This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE CCNC 2010 proceedings

A Topology-aware Application Layer Multicast Protocol

Reza Besharati†
Ilam Unit
Islamic Azad University
Ilam, Iran

Mozafar Bag-Mohammadi
Engineering Faculty
Ilam University
Ilam, Iran

Mashallah Abbassi Dezfouli
†Khozestan Sciences & Research Branch
Islamic Azad University
Ahvaz, Iran

***Abstract*—Application Layer Multicast (ALM) is considered as an alternative approach to IP multicast. Topology awareness is an important metric for ALM because it reduces link stress and delay stretch considerably. This work describes a novel, stable and low overhead ALM approach using binning technique to cluster nearby receivers, referred to as Bincast. Bincast uses a constant number of landmarks to cluster nearby nodes. It then constructs a k-ary tree between cluster members. The most stable node is selected as the local header in each cluster. It monitors cluster membership events and decides to split or merge the cluster if necessary. Detailed performance evaluation revealed that Bincast has lower delay stretch than similar methods with approximately same stress.**

Keywords – multicast, ALM, binning, stability.

I. INTRODUCTION

IP multicast needs full support from all network routers in order to work properly. One of major issues with IP multicast is its slow deployment ratio. As an alternative approach, Application Layer Multicast (ALM) does not need any special support from the underlying network [1], [2], [3], [4], [6], [7]. In ALM, the multicast service is completely implemented at end hosts by constructing an overlay over Internet. End hosts replicate multicast data packets and forward them to other hosts based on their positions in the overlay.

ALM suffers from some inherent drawbacks. First, a physical link may transit the same multicast packet more than once. For a physical link, *stress* is defined as the number of identical copies of a same data packet traversing the link. Second, the path from the source to a member on the overlay is usually longer than unicast path between the source and the member. For each member, *delay stretch* is defined as the ratio of the delay along the overlay to the delay of the direct unicast path from the source to the member. Third, ALM overlay is usually unstable due to either frequent failure of end hosts or their graceful departures. Hence, an elaborate failure detection and recovery scheme must be provided alongside the main method to deal with this problem. It should be able to handle multiple simultaneous failures with low control overhead.

In this paper we propose a location-aware and low overhead ALM method called Bincast to efficiently address aforementioned drawbacks. It uses the binning technique to cluster nearby members in a same bin. Each receiver contacts a constant number of landmarks to find its bin. The landmarks are known nodes (such as DNS servers) scattered evenly around Internet. In each bin, a stable host with maximum fan-out is selected as the local header. A host is considered as stable if its age is between t_L and t_H. These parameters could be driven from host churn analysis for some overlays [21]. Fan-out of a host is defined as the maximum number of end hosts it can accept as children in the tree. We use a k-ary tree to connect bin members. The local header is the root of the k-ary tree. The local header may turn into a midway header as a result of the bin split operation. When size of the bin exceeds a predefined threshold, the local header splits the bin into *k* new bins. Previous local header becomes a midway header. Also, *k* new local headers are selected for newly formed bins. Bincast utilizes a simple failure detection and recovery mechanism to cope with host failures. It gathers the required information for failure recovery during the join process of a new member.

Since bin members are in the same vicinity, delay stretch of Bincast is considerably less than other ALM methods. Also, the proximity of bin members reduces link stress as well. The stability of the multicast service established by Bincast is high since local headers are selected among stable members. Therefore, the probability of their failures is likely less than other bin members. Bincast constructs a secondary k-ary tree between the source, local headers and midway headers using Source-Specific Multicast (SSM) model. A clear benefit of this two-tier design is higher scalability. Our simulation confirms that Bincast has low stress, less delay stretch and lower control overhead compared to a similar method [2].

The rest of this paper is organized as follow. Section two introduces related work briefly. We then discuss Bincast tree construction method in section three. The section also contains the bin split method and host failure considerations. Section

978-1-4244-5176-0/10/$26.00 ©2010 IEEE

four presents simulation experiments with random network. Finally, section five concludes the paper.

