Transience of Peers & Streaming Media*

Mayank Bawa, Hrishikesh Deshpande, Hector Garcia-Molina
Computer Science Department, Stanford University
Stanford, CA 94305
{bawa,hrishi,hector}@cs.stanford.edu

ABSTRACT
Application level multicast schemes have traditionally been evaluated with respect to the *efficiency penalties* incurred in migrating the multicast functionality from the network layer to the application layer. We argue that the current performance measures, and therefore design strategies, are *incomplete* as they do not consider transience of peers. The routers in application level multicast systems are participant clients, and not infrastructure units. As such, the assumptions on the behavior of these application routers are significantly different from the infrastructure routing units that traditional research has dealt with, especially in a peer-to-peer setting where peers are multi-use and the management is decentralized. We argue that the transience in peer behavior has *implications on end-performance enabled*. We outline a design philosophy that seeks to *separate* policy decisions in handling peer behavior from the end-application at a basic infrastructural *peering layer*. As a proof of concept, we have implemented a peering layer prototype, which is available for download.

1. INTRODUCTION
Live streaming media will form a significant fraction of internet traffic in the near future. Recent trade reports indicate that if the current acceptance rate among end-users persists, streaming media could overtake television with respect to the size of the client base [1]. We expect the adoption trends for streaming media to be particularly remarkable in the context of peer-to-peer (P2P) systems.

Since video streams are high bandwidth applications, even a small number of clients receiving the stream by unicast are often sufficient to saturate bandwidth at the source. IP-Multicast [2] was proposed as an extension to Internet architecture to support multiple clients at network level. The deployment of IP Multicast has been slowed by difficult issues related to scalability, and support for higher layer functionality like congestion control and reliability. Several recent research projects [3, 4, 5, 6] have argued for a multicast service at the application layer (*end-host multicast*). The service is to be simulated over unicast-links between hosts, forming an overlay network over end-hosts.

The end-hosts in P2P systems form ad-hoc networks to contribute resources to the community, and in turn use resources provided by other members for a personal goal. Thus, members of a P2P system can be expected to share their bandwidth to spread a media stream to other clients. In principle then, an end-host multicast solution seems to be an ideal fit. Indeed, several such schemes [7, 8, 9] have been developed over the last year.

In practice, we believe however, that the migration of the multicast functionality from the network layer to the application layer leads to a subtle mismatch that has implications on end-application performance. The mismatch arises from a change in the characteristics of the routing infrastructure assumed by the end-application — the behavior of routing elements (clients) in end-host multicast is very different from those (routers) in IP-Multicast.

For example, consider the following scenario in streaming-media delivery via multicast. During an end-hosts multicast session, a large fraction of clients (that were acting as routers) might *unsubscribe together* in the middle of the stream, partitioning the overlay network. The time taken to repair the partitions will result in loss of packets transmitted during the transience period. Such an event, while probable in end-host multicast, is in contrast to network-level multicast where clients only occur as *leaves* in the multicast tree built on routers. Router failures are not as frequent, and the chances of simultaneous failures of a large number of routers, occurring frequently over the stream duration, is extremely small. Hence, while *packet losses* due to router failures is not a probable eventuality in IP-Multicast, it must be accounted for in an evaluation of an end-host multicast proposal. Unfortunately, conventional evaluations of such proposals have glossed over such implications on end-application performance.

It is our thesis that the primary challenge in any P2P architecture will be the *masking* of such peer transience. We will argue that end-applications are not easily insulated from the behavior of the new infrastructure units of a P2P network. A P2P network can be large with hundreds of nodes at

*This research was supported in part by NSF (Grant No. IIS-9817799). Mayank Bawa is supported by a Sequoia Capital Stanford Graduate Fellowship.

any given instant. More significantly, peers are autonomous, unpredictable, and may have short (of the order of minutes) lifetimes. Thus, it is critical for widespread adoption of any P2P application that the service be capable of good performance over a large and dynamic system, and the underlying architecture handle such transience gracefully.

