Efficient P2P Media Dissemination with Forward Error Correction

by

Phuoc Do

A Thesis submitted to

Oregon State University

University Honors College

in partial fulfillment of the requirements for the degree of

Honors Baccalaureate of Science in Computer Science

(Honors Associate)

Presented June 2, 2006

Commencement June 2006

Honors Baccalaureate of Science in Computer Science Thesis
Title: Efficient P2P Media Dissemination with Forward Error Correction
By Phuoc Do, June 2006

Approved by:

_____

Mentor, Assistant Professor, School of EECS

_____

Committee Member, Jon Herlocker, Assistant Professor, School of EECS

_____

Committee Member, Luca Lucchese, Assistant Professor, School of EECS

_____

Dean, Joe Hendricks, University Honors College

I understand that my thesis will become part of the permanent collection of Oregon State University, University Honors College. My signature below authorizes the release of my thesis to any reader upon request.

_____

Phuoc Do, Author

AN ABSTRACT OF THE THESIS OF

Phuoc Do for the degree of Honors Baccalaureate of Science in Computer Science presented on June 2, 2006. Title: Efficient P2P Media Dissemination with Forward Error Correction.

Abstract approved: _____

Thinh Nguyen

Media application on the internet has become more and more popular as the bandwidth of the network links increase. The bottleneck of the existing media systems is no longer the link bandwidth at user's end, but the server's ability to handle streaming requests. These existing streaming systems do not scale up to a large number of users. Peer-to-Peer (P2P) architecture allows the system to scale up to a large number of users by using the peer to help forward data. However, because the peers in a P2P system participate in the data forwarding process, the performance of the P2P system depends on the reliability and robustness of the peers. We propose a hybrid P2P topology that is efficient in both bandwidth utilization and scalability. This paper describes the construction and the algorithm design of our proposed data dissemination scheme. In order to achieve reliability and robustness requirements of real-time media applications, we implemented Forward Error Correction (FEC) which substantially improves the packet loss rate at the peer's end. The system can be scaled up to a large number of users and is capable of disseminating high-quality media (audio and video) stream.

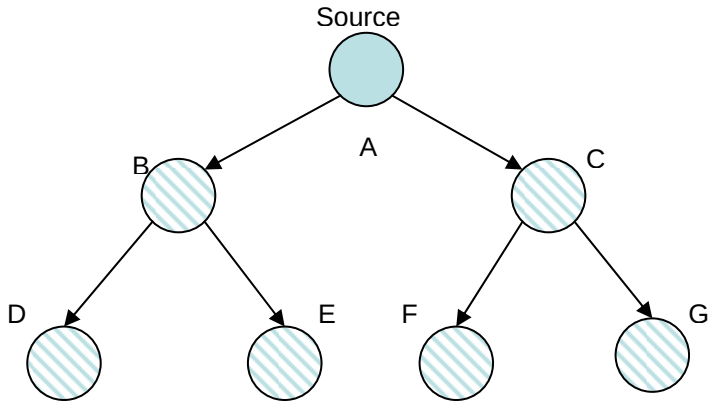
List of Figures

Table of Contents

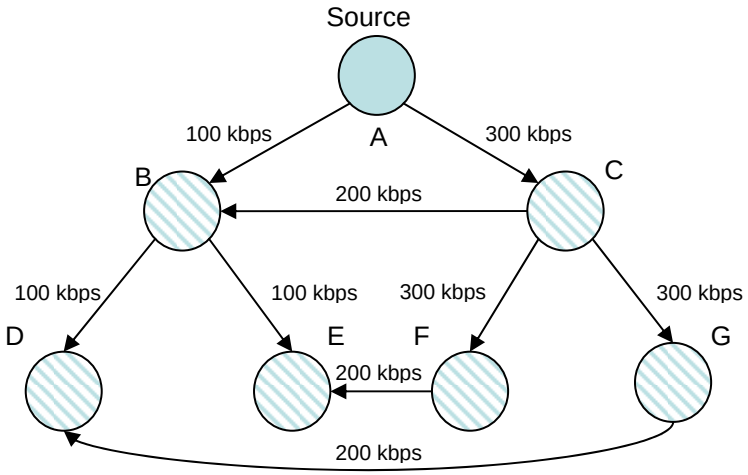
Efficient P2P Media Dissemination with Forward Error Correction

I. Introduction

Broadband internet (DSL or cable modem) has the capability of carrying 256 kbps or more which is approximately four times the speed of a digital telephone modem. Broadband internet usage in the United States grew from 6% in June 2000 to over 30% in 2003 [1]. As broadband internet access becomes more popular, media streaming application on the Internet becomes more feasible. Nevertheless, existing streaming systems face the challenge of scaling up to a large number of simultaneous receivers. IP multicast is a well-known data dissemination scheme on the Internet which attempts to overcome this challenge [2]. The primary motivations of IP multicast are to avoid wasted bandwidth and to scale with the number of receivers. However, IP multicast has compatibility issues among the autonomous systems (AS). This has led to the development of overlay multicast systems [3] where end hosts themselves form a multicast tree as shown in Figure 1 (a). The advantage of overlay multicast is that packet routing and forwarding are done at application layer, which lead to easy deployment across AS(es). Although overlay multicast systems increase bandwidth utilization, they are not optimal because (1) identical packets may travel on the same physical links and (2) the leaf nodes do not contribute their bandwidth to data forwarding.



(a)



(b)

Figure 1: (a) Overlay multicast tree (b) Mesh topology

Let us now consider a mesh topology in Figure 1 (b) where there are additional links between the nodes, particularly, one link from *C* to *B*, one link from *F* to *E*, and one link from *G* to *D*. Assume that the network has the following characteristics:

- Nodes of the left branch have low upload but high download capacity.
- Nodes of the right branch have high upload and high download capacity.
- The link from *A* (source) to *B* has capacity of 100 kbps.
- The link from *A* (source) to *C* has capacity of 300 kbps.

