DigiMetro – An Application-Level Multicast System for Multi-party Video Conferencing

Chong Luo, Jiang Li, and Shipeng Li
Microsoft Research Asia

Abstract:

The increasing demand for multi-party video conferencing has aroused the research interest in the underlying multicast support. In this paper, we propose DigiMetro, an application-level multicast system tailored to small and impromptu video conferencing. Breaking through the conventional wisdom to use shared overlay to handle multiple data sources, DigiMetro organizes the data delivery routes as source-specific trees, which are first constructed by a local greedy algorithm and then gradually improved by a global refinement procedure. Extensive simulation experiments demonstrate the efficiency of both algorithms. Moreover, DigiMetro is able to handle different video bit rates and provides with different services over voice/video streams.

I. INTRODUCTION

The Internet has become an essential part of our daily life. Many people are now stay connected through emails and instant messenger (IM) services. However, with the development of multimedia technologies, people can no longer settle for the conventional text conversations. There are increasing demands for real-time multi-party voice/video conferencing. Similar requests also rise from long-distance learning, telemedicine, and global business. They together contribute to the demand for multicast support to such small-scale video conferencing applications.

The most essential characteristic of this type of application is the few-to-few semantic. In contrast to the one-to-many content distribution systems, this type of video conferencing is typically small, with fewer than ten participants, and the membership usually changes rapidly: any member may join, leave or invite other members to the conference at any time. Besides, there are usually as many data sources as the number of conference members. For each data source, at least two types of media streams are involved, voice and video, both of which are highly bandwidth intensive. On the other hand, most Internet users have very limited bandwidth, and the Internet connections are of great diversity, including dial-up, DSL, cable modem and LAN. It is very challenging to serve all these types of users.

The more critical challenge comes from the real-time requirements. As we know, video conferencing is a real-time application involving two-way communication. It has very stringent requirements on end-to-end latency. This is different from the media streaming application, which only has one-way data transmission and allows a few seconds of buffering time at the receiver side.

In this paper, we designed DigiMetro, an application level multicast system that is tailored to the small-scale video conferencing application. The rest of the paper is organized as follows: Section II reviews the existing multicast systems and discusses why the existing techniques do not suffice to support the application we addressed. Section III gives an overview of DigiMetro. The multicast routing algorithms and protocols are described in Section IV. Section V presents results from extensive simulation experiments. We conclude and point out future directions in Section VI.

II. RELATED WORK

The evolution of multicast technology has experienced two stages. The earlier stage is known as IP multicast. The idea was first introduced by Steve Deering [1], who suggested that multicast related functionalities should be implemented at the network layer. However, some inherent architectural problems have impeded the global deployment of IP multicast, such as high complexity, poor scalability and lack of security against malicious attacks. In such a condition, some researchers made a detour by pushing the multicast related functionalities to the application layer.

Research on application level multicast (ALM) usually involves both protocol design and infrastructure planning. The latter is the focus of some early work such as Scattercast [2] and Overcast [3], two ALM systems designed for Internet content distribution. They differ from other ALM systems in that they purposely place a collection of nodes, at strategic locations in the network. This additional cost may be worthwhile for a large content distribution system. However, it is definitely too expensive for small and impromptu video conferences.

The majority of ALM systems are purely built on end systems and do not require extra infrastructure support. They are mostly targeting large-scale applications, involving thousands of end systems. Some of them, such as NICE [4] and SpreadIT [5], take the responsibility of node management as well as building multicast trees for media streaming. Some others employ the existing P2P systems for providing object management and routing functions. Systems in this category include SCRIBE [6], Bayeux [7] and CAN-multicast [8], which are built on top of Pastry, Tapestry, and CAN, respectively. For all the multicast systems listed in this paragraph, the main objective is to reduce the network resource usage and to balance the link stresses of the underlying transmission routes. However, they do not put

much effort in optimizing the end-to-end delay performance, which is very important for a real-time system.

Small-scale ALM systems provide a contrast to the large-scale ALM systems. They normally do not rely on hardware, nor do they have any specific configuration over the infrastructure. Since the group size is small, the group state maintenance can be realized simply by keeping a full member list at a rendezvous point or on every multicast participant. The focus of system design is on the multicast routing. They differ from each other in the following aspects: (1) centralized or distributed control over membership and tree maintenance; (2) mesh-first or tree-first multicast tree construction strategy; (3) share tree or source-specific tree for data delivery.