II. Related Work

Work in [17] categorized ALM methods into mesh-based protocols [1], [7] and tree-based protocols [2], [3], [4]. It then compares them and concludes that a mesh-based protocol is generally robust against host failures but suffers from poor scalability. In contrast, a tree-based protocol is more scalable and has less stress. But, it is more vulnerable against host failures. As a pioneer ALM method, NARADA [1] exploits a richly connected mesh structure between end hosts to construct the multicast tree in application layer. NARADA is robust against either host failures or departures due to additional mesh connections. Each end host maintains a list of the mesh neighbors, which is used in the case of host failures. It is shown that NARADA causes high stress on some network links and suffers from scalability issues [2].

Host multicast Tree Protocol (HMTP) constructs a tree structure between multicast members [3]. HTMP is a scalable protocol with relatively low stress overhead. But, concurrent host failures may partition the tree into multiple subtrees. NICE [2] is a tree-based ALM method, which uses a hierarchical tree structure. Level zero is composed from nearby nodes organized in a cluster. Each cluster has a header, which also participates in upper level clusters. The header is geographically centered node among cluster nodes. The source is in the highest cluster and acts as the root of the hierarchical tree. When a new member joins the group, it contacts a bootstrap node to locate some of multicast members. It then contacts those nodes and goes through series of tree optimizations to find a proper cluster. Since it starts the join process from the source down to leaf members, the order of join process is $m \log n$ in which m is the cluster size and n is the number of multicast members.

Pioneer peer-to-peer (P2P) systems, such as Pastry [11] and Tapestry [10], use topology information in overlay construction and routing. They assume that the topology satisfies the triangle inequality and use it to find a nearby node. However, triangle inequality may not hold in Internet topology as shown in [5]. Topology awareness has been used in P2P overlays such as topologically-aware CAN [12], expressway-based CAN [14] and Brocade [13]. The first scheme uses landmarks ordering to cluster overlay nodes. The second scheme constructs an auxiliary network called expressway based on AS-level topology extracted from BGP reports and a landmark numbering technique. Brocade constructs a secondary overlay on top of a Tapestry network. It considers locality in the routing process at AS level.

Topology Aware Grouping (TAG) [7] uses information about sender path (route) overlap among members to construct ALM overlay. Laptop [15] constructs a hierarchical tree-based overlay network using the geographical layout of hosts. In Laptop, a new member finds its parent by probing a small number of overlay nodes. Laptop uses a hierarchical addressing scheme to correlate the logical distance with the physical distance of nearby nodes [15]. Work [8] proposes a beaconing technique for clustering nearby application peers. More recently, work [9] uses DNS names of end hosts as a proximity measure and constructs ALM overlay accordingly. They assume that two overlay nodes with closely related DNS names are adjacent.

Our work is in close relation with the method proposed in [16]. They use landmarks ordering technique as proposed in [12] to cluster nearby nodes. Each new member must contact a rendezvous point in order to find a proper director at level one based on its landmarks order. It then contacts the director to find a proper level two director. Eventually, it finds and joins an appropriate cluster at lowest level. Number of levels, directors and clusters depends on the number of landmarks. For example, if we use four landmarks, then there are four levels, 40 directors ($P(4,1)+P(4,2)+P(4,3)$) and $P(4,4)$ or 24 clusters. Since number of clusters and identities of their members are restricted by definition, it is possible that a cluster becomes very large. As a result, the cluster is divided and two new clusters are created. The distance vector routing is used between cluster members for multicast data distribution.

