P2P streaming systems.

Zigzag is best applicable to the streaming applications such as a single media server broadcasting a live long-term sport event to many clients, each staying in the system for a long-enough period. Another application can be found in a sensor network where the live monitored data is broadcast from the sensing site to many distant nodes for processing. In the next section, we present a design for the Zigzag approach and an alternative of its. We also report the results from our simulation study. The related works are discussed afterwards, and finally, this paper is concluded with a summary.

II. Zigzag Approach

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". In this section, we propose a scheme called Zigzag which consists of two important entities: the *administrative organization* representing the logical relationships among the peers, and the *multicast tree* representing the physical relationships among them (i.e., how peers link together to receive the content). Firstly, we describe the administrative organization. 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/departure, and discuss performance optimization issues.

A. Administrative Organization

An administrative organization is used to manage the peers currently in the system and illustrated in Fig. 1(a). 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 of size in [2, $3k$].

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

(4) A non-head peer in a cluster at layer j is selected to be the **associate-head** *of that cluster. An exception holds for the highest layer where the server is both the head and the associate-head.*

As an example, in Fig. 1(a), at the highest layer (i.e., layer H-1), the server S is both the head and the associate-head. Peer 3 is the associate-head of a cluster at layer H - 2, and the head of a cluster at layer H - 3. Peer 4 is the head of a cluster at layer H-2, thus also belonging to layer H-1. The role of the associate-head will become clear when we discuss the multicast tree in the next section.

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 grow or shrink. 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 $2k$ (instead of $3k$) and the current size was $2k+1$, after the splitting, the two new clusters would have sizes k and $k+1$ and be prone to be undersize as peers leave.

The above structure implies $H \in [log_{3k}N, log_kN+1]$ for $N \geq k$, 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 is an extension of the one proposed in [3] and employed in [6], [7]. The difference in our definition is the new concept of "associate-head". Furthermore, we propose novel and better solutions to map peers into the administrative organization, to build the multicast tree based on it, and to update these two structures under network dynamics. Those are our main contributions.

Fig. 1(b) illustrates the terms we use for the rest of the paper:

- Subordinate: Non-head peers of a cluster headed by a peer X are called "subordinates" of X. E.g., in Fig. 1(a), peers 1, 2, and 3 are subordinates of peer 4; peers 4, 5, 6, and 7 are subordinates of the server.
- Foreign head: A non-head clustermate Y of a peer X at layer $j > 0$ is called a "foreign head" of layer-(j - 1) subordinates of X. E.g., in Fig. 1(a), peer 4 is a foreign head of peers 5, 6, and 7.
- Foreign subordinate: The layer-(j-1) associate-head of X is called a "foreign subordinate" of Y mentioned above. E.g., in Fig. 1(a), peer 5 is a foreign subordinate of peer 4.
- Foreign cluster: The layer-(j-1) cluster of X is called a "foreign cluster" of Y mentioned above. E.g., in Fig. 1(a), the layer-(H-3) cluster whose head is peer 1 is a foreign cluster of peers 2, 3, and 4.
- Super cluster: Suppose that the head of a cluster U appears in a cluster V at the next higher layer. We call V the "super cluster" of U.

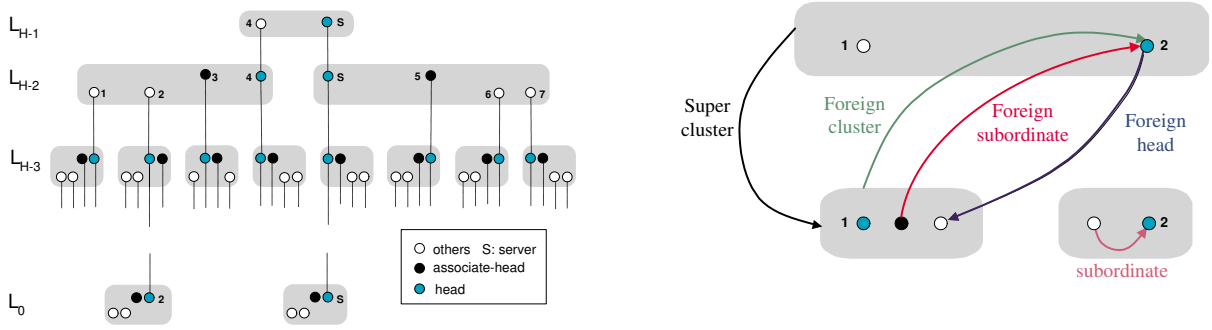
B. Multicast Tree

The multicast tree is built based on the administrative organization. In this section, we propose rules to which the multicast tree must be confined and explain the rationale behind that. The join, departure, and optimization policies must follow these rules. We call these rules C-rules[1] and define them below (demonstrated by Fig. 2(a)):

Definition 1: [C-Rules]

- *Rule 1: A peer, when not at its highest layer, neither has a link out nor a link in.* E.g., peer 4 at layer 1 and layer

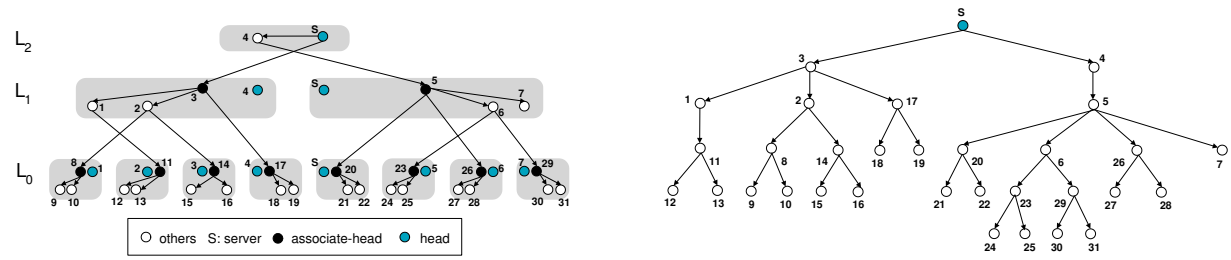
[1] "C" stands for "connectivity". Due to these rules, content goes zigzag from the server to any peer, not through the cluster heads, we name the proposed approach "Zigzag".



(a) Administrative organization of peers

(b) Relationships among clusters and peers

Fig. 1. Administrative organization of peers and relationships among them



(a) The multicast tree atop the administrative organization

(b) The multicast tree in another perspective

Fig. 2. The multicast tree atop the administrative organization (H = 3, k = 4) and its view in another perspective

0 has neither outgoing link nor incoming link because its highest layer is layer 2.

- *Rule 2: Non-head members of a cluster must receive the content directly from its associate-head. In other words, this associate-head links to every other non-head member of the cluster*. E.g., in a cluster at layer 1, associate-head 3 links to non-head members 1 and 2; in a cluster at layer 0, associate-head 8 links to non-head members 9 and 10.
- *Rule 3: The associate-head of a cluster, except for the server, must get the content directly from a foreign head*. E.g., the associate-head 8 of a cluster at layer 0 has a link from peer 2 who is a foreign head of 8; the associate-head 5 of a cluster at layer 1 has a link from its foreign head 4.

It is trivial to prove the above rules guarantee a tree structure including all the peers. Fig. 2(a) gives an example of a 31-client multicast tree built atop the administrative organization. This tree in another perspective is depicted in Fig. 2(b). 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". Furthermore, without otherwise specified, a "layer-j peer" implicitly refers to a node whose highest layer is layer j.

Theorem 1: The worst-case node degree of the multicast tree is at most $6k$ - *3.*

Proof: A node has at most ($3k$ - 1) foreign clusters, thus having at most ($3k$ - 1) foreign subordinates. Consider a node X at its highest layer j. There are three possibilities:

(1) X is the associate-head: X must link to all of its layer-j non-head clustermates (at most $3k$-2 of them) and may have links to a subset of its foreign subordinates (at most $3k$-1 of them). No other links are permitted due to the C-rules. Therefore, the degree of X cannot exceed ($3k$-2) + ($3k$-1) = $6k$ - 3.

(2) X is the head: This implies X must be the server because it would appear at layer j+1 otherwise. Since the server is also the associate-head at the highest layer, the degree of the server cannot exceed $6k$ - 3.

