Resilient Peer-to-Peer Multicast without the Cost

Stefan Birrer and Fabián E. Bustamante

Department of Computer Science, Northwestern University, Evanston, USA

ABSTRACT

We introduce Nemo, a novel peer-to-peer multicast protocol that achieves high delivery ratio without sacrificing end-to-end latency or incurring additional costs. Based on two simple techniques: (1) *co-leaders* to minimize dependencies and, (2) *triggered negative acknowledgments (NACKs)* to detect lost packets, Nemo's design emphasizes conceptual simplicity and minimum dependencies, thus achieving performance characteristics capable of withstanding the natural instability of its target environment. We present an extensive comparative evaluation of our protocol through simulation and wide-area experimentation. We contrast the scalability and performance of Nemo with that of three alternative protocols: Narada, Nice and Nice-PRM. Our results show that Nemo can achieve delivery ratios similar to those of comparable protocols under high failure rates, but at a fraction of their cost in terms of duplicate packets *(reductions > 90%)* and control-related traffic.

Keywords: Resilient Multicast, Peer-to-Peer Multicast, Scalable Multicast

1. INTRODUCTION

Multicast is an efficient mechanism to support group communication applications such as audio and video conferencing, multi-party games and content distribution. It decouples the size of the receiver set from the amount of state kept at any single node and potentially avoids redundant communication in the network. Partially in response to the deployment issues of network-layer multicast,[1,2] a number of protocols recently proposed adopt an alternative peer-to-peer, or application-layer approach, with all multicast related functionality implemented exclusively at end-systems.[3–11]

One of the most important challenges of these application-layer protocols is their ability to efficiently handle the high degree of transiency inherent to their environment.[12] As multicast functionality is pushed to autonomous, unpredictable peers, significant performance loss can result from the end hosts' higher degree of transiency when compared to routers. A good indication of transiency is the peers' *median session time*, where *session time* is defined as the time between when a peer joins and leaves the network. Measurement studies of widely used P2P systems have reported median session times ranging from an hour to a minute.[13–15] Achieving high delivery ratios under these conditions, without incurring additional costs or sacrificing end-to-end latencies has proven to be a difficult task.[15,16]

This paper makes three main contributions. First, we introduce Nemo, a novel peer-to-peer multicast protocol that aims at achieving this elusive goal. Based on two techniques: (1) *co-leaders* and, (2) *triggered negative acknowledgments (NACKs)*, Nemo's design emphasizes conceptual simplicity and minimum dependencies,[17] achieving in a cost-effective manner performance characteristics capable of withstanding the natural instability of its target environment. Nemo's approach is aimed at applications, such as real-time audio and video streaming, that can benefit from high delivery ratios within specific latency bounds, without requiring perfect reliability; a model previously termed *resilient multicast*.[18,19] Second, we present simulation-based and wide-area experimentations showing that Nemo can achieve high delivery ratios and low end-to-end latency similar to those of comparable protocols under high failures rates, while significantly reducing costs in terms of duplicate packets *(reductions > 90%)* and control related traffic, making the proposed algorithm a more scalable solution to the problem. Lastly, we describe the use of periodic probabilistic operations for overlay maintenance and discuss their effectiveness in dealing with highly dynamic environments.

After reviewing background and related work in Section 2, Section 3 outlines Nemo's approach and presents the protocol's design and operational details. We report experimental results in Section 4 and conclude in Section 5.

E-mail: {sbirrer,fabianb}@cs.northwestern.edu

2. RELATED WORK

All peer-to-peer or application-layer multicast protocols organize the participating peers in two topologies: a control overlay for group membership related tasks, and a delivery tree for data forwarding. Available protocols can be classified according to the sequence adopted in their construction.[3, 20] In a tree-first approach,[4, 21] peers directly build the data delivery tree by selecting their parents from among known peers. Additional links are later added to define the control topology. With a mesh-first approach,[3, 6] peers construct a more densely connected graph (mesh) over which (reverse) shortest path spanning trees, rooted at any peer, can be built. Protocols adopting an implicit approach[7–9] create only a control topology among the participant peers. Their data delivery topology is implicitly determined by the defined set of packet-forwarding rules.