Bincast and work in [16] differ in many ways. First, Bincast does not rely on a rendezvous point or a bootstrap mechanism for the overlay construction. Second, Bincast clustering is done based on a finer granularity scoring method than mere landmarks ordering. Also, the landmarks ordering technique restricts the number of available clusters. In order to support large multicast groups, one needs to increase number of landmarks accordingly. In contrast, Bincast dynamically adjusts the number of clusters according to the membership pattern of hosts over time and their location. Third, Bincast has a simple failure detection mechanism to address prevalent host failures. Also, the cluster routing between cluster members is not clearly discussed in [16]. It is not clear that how they use distance vector routing algorithm for data distribution.

III. Bincast

We first describe the binning mechanism. When a new member (m_{new}) decides to join the group, it must ping a predefined number of landmarks with a known order. It then combines the resulted delays using ternary computation to find its score. The score of m_{new}, i.e. S_m, is equal to:

$$S_m = \sum_i D_{li} \times 3^{i-1} \quad (1)$$

In which, D_{li} is RTT of landmark i. For example, if number of landmarks is four and measured RTTs are D_{l1} = 50ms, D_{l2} = 120ms, D_{l3} = 75ms and D_{l4} = 210ms, S_m is 6755. Suppose that the number of initial bins is four and maximum node score is 10000. We assign (0, 2500) range to bin 1, (2501, 5000) to bin 2, (5001, 7500) to bin 3 and (7501, 10000) to bin 4. So, m_{new} belongs to bin 3. When size of a bin exceeds a predefined threshold (t=10), the bin is divided into four new bins. The assigned bin range is also divided accordingly. For example, if bin 3 is divided, the resulted bins have following ranges: (5001, 5625), (5626, 6250), (6251, 6876) and (6876, 7500). It is worth noting that the described scoring function is flexible and interchangeable, i.e., it can be easily replaced by application programmer in efforts to find better clustering techniques.

We construct a k-ary tree between bin members. Each bin has a local header, which also acts as the root of the tree. It maintains a list of the bin members and their scores. The bin split decision is done by the local header. When it decides to split the bin, it becomes a midway header and selects *k* new local headers for the resulted bins. They are directly connected to the new midway header in upper level tree. Therefore, the tree at level one composed from the source, all midway headers and all local headers. As said before, local headers of new bins are selected considering their stability and fan-out.

A. Join Process

When a new member (m_{new}) joins the tree, it contacts landmarks to compute its score, i.e. S_m, according to (1). Then, it sends the score by means of a *Join* message toward the source. The source responds to the message as follow. It first finds a bin that its range contains S_m. If the bin is empty, m_{new} is the first member of the bin and is selected as its local header. Hence, the source sends a *Parent* message toward m_{new} accepting it as a direct child. Otherwise, the source sends a *Build* message to the next header in the found bin. The message contains an empty path list, which is filled up gradually. When an overlay node receives the message, it updates the path list by adding its IP address to the message. Then, if it has an empty slot for m_{new} that matches S_m, it accepts m_{new} as a child and sends a *Parent* message toward it. The message contains a copy of the updated path list. Otherwise, it refers m_{new} to one of its children by forwarding the *Build* message. The child range must contain S_m. Receiving a *Parent* message by m_{new} means that the join process is completed. It then stores the path list.

In Bincast, all non-leaf members must be aware of the last stable host in each downstream subtree. Therefore, when an overlay node forwards a *Build* message, it stores the outgoing overlay branch of the message, m_{new}, arrival time of the message and the path list in a proper data structure. It also sets a timer with value t_L. After expiration of the timer, m_{new} becomes the last stable host for that branch. Therefore, each node knows the last stable hosts in its subtrees and other overlay nodes on the path from the source to itself. This information is useful when a member fails or leaves the tree.