- The link from *C* to *B* has capacity of 200 kbps.
- The link from *B* to *D* has capacity of 100 kbps.
- The link from *B* to *E* has capacity of 100 kbps.
- The link from *C* to *F* has capacity of 300 kbps.
- The link from *C* to *G* has capacity of 300 kbps.
- The link from *F* to *E* has capacity of 200 kbps.
- The link from *G* to *D* has capacity of 200 kbps.

Using this topology, the system is able to disseminate data at a higher bit rate to the left branch of the mesh (300 kbps instead of 100 kbps). We first partition the data into two disjoint sets of packets (1/3 of the packets go to set 1 and the other 2/3 of the packets go to set 2). Packets from set 1 are sent to the left branch and packets from set 2 are sent to the right branch. Set 2 packets will then be forwarded from the right branch (nodes C, F, and G) to the left branch (nodes B, D, and E) through the additional mesh links.

As shown in previous example, although nodes from the left branch have upload capacity of 200 kbps, we can disseminate data at 300 kbps to all nodes. Mesh topology has a potential of utilizing more bandwidth. We would like to construct a network topology and design a data dissemination algorithm to achieve the following criteria:

- The total (upload and download) bandwidth, play-back delay, and out-degree of a node equals to other nodes (fair distribution).
- When a node joins or leaves the network, it should not damage the connectivity of the remaining nodes.

- In order to support real-time media applications, the delay from the source node to any other node needs to be small.
- Throughput efficiency (defined in section II.1) is optimal for all nodes.

In order to maximize bandwidth utilization and maintain fair distribution, all nodes in the network need to have similar bandwidth. The source can send media streaming of equal (or slightly smaller) size to the upload capacity of the nodes. However, in reality, not all receivers have similar bandwidth. We approach this non-homogenous bandwidth problem by first clustering the nodes according to their bandwidth capacities. Given the clusters, we then construct the structured meshes that achieve the above criteria.

The rest of the document is organized as follows. In section II, we describe the mesh topology and data dissemination algorithms that maximize the throughput efficiency and at the same time, maintain reasonable trade-off between delay and out-degree. In section III, we discuss the software implementation of the topology and algorithms described in section II. In section IV, we discuss about the packet loss evaluation and how FEC helps improve packet loss rate. Finally, section V summarizes the work.

II. Efficient P2P Topology

1. Throughput Efficiency

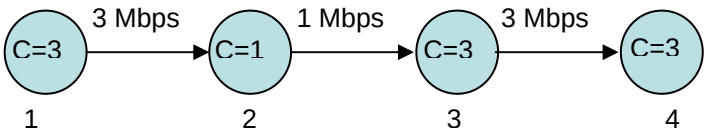
To measure the bandwidth utilization of different dissemination schemes, we define the following throughput efficiency:

Definition: Throughput efficiency is defined as

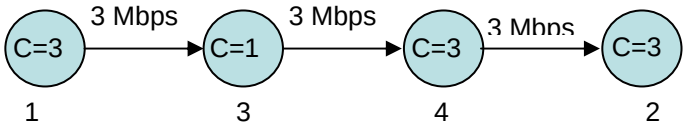
$$E \equiv \frac{\sum_{i=0}^{N} S_i}{\min(\sum_{i=0}^{N} C_i, NC_0)} \quad (1)$$

Where 0 *denotes the source node* $i = 1 \ldots N$ *denote* N *destination nodes,* S_i *and* C_i *are the useful sending rate and the sending capacity of node* i*, respectively.*

The useful sending rate S_i is the total rate at which node i is sending to all its neighboring nodes j's such that this data (received from node i at node j) is completely disjoint with the data received from all other nodes $k \neq i$. We consider a data dissemination scheme (which includes both topology and data dissemination algorithm) is not optimal if it results in duplicate data at a particular node. The numerator in Definition 1 is the total actual sending rate of all nodes, while the denominator is the minimum of the two quantities: (1) total sending capacity of all nodes and (2) the total receiving capacity.



(a)



(b)

Figure 2: (a) Chain topology with throughput efficiency of 0.5

(b) Chain topology with throughput efficiency of 0.9

Figure 2 (a) shows a chain topology with four nodes and their upload capacities C_i (C_4 is 3 Mbps). In this topology the maximum sending rate is 1 Mbps. Although the link from 1 to 2 and the link from 3 to 4 can carry 3 Mbps, the link from 2 to 3 is only capable of sending 1 Mbps. Therefore, the link from 3 to 4 only forwards data at 1 Mbps. The throughput efficiency of this topology is $\frac{3+1+1}{3+3+3+1} = 0.5$. Now, if the system discovers this inefficiency and moves node 2 to the last position in the chain as depicted in Figure 2 (b). The throughput efficiency is improved to $\frac{3+3+3}{3+3+3+1} = 0.9$. It is clearly that high throughput efficiency is desirable because each link is able to send data at its full capacity. We also note that throughput efficiency $E \leq 1$ for any data dissemination scheme [4].

2. Topology Construction

In this section, we examine five different topologies that maximize the throughput and achieve reasonable trade-off in delay and out-degree.

2.1.Fully Connected Topology

Let us consider the fully connected topology (with $N+1$ nodes) where the source node is connected to all N other destination nodes. Each destination node is connected to $N-1$other destination nodes.

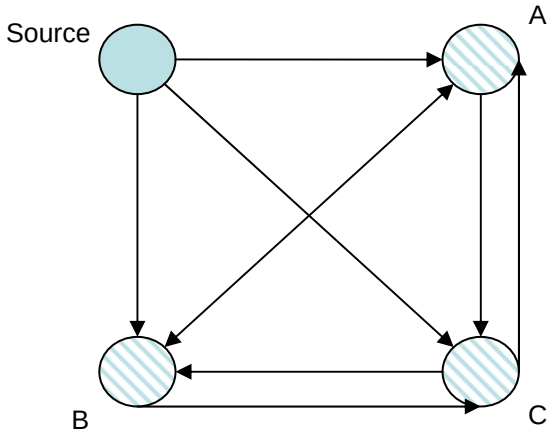


Figure 3: Fully connected topology with $N=4$

Figure 3 shows an example of a fully connected topology with $N=4$. In this topology, one possible algorithm to effectively disseminate data to all nodes is as follows:

1. The source node partitions the data it into N packets.
2. The source node then sends each packet to its neighbors.
3. Each destination node then broadcasts the data it received to all its $N-1$ neighbors.