In End System Multicast [9], end systems self-organize into an overlay structure using a fully distributed protocol named Narada. Narada adopts a mesh-first strategy in constructing multicast trees. It forms a rich connect graph (called a mesh) and then generates source-specific data distribution trees based on the mesh. The disadvantage of Narada is that there is no control over the resulting spanning tree for a given mesh. The concerns come from two aspects: (1) A high quality mesh does not necessarily result in efficient multicast trees for all data sources; (2) Narada does not consider the effect of multiple trees over the same mesh. Thus, in their later work [10] on using Narada for videoconferencing, they assume a single source at any point in time.

Differing from Narada, ALMI [11] is a centralized protocol. Each ALMI session has a session controller which takes all the responsibility of membership registration and multicast tree maintenance. The multicast tree is a shared tree constructed with a tree-first strategy. The session controller periodically re-calculates a new tree based on the end-to-end measurements collected by session members. Although a shared tree is easy to manage, it does not have as good delay properties as source-specific trees. The centralized design also causes two problems: (1) if the controller fails, the multicast tree has to stay unchanged and thus is vulnerable to network changes; (2) During the switch of multicast trees, there will be evident turbulence in performance.

The protocol for multi-sender 3D video conferencing [12] uses a hybrid approach of the above two systems. It adopts a centralized approach similar to ALMI for tree management and uses a mesh-first strategy similar to Narada for tree generation. The novel idea in this protocol is to use a double-algorithm approach for participant joining. If the local algorithm fails to attach a new receiver, the global algorithm will be used to investigate a rearrangement of all trees. However, it still has the following shortcomings that prevent it from practical deployment: (1) it suffers from the one-point-failure problem just as ALMI does; (2) it does not take the dynamic nature of the Internet into consideration and assumes static available bandwidth in calculating multicast trees; (3) The second algorithm re-arranges all trees without considering their original topologies, thus there will be inevitable jitters and long latencies during the switching of the trees.

To sum up, despite the great number of application level multicast systems, none of them are purposely designed or practically employable for video conferencing systems with few-to-few semantics.

III. DIGIMETRO FOR SMALL-SCALE VIDEO CONFERENCING

In this paper, we propose DigiMetro, an application level multicast system tailored to support small-scale multi-party video conferencing.

DigiMetro is a fully distributed protocol. All the members are logically equal: each of them maintains a complete member list and takes full charge of its own multicast tree. DigiMetro makes a clear distinction between the concept of a conference and that of a multicast session. While a conference is made up of a group of members, a multicast session is composed of a single data source and a number of receivers. Thus, in a multi-party conference, there are multiple multicast sessions, since every conference member is a data source.

To deal with the multiple data sources, the conventional wisdom is to use shared overlay as data delivery trees for all the multicast sessions. The management cost is much less than that of using multiple source-specific trees. However, shared overlay does not have as good delay properties as source-specific trees do [13]. In a small-scale real-time conferencing application, we argue that the better delay properties should prevail over the management considerations. Thus, DigiMetro opts for source-specific multicast trees to deliver media data.

In constructing the multicast trees, many factors need to be considered, such as delay, bandwidth, jitter and cost. The problem of finding the best delivery routes subject to multiple metrics has proved to be NP-complete [14]. On the other hand, to use a single metric as the indicator in path selection is obviously not sufficient for supporting multimedia applications. Thus, we designed a two-step tree construction algorithm based on both delay and available bandwidth measurements.

As is well known, the Internet is changing dynamically. The end-to-end delay and the available bandwidth between conference members are also not static. Thus, we need a mechanism to duly measure these two indicators in order to build efficient multicast trees. In DigiMetro, conference members periodically probe each other to obtain the end-to-end delay. The probing requests are also used for member failure detection: if member *A* does not receive a probing message from member *B* over a period of time, it diagnoses *B* as having a network failure. This is also referred to as the heart-beat mechanism. In DigiMetro, the heart-beat

(probing) interval is 5 seconds, and the failure diagnostic threshold is 20 seconds.

As another important metric for multicast routing, the available bandwidth along a delivery path should be examined. We adopt the Packet-Pair [15] method to measure the available bandwidth. In our current implementation, each member probes others in turn with a pair of 1.5 KB UDP packets at an interval of 10 seconds. Thus, in a typical five-member conference, the available bandwidth between every two members is determined every 40 seconds.

For every 5 seconds, each member will broadcast its updated measurements in the conference. The network overhead associated with the three recurrent tasks is estimated in Table 1. We can see that the overall overhead is less than 4 kbps, which is affordable by most Internet users.

