ZIGZAG: An Efficient Peer-to-Peer Scheme for Media Streaming

Duc A. Tran
School of Electrical Engineering
and Computer Science
University of Central Florida
Orlando, FL 32816–2362
Email: dtran@cs.ucf.edu

Kien A. Hua
School of Electrical Engineering
and Computer Science
University of Central Florida
Orlando, FL 32816-2362
Email: kienhua@cs.ucf.edu

Tai Do
School of Electrical Engineering
and Computer Science
University of Central Florida
Orlando, FL 32816-2362
Email: tdo@cs.ucf.edu

Abstract— **We design a peer-to-peer technique called ZIGZAG for single-source media streaming. ZIGZAG 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 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 and unpredictable client behaviors. ZIGZAG handles such situations gracefully requiring a constant amortized control overhead. Especially, failure recovery can be done regionally with little impact on the existing clients and mostly no burden on the server.**

I. Introduction

We are interested in the problem of streaming live bandwidth-intensive media from a single source to a large quantity of receivers on the Internet. The simplest solution dedicates an individual connection to stream the content to each receiver. This method consumes a tremendous amount of costly bandwidth and leads to an inferior quality stream for the receiver, making it nearly impossible for a service provider to serve quality streaming to large audiences while generating profits. IP Multicast [1], [2] could be the best way to overcome this drawback since it was designed for group-oriented applications. However, its deployment on the Internet is still limited due to several fundamental concerns [3], [4]. Therefore, we seek a solution that employs IP unicast only but offers considerably better performance efficiency than the dedicated-connection approach.

We presented in [14] a technique called *Chaining*. To the best of our knowledge, Chaining is the first *peer-to-peer* (P2P) streaming technique. In such a communication paradigm, 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. More recent P2P streaming techniques were introduced in [3], [5–7], [15], [16]. We will discuss them in Section IV.

The following issues are important in designing an efficient P2P technique:

- 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.
- 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.
- 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.

In this paper, we propose a new P2P streaming technique, called ZIGZAG, to address all of the above issues. ZIGZAG organizes receivers into a hierarchy of bounded-size clusters and builds the multicast tree based on that. The connectivity of this tree is enforced by a set of rules, which guarantees that the tree always has a height O($log_k N$) and a node degree O(k^2), where N is the number of receivers and k a constant. Furthermore, the effects of network dynamics such as unpredictable receiver behaviors are handled gracefully without violating the rules. This is achieved requiring a 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. This is an important benefit because the source is usually overwhelmed

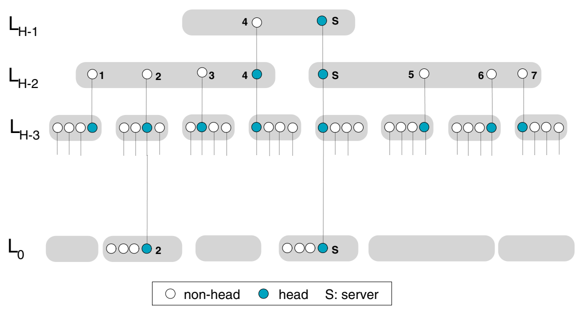


Fig. 1. Administrative organization of peers

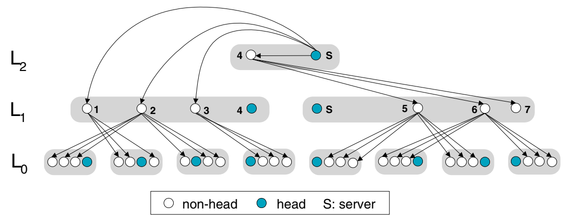


Fig. 2. The multicast tree of peers (H = 3, k = 4)

by huge requests from the network. Previous solutions such as [5–7] provide some of the above features, but none can provide all.

Besides theoretical analyses that prove the correctness of our scheme, a simulation-based study was carried out to evaluate its performance under various scenarios. In this study, we also compared ZIGZAG to the technique proposed in [5], a recent scheme for P2P streaming.

The remainder of this paper is organized as follows. Section II presents the protocol details of the ZIGZAG scheme. Section III reports the results from our performance evaluation study. Section IV discusses related work with comparisons to ZIGZAG. Finally, Section V concludes this paper with brief remarks.