Correspondingly, in this position paper, we will argue the following two points:

- We revisit the issue of evaluating an end-host multicast scheme and urge the use of *end-application metrics* to account for the loss in performance due to the transience of routing units (Section 3).
- We introduce the design philosophy of a *peering layer* that insulates peer transience from end-applications. As a proof of concept, we present an architecture for streaming media over a P2P network that uses the *peering layer* to efficiently enable a connected topology (Sections 4 and 5).

The rest of this position paper outlines the end-host multicast problem and presents conventional evaluation metrics (Section 2), summarizes current implementation status of the peering layer[1] (Section 6), and relates this work to other projects (Section 7). We conclude in Section 8.

2. PROBLEM AND PRELIMINARIES

In this section, we give an overview of a simple end-host multicast architecture to establish the vocabulary, and then point out conventional schemes used to evaluate end-host multicast proposals.

A media stream is a time ordered sequence of packets that is logically composed of two channels: *data* (served using unreliable RTP/UDP) and *control* (sent using reliable RTSP/TCP) channels. A *live* stream has the important property of being *history-agnostic*: a group-member is only interested in the stream from the instant of its subscription onwards.

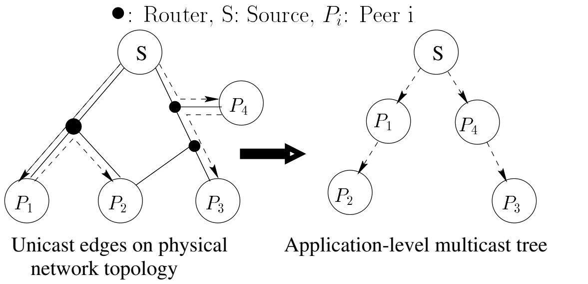

Figure 1: An application level multicast tree built on the peers

A simple architecture for an end-host multicast service organizes the group members into a source-specific spanning tree as shown in Figure 1. The figure uses dotted lines to show the flow of data packets, while the solid lines represent the underlying physical wires. Each node P_i (including the source S) forwards the stream to all its immediate children (if any). Effectively, each host acts as a multicast router, replicating and forwarding packets, and assuming responsibility for group management. The multicast tree is maintained incrementally as nodes join and leave. The scheme specifies protocols to enable an incoming client P_i to identify an existing node in the tree that can act as P_i's parent (*join policy*), and to enable a repair of partitions in the tree when a client opts out of its multicast session (*leave policy*).

The bandwidth requirements of S are now shared by the network of clients, providing a potentially scalable solution for media broadcast. However, there is a loss in efficiency entailed by moving the multicast functionality up the stack. For example, observe that the multicast tree is an overlay network — the actual forwarding paths in the physical network might visit the same network router multiple times. Thus, in Figure 1, the router on the path from S to P_1 will see each packet twice: once when S sends the packet to P_1, and next when P_1 forwards the packet to P_2. Moreover, routing packets along the overlay network results in increased delay from the source S to a client P_i.

Stemming from the influential work of [3] that identified such efficiency concerns, subsequent proposals have been evaluated with respect to the metrics of *stress, relative delay penalty,* and *normalized resource usage*. The *stress* of a physical link is defined as the number of identical copies of a packet carried by the link. The ratio of the delay between the source and a recipient along the overlay to the unicast delay between them is defined as the *relative delay penalty*. Finally, *resource usage* characterizes the network resources consumed in the process of data delivery to all recipients. Correspondingly, it is defined as $\sum_{i=1}^{L} d_i * s_i$, where L is the number of links active in multicast, d_i is the delay of link i, and s_i is the stress of link i. Observe that these metrics succinctly capture concerns on the loss in efficiency. However, note also that it is *not* possible to predict the effects of transience on end-application performance using the above efficiency metrics.

3. PERFORMANCE ANALYSIS FOR END-SYSTEM MULTICAST

In this section, we define a new set of *end-system metrics* to characterize performance of an end-host multicast solution. We then present a performance analysis scheme to enable the study of effects of router-transience on such metrics under a candidate solution.

An end-performance evaluation must, fundamentally, be influenced by the application semantics. A streaming media application exhibits the following characteristics that can stress the candidate architecture:

- *Sensitivity to data loss* that requires minimal packet loss while accommodating changes in membership, and
- *Timeliness constraints* that require changes in membership to be accommodated quickly.

