Magellan: Performance-based, Cooperative Multicast

Stefan Birrer   Fabián E. Bustamante
Northwestern University
Department of Computer Science
Evanston, IL 60201, USA
{sbirrer,fabianb}@cs.northwestern.edu

Abstract

Among the proposed overlay multicast protocols, tree-based systems have proven to be highly scalable and ef£cient in terms of physical link stress and end-to-end latency. Conventional tree-based protocols, however, distribute the forwarding load unevenly among the participating peers. An effective approach for addressing this problem is to stripe the multicast content across a forest of disjoint trees, evenly sharing the forwarding responsibility among participants. DHTs seem to be naturally well suited for the task, as they are able to leverage the inherent properties of their routing model in building such a forest. In heterogeneous environments, though, DHT-based schemes for tree (and forest) construction may yield deep, unbalanced structures with potentially large delivery latencies.

This paper introduces Magellan, a new overlay multicast protocol we have built to explore the tradeoff between fairness and performance in these environments. Magellan builds a data-distribution forest out of multiple performance-centric, balanced trees. *It assigns every peer in the system a primary tree with priority over the peer's resources. The peers' spare resources are then made available to secondary trees. In this manner, Magellan achieves fairness, ensuring that every participating peer contributes resources to the system. By employing a balanced distribution tree with $O(\lg N)$-bounded, end-to-end hop-distance, Magellan also provides high delivery ratio with comparable low latency. Preliminary simulation results show the advantage of this approach.*

1. Introduction

A recent research trend advocates an end-system, or application-level, approach to multicasting as a viable alternative to network-level multicast for a wide range of group communication applications [15, 22, 35, 3, 12, 6]. In this application-layer approach, participating nodes organize themselves into an overlay topology for data delivery. The topology is an overlay in that each edge corresponds to a unicast path between two end systems in the underlying Internet. All multicast related functionality is implemented at end systems instead of at routers, and the goal of the protocol is to construct and maintain an ef£cient overlay for data transmission. There is ample motivation for such an approach, as it delivers the scalability advantages of multicast, in terms of server load and bandwidth communication, while avoiding the deployment issues of a network-level solution [16, 18] .

Among the proposed end system multicast protocols, tree-based schemes have shown to be highly scalable and ef£cient in terms of physical link stress and end-to-end latency [25, 22, 3, 12, 11, 6]. *Conventional tree-based protocols, however, are not well suited for cooperative environments as the forwarding load tends to be unevenly distributed among the participants.* A small fraction of peers, those interior to the tree, are made only responsible for supporting the commonly needed service, while the majority of nodes act as leafs and thus contribute no resources to the forwarding task.

Castro et al. [10] proposed addressing this problem by the striping of multicast content across a forest of interior-node-disjoint trees. By making each peer an interior node in at least one tree, forwarding responsibilities can be evenly shared among the participating nodes. A key challenge of these schemes is to ef£ciently create and maintain this forest, a task for which Distributed Hash Tables (DHTs) seem naturally well suited. By leveraging the inherent properties of the DHT routing model, SplitStream [10] can provide a relatively simple and ef£cient method for creating a forest of interior-node-disjoint trees that neither requires costly network monitoring nor depends on a centralized coordinator. DHT-based schemes for tree and forest construction, however, have been found to result in deep, unbalanced tree structures. Deep structures are undesirable due their increased vulnerability to node failures/departures and potentially large application-performance overhead [10, 5].

This paper introduces Magellan, a new overlay multicast protocol targeted to cooperative applications with low-latency constraints (such as online gaming and live streaming) over large-scale, heterogenous environments. *Similar to SplitStream [10], Magellan builds a data-distribution forest over which trees it forwards stripes of the multicast content. Unlike SplitStream, Magellan constructs its forest from multiple performance-centric, balanced trees instead of DHT-based ones.* While DHT-based approaches focus on constructing/maintaining a structure based on a virtual ID space, performance-centric approaches consider primarily the application's performance (e.g. latency) when adding a new link to the overlay [5, 15, 3, 6]. With Magellan, every peer in the system is assigned a primary tree with priority over the peer's resources. A peer's spare resources are then allocated to the remaining trees in the forest. We present experimental evaluation results contrasting Magellan's performance-centric and SplitStream's DHT-based [10] approaches to cooperative multicast. To provide a baseline against which to evaluate the performance/fairness tradeoffs in Magellan, we rely on Nemo [6], a conventional performance-centric, tree-based multicast protocol. These preliminary results show that Magellan achieves its goals of striking a balance between fairness and performance.

The remainder of this paper is structured as follows. Section 2 describes the Magellan approach in more detail. We brie¤y review relevant background material in Section 3 and present the design of Magellan in Section 4. Section 5 discusses related efforts. We present simulation-based results in Section 6 and conclude in Section 7.

2. The Magellan Approach

Magellan addresses the needs of cooperative, group-communication applications in large-scale, heterogeneous environments by forwarding the stripped multicast content over a forest of performance-centric trees. Example applications include large group teleconferencing [24], large-scale virtual environments (e.g. Quake Arena) [30], and active spectator support for on-line, multiplayer games [4]. In all these examples, the need for humans to interact in realistic and natural ways within the virtual setting places bounds on the performance of the underlying group communication system.

In cooperative settings, nodes are expected to contribute resources in exchange for using a commonly needed service. Conventional tree multicast are not well suited to this model, as the load of duplicating and forwarding messages is unevenly distributed among the participating peers. To ensure a fairer load distribution, Magellan adopts a SplitStream-like solution [10], forwarding the stripped, multicast content over a forest of interweaved trees. Unlike SplitStream, Magellan follows a performance-centric instead of a DHT-based approach to overlay multicast [25, 14, 3, 6]. DHT-based tree and forest construction schemes have been found to yield deep, unbalanced tree structures in highly heterogenous and dynamic environments as those targeted by Magellan. [5]. Trees with high depths can have a signi£cant impact in application performance which, in this context, is mainly determined by the frequency of interruptions due to node failures/departures and/or congestion on an upstream link, which in turns depends on the number of ancestors a node has [10, 5].