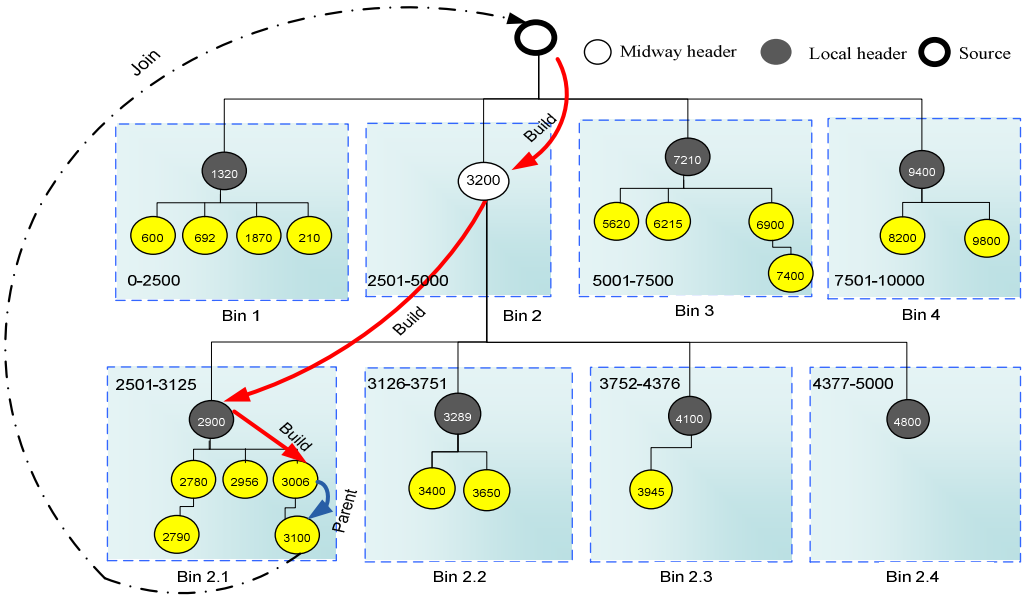

Figure 1. Join process for node with score=3100

Fig. 1 shows an example of the join process. In this figure, each node is identified by its score. The new member has score 3100 and is depicted in down-left corner of Fig. 1. It sends a *Join* message toward source. In response, the source sends a *Build* message toward midway header of bin 2 namely node 3200. This node forwards the *Build* message toward node 2900. Node 2900 is a local header of bin 2.1. It sends a *Build* message toward node 3006. This node accepts node 3100 request and sends a *Parent* message toward it.

The resulted tree is a k-ary tree. The lower bound on the depth of the tree is $\log_k^N$ where *N* is the number of end hosts and *k* is the number of bins in each level. The join process consists of a constant number of ping operations plus searching the tree depth for a parent. If we assume that the number of landmarks is *l*, then order of the join process is $l + \log_k^N$. Each non-header member must maintain a list of its children as well as a list of overlay nodes on the pass from the source to itself. The former is used for tree maintenance while the last is used for multicast data forwarding. Overall size of both lists is at most $k + \log_k^N$. Each local header must keep a list of bin members with their scores and arrival times. Since the bin size is limited by a threshold (t=10), this list is also small. Overall, Bincast scales well with the number of multicast members.

B. Split Operation

As stated earlier, when number of bin members exceeds the defined threshold, i.e. 10, the bin is divided into *k* new bins. Each bin inherits *1/k* of the old bin range. In this case, the local

header sends a *Split* message toward all bin members. When an end host received the *Split* message, it sends a *Join* message toward the local header. The local header has learned the score of all bin members during the join process. It uses the score to place each node in one of newly created bins. Then, it becomes a midway header. Local headers of new bins are selected using previously introduced criteria i.e. stability and fan-out. In Fig. 1, bin 2 is divided into four bins namely 2.1, 2.2, 2.3 and 2.4. Their ranges are also shown in the figure.

C. Member Failure and Host Deparure

Each non-leaf node must periodically update its children with a *Hello* message. Its failure is detected when its children missed three consecutive *Hello* messages. After detection of the failure, the detector sends a *Join* message toward closest upstream header. The header will ask the last stable host in related subtree to take place of the failed node. Clearly, the last stable host is a leaf member. The identities of upstream overlay nodes and the last stable host are learned during the join process as described earlier. When a host gracefully leaves the tree, the host informs its children by sending a *Remove* message toward them. They will then join the closest upstream header as before.