II. 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". In this section, we propose the ZIGZAG scheme 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 get connected). Firstly, we describe the administrative organization 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/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. 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):

- Layer 0 contains all peers.
- 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, $3k$].
- 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 [5]. 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:

- Subordinate: Non-head peers of a cluster headed by a peer X are called "subordinate" of X.
- Foreign head: A non-head (or server) clustermate of a peer X at layer $j > 0$ is called a "foreign head" of layer-(j-1) subordinates of X.
- Foreign subordinate: Layer-(j-1) subordinates of X are called "foreign subordinates" of any layer-j clustermate of X.
- Foreign cluster: The layer-(j-1) cluster of X is called a "foreign cluster" any layer-j clustermate of X.

B. Multicast Tree

Unlike in [5], the administrative organization in ZIGZAG 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):

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

- A peer, when at its highest layer, can only link to its foreign subordinates. E.g., peer 4 at layer 2 only links to peers 5, 6, and 7 at layer 1, which are foreign subordinates of 4. The only exception is the server; at the highest layer, the server links to each of its subordinates.
- 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 a foreign head. E.g., non-head peers in layer-0 cluster of peer 1 have a link from their foreign head 2; peers 1, 2 and 3 have a link from their foreign 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 to 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).

Proof: A node has at most ($3k$ - 1) foreign clusters, thus having at most ($3k$ - 1) $\times$ ($3k$ - 1) foreign subordinates. Since a non-server node X can only have outgoing links when X is at its highest layer and since these links only point to a subset of its foreign subordinates, the degree of X is no more than the number of its foreign subordinates, which is at most ($3k$ - 1) $\times$ ($3k$ - 1). The server also has links to its subordinates at the highest layer, therefore the server degree is at most ($3k$ - 1) $\times$ ($3k$ - 1) + ($3k$ - 1) = $9k^2$ - $3k$. Theorem 1 has been proved. ■

Theorem 2: The height of the multicast tree is O($log_k N$) 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. The path from the server to any layer-0 node goes through each layer only once, and does not contain horizontal links (i.e., links between layer mates) except at the highest layer. Therefore, the number of nodes on the path is at most the number of layers H plus one. Since H = O($log_k N$), the path length is at most O($log_k N$) + 1. Theorem 2 has been proved. ■

The motivation behind not using the head as the parent for its subordinates in the ZIGZAG scheme is as follows[1]. 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) = Ω($log_k N$). 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 a foreign 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.

[1]Since a peer gets the content from a foreign head, but not its head, and can only forward the content to its foreign subordinates, but not its subordinates, we named our technique ZIGZAG.

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 communicates with its layer-j clustermates, its children and parent on the multicast tree. For peers within a cluster, the exchanged information is just the peer degree. If the recipient is the cluster head, X also sends a list L = {[X_1, d_1], [X_2, d_2], ..}, where [X_i, d_i] represents that X is currently forwarding the content to d_i peers in the foreign cluster whose head is X_i. E.g., in Fig. 2, at layer 1, peer 5 needs to send a list {[S, 3], [6, 3]} to the head S. If the recipient is the parent, X instead sends the following information:

- 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, $Reachable$(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($log_k N$) 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 $\times$ $log_k N$), 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) $\times$ ($3k$ - 1) subordinates. The number of children of X is its degree, hence no more than $9k^2$ - $3k$. Consequently, the worst-case control overhead at X is upper bounded by (j + 1) $\times$ ($3k$ - 1) + $9k^2$ - $3k$ = j $\times$ ($3k$ - 1) + $9k^2$ - 1. Since j can be H - 1, the worst-case control overhead is O(k $\times$ $log_k N$).

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 $\Sigma_{j=0}^{H-1}(1/k^j \times (j \times (3k-1) + 9k^2 - 1) \rightarrow$ O(k) with asymptotically increasing N. Theorem 3 has been proved. ■

D. Client Join

The multicast tree is augmented whenever a new client joins. The new tree must not violate the rules specified in Section II-B. 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
2.    Add $P$ to the only cluster of $X$
3.    Make $P$ a new child of the parent of $X$
4. Else
5.    If $Addable(X)$
6.       Select a child $Y$:
            $Addable(Y)$ and $D(Y)$+$d(Y, P)$ is min
7.       Forward the join request to $Y$
8.    Else
9.       Select a child $Y$:
            $Reachable(Y)$ and $D(Y)$+$d(Y, P)$ is min
10.      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 $[k, 3k)$ to avoid being oversize. The end-to-end delay is attempted to be better after each node contact. In Step 5 or 8, P has to contact with at most d_X peers where d_X is the degree of X. Since the tree height is O($log_k N$) and the maximum degree is O(k^2), the number of nodes that P has to contact is only O($k^2 \times log_k N$). This proves Theorem 4 true.