The throughput efficiency of this data dissemination scheme is $E=1$ [4]. The maximum number of hops that any packet travels from the source to a final destination

node is two. In other words, the maximum delay of any data packet is two hops. However, the out-degree for the source node and any destination node are N and $N-1$, respectively.

2.2.Chain Topology

Now let us consider a chain topology similar to Figure 2 (a) and (b). Assume the links between two consecutive nodes in the chain relay data at its capacity equal to C. Because the last node in the chain is inactive, the total sending rate is NC. The denominator $\min(\sum_{i=0}^{N} C_i, NC_0)$ is also NC. Therefore, the throughput efficiency of this chain topology is $E=1$. If the link capacities are non-homogenous, we can apply clustering technique to divide the nodes into homogenous-bandwidth groups. The chain topology minimizes the out-degree for each node. However, the delay grows linearly with the number of nodes in the chain.

2.3.Balanced Mesh

Fully connected and chain topology are the extreme cases. Fully connected topology minimizes the delay but the nodes have high out-degree. Chain topology minimizes node out-degree but delay will become substantial as the number of nodes in the chain increases. In this section, we proposed the balanced mesh which achieves high throughput efficiency, reasonable trade-off between delay and out-degree. We also assume that the nodes have similar capacities C. Balanced mesh consists of a balanced tree with branching factor b with the source node being the root of the tree. The leaf nodes of the tree are connected in pairs through cross links. There are mesh links from

the leaf nodes connected up to their ancestors. Figure 4 shows an example of a balanced mesh with branching factor $b = 2$ and depth $d = 3$. In this case, the leaf nodes pairs are: 7 and 11, 8 and 12, 9 and 13, 10 and 14. The back links from the leaf nodes to their ancestors are: 7 to 3, 8 to 1, 9 to 4, 11 to 5, 12 to 2, and 13 to 6.

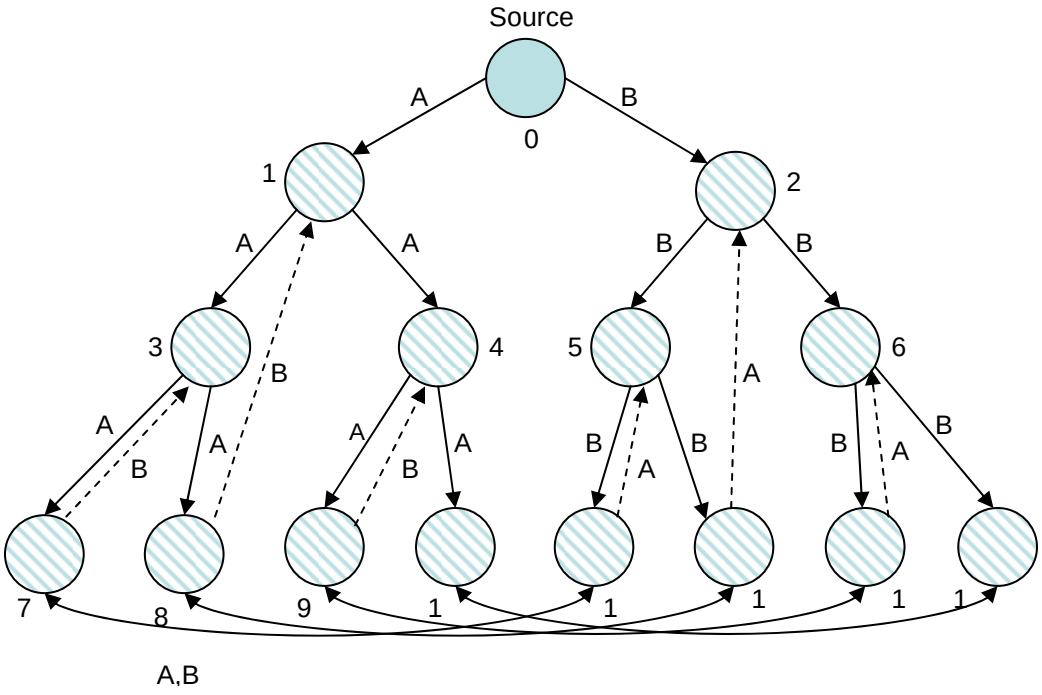


Figure 4: Balanced mesh with $b = 2$ and $d = 3$

Given the balanced mesh of Figure 4, we can disseminate data using the following algorithm:

1. The source node partitions the data into data packets of equal size and assigns the packets with incremental ID and repetitive labels of A and B. Figure 5 shows the packet partitions and assignments of IDs and labels.
2. The source node then sends A-labeled packets to the left branch of the tree and B-labeled packets to the right branch of the tree.
3. Internal node broadcasts the packets it receives from its parent to its children.

4. Leaf node forward the packets it receives *from its parent* through the cross links. For example, when node 7 receives A packets from its parent (node 3), it forwards those packets to node 11 through the cross link between 7 and 11.
5. Leaf nodes forward the packets it receives *from the cross links* to its ancestor through the back link (if any). For example, when node 7 receives B packets from node 11, it forwards those packets to node 3.
6. Destination nodes put the received packets into buffer, and reorder the packets according to their IDs.