2.1. Magellan

Magellan ensures that every participating peer contribute resources to at least one tree in the forest and that all trees have a set of assigned peers to serve as their interior nodes. The set of interior nodes to a tree is made of **primary peers**, i.e. peers for which the tree is their **primary tree**, and additional **secondary peers**.

Figure 1 illustrates the logical organization of Magellan with two trees. Each peer primarily serves in either the black or white tree (as indicated by the peer's color). In this simple scenario, all peers are able to contribute resources to the system.

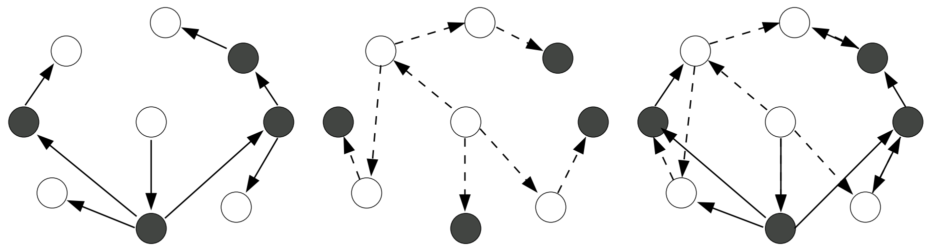
If a tree's set of primary peers does not collectively have the required resources to support the tree's stripe, e.g. due to peers with low bandwidth capacity, Magellan assigns peers with spare resources as needed. Thus, Magellan guarantees that no tree will run out of forwarding capacity before the full system is saturated while supporting the participation of non-contributors in the system. [1] By relying on balanced multicast trees, Magellan reduces the total end-to-end hop distance in the distribution topology, thus lessening the tree vulnerability to node failures and minimizing performance overhead.

For detecting peer failures/departures and repairing the topology, Magellan relies on an ef£cient, per-tree maintenance protocol. In addition to the frequency of interruptions a node experiences, the second factor determining application performance is the ef£ciency of the detection/repair protocol. All multicast messages in Magellan are uniquely identi£ed and lost messages are recovered via lateral error recovery, i.e. recovered from any of the trees, not only the forwarding one [39].

3. Background

A variety of overlay multicast protocols have been proposed covering a range of targets level of scalability, applications and primary performance metrics [14, 35, 25,

[1] Dealing with free-riders, however, is outside the scope of this work.



(a) The 'black' tree. (b) The 'white' tree. (c) Both trees overlayed.

Figure 1. Magellan with two stripes and one publisher in the middle of the network. The peers are colored according to their primary tree. Every peer serves as interior node in its primary tree and as leaf in the other one.

3, 6, 7, 36, 42, 22, 12]. Among the proposed schemes, performance-centric protocols [15, 25, 3, 33, 6, 7] have been shown to deliver excellent performance, while ef£ciently utilizing the underlying network infrastructure. Although clearly realizable with any performance-centric, tree multicast protocol, this paper describes and evaluates the design and implementation of Magellan based on Nemo [6]. Thus, before delving into Magellan's details, this section provides a brief overview of Nemo. For a complete description the reader is referred to [6].

In Nemo [6], participating peers are organized into clusters based on network proximity, with every peer being a member of a cluster at the lowest layer. [2] Each of these clusters selects a *leader* that becomes a member of the immediately higher layer. In part to avoid dependency on a single node, every cluster leader recruits a number of *co-leaders* to form its *crew*. The process is repeated, with all peers in a layer being grouped into clusters, crew members selected, and leaders promoted to participate in the next-higher layer. Thus peers can lead more than one cluster in successive layers of this logical hierarchy. Co-leaders improve the resilience of the multicast group by avoiding dependencies on single nodes and providing alternative paths for data forwarding. Figure 2 illustrates the logical organization of Nemo.

Beyond enhancing the overlay's resilience, crew members share the load of message forwarding, as messages routed to a cluster are directed to a randomly selected crew member. The receiving crew member is then responsible for forwarding the packet to all other members in the target cluster. In this manner, Nemo reduces the outdegree of cluster leaders (by comparison with protocols that do not rely on co-leaders, such as Nice [3]) and consequently improves the system scalability.

[2] Clusters vary in size between d and $3d - 1$, where d is a constant known as *degree*. This is common to both Nemo and Nice [3]; the degree bounds have been chosen to help reduce oscillation in clusters.

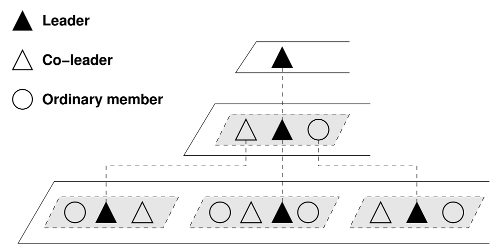


Figure 2. Nemo's logical organization. Shapes illustrates the role of a peer within a cluster: a leader at a given layer can act as co-leader or ordinary member at the next higher layer.

Nemo's data delivery topology is implicitly de£ned by the set of packet-forwarding rules. A peer sends a message to one of the leaders for its layer. Leaders (the leader and its co-leaders) forward any received message to all other peers in their clusters and up to the next higher layer. A node in charge of forwarding a packet to a given cluster must select the destination peer from among all crew members in the cluster's leader group. Figure 3 presents a sketch of the forwarding algorithm. Figure 4 illustrates the algorithm in action using the logical topology in Figure 2. At time t_0 a publisher forwards the packet to its cluster leader, which, in turn, sends it to all cluster members and the leader of the next higher layer (t_1). At time t_2, this leader will forward the packet to all its cluster members, i.e. the members of its lowest layer and the members of the second lowest layer. In the last step, the leader of the cluster on the left forwards

```
FORWARD-DATA(msg)
R ← ∅
if leader ∉ msg.sender_crew
  then R ← R ∪ leader
for each child in children
do if child ∉ msg.sender_crew
     then R ← R ∪ child
SEND(msg, R, sender_crew ← CREW-OF(self))
if IS-CREW-MEMBER(self) and leader ∉ msg.sender_crew
  then R ← ∅
       R ← R ∪ super_leader
       for each neighbor in neighbors
       do R ← R ∪ neighbor
       SEND(msg, R, sender_crew ← CREW-OF(leader))
```