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

The join procedure terminates at step 3 at some leaf 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 needed. Otherwise, this cluster has to be split so that the newly created clusters must have sizes in $[k, 3k]$. To avoid the overhead of splitting, we propose to do so periodically, not right after a cluster size becomes $3k$+1. Suppose we decide to split a layer-j ($j \in$ [1, H-2]) cluster[2] 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) subordinates of X_l. Clearly, x_{ii} = 0 for all i because of the ZIGZAG tree rules. 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) subordinates 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 II-B). 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.

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

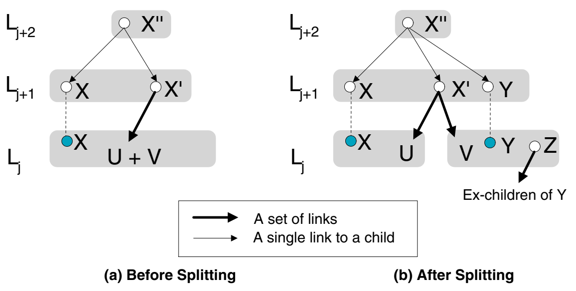


Fig. 3. Split Algorithm

It might happen that the cluster on layer j+1 becomes oversize 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).

Proof: Step 2 requires $\Sigma_{X_i \in U, X_l \in V}(x_{il} + x_{li})$ peers to reconnect. This value is at most $\Sigma_{i=1}^{i=n} x_i$ where x_i is the number of subordinates of X_i at layer $j-1$. Therefore, Step 2 requires at most $n \times (3k$ - 1) $< 6k \times (3k$ - 1) to reconnect.[3] In Step 3, the number of former children of Y is less than the number of its foreign subordinates, hence at most $(3k$ - $1)^2$ of them need to reconnect to Z. In total, the split procedure needs at most $6k \times (3k$ - 1) + $(3k$ - $1)^2$ nodes to reconnect. Theorem 5 has been proved. ∎

[3]We would not wait until $n \geq 6k$ to do the split.

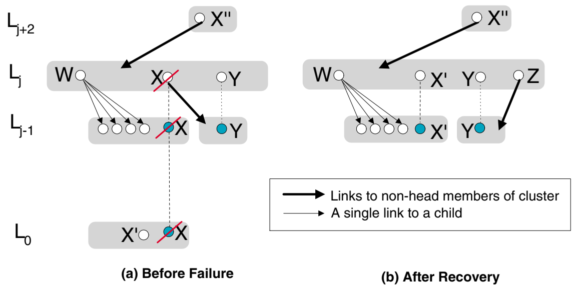


Fig. 4. Failure Recovery: Peer X fails

E. Client Departure

The new tree after a client departs must not violate the rules specified in Section II-B. 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 II-C, the parent peer of X, all subordinates 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 $j > 0$. The failure recovery procedure is exhibited in Fig. 4. 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 subordinates 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 subordinate 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. In overall, a client departure affects a few (at most ($3k$ - 1) $\times$ ($3k$ - 1)) peers at layer j-1 and mostly does not burden the server. 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:
   a) X is the head at layer j+1: For each child cluster of Y, a foreign head $Z \neq Y$ that has the minimum degree will be the new parent; Z must not be the head of this cluster.
   b) 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).

Proof: Step 2 requires at most 2 $\times$ ($3k$ - 1) peers to reconnect. Step 3 requires at most ($3k$ - 1) $\times$ ($3k$ - 1) peers to reconnect. In total, no more than ($9k^2$ - 1) peers need to reconnect. Theorem 7 has been proved. ■

F. Performance Optimization

Under the network dynamics, the administrative organization and multicast tree can be periodically reconfigured in order to provide better quality of service to clients. Consider a peer X, in its highest-layer cluster $j > 0$, is busy serving many children. It might consider switching its parenthood of some children to another non-head clustermate which is less busy. Suppose that X currently has links to foreign clusters C_1, C_2, .., C_m, each C_i having s_i non-head subordinates, respectively. We propose two strategies for handling the refinement: Degree-based Switch and Capacity-based Switch.