(3) X is neither the head nor the associate-head: X can only have links to a subset of its foreign subordinates. There are at most $3k$-1 of them, hence the degree of X is at most $3k$-1.

In any case, the degree of a node cannot exceed $6k$ - 3, thus proving the theorem true. ■

Theorem 2: The height of the multicast tree is at most $2log_kN+1$*, where* N *is the number of peers.*

Proof: The longest path from the server to a node must be the path from the server to some layer-0 node. This path visits each layer only once. In such a visit, only one link is counted, which is from an associate-head to one of its non-head clustermates. Therefore, the number of nodes, excluding the server, on the path is at most twice the number of layers

H minus one. Since $H \leq log_k N + 1$, the path length is at most $2log_k N + 1$. The theorem has been proved. ■

Theorems 1 and 2 summarize two properties any P2P multicast tree should desire. Indeed, the content from the server to any client goes through at most $2log_k N$ intermediate clients whose outgoing bandwidth contention is kept moderate because each client serves no more than a small constant number of others. However, this advantage might be diminished should the C-rules not be enforced. Therefore, as clients join and leave, we must be able to adjust the tree without violating the C-rules. Overheads incurred by this adjustment should be small to keep the system scalable.

The motivation behind not using the head as the parent for its subordinates in our approach is as follows. Suppose the members of a cluster always get the content from their head. If the highest layer of a node X is j, X would have links to its subordinates at each layer, j-1, j-2, ..., 0, that it belongs to. Since j can be H - 1, the worst-case node degree would be $H \times (3k - 1) = \Omega(klog_k N)$. Furthermore, the closer to the source, the larger degree a node would have. In other words, the bottleneck would occur very early in the delivery path. This might not be acceptable for bandwidth-intensive media streaming.

Our using the associate-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. On the other hand, if the head of a cluster fails, the other members of that cluster will not be affected because the head does not involve in transmitting the content to them. We will discuss policies for failure recovery in more detail shortly.

C. Control protocol

To maintain its position and connections in the multicast tree and the administrative organization, each node X in a layer-j cluster periodically exchanges control messages with its layer-j clustermates, its children and parent on the multicast tree. For non-head peers within the same cluster, the exchanged information X sends is just the peer degree d_X. If the recipient is the cluster head, X sends the following information instead:

- $D(X)$: the current end-to-end delay from the server observed at X.
- A subset of its layer-j clustermates $L_1(X) = \{X_1, X_2, ..\}$, where the occurrence of X_i represents that X is currently forwarding the content to X_i. E.g., in Fig. 2(a), peer 5 at layer 1 needs to send a list $\{6, 7\}$ to the head S because peers 6 and 7 are receiving the content from peer 5.
- A subset of its layer-j clustermates $L_2(X) = \{Y_1, Y_2, ..\}$, where the occurrence of Y_i represents that X is currently forwarding the content to a foreign subordinate whose head is Y_i (the subordinate must be an associate-head according to C-rules). E.g., in Fig. 2(a), peer 5 at layer 1 needs to send a list $\{S, 6\}$ to its head S because associate-head 20 of S and associate-head 23 of peer 6 are receiving the content from peer 5.

If the recipient is the parent, X sends the following information together with its degree:

- A Boolean flag $Reachable(X)$: true iff there exists a path in the multicast tree from X to a layer-0 peer. E.g., in Fig. 2(a), $Reachable(7)$ = false, $Reachable(4)$ = true.
- A Boolean flag $Addable(X)$: true iff there exists a path in the multicast tree from X to a layer-0 peer whose cluster's size is in $[k, 3k - 1]$.

The values of $Reachable$ and $Addable$ at a peer X are updated based on the information received from its children. For instances, if all children send "$Reachable$ = false" to this peer, then $Reachable(X)$ is set to false; $Addable(X)$ is set to true if X receives "$Addable$ = true" from at least a child peer.

The theorem below tells that the control overhead for an average member is a constant. The worst node has to communicate with O($klog_k N$) other nodes, which is acceptable since the information exchanged is just soft-state refreshes.

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

Proof: Consider a node X whose highest layer is j. X belongs to (j + 1) clusters at layers 0, 1, .., j, thus having at most (j + 1) × (3k - 1) clustermates. The number of children of X is its degree, hence no more than 6k - 3. Consequently, the worst-case control overhead at X is upper bounded by (j + 1) × (3k - 1) + (6k-3) + 1 = j × (3k - 1) + 9k - 3. Since j can be H - 1, the worst-case control overhead is O($klog_k N$).

However, the probability that a node has its highest layer to be j is at most $(N/k^j) / N = 1/k^j$. Thus, the amortized worst-case overhead at an average node is at most $\Sigma_{j=0}^{H-1}(1/k^j \times (j \times (3k - 1) + 9k - 3)) \rightarrow$ O(k) with asymptotically increasing N. Theorem 3 has been proved. ■

D. Client Join and Departure

The multicast tree is updated whenever a new client joins or leaves. The new tree must not violate the C-rules specified in Section II-B. We propose the join and departure algorithms below.

1) Join Algorithm: 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. A peer X pursues the below steps on receipt of a join request: (In this algorithm, $d(Y, P)$ is the delay from Y to P measured during the contact between Y and P.)

1. If X is a layer-0 associate-head
1.1. Add P to the only cluster of X
1.2. Make P a new child 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 associate-head of C. Therefore, the join algorithm stops if X is already a layer-0 associate-head. Otherwise, X considers "addable" child peers first because we want to add P to a layer-0 cluster of size $[k, 3k-1]$ to avoid oversize. Among these "addable" children, X chooses Y such that $D(Y)+d(Y, P)$ is minimum to keep the delay from the server to the new peer P as small as possible. In the case there is no "addable" child, X considers "reachable" children and pursue the same delay minimization strategy. The peer Y in the above algorithm always exists since X must be a "reachable" peer in order to receive the join request earlier.

In Step 2.1.1 or 2.2.1, P has to contact with at most d_X peers (d_X is the degree of X). Since the tree height is at most $(2log_kN + 1)$ and the maximum degree is $(6k - 3)$, the number of nodes that P has to contact is at most $(2log_kN + 1) \times (6k - 3) = O(k \times log_kN)$. This proves Theorem 4 true.

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

Proof: This theorem has been proved above. ■

The join procedure terminates at step 1.2 at some layer-0 associate-head X, 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 $[k, 3k]$, no further work is necessary. Otherwise, this cluster has to be split so that the newly created clusters have sizes in $[k, 3k]$ to maintain the structure of the administrative organization. We present the split algorithm in Section II-E.1.

2) Departure Algorithm: We consider a peer X who departs the tree either purposely or accidentally due to a failure. Suppose that X's highest layer is layer $j \in [0, H)$. As a result of the control protocol, the parent/children peers of X, and all layer-i clustermates of X for $i \in [0, j]$ must be aware of this departure. Basically, the following tasks are required for recovery: (1) The parent removes the link to X; (2) The children need a new parent to get the content; (3) Each layer-i cluster of X ($i \in [0, j\text{-}1]$) selects a new head since the head X no longer exists; and (4) The layer-j cluster needs a new associate-head if X has been its associate-head. Task (1) is trivial; we propose the detail policies for the remaining tasks below.

We first consider the case $j = 0$, thus X belongs to only one cluster. If X is not the associate-head of this cluster, no further work is required. Otherwise, the cluster head chooses among its subordinates a new associate-head which will reconnect to the current parent of X and be the parent for all the other non-head members. The peer X' to be the new associate-head is the one experiencing the best end-to-end delay, i.e., $D(X')$ is smallest. This is based on the heuristic that choosing a peer closest to the former associate-head would have a little or no negative impact on the service quality currently perceived by the others.

We now consider the case $j > 0$. For each foreign subordinate Y who is a child of X, the head Y_{head} of Y is responsible for finding a new parent for it. Y_{head} just selects Z, a layer-j non-head clustermate, that has the minimum degree, and asks it to forward data to Y. 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 non-associate-head subordinate of X at layer 0. X' will replace X as the new head for each of those clusters. X' also replaces X's position at layer j. In other words, if X used to be the associate-head, X' now becomes the associate-head and gets a link from the existing parent of X. If X is not associate-head, X' gets a link from the current associate-head of X.