Figure 3. Data Forwarding Algorithm. SEND **transmits a packet to a selected node among a destination's crew members.**

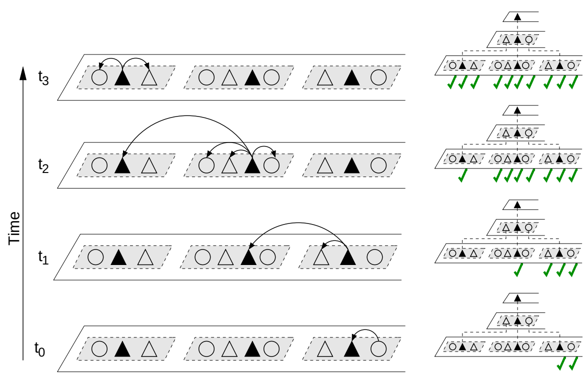


Figure 4. Basic data forwarding in Nemo. Each row corresponds to one time step.

```
CHECK-LEADERSHIP-TRANSFER(cluster, tree)
optimal ← GET-LEADER-FOR(cluster, tree)
if GET-SELF ≠ optimal
  then TRANSFER-LEADERSHIP(optimal, cluster, tree)

CHECK-CLUSTER-SPLIT(cluster, tree)
if SIZE-OF(cluster) > MAX_CLUSTER_SIZE
  then c1, c2 ← SPLIT-CLUSTER(cluster, tree)
       if GET-SELF() in c1
         then % we remain leader of c1
              TRANSFER-LEADERSHIP(
                c2, GET-LEADER(c2, tree), tree)
         else % we remain leader of c2
              TRANSFER-LEADERSHIP(
                c1, GET-LEADER(c1, tree), tree)

CHECK-LEADER-REFINEMENT(leader, candidates, tree)
optimal ← leader
cost ← GET-COST-FOR(leader, tree)
for each peer in candidates
do cost2 ← GET-COST-FOR(peer, tree)
   if cost2 < cost
     then optimal ← peer
          cost ← cost2
if optimal ≠ leader
  then TRANSFER-TO-NEW-LEADER(optimal, leader, tree)
```

Figure 5. A simpli£ed representation of Magellan's maintenance operations. SPLIT-CLUSTER **returns two cluster so that the sum of the two emerging clusters' diameter is approximately minimized.** TRANSFER-LEADERSHIP **noti£es all affected peers about the new leader.** TRANSFER-TO-NEW-LEADER **joins the new leader and noti£es the old one about the departure. The candidates for leader re£nement are the members of the super cluster, i.e. the cluster one layer above.**

the packet to its members.

4. Magellan's Design

Magellan builds a data-distribution forest over which trees it forwards stripes of the multicast content. For building the balanced multicast trees that constitute this forest, Magellan relies on Nemo, inheriting its resilience to the degree of churn inherent to the target environments. Tree construction/maintenance in Magellan makes use of a two-metric approach inspired on previous work by Wang and Crowcroft [38] and Chu et al. [13]. Following a *shortest wide-enough path* algorithm, a Magellan's node £rst selects candidate neighbor peers based on suf£cient (instead of highest) bandwidth, thus increasing the pool of alternative peers to choose from, to then choose among them those in the shortest path (lowest latency). To handle heterogeneous bandwidth constraints that may lead to "under" provisioned trees, peers with excess bandwidth capacity can be recruited to assist in secondary trees.

Based on Nemo, Magellan optimizes each of its independent trees for a generic end-to-end cost. For this purpose, it relies on the maintenance algorithms of its underlying tree protocol, Nemo, in which cluster leaders are chosen so that the sum of the costs to each cluster member is minimized, i.e. the cluster cost. Consequently, a cluster leader periodically reevaluates its owned clusters and transfers the cluster's ownership to another peer if this results in a reduced cluster cost. Additionally, cluster split operation divide the peers into two groups so that the sum of the pairwise costs is minimized for the two emerging clusters. Lastly, clients may switch clusters if they £nd a new leader with a smaller cost than their current parent, i.e. leader re£nement. Magellan's maintenance algorithms are summarized in Figure 5 with general helper functions de£ned in Figure 6.

```
GET-LEADER-FOR(cluster, tree)
  leader ← NIL
  cost ← 0
  for each peer in cluster
  do cost2 ← GET-CLUSTER-COST(peer, cluster, tree)
     if leader = NIL or cost2 < cost
        then leader ← peer
             cost ← cost2
  return leader

GET-CLUSTER-COST(leader, cluster, tree)
  cost ← 0
  for each peer in cluster
  do if peer ≠ leader
        then cost ← cost + GET-COST(leader, peer, tree)
  return cost

GET-COST(from, to, tree)
  cost ← 0
  if IS-SELF(from)
    then cost ← GET-COST-FOR(to, tree)
    else cost ← cache[from][to]
  return cost
```

Figure 6. Helper functions used in Magellan's maintenance operations. The cost cache *cache* is populated with the cluster members' announced cost to each other.

In the simple yet common scenario where cost is de£ned as the latency between two peers, Nemo, Magellan and Nice [3] use similar techniques for cluster split and leader re£nement. The periodical cluster reevaluation that prevents weak peers from monopolizing higher layers by promoting high bandwidth peers up in the tree, however, is unique to Nemo and Magellan. In bandwidth constraint environments, this technique would demote bandwidth-limited peers in favor of peers with higher available bandwidth.

Figure 7 outlines the heuristic cost function employed in Magellan's construction scheme. The cost accounts for the latency between two peers at £rst, adding a penalty if

```
GET-COST-FOR(peer, tree)
  cost ← latency(self, peer)
  if IS-LINK-OVERLOADED(self, peer)
    then cost ← cost+
           GET-OVERLOADED-PENALTY(self, peer)
  if !IS-PRIMARY-TREE(peer, tree)
    then cost ← cost+
           GET-NON-PRIMARY-TREE-PENALTY(self, peer)
  return cost
```

Figure 7. Determining the cost for a peer for promotion/demotion in a given tree.

the link is overloaded. To encourage the promotion of primary peers inside each tree structure, the cost function adds a cost penalty to secondary peers. While this scheme will favor performance over fairness, as it will opt for a nearby secondary peer over a distant primary one, experimentation results show the negative effects on fairness to be mostly neglectable.