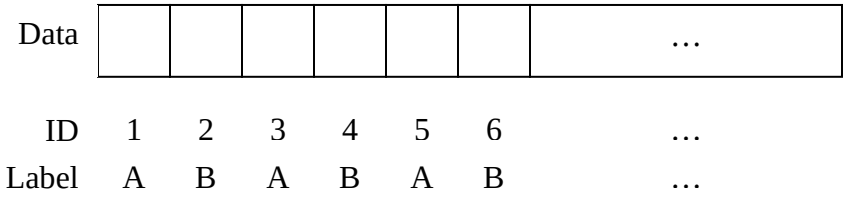


Figure 5: Data Packets with IDs and Labels

We now generalize the description of balanced mesh of branching factor b and depth d. The number of nodes $N+1$ (N denotes the number of destination nodes) in the mesh is:

$$N+1=\frac{b^{d+1}-1}{b-1}$$

(Proof in appendix VI)

Balanced tree has b^d (proof in appendix VI) leaf nodes and b branches. For every leaf node of a particular branch, we create cross links between from it and all leaf nodes of other branches. We also create back links from it to its ancestor. The back links are constructed as follows. If the leaf node is the i^{th} child of its parent where i = 1..b-1, we create a back link from it to its parent. If the leaf node is the b^{th} child of its parent, we

find out which sub tree it is the right most child of and connect it with the root of that sub tree. The data dissemination algorithm is similar to the balanced mesh of branching factor $b = 2$ with two modifications. In step 2, instead of having to send packets with only two different labels (A and B), the source node now sends packets with b different labels. In step5, when leaf node receives a particular cross-link packet, it needs to find out the back link to forward the cross-link packet. The back link and the packet label of the link are determined when the back link is constructed (section III.2.a).

Balanced mesh has the following properties [4]:

1. The **out-degree** for each node is at most b.
2. The maximum node **delay** is $\log_b((b-1)N + b) + 1$ where N is the number of destination nodes.
3. The **throughput efficiency** is $E = 1$.

2.4.Cascaded Balanced Mesh

Although balanced mesh achieves perfect throughput efficiency and good trade-off between out-degree and delay, it strictly requires the number of nodes $N + 1 = (b^{d+1} - 1)/(b-1)$. In this section, we describe cascaded balanced mesh which allows arbitrary number of nodes in the mesh. We construct cascaded balanced mesh by stacking balanced meshes on top of each other.

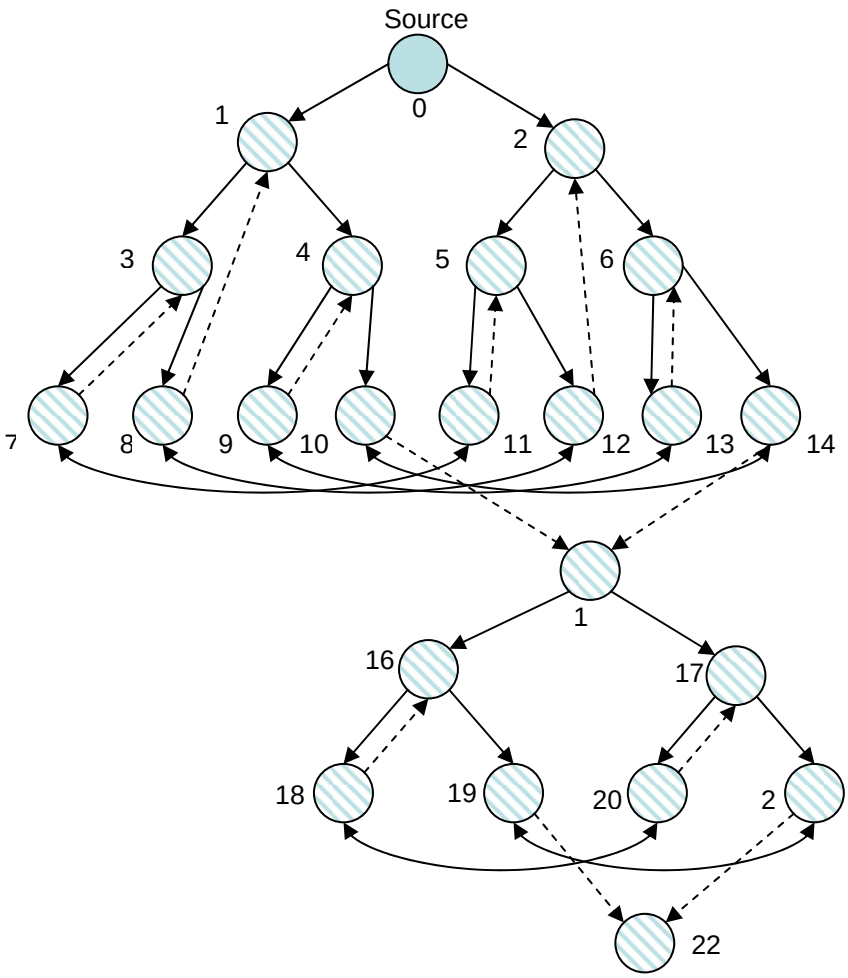


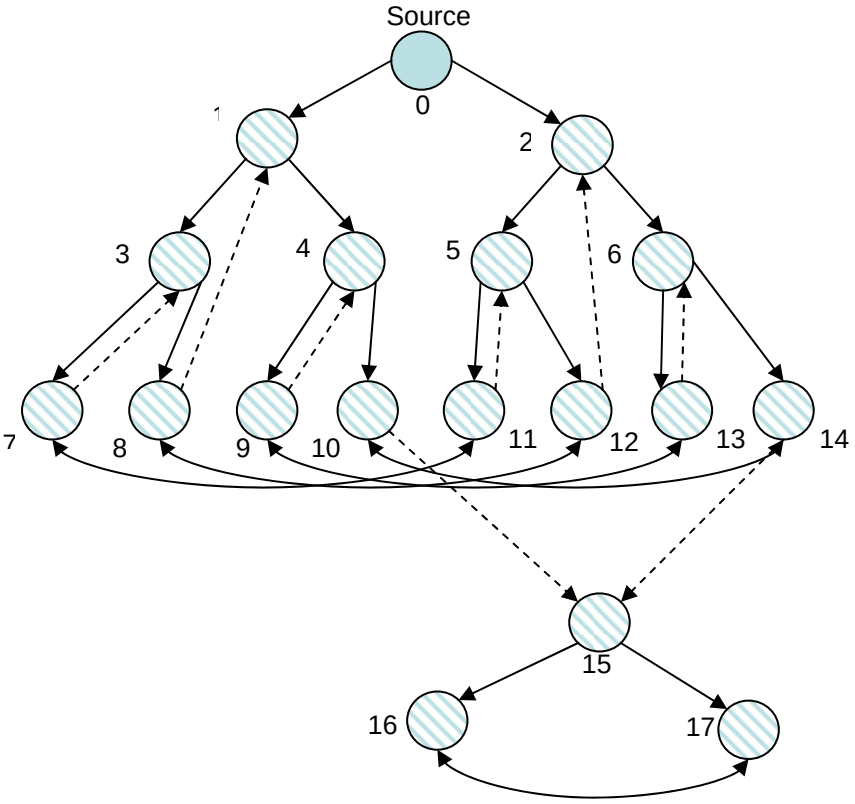
Figure 6: Cascaded balanced mesh with branching factor $b = 2$