TABLE 1 NETWORK OVERHEAD ESTIMATION

Task	Packet size	Frequency	Overhead
Delay measurement	52 bytes	8 pck/5 sec	0.665Kbps
Bandwidth measurement	1500 bytes	2 pck/10 sec	2.4Kbps
Measurements update	84 bytes	4 pck/5 sec	0.538Kbps

In addition to basic multicast support, DigiMetro offers two special features at the QoS (Quality of Service) level.

Accommodate different bit rate: This feature is designed in conformity to the diversity of current Internet connections. The access rates vary greatly from dial-up to broadband to LAN. In order to accommodate users of all types of connections in the same conference, we allow each user to use the affordable voice/video bit rate. DigiMetro is able to handle multiple data delivery trees of different bit rates.

Differentiate service over media streams: In a typical video conference, both voice and video streams are involved. While video enriches a conference, audio is a necessity for efficient communication. DigiMetro offers a different service over different types of streams and the application can assign the priority for each stream type. This is realized by constructing separate multicast trees for different streams and providing a communication mechanism for them to negotiate bandwidth usage.

IV. MULTICAST ROUTING ALGORITHMS AND PROTOCOLS

DigiMetro adopts the tree-first strategy to construct the multicast trees. DigiMetro excels over other ALM systems by using a two-step process in multicast tree maintenance: the source-specific trees are first constructed by a local greedy algorithm and then gradually improved by a global refinement algorithm. Both schemes are based upon delay and bandwidth measurements and share the same goal to optimize the data transmission latency under the given bandwidth constraints.

A. Multicast Tree Construction

DigiMetro is a distributed protocol in which each data source takes care of its own data distribution tree. When a data source *S* receives a *SUBSCRIBE* request from another member *M*, it tries to attach *M* to its multicast tree using one of the following schemes:
Scheme 0: Attach *M* to *S* directly;
Scheme 1: Attach *M* to a selected tree node *P* (inner node or leaf node) and leave all the existing links unchanged;
Scheme 2: M replaces the position of an existing node *C* and adds *C* to its own child set.

The resulting topologies of the three schemes are shown in Fig. 1.

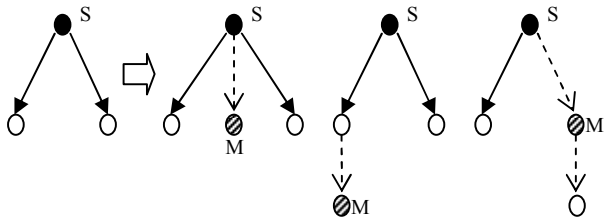


Fig. 1. Possible topologies of a multicast tree after the new subscription request is accepted

To be descriptive, the construction algorithm will attach *M* to the root of the tree if the available bandwidth of *S* allows. Otherwise, it compares the cost of the other two schemes and chooses the one with less cost. Here, the cost of a scheme is defined as the largest transmission latency from *S* to *M* or *M*'s descendants in the multicast tree produced by this scheme.

In practice, we have a preference for scheme 1 over scheme 2, because scheme 1 does not bring any change to the existing transmission routes and results in better stability. Thus, we choose scheme 2 only when its profit gain over scheme 1 exceeds a given threshold. Fig. 1 shows the three possible topologies after M joins the multicast tree rooted at node *S*.

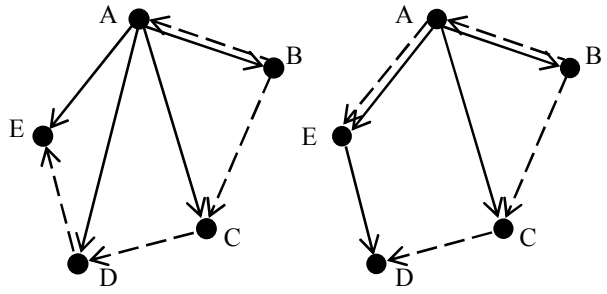
If no member has available bandwidth, the subscription request will be queued in a wait list and the multicast tree will stay unchanged. Remember that we offer different services over different streams and favor voice streams. Thus, when an audio subscription request is not fulfilled, the video tree controller will sacrifice itself by stop feeding one of its subscribers, so that the audio request can be accommodated.