To avoid overloading a peer, our scheme tracks the message drops from the outgoing queue of the ¤ow controlled stream for each peer. We consider a peer to be overloaded whenever it starts to drop messages; at this point a penalty, that dominates the latency metric, forces the tree construction algorithm to prefer non-overloaded over saturated peers. To monitor a ¤ow's performance, we rely on an implementation of a TCP-friendly, UDP transmission layer that implements TCP-friendly rate control (TFRC) [21].

Our heuristic provides a best-effort approach of ensuring that every peer contributes resources to its primary tree, while high capacity peers help out in other, secondary trees on demand. As long as there are primary peers with suf£cient bandwidth capacity within a reasonable distance, resources from secondary peers will not be required. As the group size increases, Magellan's construction operations are more likely to £nd quali£ed primary peers within a reasonable distance, thus reducing the heuristic's potential negative impact on fairness.

5. Related Work

A number of application-level multicast schemes have been proposed [14, 25, 3, 12, 36, 42, 6, 17]. Most of these schemes are based on a single application-level multicast tree with some targeting video and high-bandwidth content distribution [25, 26, 27, 17]. Nice [3] relies on balanced multicast trees to achieve high scalability. By relying on balanced trees, it is able to provide low end-to-end latency in large multicast groups. Nemo [6] introduces the concept of crew members in conjunction with negative acknowledgements to early detect and recover lost packets, thus increasing the resilience of the data delivery topology. Nemo adopts a periodic probabilistic maintenance operations, to help reduce the control overhead under high churn, and a periodic cluster reevaluation operation that demotes weak peers to guarantee an ef£cient distribution topology. Magellan builds upon Nemo to construct multiple performance-centric, balanced multicast trees. FatNemo [7], in particular, provides high-bandwidth multisource, multicast in cooperative environments by porting a fat-tree approach from parallel architecture to overlay networks. Both Magellan and FatNemo build on a balanced, performance-centric tree protocol and exposes the alternate paths introduced by co-leaders to achieve high performance and resilience. In contrast with Magellan, FatNemo does not aim at fair load distribution and, in fact, speci£cally leverages unbalanced load distribution to achieve low delivery latency under high load with potentially many publishers.

A number of protocols address multisource, multicasting for higher throughput, and some of these employ codecs for fault tolerance in bursty environments [29, 31, 32, 1, 2]. Magellan uses a fully decentralized algorithm to organizes peers into a scalable content delivery topology. While it currently does not employ any form of redundant encoding for recovery (such as Forward Error Correction), it can be easily extended for in this manner to potentially increase its performance in transient environments [8, 23].

Multiple trees have been proposed as an approach to achieve a resilient, high performance overlay mesh [41, 34, 33]. CoopNet [34, 33] uses a centralized organization scheme to build a set of distribution trees over which it stripes video using MDC. Multiple trees allows CoopNet to reduce the disruption caused by rapid node arrivals/departures common in ¤ash crowds situations. Both, interleaved spanning trees [41] and Magellan leverage a decentralized construction scheme for multi-tree, overlay creation and maintenance. In addition, Magellan relies on properties provided by the tree maintenance protocol and on the use of lateral error recovery [39] for high resilience. Furthermore, it aims at providing fairness in bandwidth contribution for individual peers and uses balanced trees for higher performance.

SplitStream [10] introduces an approach to striping content across a forest of interior-node-disjoint multicast trees. By enforcing the bandwidth constraint of each peer, SplitStream achieves contribution fairness among the cooperative peers. The implementation of SplitStream is based on Scribe [12], an application-level group communication system built upon Pastry [37], a proximity-aware DHT routing protocol. By utilizing multiple disjoint trees, SplitStream achieves fairness by making every peer contributing resources to the system. Having every peer as a contributor, however, reduces the performance as it results in suboptimal distribution topologies. Further, every peer must be able to support at least one stripe since any peer could be root for a multicast tree in Scribe. Magellan, in contrast, builds upon a performance-centric, multicast protocol which allows it to strike a balance between fairness and performance.

6. Evaluation

We study the fairness of Magellan and analyze its ability to handle heterogenous environments. Fairness is evaluated in terms of *outdegree* and *imbalance*,

- **Outdegree:** The outdegree is the fanout of a node and indicates the number of full rate streams a peer has to support in the distribution topology. Since the multicast data is split into smaller messages and peers may only forward part of them, the effective outdegree can be a non-integer value.

- **Imbalance:** The imbalance is de£ned as $stdev(outdegree)$ and illustrates how much the peer's contribution differ. A higher imbalance indicates that the peers' contribution to the system is less balanced. Thus a fair protocol aims at providing a small imbalance.

Performance, on the other hand, is analyzed in terms of the more traditional *delivery latency* and *delivery ratio*.

- **Delivery Latency:** End-to-end delay (including retransmission time) from the source to the receivers, as seen by the application. This accounts for path latencies along the overlay hops, as well as queueing delay and processing overhead at peers along the path.

- **Delivery Ratio:** Ratio of subscribers that have received a packet within a £xed time window. Disabled receivers are not accounted for.

These last two metrics are also use to illustrate the resilience of a protocol to high degrees of churn in the peer population , as node failures may result in suboptimal or interrupted delivery, and its ability to support bandwidth heterogeneous environments. The remainder of this section presents implementation details of the compared protocols and describes our experiment setup. We discuss evaluation results in Section 6.3.

6.1. Details on Protocol Implementations

We present experimental evaluation results contrasting Magellan's performance-centric and SplitStream's DHT-based [10] approaches to cooperative multicast. To provide a baseline against which to evaluate the performance/fairness tradeoffs in Magellan, we rely on Nemo [6], a conventional performance-centric, tree-based multicast protocol. For this evaluation we rely on the implementation of the protocols employed in the respective papers, Nemo [6] and FreePastry 1.3.2 [20] for SplitStream [10]. For the latter, we port the FreePastry communication layer to our simulation environment.