Banerjee et al.[7] introduce Nice and demonstrate the effectiveness of overlay multicast across large scale networks. They present the first look at the robustness of alternative overlay multicast protocols under group-membership changes. Nemo adopts the same implicit approach, and its design draws a number of ideas from Nice. Nemo focuses on resilient multicast with highly transient population, introducing co-leaders and adopting a periodic probabilistic approach to reduce the cost of membership operations.

A large number of research projects have addressed reliable and resilient multicast at the network layer.[18, 22–24] A comparative survey of these protocols is given in.[25, 26] In the context of overlay multicast, a number of protocols have been proposed aiming at high resilience.[10, 11, 19, 27] ZIGZAG[11] improves resilience by separating the control and data delivery trees between two peers, at every level, with both peers sharing responsibilities in the presence of failures. Yang et al.[27] propose to address failures by proactively choosing backup leaders ahead of time. Nemo's co-leaders serve a similar role as replacement leaders. In addition, co-leaders forward the data during the repair interval, which includes the time until the failure is detected, thus providing an uninterrupted service to the downstream peers. Probabilistic Resilient Multicast (PRM)[19] uses probabilistic randomized forwarding and NACK-based retransmission to improve resilience. In contrast, Nemo relies only on deterministic techniques for data forwarding improving resilience through the introduction of co-leaders.

SplitStream[28] and CoopNet[10] opt for splitting the multicast content across separate trees, thus improving load sharing and resilience. CoopNet also relies on a centralized organization protocol and adopts Multiple Description Coding to achieve data redundancy. Nemo is a decentralized peer-to-peer multicast protocol which offers redundancy in the delivery path with only a single control topology through the addition of co-leaders. Exploring the use of data redundancy is part of our future work.

Nemo adopts a probabilist approach to reduce the cost of overlay maintenance. Closely related, although in the context of structured peer-to-peer overlay networks, Mahajan et al.[29] proposes to dynamically adjust protocol parameters, such as heartbeat intervals and grace periods, based on the operating conditions. Nemo's refinement algorithm could benefit from their technique by adjusting the refinement interval based on the experienced membership-change rate and the measured dynamics of the underlying physical network.

3. NEMO'S APPROACH

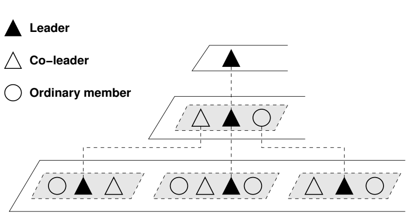


Figure 1. Nemo's logical organization. The shape illustrates the role of a peer within a cluster.

Nemo follows the *implicit approach* to building overlays for multicasting: participating peers are organized into clusters based on network proximity *, with every peer being a member of a cluster at the lowest layer. Clusters vary in size between d and $3d - 1$, where d is a constant known as *degree*.† Each of these clusters selects a *leader* that becomes a member of the immediately higherm 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. Hence peers can lead more than one cluster in successive layers of this logical hierarchy. Figure 1 illustrates the logical organization of Nemo.

Our work focuses on improving the resilience of peer-to-peer overlay systems for streaming media multicast. Our peer-to-peer approach specifically addresses the inherent high degree of transiency of peer populations. The following

*Although other factors such as bandwidth[3, 30] and expected peer lifetime[13] can be easily incorporated.

†This is common to Nemo, Nice[7] and Zigzag[11]; the degree bounds have been chosen to help reduce oscillation in clusters.

paragraphs discuss the details of our approach. We begin by explaining the dynamics of the basic tree-based protocol such as the joining and departure of peers, and briefly discuss Nemo's probabilistic approach to overlay maintenance. We then elaborate on the algorithm for data forwarding and retransmission.

3.1. Group Management and Cluster Dynamics

A new peer joins the multicast group by querying a well-known node, the rendezvous point, for the IDs of the members on the top layer. Starting there and in a iterative manner, the incoming peer continues: (i) requesting the list of members at the current layer from the cluster's leader, (ii) selecting from among them who to contact next based on network proximity, and (iii) moving into the next layer. When the new peer finds the closest leader at the bottom layer, it joins the associated cluster.