1) Degree-based Switch: As a result of the control protocol, X knows which layer-j non-head clustermate has what degree. We denote the degree of a peer Y by d_Y. The below steps are pursued by X to transfer the service load, attempting to balance the degree as much as possible:

```
1. For($i$ = 1; $i \leq m$; $i$++)
2.     Select a non-head clustermate $Y$:
           $Y$ is not the head of $C_i$
           $d_X$ - $d_Y$ - $s_i > 0$
           $d_X$ - $d_Y$ - $s_i$ is max
3.     If such $Y$ exists
4.         Redirect non-members of $C_i$ to $Y$
5.         Update $d_X$ and $d_Y$ accordingly
```

2) Capacity-based Switch: It is likely that peers have different bandwidth capacities. In this case, we define the busyness of a peer X to be $\frac{d_X}{B_X}$ where B_X is the bandwidth capacity of peer X. X follows the steps below to transfer the service load:

```
1. For($i$ = 1; $i \leq m$; $i$++)
2.     Select a non-head clustermate $Y$:
           $Y$ is not the head of $C_i$
           $(\frac{d_X}{B_X} - \frac{d_Y}{B_Y})^2 - (\frac{d_X - s_i}{B_X} - \frac{d_Y + s_i}{B_Y})^2 > 0$
           $(\frac{d_X}{B_X} - \frac{d_Y}{B_Y})^2 - (\frac{d_X - s_i}{B_X} - \frac{d_Y + s_i}{B_Y})^2$ is max
3.     If such $Y$ exists
4.         Redirect non-members of $C_i$ to $Y$
5.         Update $d_X$ and $d_Y$ accordingly
```

The capacity-based switch attempts to balance the peer busyness among all non-head peers of a cluster. In order to work with this strategy, a peer must have knowledge about not only a clustermate degree but also its bandwidth capacity. This is feasible by requiring the exchanged soft-state information to include both degree and bandwidth capacity.

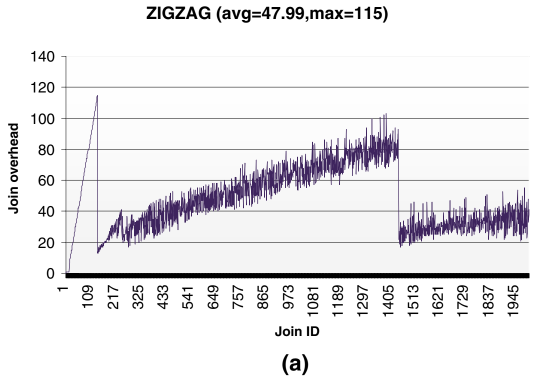
The performance optimization procedure makes the service load fairly distributed among the peers without violating the multicast tree rules. However, frequently calling it might cause many peer reconnections, which would affect the continuity of client playback. To prevent this in the case of degree-based switch, [4] a peer runs the optimization procedure when its degree becomes larger than Δ chosen as follows. We consider a layer-j ($0 < j < H$-1) cluster with non-head members X_1, X_2, .., X_n. The total number of their children must equal the total number of their layer-(j-1) non-head subordinates (due to the rules enforced on the multicast tree). Let this quantity be n'. Clearly, $n' \in$ [(k-1)(n+1), (3k-1)(n+1)]. If all X_i's are balanced in service load, the average degree will approximately be $n'/n \in$ [(k-1)(1+1/(3k-1)), (3k-1)(1+1/k)], thus $n'/n \in$ (k-1, 3k+3). We can choose Δ = 2 $\times$ (3k+3).

III. Performance Evaluation

The last section provided the worst-case analyses of ZIGZAG. 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 [3]). 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 dedicated-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 3240-node transit-stub graph as our underlying network topology. The server's location is fixed at a stub-domain node. We investigated our system with 2000 clients located randomly among the other stub-domain nodes. Therefore, the client population accounts for 2000/3240 = 62% 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 ZIGZAG system, the second investigating a ZIGZAG system allowing failures, and the third comparing ZIGZAG to NICE [5], a recent P2P streaming scheme. The performance optimization procedure was disabled in ZIGZAG. [5] We report the results in the following sections.