Correspondingly, our *end-system metrics* to characterize performance for streaming media are *packet-losses*, *median lost-block widths*, *relative delay-penalty*, and *time to first packet (response time)* defined as follows:

[1] A skeletal implementation of the peering layer is available for download from [11].

- Let N_i be the total number of packets streamed by S during a client P_i's lifetime. Let N_{li} be the number of packets that were streamed by S during P_i's lifetime, but not received by P_i. Then, packet losses observed at P_i is defined as N_{li}/N_i.
- The disruptions in the stream become visible to the end-user when packet losses at P_i are consecutive, occurring as blocks of lost packets. Hence, it is important to record the sizes of lost packet-blocks at each client P_i. The median lost-block width at a client P_i, defined as the median of all lost packet-blocks observed over the lifetime of P_i, accounts for such disruptions.
- The response-time for a client P_i is defined as the time taken to receive the first stream packet at P_i after a subscribe request was sent by P_i.
- The relative delay-penalty at a client is defined as before (Section 2).

Given the above set of performance metrics, we next outline an analysis scheme. We propose that an evaluation of a multicast proposal must answer the following questions:

- What are the effects of the join policies on the end-system metrics? What are the parameter ranges over which the join policies can scale?
- What are the effects of the leave policies on the shape of the multicast topology? Are the leave policies successful in keeping the topology connected, compact, and stable?
- Which combinations of join and leave policies perform optimally with respect to the packet loss metrics?
- How do the policies compare against a centralized unicast scheme on the response time metric of time to first packet? Does the architecture scale gracefully with increase in the size of the client-base?
- Live streaming media often gives rise to flash crowds at the source, as the clients attempt to subscribe within a short interval of the start of the event. How does the architecture respond to such flash crowds?
- The clients receiving a stream have unpredictable lifetimes. How does the architecture behave under a range of expected lifetimes of the clients? Does the topology remain stable and connected for small lifetimes of clients?
- The clients may have a range of bandwidth capacities, from T3 to symmetric DSL. What is the effect of available uplink bandwidth capacity at a node on the end-system metrics?

A third and final piece remains to be specified — the behavior model of peers in a P2P network. We believe such modeling will be the source of interesting future work. Several P2P systems enjoy considerable deployment and can be used to collect behavioral traces. With these three pieces in hand, an end-host multicast proposal can be evaluated either through analysis or simulations.

To summarize, we believe that the efficiency metrics discussed in Section 2, while essential, were designed to characterize the effects of a functionality migration on the lower (network) layer alone. Unfortunately, it is not only the lower layers that have to bear a penalty for such a migration. Current end-applications are not designed to handle such unpredictability in the behavior of the infrastructure units, resulting in degradation of end-performance returned. We believe that an evaluation of a candidate multicast scheme under the analysis scheme proposed above will enable a more realistic and accurate test of feasibility.

4. THE PEERING LAYER

As we argued in Section 1, *masking* peer transience is the primary challenge in any P2P architecture that supports an end application. In this section, we describe the *peering layer* design philosophy that serves to hide topology changes from the above layers.

The peering layer is envisaged as a basic P2P infrastructure layer that exists below the end application. The peering layer is a placeholder for policies that govern topology maintainance — insertion and deletion of peers. By layering the end service on top of a core transience management service in the peering layer, we believe P2P systems will gain in ease of development.

As a proof of concept that the peering layer functionality is useful in the development of P2P services, we have designed a simple tree-based multicast architecture layered on the peering layer. The solution, called *PeerCast*, uses the peering layers at different nodes to establish and maintain a multicast tree. The PeerCast solution can work with short-lived nodes. Importantly, as we show elsewhere[10], by separating transience management from end applications, we were able to *enable existing implementations* of media players at a peer to use the peering layer. Thus peers were conveniently enabled for stream distribution *without* changing their existing code bases.

4.1 The Design Philosophy

We start by motivating the existence of a peering layer as a practical utility. Notice that a media stream is logically composed of two channels: data and control. An instance of the data and control channels taken together, between a server and a client constitutes a *data-transfer session*. Such a session is distinguished from an *application session* which begins when a client subscribes to a stream from a source, and ends when the client unsubscribes. An application session can subsume several data-transfer sessions as the client changes servers from which it receives feed for a given stream.