IV. Performance Evaluation

We evaluated the performance of Bincast using *myns* simulator [19]. The network topologies were produced by GT-ITM topology generator [18] based on transit-stub graph model. The average node degree of generated topologies is between 3 and 4. Each topology has 10100 nodes. We ran each experiment 100 times using 10 different topologies (10 run per each topology). End hosts are attached to randomly chosen stub routers. Number of end host varied between 50 and 500. The link delays are randomly assigned to links by GT-ITM varying between 1ms and 100ms.

We compared Bincast with NICE, which is considered as an efficient hierarchical ALM method. For NICE, cluster size varies between k and 3k-1 where k=3. Therefore, each cluster has at most 8 members. For Bincast, we use 8 bins (k=8) per level and at most 10 members per bin (t=10) for fair comparison. Therefore, we construct a k-ary tree with k=8. We assumed that fan-out of all hosts is equal and greater than 8. Also, we have evaluated two other schemes. In first scheme, named "multi-unicast", the source sends a separate unicast flow to each member. For second scheme, we have implemented PIM-SSM as described in [20].

A. Performance Metrics

Delay stretch and link stress are defined per member and link respectively. This will restrict their usage as a numeric metric. Hence, we define following metrics based on them:

- ***Total data delivery of all receivers (D_{total}):*** Suppose that D_i is the data delivery delay for receiver i on the overlay. D_i is computed by adding up link delays for all links on the path between the source and the member. D_{total} is the sum of D_is for all receivers.
- ***Total stress of all links (S_{total}):*** For this metric, we send a multicast data packet toward all members. Then, we count individual link stress for all links and add them up to find S_{total}.
- ***Control Overhead (C_{total}):*** It is the total number of exchanged control packets between overlay nodes to construct and maintain the tree.

B. Simulation Result

Fig.2 shows D_{total} for various group sizes between 50 and 500. As figure suggests, Bincast produces less delay stretch than NICE for all group sizes. In another word, average path delay and length is lower compared to NICE. D_{total} of NICE is 1.45 times D_{total} of Bincast on average. The difference is even higher for larger group size. Fig. 3 compares S_{total} of Bincast and NICE with PIM-SSM. Bincast imposes slightly more stress on network links than NICE. The "multi-unicast" and PIM-SSM have worst and best link stress respectively as expected. For PIM-SSM, link stress is 1 for all links. Hence, S_{total} is the number of the tree links.

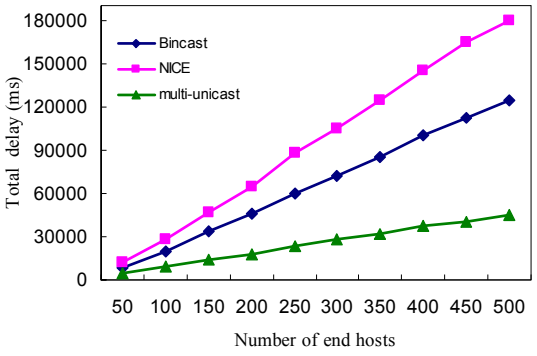

Figure 2. D_{total} for various group sizes.

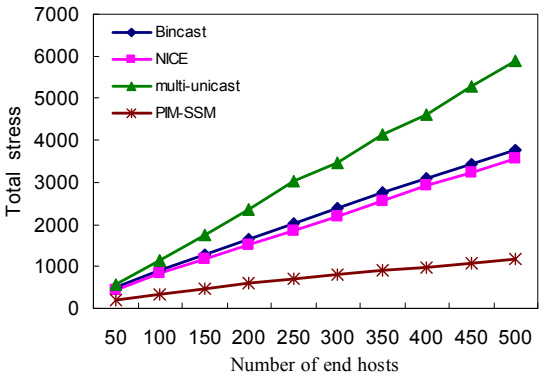

Figure 3. S_{total} for various group sizes.