Fig. 3(a) gives the resulting multicast tree and administrative organization after peer 4 fails. The original P2P system is given in Fig. 2(a). Peer 18 at layer 0 is selected to replace the position of peer 4 in every cluster peer 4 used to belong to. In this example, peer 5 first reconnects to the server because there is no other surviving non-head peer at the highest layer; after peer 18 replaces peer 4 at this layer, peers 5 will reconnect to peer 18 to follow the C-rules. Another example is given in Fig. 3(b) where peer 3, which is an associate-head, fails. Peer 15 will replace this peer, however before that peer 17 (ex-child at layer-0 of peer 3) reconnects to peer 1 (having minimum degree) for a quick playback resumption. Also to keep service continuity, peers 1 and 2 first reconnect to peer 4 at layer 1 before finally connecting to peer 15 when peer 15 becomes the associate-head in place of peer 3.

In overall, a client departure requires a few (at most one peer at layer 0 plus $3k$-1 peers at layer j-1 plus $3k$-2 peers at layer j) to reconnect, and does not burden the server. The overhead of failure recovery is consequently stated as follows:

Theorem 5: In the worst case, the number of peers that need to reconnect due to a failure is $6k$-2.

Proof: This theorem has been proved above. ■

E. Cluster Maintenance

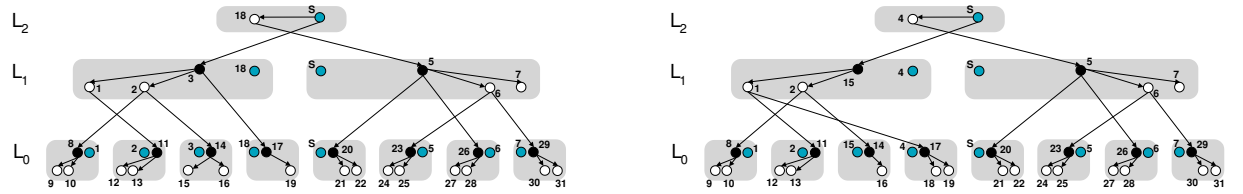
The administrative organization requires that the size of any cluster, except for the highest-layer cluster, be between k and $3k$. Due to client joins and departures, some cluster may become oversize or undersize. This cluster has to be split into smaller clusters, or be merged with another cluster to form a larger cluster, so that the size restriction is satisfied. In this section, we propose algorithms for cluster split and cluster mergence.

1) Cluster Split: Suppose we decide to split a layer-j cluster U with head X_{head}, associate-head X_{assoc}, and the other peers X_1, .., X_n. The goal is to create a new cluster V and move some peers of U to this cluster. The basic idea of this split is illustrated in Fig. 4. After moving some peers into V, we need to find two peers X'_{head} and X'_{assoc} as the head and the associate-head, respectively. All the other members of V will get the content from X'_{assoc}. The head X'_{head} will appear in the super cluster of U and get a link from the associate-head of that cluster. Furthermore, the associate-head X'_{assoc} will get a link from the parent of X_{assoc}.

Other details of the split algorithm include determining what peers to move from U to V and which of them to be the head and the associate-head of V, and making necessary reconnections to enforce the C-rules. There are three cases: $j = 0$, $j \in (0, H\text{-}1)$, and $j = H$-1.

Case 1: $j = 0$:

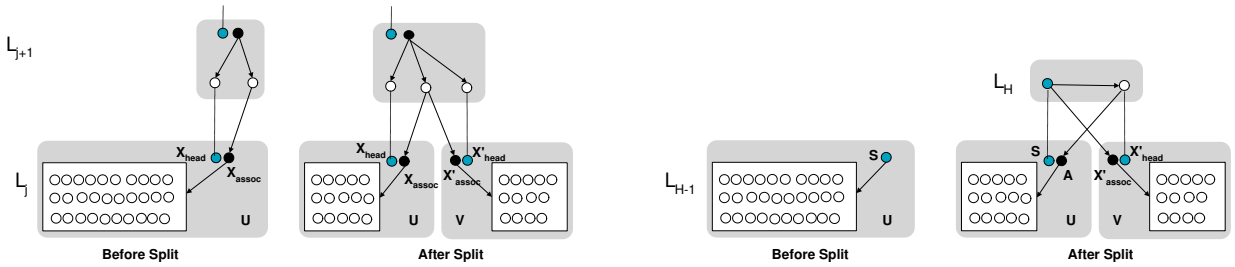
This case is handled simply by following the steps below:



(a) Peer 4 fails

(b) Peer 3 fails

Fig. 3. Failure Recovery: The original P2P tree is given in Fig. 2(a)



(a) General case

(b) Highest-layer case

Fig. 4. Cluster Split: A cluster is split into two smaller clusters, each of size between k and $3k$

(1) Sort $\{X_1, .., X_n\}$ in the non-increasing order of end-to-end delay $D(.)$. Suppose the resulting list is $\{X'_1, .., X'_n\}$. Move the first $n/2$ peers $\{X'_1, .., X'_{n/2}\}$ to cluster V. As a result, cluster U retains only peers close to its associate-head X_{assoc}. (Because $D(X_j) = D(X_{assoc}) + d(X_{assoc}, X_j)$.)

(2) Select $X'_{head} = X'_1$ and $X'_{assoc} = X'_2$ to be the head and associate-head of cluster V, respectively. In addition, make peers $X'_3, .., X'_{n/2}$ children of the associate-head X'_{assoc}. By being promoted to be the head and associate-head, X'_{head} and X'_{assoc} will be closer to the server, thus improving their end-to-end delay which used to be poorest.

Case 2: $j \in$ (0, H-1):

Let x_{il} be a Boolean value, true if and only if X_i currently links to the layer-(j-1) associate-head of X_l. Clearly, $x_{ii} = 0$ for all i because of the C-rules. We follow the steps below:

(1) Partition $\{X_1, .., X_n\}$ 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. Set u will remain in cluster U and set v will be moved to cluster V. Since after the split the peers in U cannot link to peers in V and vice versa, this condition helps reduce the number of peer reconnections resulted from the split.

(2) For each peer $X_i \in u$ and $X_l \in v$ such that $x_{il} > 0$, remove the link from X_i to the layer-(j-1) associate-head of X_l, and select a random peer in set v other than X_l to be the new parent for this associate-head. Inversely, for each peer $X_i \in v$ and $X_l \in u$ such that $x_{il} > 0$, a similar procedure takes place.

(3) We need a head for cluster V. This head will appear at the super cluster of X_{head}, thus abandoning its current children. To minimize the number of reconnections, the head is chosen to be the peer X'_{head} in v having the smallest degree. Each abandoned child will reconnect to a random foreign head appearing in v - $\{X'_{head}\}$. We also choose the peer X'_{assoc} in v having the second-smallest degree to be the associate-head for cluster V.

Case 3: $j = H$ - 1

In this case, we have $X_{head} = X_{assoc} = S$. We firstly follow the three steps as in the case $j \in$ (0, H-1). Further steps are explained as follows. Before the split, the server links to its clustermates and probably some foreign subordinates at layer j-1 = H-2. After the split, the server's highest layer becomes j+1 = H and the server can no longer link to its current children due to the C-rules. Therefore, each abandoned child at layer j-1 will reconnect to a random foreign head appearing in set u, and each abandoned child at layer j will reconnect to the new associate-head of U chosen to be a peer $A \in u$ having the minimum degree. To enforce the C-rules, this associate-head A must get the content from a foreign head. We can choose X'_{head} at layer j+1 to be this foreign head. The split procedure at the highest layer is illustrated in Fig. 4(b).

It might happen that the super cluster becomes oversize due to admitting X'_{head}, hence a similar split takes place. In this case, which is not frequent, the split is helpful anyway since the super cluster is close to be oversize. 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 small.

Theorem 6: The worst-case split overhead is O(k).