Since the multicast trees are maintained in a distributed manner, it is necessary to define a set of conventions, or so-called protocol, to regulate the behavior of changing routes: when a data source intends to change its multicast tree, it sends a new *CHILDRENSET* to the member *P* whose children set should be updated. On receiving this message, *P* accepts as many children as its available bandwidth allows. If there are some unfulfilled nodes due to the concurrent actions, it returns a *SATURATED* message to *S*, bringing along its current bandwidth capacity as well as the list of unattached nodes, so that *S* is able to rearrange these nodes based on the updated bandwidth information.

Relative to the *SUBSCRIBE* request, if a member *M* does not want to receive a certain stream anymore, it can send an *UNSUBSCRIBE* message to the data source *S*. Then *S* will remove *M* from the multicast tree and rearrange *M*'s children if it has any.

B. Multicast Tree Refinement

The tree construction algorithm is a local greedy algorithm. Each data source tends to use its bandwidth resource to transmit its own data. This results in unbalanced bandwidth usage amongst multicast trees. Fig.2. shows two multicast trees in a five-member conference before and after the refinement process. The real lines represent the multicast tree rooted at member A, while the dashed lines represent the tree rooted at member B. Before refinement (as Fig.2a. shows), member A distributes its data in Unicast way, which uses up its bandwidth resource. Thus, it is unable to relay data for member E in the multicast tree of member B. As a result, the transmission route from B to E (i.e. B→C→D→E) is extremely long.



(a) before refinement (b) after refinement

Fig. 2. An example of the multicast tree adjustment in the refinement algorithm

Moreover, the membership in each multicast session changes rapidly, as does the network dynamics. There is a need to constantly refine the multicast trees to get better performance without bringing perceptible jitters. The refinement scheme calls for cooperation among data sources, so that more subscription requests can be accommodated and the end-to-end latency on the high side can be shortened.

The refinement procedure is a five-step process. It is triggered when there are unsatisfied requests in the waiting list, or there are extremely long transmission routes that are caused by irrational topology. The pseudo code for the first two steps, simplified for clarity, is shown in Fig. 3.

The entire refinement procedure can be described as follows:

Step 1: Data source *S* generates and broadcasts an *ASKHELP* message which is a parent candidate list for *L*. Each node *N* on the list, if it has enough available bandwidth to adopt *L* as a child, can improve the transmission latency from *S* to *L*.

```
Input: data source S and node L whose latency needs to
be improved
Refine (S, L)
{
    for (each node N in S.ReceiverList)
        if (N.Depth () + Delay (N, L) < L.Depth ())
            ParentCandidates.push (N);
    S.BroadcastAskHelp(ParentCandidates,L.Depth());
}

T.ProcessAskHelp (parentCandidate, L.Depth ())
{
    for (each node N in ParentCandidates)
        cost = N.ReArrangeChildren ();
        if (cost < L.Depth ())
            OfferHelpCandidates.push (pair(N,cost));
    T.AnswerHelp (S, OfferHelpCandidates);
}
```

Fig. 3. Multicast tree refinement algorithm (partial)

Step 2: Upon receiving the *ASKHELP* message, other tree controllers *T* will examine whether they can reduce the bandwidth usage of any of the nodes on the list, and then reply with an *ANSWERHELP* message, carrying along the cost of the rearrangement.

Step 3: After node *S* has collected all the answers from other members, it compares the proposed schemes and selects the one that has the minimum cost. *S* will send a *HELP* message to the member *B* that offers the best scheme.

Step 4: *B* checks whether the offer is still valid, if so, it replies with an *ACKHELP* message after it rearranges its distribution tree; if not, it sends a *REFUSEHELP* message indicating that some conditions have changed and it is not able to keep the promise.

Step 5: If an *ACKHELP* message is returned, *S* rearranges *L*. Otherwise *S* aborts the current refinement session and waits for the next chance to improve the performance.

We can see from Fig.2b. that after the refinement procedure, the end-to-end latency on the high side (i.e. the latency along B→C→D→E) is greatly shortened. The new transmission route B→A→E is achieved by balancing the bandwidth resource usage among the multicast trees.

V. EXPERIMENTAL RESULTS

In this section, we present the experimental results from extensive simulation tests to validate the tree management algorithms in DigiMetro.

A. Simulation Setup

In our simulation, we use NS-2 as the event simulator and adopt the transit-stub topology [16] to model the network. In each test run, we randomly pick a few nodes to which the end hosts are attached. We model the last hop uplink

bandwidth for each host and use this value in constructing the multicast trees. This simplification will not affect the validity of our experiment, since the last-hop bandwidth is often the performance bottleneck for broadband users, and should be the most important consideration of applications such as overlay multicast [17].