For Nemo [6], the cluster degree, k, is set to 3 and the grace period is set to 15 seconds. For SplitStream, we use a leaf set maintenance interval of 15 seconds and a route set maintenance interval of 225 seconds. We opted for this con£guration with intervals set to values four times lower than those employed in [10], to give SplitStream an ability to detect failures comparable to that of Nemo. [3] The outdegree for SplitStream nodes is limited to the number of stripes for the **fair variant** and set to a value corresponding to their available bandwidth capacity for the **performance-optimized variant**[4]. For Magellan, we multiply the grace period (and all other maintenance intervals) by the number of stripes in an effort to keep the overhead at the same level than Nemo.

We evaluate Magellan and SplitStream with different number of stripes (trees) - we employ S2, S4, S8 and S16 to denote the con£guration with 2, 4, 8 and 16 stripes, respectively.

6.2. Experimental Setup

We performed our evaluations through detailed simulation using SPANS, a locally written, packet-level, event-based simulator. For the purpose of this evaluation, SPANS models latency and bandwidth constraints at the link-level.

We ran simulations using GridG [28] topologies with 8115 nodes and a multicast group of 1024 members. GridG leverages Tiers [19, 9] to generate a three-tier hierarchical network structure, before applying a power law enforcing algorithm that preserves the hierarchical structure. Members are randomly attached to nodes at the lowest tier, and a random delay between 0.1 and 80 ms is assigned to every link. The links use drop-tail queues with a buffer capacity of 0.5 sec. Each end-host buffers up to 200 packets waiting for transmission, tail-dropping data packets in the presence of congestion. We con£gure GridG to assign different bandwidth distributions to different link types [26]. We assume that the core of the Internet has higher bandwidth capacities than the edges. For every link type, bandwidth is sampled from a uniform distribution. The scenario employed and the bandwidth ranges for each of link types are listed in Table 1. For the purpose of our evaluation of fairness-optimized variants, we set the minimal bandwidth capacity of a peer to be larger than one full stream rate.

Each simulation experiment lasts for 30 minutes of simulation time. All peers join the multicast group by contacting the rendezvous point at uniformly distributed, random times during the £rst 300 seconds of the simulation. For all the simulations, the warm-up interval of 24 minutes is omitted from the £gures. While the tree-based protocols enable the multicast layer immediately after joining, SplitStream is activated after Pastry runs for 10 minutes [10]. In an effort to eliminate the effect of potential implementation artifacts on the convergence time, we opted for a relatively long warm-up interval in order to provide each evaluated protocol with suf£cient time to optimize its distribution topology.

Starting at 18 minutes and lasting to the end of the simulation, each simulation run has a membership changing phase over which the evaluated protocols are exercised with and without end system failures. Node failures are independent and their time is sampled from an exponential distribution (with mean, *Mean Time To Failure*, equal to 60 min.). Failed nodes rejoin shortly after with a rejoin delay sampled from an exponential distribution with mean, *Mean Time To Repair*, equal to 10 min. The two means are chosen asymmetrically to allow, on average, 6/7 of all members to be up during this phase. Failure rates were obtained from a published report of £eld failures for networked systems [40]. The failure event sequence is generated a priori based on the above distribution and used for all protocols and all runs.

In all experiments, we model single source multicast streams to a group. Each source sends constant bit rate (CBR) traf£c of 1000 B payload at a rate of 10 packets per second. The buffer size is set to 16 packets for Nemo and Magellan, which corresponds to the usage of a 1.6-second buffer, a realistic scenario for applications such as video conferencing. SplitStream, on the other hand, is con£gured without a bounded buffer size to avoid it from being penalized, in terms of delivery ratio, by excessively long delivery latencies on some stripes.

6.3. Experimental Results

The outdegree of the participating peers well illustrates the fairness of a multicast protocol. Using multiple trees distribute the forwarding load more evenly among the participants as illustrated in Figure 8, which shows the cumulative distribution function of the outdegree for Nemo, our conventional tree-based multicast protocol, and four different con£gurations of Magellan with 2, 4, 8 and 16 stripes. Note that the outdegree of a given peer may be a non-integer value, as the multicast data is split into smaller messages and every peer may forward only parts of it. The outdegree CDF of a protocol that perfectly distributes the forwarding load among all participating peers would appear as a vertical line, thus all peers having an outdegree of 1.

The observation by Castro et al. [10] on the mismatch between conventional tree-based multicast and cooperative environments is well illustrated by Nemo's curve in Figure 8. Close to 50% of peers in Nemo are non-contributors or leaves, i.e. with an outdegree of zero. Magellan's forest of interweaved performance-centric trees, on the other hand, signi£cantly reduce the fraction of non-contributors when compared to a basic single tree approach even with only two stripes (i.e. two parallel trees).

Using more trees reduces the forwarding load within each of them and thus enables more low bandwidth peers to

[3] While this increases the overhead through control-related messages, it results also in better optimization of the underlying routing substrate.

[4] Both of these two variants were proposed in the original publication of SplitStream [10], although not labeled in the same manner.

	Scenario
	High Bandwidth
Nodes	8115
Links	16450
Client-Stub	1000-15000
Stub-Stub	10000-30000
Transit-Stub	10000-50000
Transit-Transit	50000-100000

Table 1. Our simulation scenario. The bandwidth is expressed in Kbps.

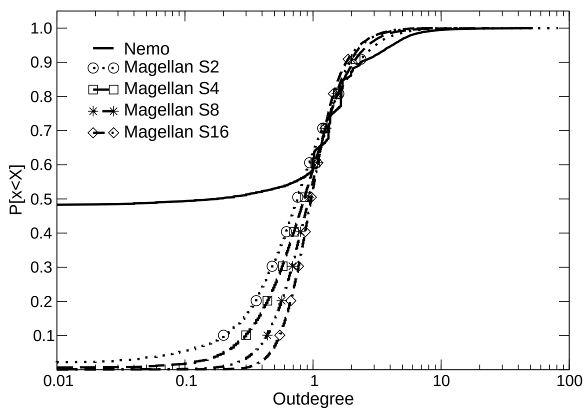


Figure 8. Outdegree CDF (Simulation, high bandwidth scenario).