However, current implementations of most applications assume a unicast service, and bind the two sessions irrevocably together. We introduce a basic P2P infrastructure layer between the application and the transport layers for streaming that breaks the coupling between the two sessions. All communication between application and transport layers passes through the peering layer, as shown in Figure 2.

Data transfer sessions are established between the peering layers at the two nodes. The end-points are identified by the tuple ⟨ *Server IP-address, Server port, Client IP-address, Client port, Stream URL* ⟩. The first four components identify the transport end-points involved in the session. The last component serves to identify the specific stream data is intended for.

Application sessions are established between the peering layer and the end application. The peering layer allows the appli-

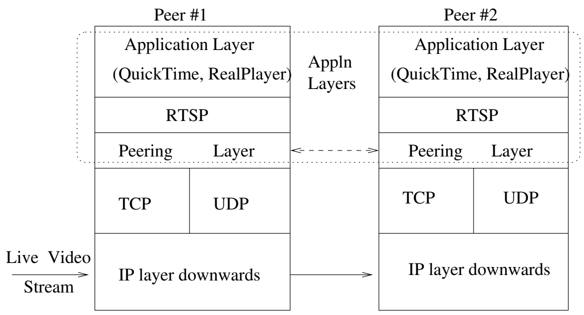

Figure 2: A layered architecture of a peer

cation layer above to specify the stream to obtain, through a "get-stream" interface. Given a stream URL, the peering layer locates a server which can provide the stream, and establishes a data-transfer session with the server. In the event of a data-transfer session termination, it locates a new server for data feed, and restarts the flow of data. If an alternate server cannot be found, an error is flagged to the application above. The application session survives these changes, leaving the end application *unaware* and, indeed, *unconcerned*, of shifts in the underlying multicast topology, as it should be.

The guarantees provided by the peering layer reflect those of the transport layer protocol used in the data-transfer session (for which it has to maintain state). In addition, it guarantees that the data feed to the application above will be maintained as long as an unsaturated server can be found.

4.2 The Redirect Primitive

The peering layer supports a simple, lightweight redirect primitive used to effect changes in the topology. The peering layer uses redirects as hints, to enable discovery of an unsaturated node in the network.

A redirect message is sent by a peer p to another peer c which is either opening a data-transfer session with p, or has a session already open. The message specifies a target peer t. On receipt of the redirect message, c closes its data-transfer session with p, and tries to establish a data-transfer session with t, for the same stream URL. Note that such redirect messages are produced and used at the peering layer, and the application above is unaware of such messages. Thus, the application session persists despite changes in the data-transfer sessions.

4.3 Discovering an Unsaturated Server

A new node n seeking the live stream needs to be able to discover an unsaturated node in the multicast group. The node n contacts the source s of the stream at the known URL. If s is unsaturated, it accepts n as its child and establishes a data-transfer session with n. Otherwise, s redirects n to one of its immediate children c. Then, n attempts to setup a data-transfer session with c. The process continues iteratively, until n gets accommodated. If n is unable to find an unsaturated node within some specified number of tries, the peering layer flags a resource unavailable error to the upper application-layer.

4.4 Managing Unsubscribe of Nodes

If a node n wishes to unsubscribe from the stream, it sends a leave message to its parent p. The parent p frees up resources dedicated to n at its side. However, the descendants of n are now disconnected from s, and experience a break in the stream. To enable a recovery following its unsubscribe, n sends redirects all its immediate children C to some target t (e.g., the source s, or the parent of n). The nodes in C then start the process of finding an unsaturated server by contacting t, as discussed above.

4.5 Handling Failures of Nodes

Intermediate nodes might fail, without being either able to inform their parent, or send redirect messages to their children. The overlay network needs to first detect such a failure, and then recover from it. The peering layer at a node uses a heart-beat mechanism to detect failures, and recovers by following the unsaturated-server discovery process as mentioned in Section 4.3.

5. POLICIES FOR TOPOLOGY MAINTENANCE

The peering layer discussed in Section 4 was said to serve as a placeholder for join and leave policies. Clearly, the nature of policies depend on the unique end application supported. In this section, we discuss simple candidate policies to show that the twin goals of performance and efficiency can be attained simultaneously by decisions *confined* at the peering layer.