Reference [17] also indicates that the average uplink bandwidth is 212Kbps. Thus, we model the bandwidths of broadband users as a uniform distribution averaged at 212Kbps. The lower bound is 80Kbps, which is approximately the sum of the voice and video bit rates (13Kbps and 56Kbps respectively). Besides broadband users, we consider LAN users as well. For them, the last hop bandwidth is not the transmission bottleneck, yet we limit their total upload bandwidth to 1Mbps.

There are a few varying parameters that will affect the performance of DigiMetro. They are:

1) Size of the topology (S_T): the larger the size is, the better it simulates the real Internet. In our simulations, we used S_T=100 and 600.

2) Size of the conference (S_C): we fix the number of conference members at 5 except for the scalability test.

3) Percentage of LAN users (P_L): the larger this percentage is the more abundant the bandwidth resources. In our experiments, P_L=50% and 25% are used. \

B. Evaluation Metrics

In order to evaluate the performance of the multicast trees produced by DigiMetro, we compared our multicast routes with that of IP-multicast and naïve Unicast on the following metrics:

1) Relative Delay Penalty (RDP): the ratio of the delay between two members along the multicast tree to the unicast delay between them [9].

2) Rejection Rate: the percentage of rejected media (audio/video) subscription requests by a given routing algorithm [18].

By definition, both IP-multicast and naïve Unicast give the lower bound of RDP at 1.0. In terms of the rejection rate, IP-multicast gives the lower bound at 0% while Unicast reaches the upper bound for source-specific approaches. Other ALM systems also evaluate on relative resource usage and link stress [9]. However, these metrics are not important for a small-scale conferencing application.

C. Experimental Results

1) Tree construction algorithm

The first experiment is conducted on four types of data sets. In each test run, five conference members join the conference at fixed intervals. Every member subscribes to all the other audio/video streams upon joining. After all the members have stabilized, we measure the rejection rate for every data source and the RDP for each source-receiver pair. Table 2 summarizes the results.

TABLE 2
EXPERIMENTAL RESULTS ON TREE CONSTRUCTION ALGORITHM

Data set (TS-S_T-P_L)	Rejection rate			RDP		
	Unicast	DigiMetro		Unicast	DigiMetro	
	A & V	Audio	Video	A & V	Audio	Video
TS-100-50%	0.17	0	0.01	1.0	1.05	1.14
TS-600-50%	0.17	0	0.01	1.0	1.06	1.25
TS-100-25%	0.27	0	0.09	1.0	1.12	1.43
TS-600-25%	0.27	0	0.09	1.0	1.11	1.39

In this table, the first column describes the data set. They are featured by the topology size and LAN user percentage. On each data set, 100 tests are conducted and the average results are listed. We can see that DigiMetro can achieve a very low rejection rate at small RDP, even when the bandwidth resource is limited.

The table also shows that the topology size does not have much influence on the rejection rate and RDP performance. However, the availability of bandwidth resource plays an important role in both evaluation metrics.

We do not compare the performance of our source-specific trees with that of a shared overlay as used in ALMI [11]. The reason is that, on more than half of the data sets, the shared overlay cannot even be constructed because of the great heterogeneity of the last-hop bandwidths.

Besides, DigiMetro discriminates in favor of the audio stream. It guarantees zero rejection rates for audio subscription requests and keeps a low RDP for audio transmission.

2) Refinement algorithm

We did 100 tests on the TS-100-25% data set. After all members have joined the conference, each member randomly unsubscribes from an audio and a video stream. Then, the conference members are allowed to start the refinement procedure. We compared the performance metrics before and after the refinement procedure.

The results show that the refinement procedure can reduce the RDP of the member that has the maximal end-to-end delay to data source by 21.5% and 26.6% for voice and video respectively. The improvement over the maximal absolute end-to-end latency is 6.3% and 7.5% for voice and video respectively.

3) Varied bit rate

DigiMetro is able to handle multiple bit rates in the same conference so as to fit different types of network users. In this experiment, we demonstrate the advantages of accommodating varied bit rate by offering three options for the video bit rate: 28.8Kbps, 56Kbps and 128Kbps. Each member selects the bit rate according to its bandwidth capacity, e.g. LAN users choose 128Kbps; users having less

than 140Kbps bandwidth choose 28.8Kbps and others pick 56Kbps.