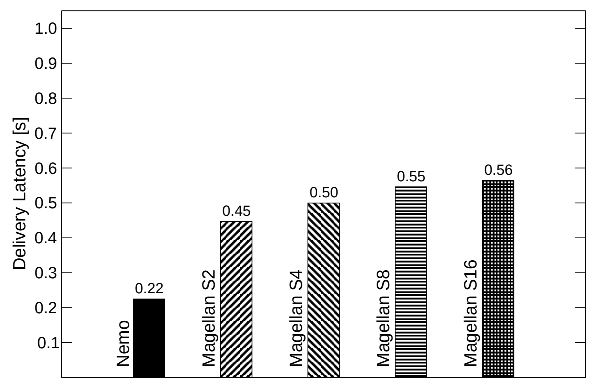


Figure 9. Delivery Latency (Simulation, high bandwidth scenario).

participate as interior nodes. Figure 8 illustrates the increasing fairness achieved by using more trees. As expected, when the number of stripes increases, Magellan's outdegree graph approaches the ideal fairness scenario with a signi£cantly small number of non-contributors.

Using multiple distribution trees, however, may negatively affect the delivery latency of a multicast protocol as only a subset of all peers may server as an interior node in each tree. To better understand the performance impact of using multiple trees, Figure 9 shows the mean delivery latencies of Nemo and Magellan's four alternative con£gurations. It is clear that fair distribution of forwarding responsibilities comes at a cost in terms of delivery latency, as Magellan's tree construction algorithm cannot recruit potential peers as internal nodes based solely on performance, but must account for primary/secondary peers for each tree. This effect is particularly pronounced when comparing Nemo to a two-stream Magellan, but it becomes less signi£cant as additional trees are added.

Figure 10 contrast the outdegree of Magellan with that of the fair and performance-optimized variants of SplitStream. As reported in [10], the fair variant of SplitStream, where the outdegree of a node is limited to the number of stripes, achieves near optimal fairness with either 2 or 16 stripes. The minor variations from the ideal outdegree of 1 can be explained by small differences among nodes, with some of them unable to deliver the full rate. As expected, the performance-based variants of SplitStream, where the outdegree of a node is set to value corresponding to their available bandwidth, sacri£ce ideal, fair load sharing for performance, resulting in more than 60% non-contributors. *Without sacri£cing end-to-end latency or delivery ratio, Magellan achieves fairness close to the fair variant of SplitStream with nearly zero non-contributors.*

Table 2 offers a more complete picture comparing Magellan and SplitStream in terms of fairness, as indicated by imbalance and delivery ratio. Imbalance, the standard deviation (stdev) of the outdegree, illustrates how much the contributions of peers differ. Again, the fair variant of SplitStream exhibits the lowest imbalance at a good delivery ratio. Magellan results in less imbalance than the performance-optimized variant of SplitStream, but considerably more than SplitStream's fair variant. For delivery ratio, Magellan employs a 1.6-second receiving window, i.e. packet delivered later than 1.6 seconds out of order will not

Protocol	Stripes							
	2		4		8		16	
	Imbalance	Delivery Ratio	Imbalance	Delivery Ratio	Imbalance	Delivery Ratio	Imbalance	Delivery Ratio
Magellan	1.706	0.97	0.982	0.97	0.749	0.97	0.657	0.97
SplitStream P	2.626	1.00	2.590	1.00	3.234	0.92	3.436	0.54
SplitStream F	0.394	1.00	0.212	1.00	0.105	1.00	0.145	1.00

Table 2. Imbalance and Delivery Ratio (Simulation, high bandwidth scenario).

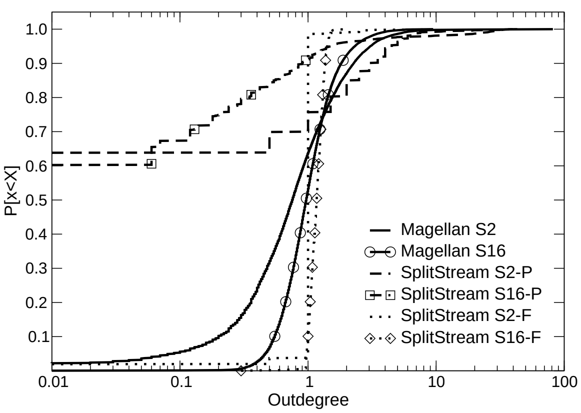


Figure 10. Outdegree CDF (Simulation, high bandwidth scenario). S2-P and S16-P denote performance-optimized tree constructions schemes, i.e. the outdegree is set according to the peer's available bandwidth, while S2-F and S16-F stand for the fair construction scheme, i.e. the outdegree is set equal to the number of stripes.

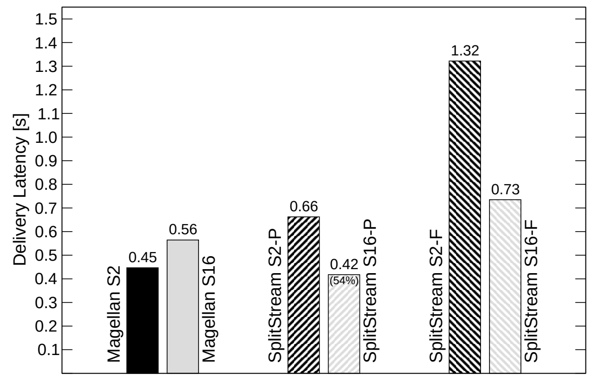


Figure 11. Delivery Latency (Simulation, high bandwidth scenario). S2-P and S16-P denote performance-optimized tree constructions schemes, i.e. the outdegree is set according to the peers available bandwidth, while S2-F and S16-F stand for the fair construction scheme, i.e. the outdegree is set equal to the number of stripes.

be accounted as successfully delivered. Despite the time constraints, Magellan delivers on average 97% of all packets.