Figure 6 shows an example of cascaded balanced mesh with $b = 2$. The mesh is constructed by stacking three balanced mesh on top of each other. The first balanced mesh has 15 nodes (0-14). The second balanced mesh has 7 nodes (15-21). The third balanced mesh has one node (22). The links 10-to-15 and 14-to-15 connect the second mesh with the first mesh. The links 19-to-22 and 21-to-22 connect the third mesh with the second mesh. Data packets are forwarded from the first mesh to the second mesh and from the second mesh to the third mesh through these links. Cascaded balanced mesh has the same throughput efficiency and out-degree as the balanced mesh. However, the delay

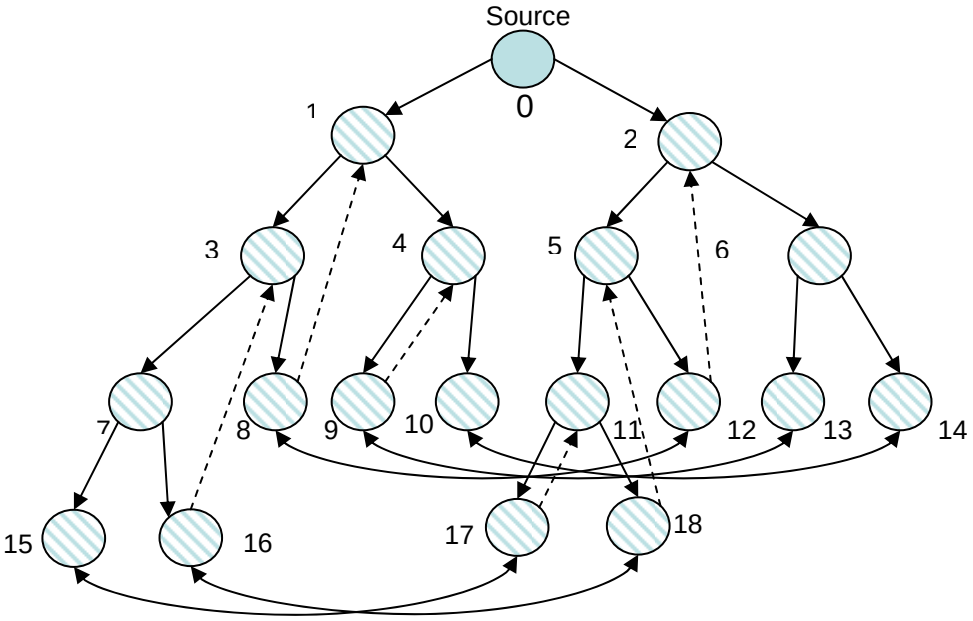
of cascaded balanced mesh is greater because data packets need to travel through all of the balanced meshes. The delay can be shown to be on the order of $O((\log_b N)^2)$ [4].

2.5.*b*-Unbalanced Mesh

In addition to the problem of long packet delay, maintaining the structure of cascaded balanced mesh is also challenging in terms of the number of affected nodes, the complexity of the algorithm, and the number of control messages. A join or leave of node may require a large portion of the mesh to be rebuilt. We propose *b*-Unbalanced mesh which reduces delay and the number of affected nodes. In order to reduce the delay of cascaded balanced mesh, we do not want to have too many balanced meshes. We notice that as the depth of balanced mesh increases, the number of nodes in a layer increases exponentially. For example, consider the balanced mesh with $b = 2$ and $d = 3$ (Figure 2), the number of nodes in the 3rd layer of the mesh is 8, the number of nodes in the 4th layer of the mesh is 16, the number of nodes in the 5th layer of the mesh is 32, and so on. If we can merge the nodes of subsequent balanced mesh to the first balanced mesh, we will be able to reduce the delay. We denote the balanced mesh containing the source node as *primary* mesh and other meshes connected to the primary mesh as *secondary* meshes.


Source
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17

(a)


Source
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18

(b)

Figure 7: (a) The mesh before the deconstruction of the secondary mesh

(b) The mesh after the deconstruction of the secondary mesh

When a new node joins, it will be added to the secondary mesh. Once the number of nodes of the secondary mesh reaches b^2, we break it and merge the nodes with the primary mesh. Figure 7 (a) and (b) depict the *b*-Unbalanced mesh with $b = 2$ in two scenarios. Figure 7 (a) shows the mesh before node 18 is added. At this time, the secondary mesh has three nodes: 15, 16, and 17. When node 18 requests to join the network, because the number of nodes in the secondary mesh is now 4 or b^2, we break the secondary mesh and attach its nodes to the primary mesh. Figure 7 (b) shows the mesh after node 18 is added. If node 19 requests to join, it will be added to the secondary mesh and connected with the primary mesh from node 10 and node 14. When a node leaves, we use the following algorithm to remove the node out of the network without affecting the data flow of other nodes.

If the secondary mesh is not empty, we perform the following steps:

1. Swap the removed node with the last node in the secondary mesh.
2. Remove the last node of the secondary mesh.

If the secondary mesh is empty, we perform the following steps:

1. Break b^2 nodes from the primary mesh
2. Rebuild the secondary mesh using these nodes
3. Swap the removed node with the last node in the secondary mesh
4. Remove the last node of the secondary mesh.

It can be shown that node insertion and deletion of the above algorithms can affect at most $b^2 + 2b$ nodes and the delay is at most $\lfloor \log_b (N+1) \rfloor + 3b - 4$ where N is the number of destination nodes [4].