5.1 Improving End-Application Performance

As we observed in Section 3, the end-system performance is characterized by packet losses, packet delays, and response times. It can be shown that the shape of the multicast tree has implications on the end-system metrics. In fact, it is easy to derive the following result.

THEOREM 1. *Under homogeneous unicast edge and node characteristics, an almost-complete spanning tree is the optimal overlay tree for packet loss, packet delay, and time to first packet metrics.*

The shape of the multicast tree is determined by the topology maintaining policy. Thus, an optimal policy is one that maintains an almost-complete spanning tree. Such a policy can then be implemented using the mechanisms we discussed in Section 4.

5.2 Improving Overlay Efficiency (or Adapting to Network Dynamics)

The multicast tree that has been formed using the policies discussed in Section 4 does not consider network attributes like link latencies, congestion, or peer bandwidths. Moreover, these quantities are dynamic, and it is important that the topology is rearranged in keeping with dynamic measurements of these quantities. Indeed, a lot of the previous work [3, 4, 5, 6, 12] in end-hosts multicast has focussed on these aspects, and suggested useful heuristics that can be included in the protocol to enable low penalties in efficiency metrics. We indicate below how such heuristics can be used with our tree management policies of Section 4.

Nodes x and y that seek to establish a data-transfer session from x to y can compute a cost function $F(x, y)$ to characterize network proximity, unicast latency, and bandwidth capacity between them [4]. Given such a cost value, new members can join the multicast tree as suggested below.

- *Knock-Downs* : Each node n periodically recomputes $F(n, c)$ for each of its children c. For each incoming peer x that contacts a node n in the tree, $F(n, x)$ is computed. If n is unsaturated, it accepts x as a child. If n is saturated, it compares $F(n, x)$ with the maximum $F(n, c)$ among all of its children c. If $F(n, x)$ is cheaper, then n accepts x as its child, and redirects the child with the most cost. Otherwise, x contacts each of n's children, and tries to connect to the least cost candidate node.
- *Join-Flip*: Each internal node n periodically computes $F(p, n)$ for its parent p. A new incoming client x sends a $F(p, x)$ along with its request to n. If $F(p, x) + F(x, n)$ is less than $F(p, n) + F(n, x)$, n closes its data transfer session with p, redirects x to p, and then itself to x. Such a policy is used in [6] to form efficient trees.

We can also define incremental optimization heuristics that rearrange an *existing* overlay to improve efficiency.

- *Maintain-Flip*: This policy is similar to the *Join-Flip* discussed above. Instead of working with a new incoming client, existing nodes in the overlay tree are swapped. Each node n computes $F(p, n)$, $F(n, c)$ for its parent p and each of its children c periodically. If $F(p, n) + F(n, c)$ is more than $F(p, c) + F(c, n)$ for some child c, the positions of n and c are swapped. The work in [6] provides similar policies to build near-optimal trees. The work in [3, 5] uses such measures as part of their routing protocols to build a spanning tree with low efficiency penalties.
- *Leaf-Sink*: The authors in [13] propose a centralized algorithm to sink unstable or low-bandwidth nodes to the bottom of the tree. They also propose Redundant Virtual Links in the tree, which allow nodes to receive the same packet from multiple different sources, leading to decreased packet losses.

6. STATUS

The PeerCast system as described is under development. A few simple join and leave policies have been designed, implemented, and field tested. A discrete event based simulator has been implemented to evaluate PeerCast as outlined in Section 3. The initial results from field tests and simulations are encouraging. We are in the process of designing and conducting more detailed simulations.

7. RELATED WORK

To place our work in the context of proposed end-host multicast solutions, we cartooned existing proposals as in Figure 3 that illustrates the domain characteristics catered to. The horizontal axis plots the size of multicast group that can be supported. The vertical axis plots the assumed lifetime of hosts over which the overlay network is formed. The solutions indicated in Figure 3 are a representative set of wider work done in the field.