Although *we do not enforce the receiving window for SplitStream,* which could signi£cantly reduce its delivery ratio, the performance-based variant of SplitStream still has a noticeable drop in delivery ratio when the number of strips increases. We believe this is due to the appearance of hot spots in the network. SplitStream assumes that bottlenecks appear only at the end hosts and does not include a mechanism to detect and react to bottlenecks inside the network. These hot spots are created as a side effect of the tree construction mechanism, which connects new peers to the delivery tree by routing a message toward the root. Since Pastry optimizes its routing table based on latency, it is likely that the join messages are routed through the core of the network and consequently are responded to by a peer located in the center of the network. Packet losses arise from the fact that the sum of all last mile capacities may be larger than the capacity of the core links, thus creating hot spots inside the stub network. Using fewer delivery trees, on the other hand, reduces this effect and consequently prevents the system from creating hot spots.

Figures 11 and 12 compare the different con£gurations of Magellan and SplitStream in terms of delivery latency without failures and delivery ratio with failures, respectively. Figure 11 shows delivery latency for the different con£gurations and variants. Magellan with two strips clearly outperforms both variants of SplitStream. The SplitStream variants with 16 stripes offer a lower delivery latency as a side-effect of the substantial number of dropped packets. This large number of lost packets may result, at least in part, from the fact that the performance-based variant of SplitStream looks only at the last mile capacity of the peer, not accounting for a bottleneck that may arise within the network. The fair SplitStream variant reduces delivery latency as new stripes are added. This can be explained, in part, from the fact that using more trees distribute the load better and reduces the effect of queues inside the network.

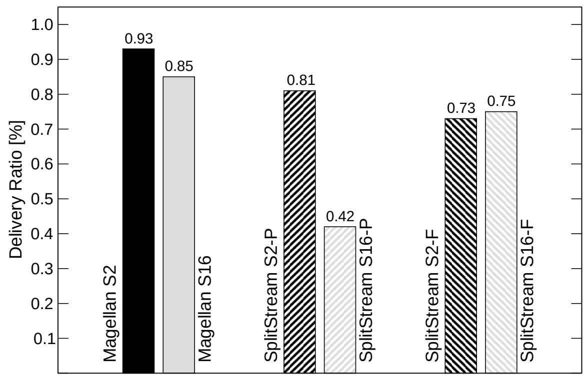


Figure 12. Delivery Ratio with Failures (Simulation, high bandwidth scenario). S2-P and S16-P denote performance-optimized tree constructions schemes, i.e. the outdegree is set according to the peers available bandwidth, while S2-F and S16-F stand for the fair construction scheme, i.e. the outdegree is set equal to the number of stripes.

Figure 12 shows the delivery ratios achieved by the Magellan's and SplitStream's different con£gurations. Magellan achieves higher delivery ratios than the two SplitStream variants. The trees in Magellan are not disjoint, and the delivery ratio decreases as a consequence of the growing dependencies on single nodes. For the performance-optimized SplitStream variant, which by opting for performance over fairness does not create disjoint trees, the same dependencies on single node result in a lower delivery ratio with more trees. The fair SplitStream variant enforces fairness and disjointness the best among all the protocols. Enforcing fairness, however, yields deep unbalanced trees with the resulting overhead on application performance.

Throughout this section, we have evaluated two setups of SplitStream: the fairness-optimized and the performance-optimized variant. While a combination of these two variants could be employed, e.g. by relaxing the fair variant so that every peer contributes at most twice the basic stream rate, the performance of any mixed approach would be bound by that of the performance of the optimized variant which de£nes the lowest achievable delivery latency. Trying to improve load distribution would certainly result on increased delivery latencies, toward the upper bound de£ned by the performance of the "pure" fair variant. Similarly, one could explore the range of fairness/performance tradeoff in Magellan, by changing the penalty for non-primary trees (see Fig. 7). In sum, both Magellan and SplitStream enable a £ne grained control of the fairness/performance tradeoff, although Magellan starts by offering lower delivery latency, with high resilience under churn, at comparable fairness.

7. Conclusions and Future Work

We have presented Magellan, a performance-centric multicast protocol capable of striking a balance between fairness and performance in heterogeneous, cooperative environments. Simulation results show that Magellan, even when relying on a very simple heuristic for forest creation, achieves a high degree of fairness at signi£cantly lower cost in terms of application performance than comparable approaches. Furthermore, Magellan is able to honor heterogenous bandwidth constraints without knowing them a priori. We have studied the cost of using multiple trees in terms of delivery latency and illustrated its advantage in distributing the load among all participating peers. Our evaluation illustrates that even two stripes offer a high degree of fairness, while providing signi£cantly better performance than DHT-based, fairness-optimized counterparts. Although, we have presented and analyzed an implementation of Magellan that relies on Nemo, Magellan's design could potentially be implemented with any other performance-centric multicast tree protocol. We have started to evaluate the implications of alternative policies for forest creation and maintenance, as part of our future work. Beyond this, we plan to thoroughly explore the performance/fairness tradeoff in overlay multicast for cooperative environment.

References

[1] J. G. Apostolopoulos. Reliable video communication over lossy packet networks using multiple state encoding and path diversity. In *Visual Communications and Image Processing*, January 2001.

[2] J. G. Apostolopoulos and S. J. Wee. Unbalanced multiple description video communication using path diversity. In *International Conference on Image Processing*, October 2001.

[3] S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable application layer multicast. In *Proc. of ACM SIGCOMM*, August 2002.

[4] A. R. Bharambe, V. Padmanabhan, and S. Seshan. Supporting spectators in online multiplayer games. In *Proc. of the 3rd Workshop on Hot Topics in Networks (HotNets-III)*, November 2004.

[5] A. R. Bharambe, S. G. Rao, V. N. Padmanabhan, S. Seshan, and H. Zhang. The impact of heterogeneous bandwidth constraints on DHT-based multicast protocols. In *Proc. of IPTPS*, February 2005.

[6] S. Birrer and F. E. Bustamante. Resilient peer-to-peer multicast without the cost. In *Proc. of MMCN*, January 2005.