III. Software Components

In section II, we constructed of different topologies and discussed their properties. The *b*-Unbalanced mesh is better than other topologies because of high throughput efficiency, low delay, and small out-degree. Moreover, it is easier to maintain the structure of the *b*-Unbalanced mesh. In this section, we show our software implementation of the *b*-Unbalanced mesh.

1. MeshManager

The *b*-Unbalanced mesh is wrapped inside MeshManager class. User of the algorithm component can access the *b*-Unbalanced mesh through this class. Figure 8 shows the topology of the *b*-Unbalanced mesh which contains a primary mesh (BalancedMesh) and a secondary mesh (CascadedBMesh).

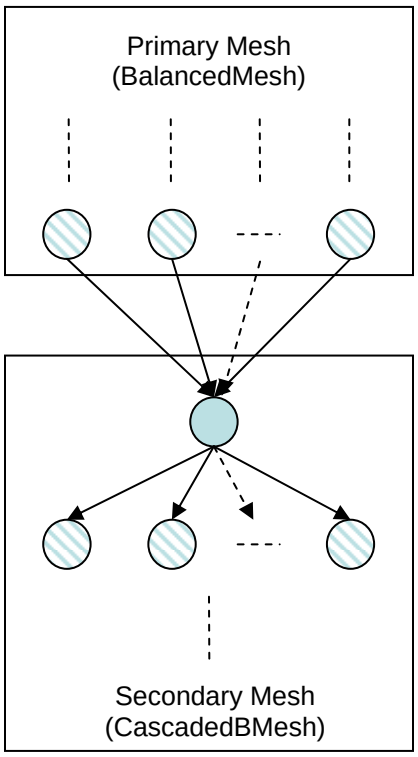


Figure 8: Topology of *b*-Unbalanced mesh

MeshManager allows the following operations:

a. Add a node to the mesh: The algorithm for adding a node is as follows:

If the primary mesh is empty

Make the new node the root of the primary mesh

End if

If the primary mesh is not empty

If the secondary mesh has $b^2 - 1$ nodes

Break the secondary mesh and merge the

nodes (including the new node) to the primary mesh

Else

Add the new node to the secondary mesh

If it is the first node in the secondary mesh

Connect the leaf nodes of the primary mesh to

the new node

End if

End if

End if

b. Remove a node from the mesh: The algorithm for removing a node is as follows:

If the node is in the secondary mesh

If the node is the root of the secondary mesh

Disconnect the links from the primary mesh to it

End if

Remove the node from the secondary mesh

End if

If the node is in the primary mesh

If the secondary mesh is empty

Remove b^2 nodes from the primary mesh

Construct the secondary mesh using these nodes

If the node is in the primary mesh

Swap the node with the last node in the secondary mesh

Remove the last node of the secondary mesh

End if

If the node is in the secondary mesh

Swap the node with the last node in the secondary mesh

Remove the last node of the secondary mesh

End if

End if

If the secondary mesh is not empty

Swap the removed node with the last node of the secondary mesh

Remove the last node of the secondary mesh

End if

End if

c. Check to see if the mesh contains a given node: This operation returns a boolean value indicating the result of the check. Internally, it performs the check on the primary mesh and the secondary mesh.

2. BMesh

BMesh represents the most basic unit of the *b*-Unbalanced mesh. It is used by the BalancedMesh and CascadedMesh. BMesh can be in two different states: LINEAR and SPAN. Figure 8 (a) and (b) depict BMesh in LINEAR and SPAN states, r75 espectively.

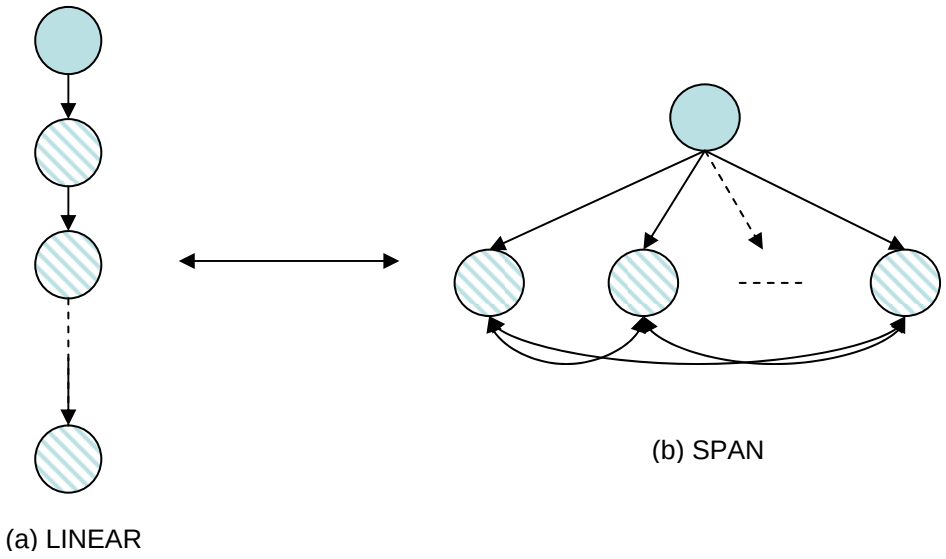


Figure 9: BMesh in (a) LINEAR and (b) SPAN states

BMesh has the following operations:

a. Add a node to the mesh: BMesh allows only $b+1$ nodes to be added into the mesh. This operation returns false if the number of nodes inside the mesh is equal to $b+1$. If the number of nodes in a BMesh is less than $b+1$, the mesh is in LINEAR state. Once the number of nodes in a BMesh reaches $b+1$, the mesh converts itself to SPAN state.

b. Remove a node from the mesh: If the number of nodes drops below $b+1$, the mesh converts itself back to LINEAR state.

c. Check to see if the mesh contains a given node: This operation returns a boolean value indicating result of the check.

d. Break the mesh: This operation returns a list of all nodes inside the mesh. Their links, parents, and children will be deleted. The operation is used when the algorithm needs to break the secondary mesh and merge the nodes to the primary mesh.

e. Establish links from the leaves to a given node: This operation connects the leaf nodes of the mesh to an external node. It is used when the leaf nodes of the BMesh need to forward data to the next BMesh in the cascaded chain.

f. Delete links from the leaves to a given node.