Members can leave Nemo in announced or unannounced manner. Since a common member has no responsibilities towards other peers, it can simply leave the group after informing its cluster's leader. A leader, on the other hand, must first elect a replacement leader for all clusters it owns and inform its own cluster's leader before leaving. To detect unannounced leaves, Nemo relies on heartbeats exchanged among the cluster's peers. After a given *grace period*, unreported members are considered dead and a repair algorithm is initiated. If the failed peer happens to be a leader, the tree itself must be repaired: the members of the victim's cluster must elect the replacement leader from among themselves.

To deal with dynamic changes in the underlying network, every peer periodically checks the leaders of the next higher layer and switches clusters if another leader is closer than the current one (with thresholds used to prevent oscillation). Similarly, in a continuous process of refinement, every leader checks its highest owned cluster for better suited leaders and transfers leadership if such a peer exists.

Due to membership changes, clusters may grow/shrink beyond the cardinality bounds defined by the clusters' degree; such clusters must be dealt with to guarantee the hierarchical properties of the protocol. Undersized clusters are merged with others while oversized ones are split into two new ones. Both operations are carried on by the cluster's leader. If the size of a cluster falls bellow its lower bound, the cluster leader redirects all cluster members to the leader of a neighbor cluster. When a cluster's cardinality exceeds its upper bound, the leader must split it into two new clusters, each of size at least $3k/2$. The two new clusters are defined so as to minimize the total sum of path latencies (i.e. the cluster's diameter). For this we employ a genetic algorithm[31] that yields near optimal results at a fraction of the cost [‡] of more thorough alternatives.

3.2. A Probabilistic Approach to Overlay Maintenance

Cluster split and merge operations can be triggered by each arrival/departure event. While potentially guaranteeing the protocol's operational parameters (such as cluster sizes), this *proactive* approach to overlay management can add significant additional stress to an already stressed network.[16] For most costly maintenance operations, such as splitting, merging and refinement, Nemo relies instead on a *periodic probabilistic* approach where operations are executed with some probability or, alternatively, deferred to the next interval. In the presence of high transiency, many of these operations can not only be deferred, but completely avoided as follow-up changes may revert a previously triggering situation. A stream of member departures/failures, for example, may leave a cluster undersized, triggering a relatively expensive merge operation. If deferred, a series of subsequent member joins may push the cluster's cardinality beyond its merge-triggering bound.

```
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 ← crewOf(self))
  if isCrewMember(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 ← crewOf(leader))
```

Figure 2. Data Forwarding Algorithm

3.3. Data Forwarding

Nemo's data delivery topology is implicitly defined by the adopted 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

[‡] The algorithm runs in cn^2 time, with a small constant c, where n is the size of the cluster.

in their clusters and up to the next higher layer. When forwarding a packet to a given cluster, the forwarding peer must select the destination peer among all crew members in the cluster. The algorithm is summarized in Fig. 2. It first adds the leader and the children to the receiver set R, originally empty, if and only if they are not part of the sender's crew (lines 2-6). Once the receiver set is populated, the message (msg) is forwarded to each receiver in R, using its own crew as the sending crew for the message. If the forwarding peer is not the leader of its highest layer, but part of the crew, the peer is responsible for forwarding the message to its neighbors and to the cluster one layer above. As the peer does this on behalf of its leader, the sender crew must be set accordingly (lines 8-13).

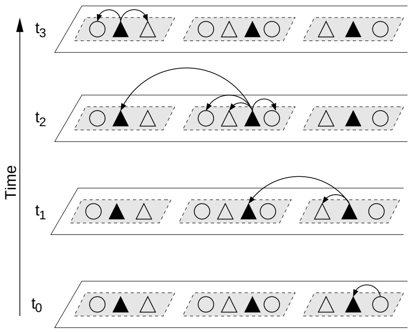


Figure 3. Data forwarding in Nemo.

Figure 3 illustrates the data forwarding algorithm using the logical topology from Figure 1; each row in the figure corresponds to one time step. Although we employ leaders for this example, the explanation is also valid when routing through co-leaders. Each row corresponds to one time step. 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 and second lowest layer. In the last step, the leader of the cluster on the left has to forward the packet to all its members.