[7] S. Birrer, D. Lu, F. E. Bustamante, Y. Qiao, and P. Dinda. FatNemo: Building a resilient multi-source multicast fattree. In *Proc. of IWCW*, October 2004.

[8] J. W. Byers, J. Considine, M. Mitzenmacher, and S. Rost. Informed content delivery accross adaptive overlay networks. In *Proc. of ACM SIGCOMM*, August 2002.

[9] K. L. Calvert, M. B. Doar, and E. W. Zegura. Modeling internet topology. *IEEE Communications Magazine*, 35(6):160–163, June 1997.

[10] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. Splitstream: High-bandwidth multicast in cooperative environments. In *Proc. of the 19th ACM SOSP*, October 2003.

[11] M. Castro, M. B. Jones, A.-M. Kermarrec, A. Rowstron, M. Theimer, H. Wang, and A. Wolman. An evaluation of scalable application-level multicast built using peer-to-peer overlays. In *Proc. of IEEE INFOCOM*, March 2003.

[12] M. Castro, A. Rowstron, A.-M. Kermarrec, and P. Druschel. SCRIBE: A large-scale and decentralised application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communication*, 20(8), October 2002.

[13] Y.-H. Chu, S. G. Rao, S. Seshan, and H. Zhang. Enabling conferencing applications on the Internet using an overlay multicast architecture. In *Proc. of ACM SIGCOMM*, August 2001.

[14] Y.-H. Chu, S. G. Rao, S. Seshan, and H. Zhang. A case for end system multicast. *IEEE Journal on Selected Areas in Communication*, 20(8), October 2002.

[15] Y.-H. Chu, S. G. Rao, and H. Zhang. A case for end system multicast. In *Proc. of ACM SIGMETRICS*, June 2000.

[16] S. E. Deering. Multicast routing in internetworks and extended LANs. In *Proc. of ACM SIGCOMM*, August 1988.

[17] H. Deshpande, M. Bawa, and H. Garcia-Molina. Streaming live media over a peer-to-peer network. Technical Report 2001-20, Stanford U., 2001.

[18] C. Diot, B. N. Levine, B. Lyles, H. Kassem, and D. Balensiefen. Deployment issues for the IP multicast service and architecture. *IEEE Network*, 14(1), January/February 2000.

[19] M. B. Doar. A better model for generating test networks. In *Proc. of Globecom*, November 1996.

[20] P. Druschel, E. Engineer, R. Gil, J. Hoye, Y. C. Hu, S. Iyer, A. Ladd, A. Mislove, A. Nandi, A. Post, C. Reis, A. Singh, and R. Zhang. Freepastry 1.3.2. freepastry.rice.edu, February 2004.

[21] S. Floyd, M. Handley, J. Padhye, and J. Widmer. Equation-based congestion control for unicast applications. In *Proc. of ACM SIGCOMM*, August 2000.

[22] P. Francis. Yoid: Extending the Internet multicast architecture. http://www.aciri.org/yoid, April 2000.

[23] J. Gemmell, E. Schooler, and J. Gray. Fcast multicast £le distribution. *IEEE Network*, 14(1):58–68, January 2000.

[24] L. Gharai, C. Perkins, R. Riley, and A. Mankin. Large scale video conferencing: A digital amphitheater. In *Proc. 8th International Conference on Distributed Multimedia Systems*, September 2002.

[25] 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 *Proc. of the 4th USENIX OSDI*, October 2000.

[26] D. Kostić, A. Rodriguez, J. Albrecht, and A. Vahdat. Bullet: High bandwidth data dissemination using an overlay mesh. In *Proc. of the 19th ACM SOSP*, October 2003.

[27] D. Kostić, A. Rodriguez, J. Albrecht, and A. Vahdat. Maintaining high bandwidth under dynamic network conditions. In *Proc. of USENIX ATC*, April 2005.

[28] D. Lu and P. A. Dinda. Synthesizing realistic computational grids. In *Proc. of SC2003*, November 2003.

[29] P. Maymounkov and D. Mazieres. Rateless codes and big downloads. In *Proc. of IPTPS*, February 2003.

[30] D. McGregor, A. Kaplka, M. Zyda, and D. Brutzman. Requirements for large-scale networked virtual environments. In *Proc. of the International Conference on Telecommunications*, June 2003.

[31] T. Nguyen and A. Zakhor. Distributed video streaming. In *Proc. of Multimedia Computing and Networking*, San Jose, CA, January 2002.

[32] T. Nguyen and A. Zakhor. Distributed video streaming with forward error correction. In *Proc. of Packet Video Workshop*, Pittsburgh, PA, 2002.

[33] V. N. Padmanabhan, H. J. Wang, and P. A. Chou. Resilient peer-to-peer streaming. In *Proc. of IEEE ICNP*, November 2003.

[34] V. N. Padmanabhan, H. J. Wang, P. A. Chou, and K. Sripanidkulchai. Distributing streaming media content using cooperative networking. In *Proc. of NOSSDAV*, May 2002.

[35] D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel. ALMI: An application level multicast infrastructure. In *Proc. of USENIX USITS*, March 2003.

[36] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Application-level multicast using content-addressable networks. In *Proc. of NGC*, November 2001.

[37] A. Rowstron and P. Drushel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM Middleware*, November 2001.

[38] Z. Wang and J. Crowcroft. Bandwidth-delay based routing algorithms. In *IEEE GlobeCom*, November 1995.

[39] K.-F. S. Wong, S. G. Chan, W.-C. Wong, Q. Zhang, W.-W. Zhu, and Y.-Q. Zhang. Lateral error recovery for application-level multicast. In *Proc. of IEEE INFOCOM*, March 2004.

[40] J. Xu, Z. Kalbarczyk, and R. K. Iyer. Networked Windows NT system £eld failure data analysis. In *Proc. of PRDC*, December 1999.

[41] A. Young, J. Chen, Z. Ma, A. Krishnamurthy, L. Peterson, and R. Wang. Overlay mesh construction using interleaved spanning trees. In *Proc. of IEEE INFOCOM*, March 2004.

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