[4]The case of capacity-based switch can be handled similarly.



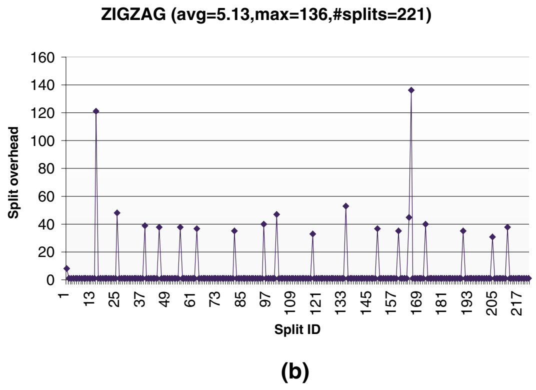


Fig. 5. 2000 Joins: Join and Split Overhead

A. Scenario 1: No Failure

In this scenario, as 2000 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. 5(a) shows that on the average, a new client needs to contact 48 clients, only 2.4% of the client population. In the worst case, a new client has to contact 115 clients, or 5.7% of the population. It is interesting that the worst case occurs early when there are only 136 clients in the system. This is because, at this moment, the server has too many children and a new client has to ask all of them,

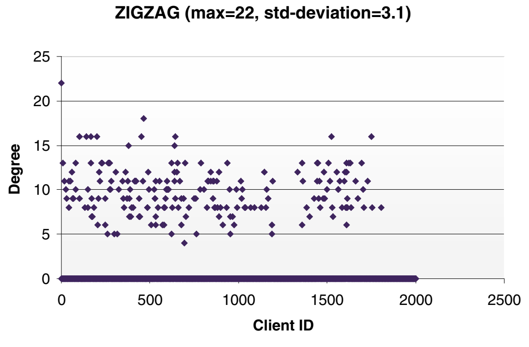
[5]We did study the case where the performance optimization was enabled in ZIGZAG and the results were significantly improved. However, to avoid any bias in comparison with NICE, only the study with performance optimization disabled is reported in this paper.

thus resulting in many contacts. However, it becomes a lot better then (we can see a "downfall" right after the 136th join in Fig. 5(a)). This is understandable since the split procedure takes place after detecting a cluster at the second-highest layer is oversize. As the result of this split, the server will have very few children. We can see the correlation between the join procedure and split procedure from Fig. 5(a) and Fig. 5(b). Each run of the split procedure helps reduce the overhead incurred by a client join. The two "downfalls" in Fig. 5(a) corresponds to the two peak points in Fig. 5(b). We can conjecture that the join-overhead curve would continue going up slowly as more clients join until a constant point (e.g., when overhead approximates 90) when it would fall down to a very low value (e.g, overhead approximates 20). This behavior would repeat, making the join algorithm scalable with the client population.

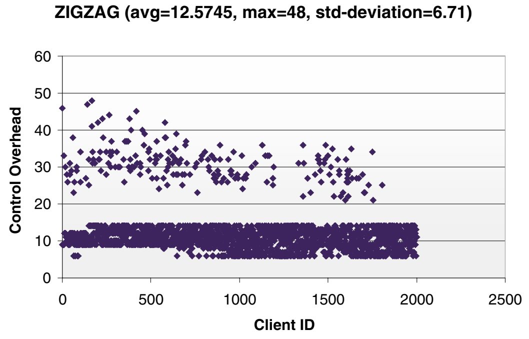
In terms of split overhead, since we wanted to study the worst scenario, we opted to run a split whenever detecting a cluster is oversize. However, as illustrated in Fig. 5(b), small split overhead is incurred during the joins of 2000 clients. The worst case requiring 136 reconnections is when the server is overloaded by many children, but after splitting, the server bottleneck is resolved very well (we can see the two downfalls in Fig. 5(a), which are consequences of the 16th and 166th splits.) Hence, most of the time, a split requires about 5 peers, or 0.25% of client population, to reconnect. Although our theoretical analysis in Section II-D shows a worst-case split overhead of O(k^2), the real result turns out to be a lot smaller than this bound.