In Fig. 4, we plotted a cumulative distribution of stress for 250 members. Overall behavior of both protocols is comparable. But, Bincast exhibits a higher value for low link stresses (less than 4). There are two peaks in Bincast plot near 7 and 8 due to using k-ary tree with k=8. These facts mean that Bincast efficiently clustered the members. Total link stress of Bincast is higher, which is also in conformation with Fig. 3. The maximum link stress for Bincast and NICE were 14 and 38 respectively in this experiment.

Now, we compare scalability of NICE and Bincast. Fig. 5 depicts control overhead or C_{total} in logarithmic scale. In this experiment, we set *Hello* messages interval for Bincast so that both protocols have the same failure detection times. Also, we ran simulation long enough allowing NICE to stabilize cluster members. Bincast control overhead is an order of magnitude less than NICE. The resulted saving can be used to accelerate failure detection mechanism using lower *Hello* interval. Also, the difference between the tree construction overhead of NICE and Bincast is more due to several overlay optimizations that take place in NICE.

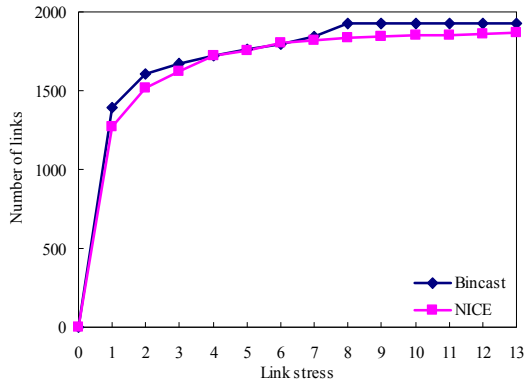

Figure 4. Cumulative distribution of link stress.

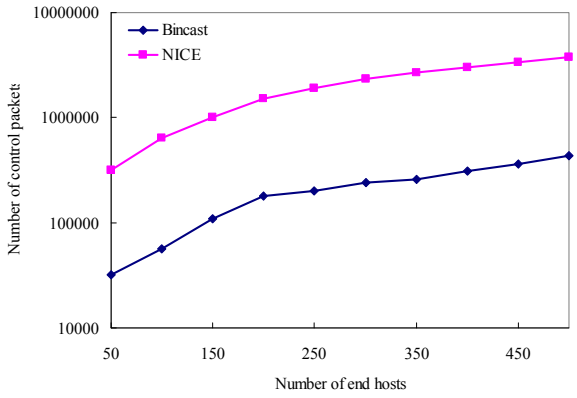

Figure 5. Control overhead.

V. CONCLUSION

Hierarchical tree-based ALM methods are highly scalable [17] due to low control overhead and sensible data distribution performance. Their main disadvantage is low tree stability compared to mesh-based methods. In this paper, we proposed a simple, efficient, robust, scalable and low overhead ALM protocol. Bincast puts nearby nodes in a same bin to take advantage from proximity of bin members. Each bin has a header, which is selected based on node stability metric and fan-out. We construct a k-ary tree between local and midway headers at upper level. A same tree is constructed between bin members at lowest level. This simple hierarchy makes Bincast extremely scalable while topology awareness preserves main performance indices such as delay stretch and link stress. Since the tree at upper levels is composed from highly stable nodes, i.e. local and midway headers, it is more robust against host failures. The control overhead of proposed method is an order of magnitude less than NICE. Also, the delay stretch is reduced by a factor of 1.5 compared to NICE while producing approximately same link stress. As future work, we will consider adding more sophisticated binning methods, a bin merge mechanism and cross-bins data distribution. The last suggestion combined with useful techniques such as network coding is helpful for data recovery in the case of host failures.

REFERENCES

[1] Y-H. Chu, S.G. Rao and H. Zhang, "A Case For End System Multicast" , Proc of ACM SIGMETRICS, June 2000.