Proof: Since computations for the other two cases are similar, let us compute the worst-case split for the case $j \in (0, H\text{-}1)$. In the algorithm for this case, Step 2 requires $\Sigma_{X_i \in u, X_l \in v}(x_{il} + x_{li})$ peers to reconnect. This value must equal the number of associate-heads at layer $j-1$. Therefore, Step 2 requires at most n peers to reconnect. In Step 3, the number of former children of X'_{head} is less than or equal to the number of its foreign subordinates, hence at most ($3k$ - 1) of them need to reconnect. Furthermore, at most ($3k$-2) peers need to reconnect to X'_{assoc}. X'_{head} and X'_{assoc} need to reconnect too. In total, the split procedure needs at most n + ($3k$ - 1) + ($3k$ - 2) + 2 = n + $6k$ - 1 peers to reconnect. Since n is greater but close to $3k$, the theorem has been proved. ■

2) Cluster Mergence: Cluster mergence is always carried out from top to bottom in the administrative organization. In other words, a mergence at layer j is only accomplished only if every layer-(j+1) cluster has a size at least k. In case a layer-(j+1) cluster is undersize, it will be merged first before the layer-j mergence takes place.

We consider an undersize layer-j ($j \in [0, H\text{-}2)$) [2] cluster U having head X_U and associate-head Y_U. We combine it with a cluster V that is at the same layer and has the smallest size. Suppose that cluster V has head X_V and associate-head Y_V. There are two cases: (1) Either X_U or X_V is the head at layer j+1, and (2) Neither of them is.

In the first case (illustrated in Fig. 5(a)), without loss of generality, suppose that X_U is the head at layer j+1. The following changes are made to merge U and V:

(1) X_U and Y_V are chosen to be the head and associate-head of the new cluster U+V, respectively. To enforce the C-rules, all the other members of U+V reconnect to Y_V.

(2) Furthermore, X_V cannot appear at layer j+1 anymore, thus abandoning its current layer-j children if any. To overcome this, the head of each layer-j child (except for Y_U if it happens to be such a child) selects a non-head clustermate at layer-(j+1), that currently has the minimum degree, to be the new parent of that child.

(3) At layer j+1: In case X_V used to be the associate-head, we need to elect a new associate-head. The head X_U simply selects the clustermate having the smallest degree as the associate-head, and informs the other clustermates to reconnect to this associate-head.

In the second case (illustrated in Fig. 5(b)) where neither X_U nor X_V is the head at layer j+1, if one of them is the associate-head at layer j+1, we select it to be the head of U+V. If neither is associate-head, supposing X_U has a higher degree, we select X_U to be the head of U+V, and vice versa. Without loss of generality, we assume X_U is the head of U+V. The subsequent steps are below:

(1) Since X_V no longer appears at layer j+1, all its layer-j children, if any, must reconnect. These children are simply redirected to X_U at layer j+1.

(2) We also need an associate-head for U+V. If the size of U is larger than that of V, Y_U becomes the associate-head of U+V. Otherwise, Y_V is chosen as the head. Supposing Y_V is the associate-head, all the other members of U+V except for X_U will reconnect to Y_V.

(3) No further work is necessary if the current parent of Y_V is not X_U. However, if this happens Y_V needs a new parent because after the mergence Y_V and X_U belong to the same cluster at layer j. This new parent is chosen to be a layer-(j+1) non-head clustermate of X_U, that has the smallest degree.

The merge procedure runs centrally at the head of U with assistance from the head of V. Since the number of peers involves is at most 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 7k.

Proof: In either case, the number of peers required to reconnect is at most the number of peers in U+V plus the number of layer-j children of X_U (or X_V). Therefore, no more than k + $3k$ + $3k$ = $7k$ peers need to reconnect. The theorem has been proved. ■

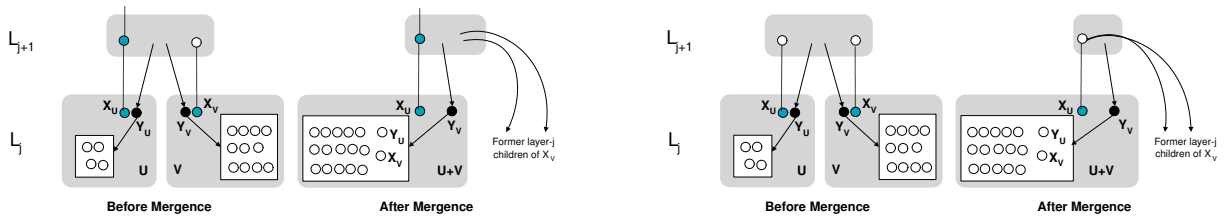
F. Performance Optimization

Under network dynamics, the administrative organization and multicast tree can be periodically reconfigured to provide better quality of service to clients. A strategy we can follow is to dynamically balance the service load among the peers, thus preventing network bottleneck. For instance, a peer busy serving many children might consider switching its parenthood for some children to another non-head clustermate which is less busy. Service load should be balanced based on not only the number of children a peer has but also its bandwidth capacity. To this goal, we compute for each peer the ratio between its degree to its bandwidth capacity and attempt to equalize this ratio among peers.

Let us focus on balancing the service load among peers in a layer-j cluster ($j > 0$). According to the C-rules, the head of this cluster, except for the server at layer H-1, does not link to any other peer. Therefore, only its subordinates are eligible for transferring service load. Additionally, since the associate-head must always be the parent of all other non-head clustermates, we should only consider transferring service load from a non-head non-associate-head member X to another non-head member Y. Suppose that X currently links to foreign subordinates x_1, x_2, .., x_m, and probably to some other clustermates. Hence, its degree $d(X)$ is m + δ, where δ equals zero for the case X is not the associate-head, and equals the number of such clustermates otherwise. X follows the steps below to transfer the service load, whose result does not violate the C-rules: (In this algorithm, $B(.)$ defines the bandwidth capacity of a peer.)

1. For(i = 1; $i \leq m$; i++)
1.1. Select a non-head clustermate Y:
Y is not the head of x_i
$(\frac{d_X}{B_X} - \frac{d_Y}{B_Y})^2 - (\frac{d_X-1}{B_X} - \frac{d_Y+1}{B_Y})^2 > 0$
$(\frac{d_X}{B_X} - \frac{d_Y}{B_Y})^2 - (\frac{d_X-1}{B_X} - \frac{d_Y+1}{B_Y})^2$ is max
1.2. If such Y exists
1.2.3. Redirect x_i to Y
1.2.4. Update d_X and d_Y accordingly

[2]The case $j = H\text{-}2$ is handled similar with minor modification.



(a) X_U is the head at layer j+1

(b) Neither X_U nor X_V is the head at layer j+1

Fig. 5. Cluster Merge: A small cluster is merged with another cluster to make a larger cluster whose size is between k and $3k$

In this algorithm, X transfers some children to another clustermate Y if doing so minimizes the difference between the service load of X and that of Y. Even though this only affects a few peers (at most $3k$-1), frequently activating the algorithm might cause many peer reconnections and thus result in discontinuity of client playback. Therefore, we suggest that a peer run the load-balancing procedure when its degree becomes larger than a value Δ chosen appropriately, such as what follows. We consider a layer-j ($j < H$-1) cluster with n members excluding the head: X_1, X_2, .., X_n where X_1 is the associate-head. Due to the C-rules, we must have $\Sigma_{i=1}^{i=n} d_{X_i}$ = $2n$. Since we would like to balance $\frac{d_{X_i}}{B_{X_i}}$, the best value for d_{X_i} is d^* = $2n \times B_{X_i}$ / $\Sigma_{j=1}^{j=n} B_{X_j}$. Since it is feasible to include the bandwidth capacity together with the degree in the control information exchanged in the control protocol, this value can be computed easily. We can choose $\Delta = 2d^*$.

III. Alternative Solution: Direct Zigzag

We presented the first design of the Zigzag approach in [6], [7]. In this early alternative, the C-rules is the same except that all the non-head members of a cluster receive the content directly from a foreign head in the super cluster, hence no need for the role of associate-head. We call this design *direct* Zigzag and therefore call the design in the previous section *indirect* Zigzag because the non-head members of a cluster get the content indirectly from a foreign head through their associate-head. For abbreviation, we refer to these two schemes as D-Zigzag and I-Zigzag, respectively.