3. BalancedMesh

BalancedMesh represents the primary mesh inside the *b*-Unbalanced mesh. BalanceMesh is restrictive because it allows addition and removal of $b+1$ or b^2 number of nodes. Figure 2 shows an example of a BalancedMesh with $b=2$ and $d=3$.

BalancedMesh has the following operations:

a. Add $b+1$ nodes: This operation can be used only when the mesh is empty. Adding $b+1$ nodes creates the root and sub-roots of the primary tree. This operation also creates cross links between the leaf nodes.

b. Add b^2 nodes: Contrasting with adding $b+1$ nodes, this operation can be used only when the mesh is not empty. It also creates cross links between the leaf nodes, and back links from the leaf nodes to their ancestors.

c. Remove $b+1$ nodes: This operation can be used only when the mesh has $b+1$ nodes. It returns false if the condition does not hold.

d. Remove b^2 nodes: This operation can be used only when the mesh has more than $b+1$ nodes. It returns false if the condition does not hold.

e. Check to see if the mesh contains a given node: This operation returns a boolean value indicating the result of the check.

f. Establish links from the leaf nodes to a given node: This operation connects the leaf nodes of the mesh to an external node. It is needed for connecting the leaf nodes of the primary mesh with the root of the secondary mesh.

g. Delete links from the leaf nodes to a given node.

4. CascadedBMesh

CascadedBMesh represents the secondary mesh of the *b*-Unbalanced mesh. A CascadedBMesh is a series of BMeshes connected together. Primary mesh's leaf nodes are connected to the root of the first BMesh in the series and the leaf nodes of the first BMesh are connected to the root of the second BMesh. The data is transferred from the primary mesh to the secondary mesh through these links. Figure 9 depicts the topology of CascadedBMesh. For visual simplicity, we hide the cross links between the leaf nodes of the internal BMeshes.

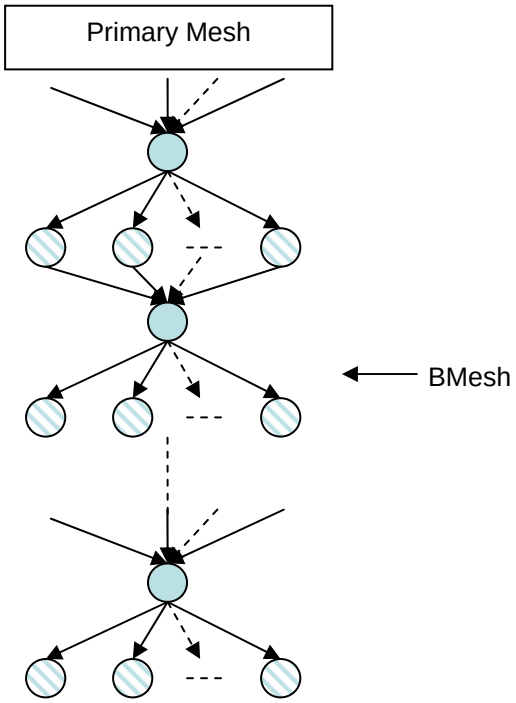


Figure 10: CascadedBMesh

CascadedBMesh has the following operations:

a. Add a node to the mesh: This operation adds the new node to the last BMesh of the chain. If the last BMesh is full (number of nodes = $b+1$), a new BMesh will be created, added to the chain, and the new node is added to the new BMesh. Once the number of nodes in the CascadedBMesh reaches b^2, we break it and merge the nodes to the primary mesh.

b. Remove a node from the tree: In order to prevent data corruption after a node is removed, we need to perform 3 steps. Firstly, we find out whether the node exists in the mesh. Secondly, if the node exists in the mesh, we swap it with the last node of the last BMesh of the chain. Finally, we remove the last node of the last BMesh of the chain.

c. Remove b^2 nodes: This operation is used when the number of nodes in the CascadedBMesh reaches b^2 and we need to merge the secondary mesh to the primary mesh. Internally, this operation breaks up each BMesh in the chain and puts the nodes to a list.

d. Check to see if the mesh contains a given node: This operation returns a boolean value indicating the result of the check. Internally, it goes through the BMesh chain and performs the check on each BMesh.

IV. Packet Loss Evaluation

For real-time application, we use UDP to transmit data from one node to another. Low loss rate is important to ensure quality of service. In peer-to-peer system, loss rate is highly unpredictable because nodes participate into the data forwarding process. An unreliable node will affect data transmission of its immediate nodes. In *b*-Unbalanced mesh, a bad node can potentially affect data transmission of half the nodes. We conduct an experiment on PlanetLab to measure the loss rate of the scheme. The experiment configuration is as follows: 7 nodes, branching factor $b = 2$, and packet size = 500 bytes. Figure 11 shows the network topology of the experiment. We selected the following nodes from PlanetLab:

- se1: planet1.seattle.intel-research.net (source)
- wa1: planetlab01.cs.washington.edu
- wa2: planetlab02.cs.washington.edu
- be1: planet1.berkeley.intel-research.net
- se2: planet2.seattle.intel-research.net
- be2: planet2.berkeley.intel-research.net
- wa3: planetlab03.cs.washington.edu

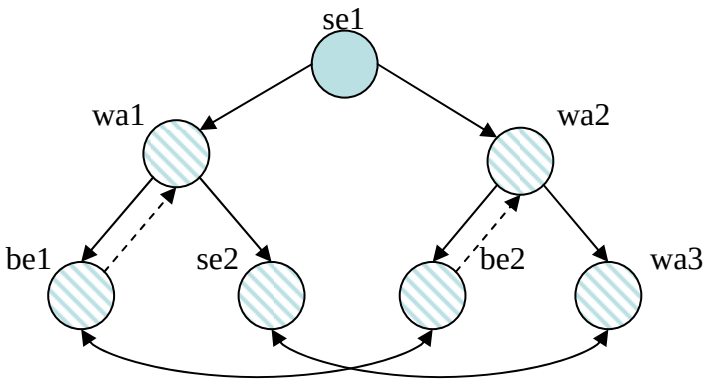


Figure 11: Network topology for PlanetLab experiment

In addition, we implemented forward error correction (FEC) with 30% redundancy. Figure 12 shows how loss rate changes as sending bit rate increases. For every bit rate, we ran two different sessions and measure the loss rate. A session's loss rate will be the average of all of the receiving peers inside the mesh. Loss rate of a particular bit rate will be the average of the two sessions. As shown in Figure 12, loss rate occasionally decreases due to traffic condition of the nodes on PlanetLab. However, the overall loss rate tends to increase as bit rate increases. FEC does help reduce loss rate. At sending rate of 33 Kbytes/sec, loss rate with FEC is about 2.0% while loss rate without FEC is about 7.8%.