[2] S.Banerjee, B.Bhattacharjee and C. Kommareddy, "Scalable Application Layer Multicast", Proc of ACM Sigcomm, August 2002.

[3] B. Zhang, S. Jamin, and L. Zhang, "Host Multicast: A Framework for Delivering Multicast to End User", Proc of IEEE INFOCOM, June 2002.

[4] P. Francis, "Yoid: Extending the Internet Multicast Architecture", ACIRI April, 2002.

[5] S. Savage, A. Collins, E. Hoffman, J. Snell , T. Anderson, "The end-to-end effects of Internet path selection", SIGCOMM Computer Communication Review, vol. 29, No. 4, pp. 289–299, 1999.

[6] M. Castro, P. Druschel, A-m. Kermarree, and A. Rowstron, "SCRIBE: A Large Scale and Decentralized Application Level Multicast Infrastructure", IEEE Journal on Selected Areas in communications (JSAC), Vol. 20, No.8, pp. 1489-1499, 2002.

[7] M. kwon, and S. Fahmy, "Topology-Aware Overlay Network for Group Communication", Proc of NOSSDAV, Miami Florida USA, May 2002.

[8] C. Kommareddy, N. Shankar and B. Bhattacharjee, "Finding Close Friends on the Internet", Proc of ICNP, November 2001.

[9] S. Horng, L. Chun, C. Yang and H-L Hsu, "A DNS-aided Application Layer Multicast protocol", Proc of IAENG International Conference on Communication Systems and Applications , IMECS , Hong Kong, vol. 2, pp. 1076-1082, 19-21 March 2008.

[10] B.Y. Zhao, J. Kubiatowicz and A.D. Joseph, "Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Tech. Rep. UCB/CSD-01-1141, University of California at Berkeley, Computer Science Division, April 2001.

[11] A. Rowstron, and P. Druschel, "Pastry: Scalable, Distributed Object Location and Routing for Large-scale Peer-to-Peer Systems", Proc of IFIP ACM International Conference on Distributed System Platform (Middleware), November 2001.

[12] S. Ratnasamy, M. Handly, R. Karp, and S. Shenker, "Topologically-Aware Overlay Construction and Server Selection", Proc of INFOCOM, June 2002.

[13] B. Y. Zhao, Y. Duan, L. Huang, A. Joseph, and J. Kubiatowicz, " Brocade: Landmark Routing on Overlay Networks,", Proc of International Workshop on Peer-to-Peer Systems (IPTPS'02), 2002.

[14] Z. Xu, M. Mahalingam, M. Karlsson, "Turning Heterogeneity into an Advantage in Overlay Routing," , Proc of IEEE INFOCOM, vol. 2, pp. 1499-1509, San Francisco, CA, 2003.

[15] C. Wu1, D. Liu and R. Hwang, "A location-aware peer-to-peer overlay network", Int. J. Communication Systems, No. 20, pp. 83–102, 2007;

[16] K. Yusung, C. Kilnam, "Scalable and Topologically-aware Application layer Multicast" , IEEE GLOBECOM, Vol. 2, pp. 1266 – 1270, 2004.

[17] S. Banerjee, B. Bhattacharjee, " Comparative study of application layer multicast protocols", available online.

[18] K. Calvert, E. Zegura, and S. Bhattacharjee, " How to Model an Internetwork ", Proc of IEEE Infocom, 1996.

[19] S. Banerjee, myns Simulator. Available from: http://www.cs.umd.edu/~suman/research/myns/index.html.

[20] S. Bhattacharyya et al., "A Framework for Source-Specific IP Multicast Deployment", draft-bhattach-pim-ssm-00.txt, 2000.

[21] D. Stutzbach and R. Rejaie, "Understanding Churn in Peer-to-Peer Networks," Proc of ACM IMC, pp. 189–202, Oct. 2006.