The forwarding responsibility is evenly shared among the leaders by alternating the message recipient among them. Thus, in case of a failed crew member, the remaining crew members can still forward their share of messages through the tree.

3.4. Data Retransmission

Similar to other protocols aiming at high resilience,[19, 22] Nemo relies on sequence numbers and triggered NACKs to detect lost packets. Every peer piggybacks a bit-mask with each data packet reporting the previously received packets. In addition, each peer maintains a cache of received packets and a list of missing ones. The *receiver window* defines the total number of cached and listed packets and is set based on the application's requirements. Once a gap (relative to a peer's upstream neighbors) is detected in the packet flow, the absent packets are considered missing after some fixed period of time. This time is selected to reduce the effect of jitter and processing delays. [§] For each missing packet, a NACK is sent to a peer caching it, and a different timeout is set for retransmission.

```
RECOVER-DATA(msg)
1  d ← DOWNWARD ∪ UPWARD
2  for each packet in recent_packets
3  do if isSameStream(packet, msg)
4        then if isChild(packet.sender_crew)
5              then d ← d ∩ ¬ DOWNWARD
6           if isLeader(packet.sender_crew)
7              then d ← d ∩ ¬ UPWARD
8  FORWARD(msg, direction ← d)
```

Figure 4. Data Recovery Algorithm.

The peer requesting the retransmission may not be the only one missing the packet. Ideally, the requesting peer should forward the recovered packet to all other known peers (and only to those) who need it. To achieve this we have designed an efficient algorithm that, without incurring additional control traffic, helps us improve latency in delivery for retransmitted packets while reducing the number of duplicates. The algorithm (sketched in Figure 4) takes advantage of the fact that, over a reasonably small window of time, a peer sees the packets from one source flowing in one logical direction. After a peer has detected a lost packet, it initiates the recovery protocol and decides the direction in which to forward the recovered packet, if at all. No forwarding would be required if all other peers have successfully received the packet. Otherwise, the forwarding direction will be decided based on the flow direction of the follow-up packets: *upward*/*downward* if the source is logically below/above the recovering peer.

The algorithm first sets the forwarding direction both *upward* and *downward*. It then checks all recently received packets (*for*-loop, lines 2-7). If the inspected packets belong to the same stream, the algorithm uses the peer's position with respect to the packet sender's crew to deduce the forwarding direction for the recovered packet. If a recently forwarded packet had a sender from the peer's successors in the tree, the recovered packets need only be transmitted to the peers higher up in the tree. Alternatively, when the packet's sender is an ancestor in the tree, the packet needs only be relayed downwards. The algorithm helps reduce the number of duplicate packets generated indirectly by packet losses.

[§] In our current implementation this time is set to twice the round trip time ($2 \cdot RTT$) to the farthest crew member.

4. EVALUATION

We have evaluated Nemo's performance through detailed simulation and in real Internet environment in the PlanetLab testbed.[32] We compare Nemo's performance to that of three other protocols – Narada,[3] Nice[7] and Nice-PRM.[19] [¶]

Our implementations of the alternative protocols closely follow the descriptions from the corresponding literature, and have been validated by contrasting our results with the published values. However, there are a number of improvements to the common algorithms, such as the use of periodic probabilistic operations, that while part of Nemo were made available to all protocols in our evaluations and help explain the performance improvements with respect to those reported in their original publications.[3,7,19] We opted for this to isolate the contribution of co-leaders to the overall resilience of the multicast protocol. By carefully abstracting the protocols' logic, the difference in the code-base of the wide-area and the simulator implementations for any given protocol is limited to how the communication between two peers is realized.

The effectiveness of the different protocols is evaluated in terms of performance improvements to the application and the protocol's overhead. *Delivery Ratio*, the ratio of subscribers which have received a packet within a fixed time window, and *connectivity*, the percentage of nodes that have received at least one packet on the last 10-seconds interval, intend to capture reliability improvements. *Latency* represents the end-to-end delay from source to receivers, including retransmission time, queueing delay and processing overhead. Protocol's overhead is measured in terms of *duplicate packets*, the number of duplicate packets per sequence number for all receivers [‖].