Fig. 6(a) shows that not only the node degrees in a ZIGZAG multicast tree are small, but also they are quite balanced. The thick line at the bottom represents the degrees of the leaves, which are 0-degree. For those peers forwarding the content to other, they forward to about 10 other peers. This study shows that ZIGZAG handles peer bottleneck efficiently, and distributes the service load among the peers fairly. In the worst case, a peer has to transmit the content to 22 others, which is tiny to the client population of 2000 clients. In terms of control overhead, as shown in Fig. 6(b), most peers have to exchange control states with only 12 others. The dense area represents peers at layers close to layer 0 while the sparse area represents peers at higher layers. Those peers at high layers do not have a heavy control overhead either; most of them communicate with around 30 peers, only 1.5% of the population. This can be considered lightweight, taking the fact that the control information is very small in size.

The study on link stress and peer stretch results in Fig. 7. ZIGZAG has a low stretch of 3.45 for most of the clients, and a link stress of 4.2 for most of the underlying links used. Especially, these values are quite fairly distributed. We recall that the client population in our study accounts for 62% 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 very promising. We also studied the case where the number of clients is small (fewer than 100), its results showed that the stretch was no more than 1.2 on average and 4.0 in the worst case. The stress was also very small, less than 2.1 on average and 9.2 in the worst case. Since we focus on a large client population and due to paper length restriction, we decided not to show the results for small P2P networks in this section.



(a)



(b)

Fig. 6. 2000 Clients: Node Degree and Control Overhead

B. Scenario 2: Failure Possible

In this scenario, we started with the system consisting of 2000 clients, which was built based on the first scenario study. We let a number (200, 400, .., 1000) of peers fail sequentially and evaluated the overhead for recovery and the overhead of mergence during that process. Fig. 8(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 mostly less than 20 reconnections (no more than 2% of client population). Furthermore, the overhead to recover a failure does not depend on the number of clients in the system. On average, the recovery overhead is always between 0.95 and 0.98, when the system has 1800, 1600, .., and 1000 clients left, respectively. This substantiates our theoretical analysis in Section II-E that the recovery overhead is always bounded by a constant regardless of the client population size.

In terms of merge overhead, the result is exhibited in Fig. 8(b). There are totally 62 calls for cluster mergence, each requiring 11 peers on average to reconnect. In the worst case,