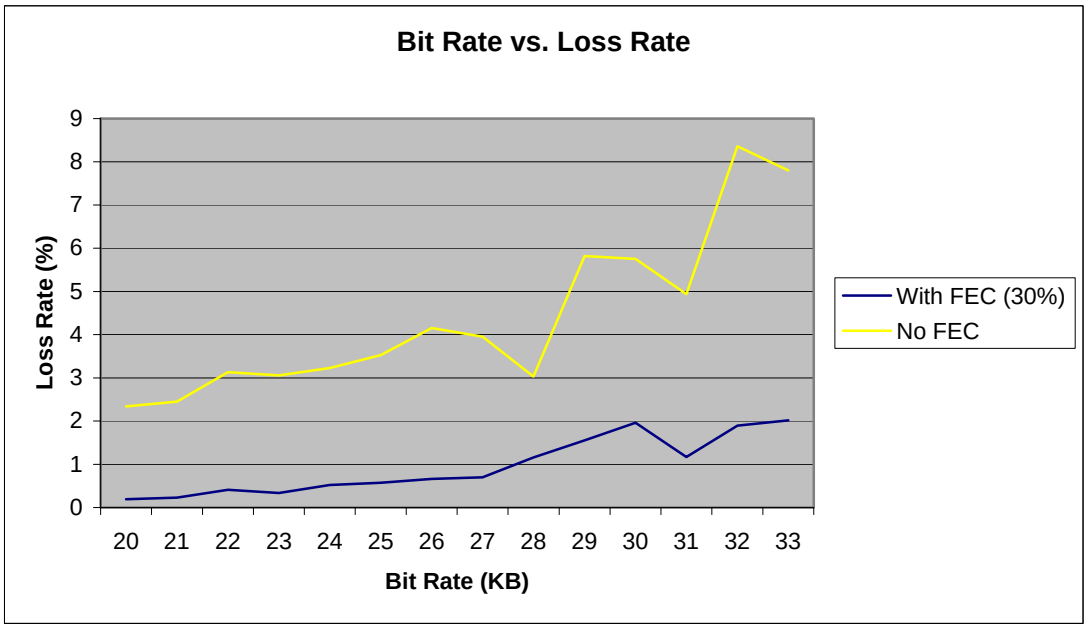


Figure 12: Bit rate vs. loss rate

V. Summary

In summary, in this paper, we visit three aspects of building a scalable and error-resilient media dissemination system. In section II, we examine various network topologies, their pros and cons. In particular, we look at the throughput efficiency, delay, and out-degree of the topology. We conclude that the *b*-Unbalanced mesh is better than other topologies. In section III, we describe our software implementation of the *b*-Unbalanced mesh. In section IV, we discuss the PlanetLab experiment to measure loss rate and how FEC helps reduce the loss rate.

VI. Appendix

1. Proof: the number of nodes at layer (depth) d of BalancedMesh with branching factor b is b^d.

We prove this by induction. Let us consider layer (depth) 0 of BalancedMesh. At this layer we have only the source node or $b^0 = 1$ node.

Now, let us assume that at layer d, we have b^d nodes. We need to prove that at layer $d+1$, we have b^{d+1} nodes. We know each node at layer d has b children at layer $d+1$. Therefore, the number of nodes at layer $d+1$ is $b^d \cdot b = b^{b+1}$.

2. Proof: the number of nodes of BalancedMesh with branching factor b and depth d is $N+1 = \frac{b^{d+1}-1}{b-1}$ where N is the number of destination nodes.

We prove this by induction. In the simplest case, in BalancedMesh with depth $d = 0$, we only have one source node in the tree with no destination node. We have

$$N+1 = \frac{b^{d+1}-1}{b-1} = \frac{b^1-1}{b-1} = 1.$$

Now, let us assume that the BalancedMesh depth of d has $N+1 = \frac{b^{d+1}-1}{b-1}$ nodes.

We need to prove that BalancedMesh of depth $d+1$ has $N+1 = \frac{b^{d+2}-1}{b-1}$ nodes.

From previous proof, we know that at layer (or depth) $d+1$, there are b^{d+1} nodes.

Therefore, the total number of nodes of BalancedMesh of depth $d+1$ is

$$N+1 = \frac{b^{d+1}-1}{b-1} + b^{d+1} = \frac{b^{d+1}-1+b^{d+2}-b^{d+1}}{b-1} = \frac{b^{d+2}-1}{b-1}.$$

Bibliography

[1] Mary Madden, “The changing picture of who’s online and what they do,” Pew Internet & American Life Project, December 2003. http://www.pewinternet.org/pdfs/PIP_Online_Pursuits_Final.PDF

[2] Stephen Deering, Deborah L. Estrin, Dino Farinacci, Van Jacobson, Ching-Gung Liu, and Liming Wei, “The pim architecture for wide-area multicast routing,” *IEEE/ACM Transactions on Networking*, vol. 4, pp. 153-162, April 1996.

[3] Suman Banerjee, Christopher Kommareddy, Koushik Kar, Bobby Bhattacharjee, and Samir Khuller, “Construction of an Efficient Overlay Multicast Infrastructure for Real-time Applications,” in IEEE INFOCOM, 2003.

[4] Thinh Nguyen, Duc Tran, and Sen-ching S. Cheung, “Efficient P2P Data Dissemination Using Structured Meshes,” International Conference on Multimedia Services Access Networks, June 2005.