We ran our simulations using Inet,[33] Transit-Stub and AS Waxman[34] topologies with 3,072 and 8,192 nodes and multicast groups of varying sizes ranging between 128 and 4,096 members. Due to space constraints we only report results based on Inet topologies with 3,072 nodes and a multicast group of 512 members. [**] Members are randomly attached to routers, and a random delay of between 1 and 4 ms is assigned to every link. Each simulation experiment lasts for 40 minutes (simulation time). All peers join the multicast group by contacting the rendezvous point at uniformly distributed, random times within the first 200 seconds. Starting at 600 seconds and lasting for about 1200 seconds, each simulation run has a phase with rapid membership changes. During this time each protocol is exercised under two different failure rates derived from Xu et al.[36] study on networked system failure and related research on resilient multicast.[19,36] A warm-up time of 300 seconds is omitted from the figures. Under a *high-failure rate*, nodes fail independently at a time sampled from an exponential distribution with *mean time to failure (MTTF)* equal to 5 minutes, to rejoin shortly after (time sampled from an exponential distribution with *mean time to repair (MTTR)* of 2 minutes). The same set of simulations is also run with a *low-failure* rate defined by a MTTF of 60 minutes and a MTTR of 10 minutes. The means for each failure rate are chosen asymmetrically to allow, on average, 5/7 (6/7) of all members to be up during this phase. We generate a failure event sequence a priori, based on the $(MTTF, MTTR)$ pairs. The same sequence is used for all protocols and all runs.

For the wide-area experiments we restrict our comparison to Nemo, Nice, and Nice-PRM(3,0.02) with between 60 and 200 end-hosts distributed across 72 PlanetLab sites. We opted for 2% forwarding probability for Nice-PRM, since this offers the best tradeoff between delivery ratio and rate of duplicate packets. The results presented here are based on twenty-five runs each, done at different times of day, where each run is a set of three experiments (one per protocol) in a random order. We inject failures at the high-failure rate used for simulation. Each protocol runs for 30 minutes that includes a 10-minute period with failures. The failure events are drawn randomly from an exponential distributed with means set to the $(MTTF, MTTR)$ pair of the high-failure case. The end-to-end delay was estimated using a global time server, based on an algorithm inspired by Mills' .[37]

All experiments were run with payloads of 100 and 1,000 Bytes. We opted for these relatively small packet sizes to avoid saturation effects in PlanetLab. For simulations, we assume infinite bandwidth per link and only model link delay, thus the packet size is secondary. We employ a buffer size of 32 packets and a rate of 10 packets per second. This corresponds to the usage of a 3.2-second buffer, a realistic scenario for applications such as multimedia streaming.

4.1. Evaluation Results

Tables 1 and 2 summarize the results from simulation and wide-area experiments for both low and high failure rates. Nemo and Nice-PRM, with 2 and 3% probability, achieve comparably high delivery ratio, significantly better than the

[¶] For Narada we employed the latency-only scheme to construct the overlay. For Nice and Nice-PRM the cluster degree, k, was set to 3. For Nemo, the cluster degree k and the crew size were set to 3.

[‖] Packets arrived outside of the delivery window are counted as duplicates, since the receiver has already assumed them lost.

[**] Comparable results were obtained with the alternative configurations. For the complete set of results please see.[35]

Table 1. Simulation Results with Low- and High-Failure Rate (512 end hosts).

Protocol	Low-Failure Rate			High-Failure Rate		
	Delivery ratio Mean (Std)	Connectivity Mean (Std)	Duplicate packets Mean (Std)	Delivery ratio Mean (Std)	Connectivity Mean (Std)	Duplicate packets Mean (Std)
Nemo	1.000 (0.12E-3)	1.0000 (0.02E-3)	0.34 (0.09)	0.998 (0.89E-3)	0.9998 (0.21E-3)	3.16 (0.29)
Nice-PRM(3,0.01)	0.999 (0.56E-3)	0.9999 (0.12E-3)	6.42 (0.38)	0.993 (1.25E-3)	0.9989 (0.28E-3)	12.47 (1.04)
Nice-PRM(3,0.02)	0.999 (0.36E-3)	0.9999 (0.07E-3)	12.00 (0.66)	0.994 (1.23E-3)	0.9993 (0.22E-3)	18.20 (0.77)
Nice-PRM(3,0.03)	0.999 (0.27E-3)	0.9999 (0.05E-3)	16.74 (0.65)	0.994 (1.01E-3)	0.9993 (0.17E-3)	24.22 (1.82)
Nice	0.999 (0.52E-3)	0.9998 (0.16E-3)	1.29 (0.40)	0.992 (1.95E-3)	0.9975 (1.02E-3)	7.10 (0.71)
Narada	0.950 (38.3E-3)	0.9548 (36.2E-3)	0.00 (0.00)	0.852 (60.3E-3)	0.8534 (57.9E-3)	0.00 (0.00)