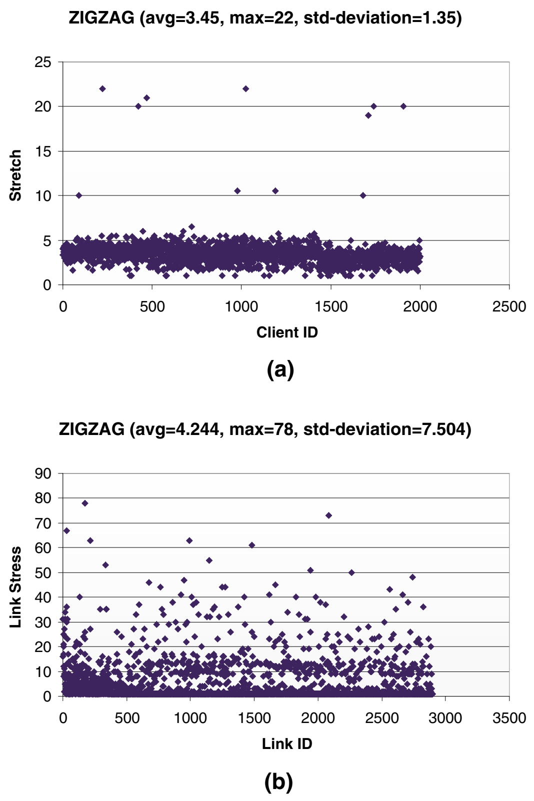


Fig. 7. 2000 Clients: Link Stress and Peer Stretch

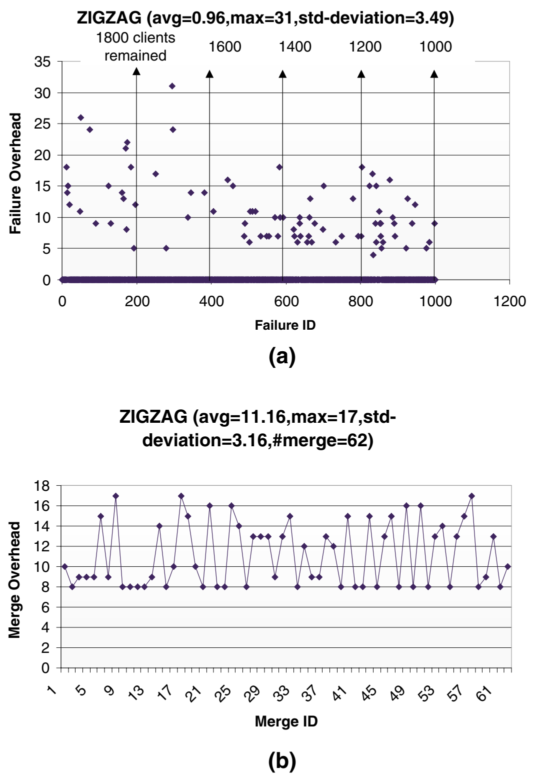


Fig. 8. Failure and merge overhead as 1000 peers fail

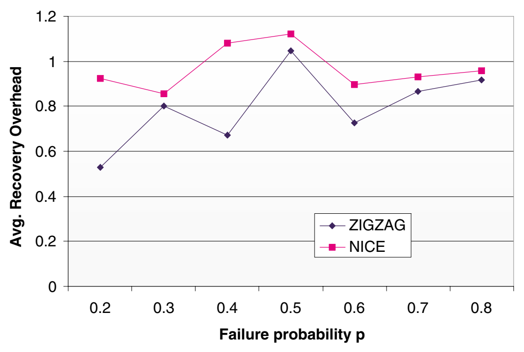


Fig. 9. ZIGZAG vs. NICE: Failure Recovery Overhead

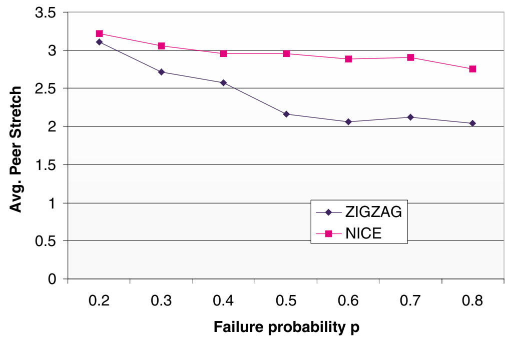
only 17 peers need to reconnect, which accounts for no more than 1.7% of the client population. This study is consistent with our theoretical analysis in Section II-E that the merge overhead is always small regardless of the client population size. Indeed, the final merge call and the first merge call require the same number (only 10) of reconnections, even though the system has different numbers of clients before those calls take place.

C. Scenario 3: ZIGZAG vs. NICE

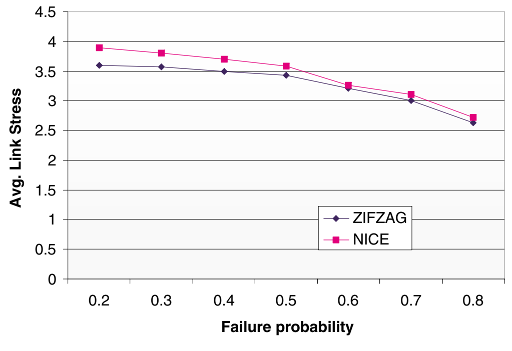
We compared the performances between ZIGZAG and NICE. NICE was recently proposed in [5] as an efficient P2P technique for streaming data. NICE also organizes the peers in a hierarchy of bounded-size clusters as ZIGZAG does. However, NICE and ZIGZAG are fundamentally different due to their own multicast tree construction and maintenance strategies. For example, NICE always uses the head of a cluster to forward the content to the other members, whereas ZIGZAG uses a foreign head instead. According to the analyses in [5], NICE has a worst-case node degree O($logN$), worst-case control overhead O($logN$), average control overhead O(k), and a worst-case join overhead O($logN$). Obviously, ZIGZAG is no worse than NICE in terms of join overhead and control overhead. Furthermore, ZIGZAG is significantly better than NICE in terms of node degree. For comparisons in terms of failure recovery overhead, peer stretch, and link stress, we report the results drawn from our simulation in this section.

We worked with the following scenario. The system initially contained only the server and stabilized after 1000 clients join sequentially. Afterwards, we ran an admission control algorithm, which is a loop of 1000 runs, each run letting a client to fail or a new client to join. The probability that a client fails is p (ranging between 0.2 and 0.8), thus a new client joins with a probability 1 - p. After the admission control algorithm stopped, we collected statistics on the trees generated by ZIGZAG and NICE, respectively. Recovery overhead was measured for each failure during the period from when the system was initialized to when it was stopped.