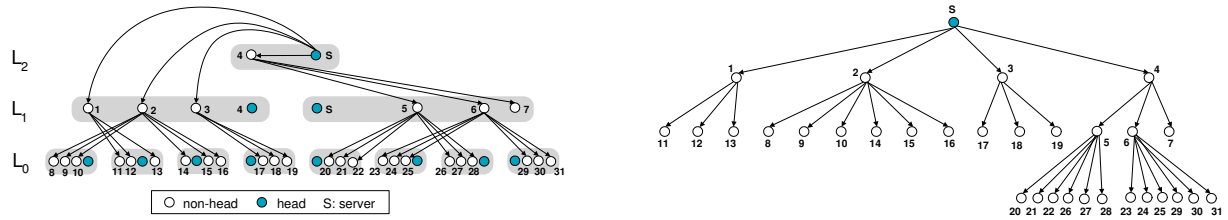
Fig. 6 illustrates an example of the multicast tree resulted from D-Zigzag. The longest path from the server to any peer has to visit each layer only once and in such a visit goes through one and only one node. The only exception applies to the highest layer where the server links to all of its subordinates. Therefore, the multicast tree height is at most the number of layers $H \leq log_k N$+1. In other words, D-Zigzag results in a shorter multicast tree than I-Zigzag does. However, we can intuitively see that the peer degree in D-Zigzag is higher than that in I-Zigzag, thus the former does not handle peer bottleneck as well as the latter does. Table I gives a theoretical comparison between these two alternatives, which shows a significant improvement in I-Zigzag over the other. The algorithms of D-Zigzag and proofs for its performance analyses can be found in [7].

It is our suggestion that I-Zigzag should be used for most P2P streaming systems due to its scalability, robustness, and efficiency. However, for a live streaming system consisting of high-capable clients, the peer bottleneck and control overhead may not be severe. In this case, D-Zigzag is more preferable because this scheme would provide a high level of content liveness.

IV. Simulation Study

The last section provided the worst-case analyses of the Zigzag approach. To investigate its performance under various scenarios, we carried out a simulation-based study. Besides evaluating performance metrics mentioned in the previous sections, i.e., peer degree, join/failure overhead, split/merge overhead, and control overhead, we also considered Peer Stretch and Link Stress (defined in [2]). Peer Stretch is the ratio between the length of the data path from the server to a peer in our multicast tree to the length of the shortest path between them in the underlying network. The pure unicast approach always has the optimal peer stretch. The stress of a link is the number of times the same packet goes through that link. An IP Multicast tree always has the optimal link stress of 1 because a packet goes through a link only once. An application-level multicast scheme should have small stretch and stress to keep the end-to-end delay short and the network bandwidth efficiently utilized.

We used the GT-ITM Generator [8] to create a 10,000-node transit-stub graph as our underlying network topology. The server's location is fixed at a stub-domain node. We investigated a system with 5000 clients located randomly among the other stub-domain nodes. Therefore, the client population accounts for 5000/10,000 = 50% of the entire network. We set the value k to 5, hence each cluster has at most 15 and no less than 5 peers. We studied three scenarios, the first investigating a failure-free system running Zigzag, the second investigating a system running Zigzag and allowing failures, and the third comparing two systems, one running Zigzag and the other running NICE [3]. We found that the I-Zigzag alternative provided better performance values than the D-Zigzag alternative did, which backs our theoretical comparison shown in Table I. Due to paper length restrictions, we only report the results for the I-Zigzag scheme in the following sections and invite the reader to [7] for the D-Zigzag scheme's results.



(a) Multicast tree atop the administrative organization

(b) The tree under another perspective

Fig. 6. Direct Zigzag: All non-head members of a cluster must receive the content from one of their foreign heads (k=4, H=3)

TABLE I
I-ZIGZAG VERSUS D-ZIGZAG: WORST-CASE OVERHEAD ANALYSES

Zigzag Alternatives	Join	Failure	Degree	Merge/Split	Control
Direct	O($klog_kN$)	O(k^2)	O(k^2)	O(k^2)	O($klog_kN$)
Indirect	O($klog_kN$)	O(k)	O(k)	O(k)	O($klog_kN$)

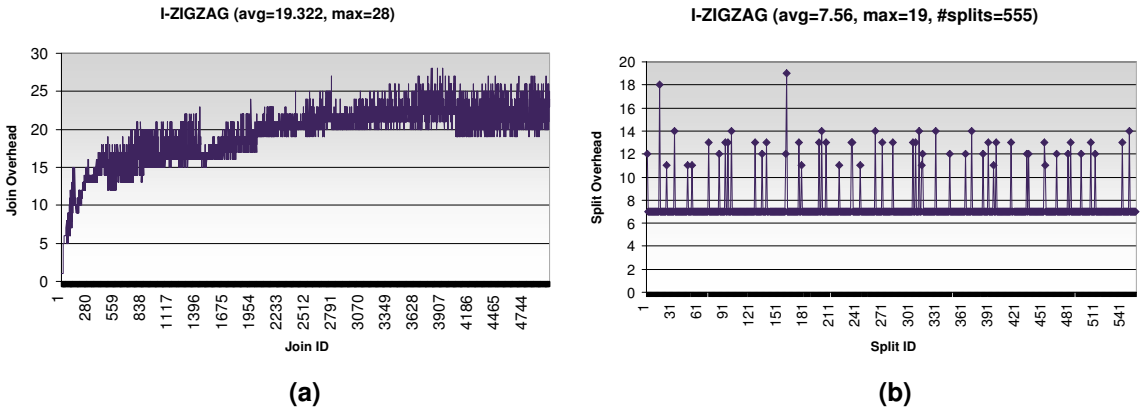


Fig. 7. 5000 Joins: Join and Split Overhead

A. Scenario 1: No Failure

In this scenario, as 5000 clients joined the system, we collected statistics on control overhead, node degree, peer stretch, and link stress. We also estimated the join overhead and split overhead accumulated during the joins.

The overhead of a join is measured as the number of peers that the new client has to contact before being added to the multicast tree. Fig. 7(a) shows that on the average, a new client needs to contact 20 clients. In the worst case, a new client has to contact 28 clients, or 0.56% of the client population. The join overhead increases slowly as more clients join. However, this manner is not monotonic; in fact, the overhead for a join after a significant split takes place is rapidly reduced. This is understandable because a split on an oversize cluster helps reduce the node degree, thus the new client needs to contact fewer peers. We can see this correlation between the join procedure and split procedure in Fig. 7(a) and Fig. 7(b): a significant increase in split overhead corresponds to a rapid decrease in join overhead.

In terms of split overhead, since we wanted to study the worst scenario, we opted to run a split whenever detecting a cluster is oversize. However, as illustrated in Fig. 7(b), small split overhead is incurred during the joins of 5000 clients. There are totally 555 splits and the worst-case split requires only 19 reconnections. Most of the time, a split requires about 8 peers to reconnect. This accounts for only 8/5000 = 0.16% of the client population.

Fig. 8(a) shows the out-degrees for all 5000 peers. The thick line at the bottom represents the degrees of the leaves which are 0-degree. Although the theoretical analysis in Section II-B shows that the worst-case degree is $6k$-3 = 27, our simulation found that all peers forward the content to no more than 13 other peers. Thus, I-Zigzag handles peer bottleneck efficiently, and furthermore distributes the service load among the peers fairly (stdev = 2.648). In terms of control overhead, as shown in Fig. 8(b), most peers have to exchange control states with only 13 others. The dense area represents peers at layers close to layer 0 while the sparse area represents peers

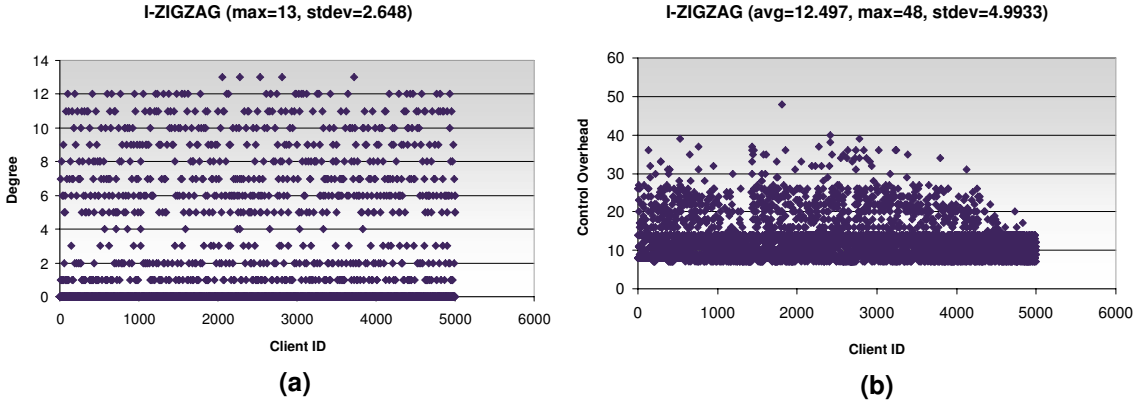


Fig. 8. 5000 Clients: Node Degree and Control Overhead

at higher layers. Those peers at high layers do not have a heavy control overhead either; most of them communicate with around 20 peers, only 0.4% of the client population. This can be considered lightweight, taking the fact that the exchanged control information is small in size.