Table 2. Wide-Area Results (PlanetLab, ∼70 end hosts, 8 Kbps). Statistics include packets with sequence numbers from 6,000-12,000. Nice-PRM is configure with 3 random peers and 2% probability (*PRM(3,0.02)*).

Protocol	Low-Failure Rate		High-Failure Rate	
	Delivery ratio Mean (Std)	Duplicate packets Mean (Std)	Delivery ratio Mean (Std)	Duplicate packets Mean (Std)
Nemo	0.996 (0.003)	0.52 (0.27)	0.979 (0.010)	1.27 (0.56)
Nice-PRM	0.997 (0.003)	1.37 (0.15)	0.953 (0.024)	2.02 (0.57)
Nice	0.991 (0.011)	0.30 (0.23)	0.939 (0.032)	1.06 (0.47)

non-resilient alternatives, Nice and Narada. Nemo features the highest delivery ratio with 99.8% under high failure rate (100% under low failure rate). As the third and sixth columns in Table 1 show, Nemo also offers the highest connectivity of all compared protocols under both scenarios. In the wide-area, Nemo also outperforms both Nice (93.9%) and Nice-PRM (95.3%) with a delivery ratio of 98% under high failure rates. Nemo's high connectivity and delivery ratio are due to the alternate data paths offered by crew members, as they enable the early detection and retransmission of lost packets within the allowed timeout interval. Previous work[19] accounted link losses in simulation and reported a similar delivery ratio for Nice-PRM, but a substantial lower delivery ratio for Nice without retransmission. Our simulation results clearly show that without losses at the network layer, the effectiveness of PRM is only minor for a protocol such as Nice. PRM's improvements to Nice's resilience, however, become clear in the more realistic setting of the wide-area testbed.

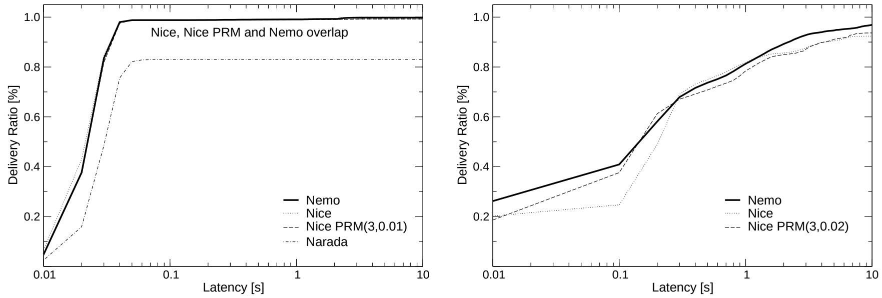
The cost of a resilient multicast protocol can be partially measured in terms of duplicate packets. The last columns for each failure rate in both tables (Tables 1 and 2) show this overheads. Nemo's approach results in substantially lower duplicates per sequence number (3.16) than Nice-PRM while providing comparable (and sometimes higher) delivery rates. Nemo's advantage holds in the wide-area, where it offers comparable low number of duplicates (1.27 and 0.52 per sequence number for high and low failure rate, respectively). Under low failure rate the three protocols offer similar delivery ratio, although Nemo achieves this with a substantially lower rate of duplicate packets (0.52) when compared with Nice-PRM (1.37). Nemo's multiple crew members and its retransmission approach help explain these improvements, as the former results in alternate paths that increase the effectiveness of NACKs and the latter ensures delivery to other peers (potentially) missing the packet, thus reducing the total amount of retransmissions.