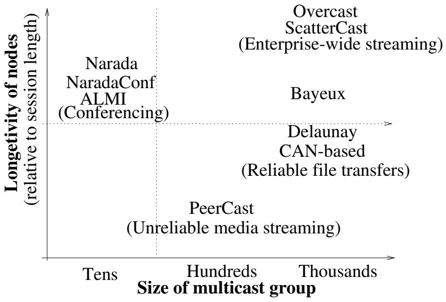

Figure 3: Classifying different end-host multicast proposals.

Solutions in the left portion of the grid [3, 4, 12] have been proposed for multicast over a group of tens of nodes, most of which live through the duration of the session. An example application domain is video conferencing, wherein participants are few (tens of members), and members live through most of the session. The top-right portion of the grid lists proposals [5, 6] that can scale to thousands of nodes, but assume existence of reliable, long-lived infrastructure hosts over which overlays are constructed. As such, these solutions are not adaptable to P2P systems. The middle-right portion lists proposals [7, 8, 9] that rely on primitives proposed in a P2P context to accommodate thousands of clients. These solutions have been proposed and tested in domains in which most of the members are expected to persist through the session. To the best of our knowledge, these proposals have not been proven to scale for hundreds of short-lived nodes forming a *dynamic* group.

The solution we propose, PeerCast, by being aware of the need to manage and mask peer transience from the end-application is able to push down on the longetivity axis in the grid. PeerCast is designed to scale to hundreds of short lifetime nodes that participate in long-durationed multicast session transmitted over unreliable unicast UDP/RTP.

8. CONCLUSIONS

In this paper, we argued that the migration of the multicast functionality from the network layer to the application layer results in a fundamental shift in the underlying infrastructure. The infrastructure units are now participating clients, that are known to be more whimsical and transient that network routers. The observation is especially true for P2P networks, where the participating peers are the providers of an end-service. We urge that the end-host multicast schemes be evaluated on end-system performance metrics of relative delay-penalty, response time, packet losses, and lost-block widths. We indicated that masking peer transience is a primary challenge in such a domain. We introduced the design philosophy of a peering layer at each peer that isolates policies for maintaining the topology from end-application functionality. The applications at a peer now need not be aware of a change in the server providing its data feed.

9. REFERENCES

[1] Shannon Dorey, "Streaming media - will it overtake television?,"

http://www.the-surfs-up.com/news/news4p1.html, 2001.

[2] S. Deering, "Multicast routing in a datagram internetwork," *PhD Thesis, Stanford University, California*, 1991.

[3] Y. Chu, S. Rao, and H. Zhang, "A case for end system multicast," in *Measurement and Modeling of Computer Systems*, 2000, pp. 1–12.

[4] Y. Chu, S. G. Rao, S. Seshan, and H. Zhang, "Enabling conferencing applications on the internet using an overlay multicast architecture," in *Proc. of the ACM SIGCOMM*, 2001.

[5] Y. Chawathe, "Scattercast: An architecture for internet broadcast distribution as an infrastructure service," *PhD Thesis, University of California, Berkeley*, 2000.

[6] J. Jannotti, D. Gifford, K. Johnson, M. Kaashoek, and J. O'Toole, "Overcast: Reliable multicasting with an overlay network," in *Proceedings of the Fourth Symposium on Operating Systems Design and Implementation*, 2000, pp. 197–212.

[7] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker, "Application-level multicast using content-addressable networks," in *Networked Group Communication*, 2001, pp. 14–29.

[8] S. Zhuang, B. Zhao, A. Joseph, R. Katz, and J. Kubiatowicz, "Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination," in *Proceedings of the Eleventh International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV 2001)*, 2001.

[9] J. Liebeherr, M. Nahas, and W. Si, "Application-level multicast with delaunay triangulations," Tech. Rep. CS-2001-26, University of Virginia, CS Dept., 2001.

[10] H. Deshpande, M. Bawa, and H. Garcia-Molina, "Streaming live media over peers," Tech. Rep. 2001-31, CS Dept., Stanford University, 2001.

[11] "http://www-db.stanford.edu/peers/," .

[12] 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 and Systems (USITS)*, 2001, pp. 49–60.

[13] V. Roca and A. El-Sayed, "A host-based multicast (hbm) solution for group communications," in *1st IEEE International Conference on Networking*, 2001.