The study on link stress and peer stretch results in Fig. 9. I-Zigzag has a low stretch of 6.13 for most of the clients, and a link stress of 4.0 for most of the underlying links used. Especially, these values are quite fairly distributed (see the two dense areas in both figures). We recall that the client population in our study accounts for 50% of the entire network in which a pair of nodes have a link with a probability of 0.5. Therefore, the results we got are promising.

B. Scenario 2: Failure Possible

In this scenario, we started with the system consisting of 5000 clients, which was built based on the first scenario study. We let a number (500, 1000, 1500, 2000, 2500) of peers fail sequentially and evaluated the overhead for recovery and the overhead of mergence during that process. Fig. 10(a) shows the results for recovery overhead as failures occur. We can see that most failures do not affect the system because they happen to layer-0 peers (illustrated by a thick line at the bottom of the graph). For those failures happening to higher-layer peers, the overhead to recover each of them is small and less than 14 reconnections (no more than 0.5% of the client population). Furthermore, the overhead to recover a failure does not really depend on the number of clients in the system. This substantiates our theoretical analysis in Section II-D.2 that the recovery overhead is always bounded by $6k$-2 = 28 regardless of the client population size. Even though the theoretical upper bound overhead is 28, the worst-case overhead from our simulation study turns out to be only a half (only 14 reconnections).

In terms of merge overhead, the result is exhibited in Fig. 10(b). There are totally 148 calls for cluster mergence, each requiring 5 peers on average to reconnect. In the worst case, only 11 peers need to reconnect, which accounts for no more than 0.44% of the client population. This study is consistent with our theoretical analysis in Section II-D.2 that the merge overhead is always small regardless of the client

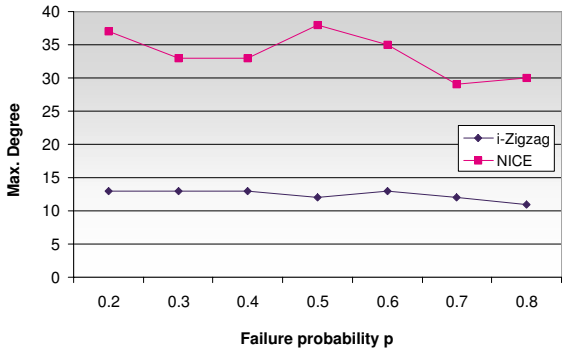


Fig. 11. I-Zigzag vs. NICE: Maximum Node Degree

population size. More interestingly, the worst merge overhead in our simulation (11 reconnections) is a lot smaller than the theoretical worst-case overhead ($7k$ = 35 reconnections) implied in Theorem 7.

C. Scenario 3: I-Zigzag versus NICE

We compared the performances between I-Zigzag and NICE. NICE was recently proposed in [3] as an efficient P2P technique for streaming data. NICE also organizes the peers in a hierarchy of bounded-size clusters as in our approach. However, NICE and I-Zigzag are fundamentally different due to their own multicast tree construction and maintenance strategies. For example, NICE always uses the head of a cluster to forward the content to the other members, whereas we use the associate-head instead.

We worked with the following scenario. The system initially contained only the server and stabilized after 3000 clients join sequentially. Afterwards, we ran an admission control algorithm, which is a loop of 2000 runs, each run letting a client fail with probability $p \in [0.2, 0.8]$ or a new client join with probability 1-p. After the admission control algorithm stopped, we collected statistics on the trees generated by I-Zigzag and NICE, respectively.

Fig. 11 exhibits an advantage of I-Zigzag over NICE in terms of peer bottleneck. I-Zigzag has a maximum peer degree only half of the maximum degree of NICE. This simulation

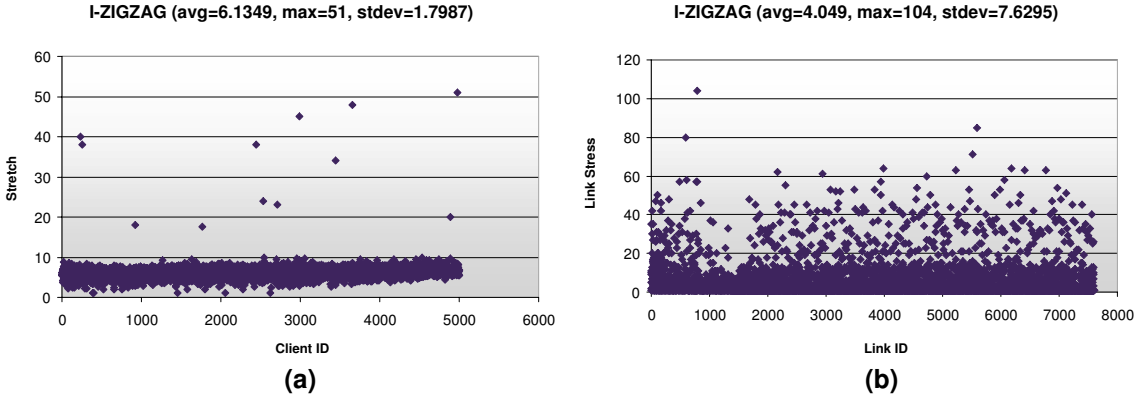


Fig. 9. 5000 Clients: Link Stress and Peer Stretch

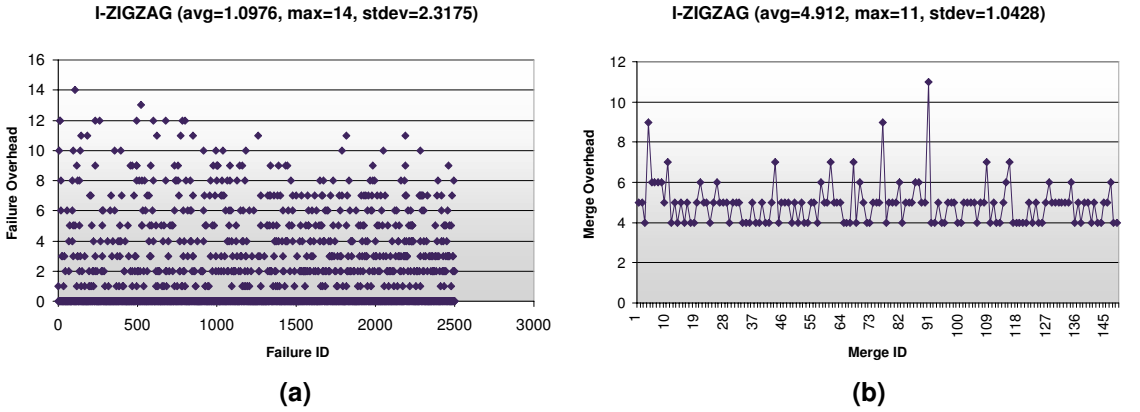


Fig. 10. Failure and merge overhead as 2500 peers fail

result is consistent with the theoretical analyses. Indeed, [3] shows that the peer degree in NICE is O($klog_kN$) while our theoretical analyses have shown that the peer degree in I-Zigzag is O(k) only.

Fig. 12 and Fig. 13 show the average-case and worst-case results, respectively, for other metrics including control overhead, join overhead, failure overhead, and link stress. Though the two approaches provide close average-case results, I-Zigzag is superior to NICE because the former deals with the worst-case scenarios better. The maximum control overhead of our technique is about 60% of that of NICE (Fig. 13(a)), and the join overhead is always about 3 or 4 reconnections fewer than that of NICE in both average and worst cases (Fig. 12 and Fig. 13(b)).