Table 3 shows the wide-area results obtained with a considerable higher data rate of 80 Kbps. The time scale for these run was reduced by half, in part to minimize their impact on concurrent PlanetLab experiments. While this larger data rate results in a significant drop on delivery ratio for all protocols, Nemo is able to sustain a considerable higher delivery ratio than the alternative protocols (twice as much) at very low cost in terms of duplicates, even lower than Nice.

Nemo's higher resilience with low overhead comes also at no cost in terms of latency, as can be observed in Fig. 5. The graph shows the cumulative distribution function (CDF) of latency for all received packets under high failure rate in one of the simulation and one of the wide-area experiments (buffer of 32 packets, which corresponds to a 3.2-second

Table 3. Wide-Area Results (PlanetLab, ∼80 end hosts, 80 Kbps). Statistics include the packets during the failure interval. Nice-PRM is configure with 3 random peers and 2% probability (*PRM(3,0.02)*).

Protocol	Low Failure Rate	
	Delivery ratio Mean (Std)	Duplicate packets Mean (Std)
Nemo	0.814 (0.124)	12.82 (5.35)
Nice-PRM	0.427 (0.137)	20.71 (8.23)
Nice	0.485 (0.133)	13.87 (3.12)



(a) Simulation - 512 end hosts (b) Wide-Area - $\sim$72 end hosts

Figure 5. Latency CDF under high-failure rate.

delay). Nemo introduces alternative paths to improve resilience. These alternative paths, with delays higher than or equal to the optimal and only choice in Nice, might introduce some latency penalties for packet delivery. As can be observed in the graph, the advantage of delivering more packets outweighs the potential latency cost, and Nemo suffers no noticeable additional delays for packets delivered without retransmission as seen on the left side of the plot. The wide-area graph confirms the trends observed through simulation. We see that the slope of the curve starting at 0.2 seconds latency is steeper to the right for Nemo, due in part to its fast packet recovery.

5. CONCLUSIONS

We have described Nemo, a highly resilient overlay multicast protocol for media streaming, and demonstrated the effectiveness of its approach through simulation and wide-area experimentation under different stress scenarios. Through the introduction of co-leaders and the use of triggered NACKs, Nemo achieves its resilience goals without costs in terms of duplicate packets and control-related traffic, and without sacrificing end-to-end latency. Nemo's probabilistic approach to overlay maintenance has shown to help handle the high levels of churn we expect to see in real deployments, while still resulting in efficiently optimized overlays.

REFERENCES

1. S. E. Deering, "Multicast routing in internetworks and extended LANs," in *Proc. of ACM SIGCOMM*, August 1988.
2. C. Diot, B. N. Levine, B. Lyles, H. Kassem, and D. Balensiefen, "Deployment issues for the IP multicast service and architecture," *IEEE Network* **14**, January/February 2000.
3. 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**, October 2002.
4. J. Jannotti, D. K. Gifford, K. L. Johnson, M. F. Kaashoek, and J. W. O'Toole Jr, "Overcast: Reliable multicasting with and overlay network," in *Proc. of the 4th USENIX OSDI*, October 2000.
5. P. Francis, "Yoid: Extending the Internet multicast architecture." http://www.aciri.org/yoid, April 2000.
6. Y. Chawathe, *Scattercast: an architecture for Internet broadcast distribution as an infrastructure service*. Ph.D. Thesis, U. of California, Berkeley, CA, Fall 2000.
7. S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "Scalable application layer multicast," in *Proc. of ACM SIGCOMM*, August 2002.
8. 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), 2002.
9. S. Ratnasamy, M. Handley, R. Karp, and S. Shenker, "Application-level multicast using content-addressable networks," in *Proc. of NGC*, November 2001.
10. V. N. Padmanabhan, H. J. Wang, and P. A. Chou, "Resilient peer-to-peer streaming," in *Proc. of IEEE ICNP*, 2003.
11. D. A. Tran, K. A. Hua, and T. Do, "ZIGZAG: An efficient peer-to-peer scheme for media streaming," in *Proc. of IEEE INFOCOM*, April 2003.