Results on failure overhead are illustrated in Fig. 9. By enforcing our distinguishing multicast tree rules, ZIGZAG's



(a)



(b)

Fig. 10. ZIGZAG vs. NICE: Peer Stretch and Link Stress

recovery algorithm is more efficient than that of NICE. Indeed, a failure happens to a peer at its highest layer j in NICE requires $j \times$ O(k) peers to reconnect. j can be the highest layer, thus the worst-case overhead is $\Omega(logN)$. According to our theoretical analyses, ZIGZAG 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. 9 that ZIGZAG clearly prevails NICE. 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.

Fig. 10(a) shows the results of average peer stretch. Recall that the peer-stretch metric is defined as the ratio of path-length from the server to the client along the overlay to the direct unicast path. In its join algorithm, ZIGZAG always tries to keep the distance from the server to the joining client small. Meanwhile, in NICE's join algorithm, distance is also considered, but only between the joining client and the joined client. This does not guarantee a reduced distance between the server and the joining client. Consequently, average peer stretch of ZIGZAG is better than that of NICE.

As shown in Fig. 10(b), the average link stress of ZIGZAG is slightly better than that of NICE. This is no way by accident, but is rooted from the degree bound of each scheme. The worst case degree is O($klog_kN$) in NICE, while bounded by O(k^2) in 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 $N \leq$ 2000, 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 ZIGZAG will be sharply better than that in NICE.

IV. 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 (P2P) approach.

In the overlay-router approach [4], [9–13], 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 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 ZIGZAG and the existing P2P techniques below.

[14] introduced the first P2P technique for streaming applications. This early design, however, did not address the stability of the system under network dynamics. [6] 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 [7] employs a multi-description coding method for the media content. In this method, a media signal is encoded 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 [3], [15] 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 [16] using a two-layer hierarchical topology. To better reduce cluster size, whereby reducing the control overhead at a peer, the scheme NICE in [5] 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 subordinates, 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 could become O($log_k N \times log_k N$). ZIGZAG, no worse than NICE in terms of the other metrics, has a worst-case tree height of O($log_k N$) while keeping the bottleneck bounded by a constant. Furthermore, the failure recovery overhead in ZIGZAG 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.

V. Conclusions

We were interested in 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. Our proposed solution called 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 use of a foreign head other than the head of a cluster to forward the content to the other members of that cluster. With this key in mind, our algorithms were developed to achieve the following desirable properties, which were substantiated by both theoretical analyses and simulation studies:

- Short end-to-end delay: The end-to-end delay 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. ZIGZAG keeps 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 a constant number of reconnections and 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.

It is still open to improve the performance. In the current version of ZIGZAG, if a peer X has to forward the content to non-head members of a foreign cluster, a star topology is used. (i.e., an individual link is created from X to each such member.) In our future work, we can improve this by organizing X and those members in a non-star tree to better reduce the degree at X. Furthermore, we are interested in extending ZIGZAG for dealing with peer heterogeneity.

Acknowledgment

The authors would like to thank the US National Science Foundation for partially funding the work in this paper under grant ANI-0088026.

References

[1] S. Deering, "Host extension for ip multicasting," *RFC-1112*, August 1989.

[2] B. Quinn and K. Almeroth, "Ip multicast applications: Challenges and solutions," *Internet Engineering Task Force (IETF) Internet Draft*, March 2001.

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

[4] 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.

[5] S. Banerjee, Bobby Bhattacharjee, and C. Kommareddy, "Scalable application layer multicast," in *ACM SIGCOMM*, Pittsburgh, PA, 2002.

[6] H. Deshpande, M. Bawa, and H. Garcia-Molina, "Streaming live media over a peer-to-peer network," in *Submitted for publication*, 2002.

[7] 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.

[8] Ellen W. Zegura, Ken Calvert, and S. Bhattacharjee, "How to model an internetwork," in *IEEE Infocom*, San Francisco, CA, 1996.

[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.

[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.

[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.

[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.

[14] 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.

[15] 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.

[16] S. Jain, R. Mahajan, D. Wetherall, and G. Borriello, "Scalable self-organizing overlays," Tech. Rep., University of Washington, 2000.