Recovery overhead was measured for each failure during the period of the 2000-run loop. By enforcing our C-rules, our recovery algorithm is more efficient than that of NICE. Indeed, a failure happens to a peer at its highest layer j in NICE requires $j \times 3k$ peers to reconnect. Since j can be log_kN, the recovery overhead in this scheme can be $3klog_kN$. According to our theoretical analyses, the Zigzag approach requires at most a constant number of reconnections in a recovery phase, regardless of how many peers are in the system. Consequently, we can see in Fig. 13(c) that I-Zigzag clearly prevails NICE in terms of maximum failure overhead. We note that the average failure overhead values for both schemes can be smaller than 1 because there are many layer-0 peers and their failure would require zero reconnection.

The average link stress of I-Zigzag is close to that of NICE, however the worst-case link stress of the former is slightly better, as shown in Fig. 13(d). This is no way by accident, but is rooted from the degree bound of each scheme. The worst case degree is O($klogN$) in NICE, while bounded by O(k) in I-Zigzag. Hence, it is more likely for NICE to have many more identical packets being sent through an underlying link near heavy loaded peers. In this study, where k = 5 and $N \leq$ 5000, the two curves are quite close because log_kN is close to k. If the system runs in a larger underlying network with many more clients, log_kN will be a lot larger than k, and we can expect that link stress in our technique will be sharply better than that in NICE.

V. Related Work

Several techniques have been proposed to address the problem of streaming media on the Internet. Most of them try to overcome the lack of IP Multicast, which makes the problem challenging, by implementing the multicast paradigm at the application layer based on IP Unicast services only. They can be categorized into two classes: overlay-router approach and peer-to-peer approach.

Failure Probability	Performance Metrics (*Average Case*)							
	Control Overhead		Join Overhead		Failure Overhead		Link Stress	
	Zigzag	Nice	*Zigzag*	Nice	*Zigzag*	Nice	*Zigzag*	Nice
0.2	*12.08536*	12.35147	*18.92733*	22.5788	*1.014888*	0.981865	*3.851808*	3.886655
0.3	*11.61025*	12.0441	*19.13317*	22.1099	*1.081761*	0.97099	*3.781343*	3.809562
0.4	*11.56818*	12.1305	*18.83806*	22.32767	*1.085608*	1.139364	*3.66275*	3.655614
0.5	*11.31591*	11.62467	*18.42239*	21.50463	*1.178357*	0.940239	*3.57236*	3.535413
0.6	*11.03657*	11.32126	*18.63507*	21.80835	*1.100749*	1.054276	*3.393512*	3.399871
0.7	*10.44709*	10.82249	*18.05811*	21.72979	*1.233935*	0.945927	*3.243787*	3.273842
0.8	*9.8914*	10.25358	*18.22372*	21.44774	*1.133082*	0.983679	*3.095078*	3.13906

Fig. 12. I-Zigzag vs. NICE: Average Case

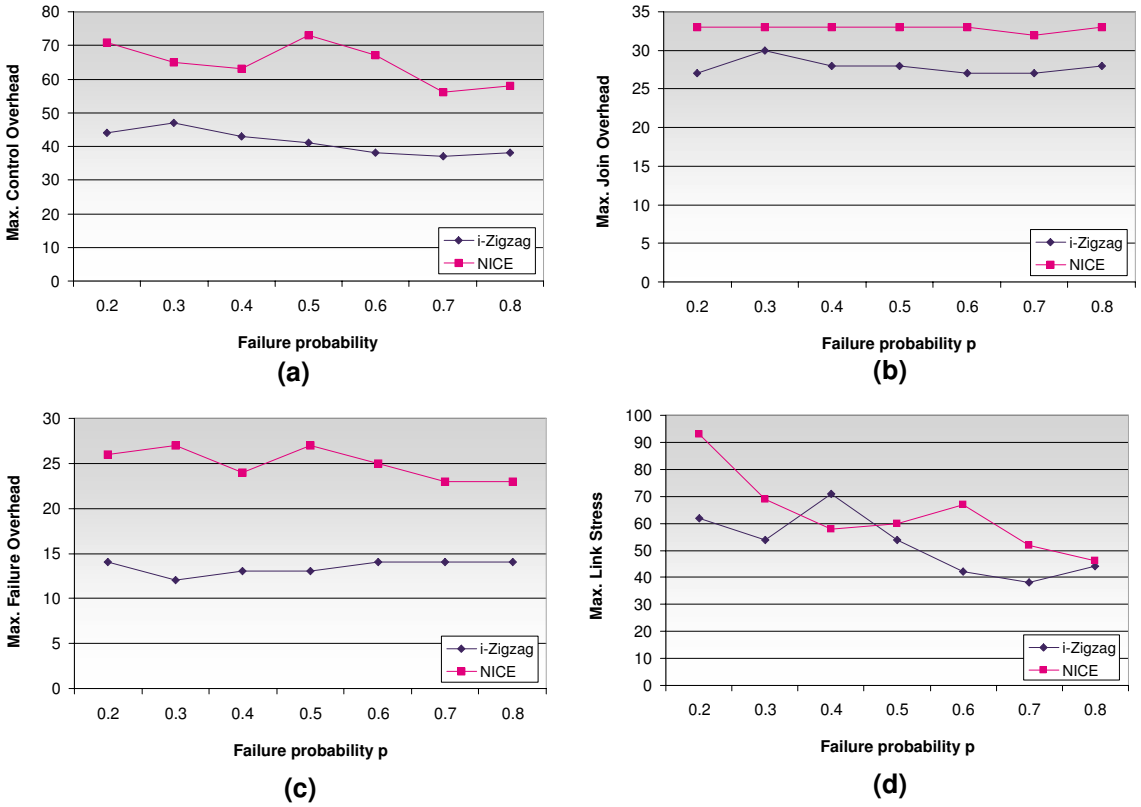


Fig. 13. I-Zigzag vs. NICE: Worst Case

In the overlay-router approach [9], [10], [11], [12], [13], [14], a number of reliable servers are installed across the network to act as the software routers with multicast functionality. These routers are interconnected according to a topology which forms an overlay for running the services. The content is transmitted from the source to a set of receivers on a multicast tree consisting of the overlay routers. A new receiver joins an existing media stream by connecting to an overlay router appearing on the delivery path to an existing receiver. This approach is designed to be scalable since the receivers can get the content not only from the source, but also from software routers, thus alleviating bandwidth demand at the source.

The peer-to-peer (P2P) approach assumes no extra resources such as the dedicated servers mentioned above. A multicast tree involves only the source and the receivers, thus avoiding the complexity and cost of deploying and maintaining extra servers. Since we employ this approach, we discuss the differences between our technique and the existing P2P techniques below.

[15] introduced a simple P2P scheme for video-on-demand services, which seems to be the first P2P proposal to support streaming applications. However, it neither mentions the stability of the system under network dynamics nor analyzes the protocol overheads involved. [4] proposed SpreadIt which builds a single distribution tree of the peers. A new receiver joins by traversing the tree nodes downward from the source until finding one with unsaturated bandwidth. Spreadit has to get the source involved whenever a failure occurs, thus vulnerable to disruptions due to the severe bottleneck at the source. Additionally, orphaned peers reconnect by using

the join algorithm, resulting in a long blocking time before the their service can resume. CoopNet [5] employs a multi-description coding method for the media content. In this method, a media signal is encoded into several separate streams, or descriptions, such that every subset of them is decodable. CoopNet builds multiple distribution trees spanning the source and all the receivers, each tree transmitting a separate description of the media signal. Therefore, a receiver can receive all the descriptions in the best case. A peer failure only causes its descendant peers to lose a few descriptions. The orphaned are still able to continue their service without burdening the source. However, this is done with a quality sacrifice. Furthermore, CoopNet puts a heavy control overhead on the source since the source must maintain full knowledge of all distribution trees.

Narada [2], [16] focuses on multi-sender multi-receiver streaming applications, maintains a mesh among the peers, and establishes a tree whenever a sender wants to transmit a content to a set of receivers. Narada only emphasizes on small P2P networks. Its extension to work with large-scale networks was proposed in [17] using a two-layer hierarchical topology. To better reduce cluster size, whereby reducing the control overhead at a peer, the scheme NICE in [3] focuses on large P2P networks by using the multi-layer hierarchical clustering idea as we do. However, NICE always uses the head to forward the content to its members, thus incurring a high bottleneck of O($log_k N$). Though an extension could be done to reduce this bottleneck to a constant, the tree height would become O($log_k N \times log_k N$). Our approach, no worse than NICE 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 approach is upper bounded by a constant while NICE requires O($log_k N$). All these are a significant improvement for bandwidth-intensive applications such as media streaming.