12. M. Bawa, H. Deshpande, and H. Garcia-Molina, "Transience of peers & streaming media," in *Proc. of HotNets-I*, October 2002.
13. F. E. Bustamante and Y. Qiao, "Friendships that last: Peer lifespan and its role in P2P protocols," in *Proc. of IWCW*, October 2003.
14. K. P. Gummadi, R. J. Dunn, S. Saroiu, S. D. Gribble, H. M. Levy, and J. Zahorjan, "Measurement, modeling and analysis of a peer-to-peer file-sharing workload," in *Proc. of ACM SOSP*, December 2003.
15. Y.-H. Chu, A. Ganjam, T. S. E. Ng, S. G. Rao, K. Sripanidkulchai, J. Zhan, and H. Zhang, "Early experience with an Internet broadcast system based on overlay multicast," in *Proc. of USENIX ATC*, June 2004.
16. S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz, "Handling churn in a DHT," in *Proc. of USENIX ATC*, December 2004.
17. T. Anderson, S. Shenker, I. Sotica, and D. Wetherall, "Design guidelines for robust Internet protocols," in *Proc. of HotNets-I*, October 2002.
18. X. Xu, A. Myers, H. Zhang, and R. Yavatkar, "Resilient multicast support for continuous-media applications," in *Proc. of NOSSDAV*, May 1997.
19. S. Banerjee, S. Lee, B. Bhattacharjee, and A. Srinivasan, "Resilient multicast using overlays," in *Proc. of ACM SIGMETRICS*, June 2003.
20. S. Banerjee and B. Bhattacharjee, "A comparative study of application layer multicast protocols," 2002. Submitted for review.
21. D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel, "ALMI: An application level multicast infrastructure," in *Proc. of USENIX USITS*, March 2001.
22. S. Paul, K. K. Sabnani, J. C.-H. Lin, and S. Bhattacharyya, "Reliable multicast transport protocol (RMTP)," *IEEE Journal on Selected Areas in Communication* **15**, April 1997.
23. B. N. Levine, D. B. Lavo, and J. J. Garcia-Luna-Aceves, "The case for reliable concurrent multicasting using shared ack trees," in *ACM Multimedia*, November 1996.
24. S. Floyd, V. Jacobson, C.-G. Liu, S. McCanne, and L. Zhang, "A reliable multicast framework for light-weight sessions and application level framing," *IEEE/ACM Transactions on Networking* **5**, December 1997.
25. B. N. Levine and J. Garcia-Luna-Aceves, "A comparison of reliable multicast protocols," *Multimedia Systems Journal* **6**, August 1998.
26. D. Towsley, J. F. Kurose, and S. Pingali, "A comparison of sender-initiated and receiver-initiated reliable multicast protocols," *IEEE Journal on Selected Areas in Communication* **15**, April 1997.
27. M. Yang and Z. Fei, "A proactive approach to reconstructing overlay multicast trees," in *Proc. of IEEE INFOCOM*, March 2004.
28. 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.
29. R. Mahajan, M. Castro, and A. Rowstron, "Controlling the cost of reliability in peer-to-peer overlays," in *Proc. of IPTPS*, February 2003.
30. Z. Wang and J. Crowcroft, "Bandwidth-delay based routing algorithms," in *Proc. of IEEE GlobeCom*, November 1995.
31. D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison Wesley, January 1989.
32. L. Peterson, T. Anderson, D. Culler, and T. Roscoe, "A blueprint for introducing disruptive technology into the Internet," in *Proc. of ACM HotNets-1*, October 2002.
33. C. Jin, Q. Chen, and S. Jamin, "Inet: Internet topology generator," Technical Report CSE-TR-433-00, U. of Michigan, Ann Arbor, MI, 2000.
34. E. W. Zegura, K. Calvert, and S. Bhattacharjee, "How to model an internetwork," in *Proc. of IEEE INFOCOM*, March 1996.
35. S. Birrer and F. E. Bustamante, "Resilient overlay multicast from the ground up," Tech. Report NWU-CS-03-22, Northwestern U., July 2003.
36. J. Xu, Z. Kalbarczyk, and R. K. Iyer, "Networked Windows NT system field failure data analysis," in *Proc. of PRDC*, December 1999.
37. D. L. Mills, "Improving algorithms for synchronizing computer network clocks," in *Proc. of ACM SIGCOMM*, August 1994.