We did 100 tests on the TS-100-25% data to compare our varied bit rate scheme with the conventional fixed bit rate (56Kbps) scheme. The rejection rate of video subscription requests is reduced from 9% to 7%, the average RDP of a video distribution tree is reduced from 1.43 to 1.26, and the average video throughput is increased from 208.8Kbps to 265.7Kbps.

The result indicates that by serving different users with different bit rates, we can greatly reduce the average RDP and rejection rate while achieving even larger throughput.

VI. Conclusion and Future Work

In this paper, we introduced the system architecture and protocol design of DigiMetro, an application-level multicast system for multi-party real-time communication applications. We broke through the conventional wisdom to use shared overlay to handle multiple data sources and adopted source-specific trees for each voice/video source. The multicast trees are constructed by a local greedy algorithm followed by a global refinement process.

In view of the heterogeneity of the Internet connections, DigiMetro offers differentiated service over different streams and discriminates in favor of audio data. In addition, DigiMetro is able to handle different bit rates, thus allows low bandwidth users to participate in a video conference with a transmission rate they can afford.

Directions for future work may include auto-adjustment of voice/video bit rate in response to the dynamic changes of network conditions and multicast membership.

References

[1] S. Deering, “Multicast routing in internetworks and extended LANs,” *in Proceedings of the ACM SIGCOMM 88*, pp. 55-64, August 1988.

[2] Y. Chawathe, S. McCanne, and E. Brewer, "An architecture for Internet content distribution as an infrastructure service," Ph.D. Thesis, University of California, Berkeley, December 2000.

[3] J. Jannotti, D.K. Gifford, K.L. Johnson, M.F. Kaashoek, and J.W. O'Toole, Jr. "Overcast: Reliable Multicasting with an Overlay Network," *In Proceedings of the Fourth Symposium on Operating System Design and Implementation (OSDI),* October 2000

[4] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "Scalable application layer multicast," Technical report, UMIACS TR-2002

[5] H. Deshpande, M. Bawa, and H. Garcia-Molina, "Streaming live media over peer-to-peer network," Technical report, Stanford University, 2001

[6] M. Castro, P. Druschel, A.M. Kermarrec, and A. Rowstron, "SCRIBE: A large-scale and decentralized application-level multicast infrastructure," *IEEE Journal on Selected Areas in communications (JSAC),* 2002

[7] S.Q. Zhuang, B.Y. Zhao, A.D. Joseph, R.H. Katz, and J. Kubiatowicz, "Bayeux: an architecture for scalable and fault-tolerant wide-area data dissemination," *In Proc. of NOSSDAV’01,* June 2001

[8] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker, “Application-level multicast using content-addressable networks,” *In Proc. of NGC '01,* London, England, 2001.

[9] Y. Chu, S. Rao, and H. Zhang, "A case for end system multicast," *in Proceedings of ACM SIGMETRICS,* pp. 1-12, June 2000.

[10] Y. Chu, S. G. Rao, S. Seshan, and H. Zhang. “Enabling conferencing applications on the Internet using an overlay multicast architecture,” *In Proc. ACM SIGCOMM 2001*, San Diago, CA, August 2001.

[11] D. Pendarakis, S. Shi, D. Verma and M. Waldvogel, "ALMI: an application level multicast infrastructure," *in Proceedings of the 3rd Usenix Symposium on Internet Technologies & Systems (USITS),* March 2001.

[12] M. Hosseini, N. Georganas, "Design of a multi-sender 3D videoconferencing application over an end system multicast protocol," *in Proc. of ACM Multimedia,* November 2003.

[13] L. Wei and D. Estrin, “The trade-offs of multicast trees and algorithms,” *in Proc. of ICCCN '94,* 1994

[14] Z. Wang and J. Crowcroft, "Bandwidth-delay based routing algorithms," *pp. 2129-2133, GLOBECOM'95,* 1995.

[15] K. Lai, M. Baker, "Measuring link bandwidths using a deterministic model," *ACM SIGCOMM'00,* vol. 30, issue 4, Auguest 2000

[16] E. Zegura, K. Calvert, S. Bhattacharjee, "How to model an internetwork," *in Proc. of INFOCOM'96*, pp594-602, vol.2, March 1996.

[17] K. Lakshminarayanan, V.N. Padmanabhan, "Some findings on the network performance of broadband hosts," *in Proc. of IMC'03,* Oct 2003.

[18] S. Shi and J. Turner, “Routing in overlay multicast networks,” *in Proc. of INFOCOM’02,* vol. 3, pp. 1200-1208, June 2002.