VTrails [18] is a commercial P2P streaming product. The broadcast source vTCaster automatically creates a tree structure based on receiver location and connection type. vTCaster collects packet-level information from each receiver, dynamically optimizing the tree in order to serve those with high-speed connections first and connect receivers who are close, network-wise (within the same ISP network, same company, etc.). Since vTCast is the central processing server for maintaining the entire tree, it is doubtful this technique can work efficiently with a large group of transient receivers. Another streaming product based on P2P is AllCast [19]. However, it is hard for us to do a specific comparison with AllCast in the absence of published information.

A recent work [20] takes into account the heterogeneity of peers in their bandwidth capacity. Due to this heterogeneity, a peer may have to receive the content from multiple supplying peers. [20] investigates an interesting problem of deciding what media data segments these supplying peers need to send to the receiving peer. [20] proposed an optimal solution for this assignment, and techniques to amplify the total system streaming capacity. Although different from our problem, theirs motivates us to extend our scheme to consider the case where a peer may receive the content collectively from more than one peer.

VI. Summary

This paper discussed the problem of streaming live media in a large P2P network. We focused on a single source only and aimed at optimizing the worst-case values for important performance metrics. The proposed solution, Zigzag, uses a novel multicast tree construction and maintenance approach based on a hierarchy of bounded-size clusters. The key in Zigzag's design is the introduction of C-rules. Our algorithms were developed to achieve the following desirable properties:

- High liveness: The end-to-end delay from the media server to a client is not only due to the underlying network traffic, but largely depends on the local delays at intermediate clients due to queuing and processing. The local delay at such an intermediate client is mostly affected by its bandwidth contention. We keep the end-to-end delay small because the multicast tree height is at most logarithm of the client population and each client needs to forward the content to at most a constant number of peers.
- Low control overhead: Each client periodically exchanges soft-state information only to its clustermates, parent, and children. Since a cluster is bounded in size and the client degree bounded by a constant, the control overhead at a client is small. On average, the overhead is a constant regardless of the client population.
- Efficient join and failure recovery: A join can be accomplished without asking more than O($logN$) existing clients, where N is the client population. Especially, a failure can be recovered quickly and regionally with fewer than a constant number of reconnections and mostly no affection on the server.
- Low maintenance overhead: To enforce the rules on the administrative organization and the multicast tree, maintenance procedures (merge, split, and performance refinement) are invoked periodically with very low overhead. Fewer than a constant number of clients need to relocate in such a procedure.

We provided both theoretical proofs and simulation studies to verify the above merits of Zigzag. We also compared Zigzag to NICE [3], finding that Zigzag is promising in terms of most performance metrics.

References

[1] Stefan Saroiu, P. Krishna Gummadi, and Steven D. Gribble, "A measurement study of peer-to-peer file sharing systems," in *ACM/SPIE Multimedia Computing and Networking*, San Jose, CA, USA, January 2002.

[2] Yang-Hua Chu, Sanjay G. Rao, and Hui Zhang, "A case for end system multicast," in *ACM SIGMETRICS*, 2000, pp. 1–12.

[3] S. Banerjee, Bobby Bhattacharjee, and C. Kommareddy, "Scalable application layer multicast," in *ACM SIGCOMM*, Pittsburgh, PA, 2002, pp. 205–217.

[4] 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.

[5] 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, pp. 177–186.

[6] Duc A. Tran, Kien A. Hua, and Tai T. Do, "Scalable media streaming in large peer-to-peer networks," in *ACM Multimedia Conference*, Juan Les Pins, France, December 2002, pp. 247–250.
[7] Duc A. Tran, Kien A. Hua, and Tai T. Do, "Zigzag: An efficient peer-to-peer scheme for media streaming," in *(CD-Rom) Proc. of IEEE INFOCOM*, San Francisco, CA, March-April 2003.
[8] Ellen W. Zegura, Ken Calvert, and S. Bhattacharjee, "How to model an internetwork," in *Proceedings of IEEE INFOCOM*, San Francisco, CA, 1996, pp. 594–602.
[9] Y. Chawathe, S. McCanne, and E. Brewer, "An architecture for internet content distribution as an infrastructure service," Unpublished work, February 2000.
[10] Duc A. Tran, Kien A. Hua, and Simon Sheu, "A new caching architecture for efficient video services on the internet," in *IEEE Symposium on Applications and the Internet*, Orlando, FL, USA, 2003, pp. 172–181.
[11] Kien A. Hua, Duc A. Tran, and Roy Villafane, "Overlay multicast for video on demand on the internet," in *ACM Symposium on Applied Computing*, Melbourne, FL, USA, 2003, pp. 935–942.
[12] S. Q. Zhuang, B. Y. Zhao, and A. D. Joseph, "Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination," in *11th ACM/IEEE NOSSDAV*, New York, June 2001, pp. 11–20.
[13] D. Pendakaris and S. Shi, "ALMI: An application level multicast infrastructure," in *USENIX Symposium on Internet Technologies and Systems*, Sanfrancisco, CA, March 26-28 2001, pp. 49–60.
[14] J. Jannotti, D. K. Gifford, and K. L. Johnson, "Overcast: Reliable multicasting with an overlay network," in *USENIX Symposium on Operating System Design and Implementation*, San Diego, CA, October 2000, pp. 197–212.
[15] S. Sheu, Kien A. Hua, and W. Tavanapong, "Chaining: A generalized batching technique for video-on-demand," in *Proc. of the IEEE Int'l Conf. On Multimedia Computing and System*, Ottawa, Ontario, Canada, June 1997, pp. 110–117.
[16] Yang-Hua Chu, Sanjay G. Rao, S. Seshan, and Hui Zhang, "Enabling conferencing applications on the internet using an overlay multicast architecture," in *ACM SIGCOMM*, San Diego, CA, August 2001, pp. 55–67.
[17] S. Jain, R. Mahajan, D. Wetherall, and G. Borriello, "Scalable self-organizing overlays," Tech. Rep., UW-CSE 02-06-04 University of Washington, 2000.
[18] vTrails, "A p2p streaming product," http://www.vtrails.com.
[19] AllCast, "A p2p streaming product," http://www.allcast.com.
[20] Dongyan Xu, Mohamed Hefeeda, Susanne Hambrusch, and Bharat Bhargava, "On peer-to-peer media streaming," in *IEEE Conference on Distributed Computing and Systems*, July 2002, pp. 363–371.

Duc A. Tran graduated from the University of Central Florida (UCF) in May 2003 with a Ph.D. degree in Computer Science. At UCF, he received the 2003 Order of Pegasus from UCF and the 2002 IEEE Outstanding Graduate Student from IEEE Orlando Section. His research work, partially funded by the National Science Foundation, includes some 20 refereed papers mainly in the areas of multimedia networking, multimedia databases, and P2P systems. Dr. Tran is a member of IEEE and ACM, and going to join the University of Dayton as an Assistant Professor of Computer Science in August 2003.

Kien A. Hua received his Ph.D. degree in Electrical Engineering from the University of Illinois at Urbana-Champaign in 1987. Dr. Hua worked for IBM from 1987 to 1990, where he led a project to develop a highly parallel computer which was a precursor to the series of products known as SPx (i.e., SP1, SP2, etc.) He also led another project to design a high-performance processor for mainframe computers. This work led to a Best Paper Award and a Best Presenter Award from the 1990 IEEE International Conference on Computer Design. Dr. Hua has been with the University of Central Florida since 1990 and is now a professor of Computer Science directing the Data Systems Laboratory. His current research interests are multimedia systems, Internet computing, and mobile computing. He has published over 100 articles, including four recognized as best/top papers at international conferences. Dr. Hua is a senior member of IEEE.

Tai T. Do is a Ph.D. student in Computer Science at the University of Central Florida, working in the Data Systems Laboratory. He received a B.S. degree in Electrical Engineering from the University of Oklahoma in 2001. His main research interests are image retrieval/indexing, multimedia networking, and clustering and broadcasting techniques in mobile ad hoc networks. He is a student member of ACM.