NORTHWESTERN
UNIVERSITY

Computer Science Department

Technical Report
NWU-CS-03-22
December 13th, 2003

Resilient Peer-to-Peer Multicast from the Ground Up

Stefan Birrer and Fabián E. Bustamante

Abstract

One of the most important challenges of peer-to-peer multicast protocols is the ability to efficiently deal with the high degree of churn inherent to their environment. As multicast functionality is pushed to autonomous, unpredictable peers, significant performance losses can result from group membership changes and the higher failure rates of end-hosts when compared to routers. Achieving high delivery ratios without sacrificing end-to-end latencies or incurring additional costs has proven to be a challenging task.

This paper introduces Nemo, a novel peer-to-peer multicast protocol that aims at achieving this elusive goal. We present an extensive comparative evaluation of our protocol through simulation and wide-area experimentation. We compare the performance of Nemo with that of three alternative protocols: Narada, Nice and Nice-PRM. Our results show how Nemo can achieve delivery ratios similar to those of comparable protocols (up to 99.98%) under different failure rates, but at a fraction of their cost in terms of duplicate packets (reductions > 85%) and control-related traffic.

Keywords: Peer-to-peer, overlay multicast, resilience, churn.

Resilient Peer-to-Peer Multicast from the Ground Up

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

December 13th, 2003

Abstract

One of the most important challenges of peer-to-peer multicast protocols is the ability to efficiently deal with the high degree of churn inherent to their environment. As multicast functionality is pushed to autonomous, unpredictable peers, significant performance losses can result from group membership changes and the higher failure rates of end-hosts when compared to routers. **Achieving high delivery ratios without sacrificing end-to-end latencies or incurring additional costs has proven to be a challenging task.**

This paper introduces Nemo, a novel peer-to-peer multicast protocol that aims at achieving this elusive goal. We present an extensive comparative evaluation of our protocol through simulation and wide-area experimentation. We compare the 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 (up to 99.98%) under different failure rates, but at a fraction of their cost in terms of duplicate packets (reductions $> 85\%$) and control-related traffic.

1 Introduction

Multicast is an efficient mechanism to support group communication. 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, promising to make possible large scale multi-party applications such as audio and video conference, research collaboration and content distribution. More of a decade after first being proposed, however, IP Multicast [14] is not yet widely available due in part to a number of both technical and non-technical issues [15].

In recent years a number of researchers have proposed an alternate, peer-to-peer architecture for supporting group communication applications over the Internet. In this middleware, or application-layer, approach participating peers organize themselves into an overlay topology for data delivery. The topology is an overlay in the sense that each edge corresponds to a unicast path between two

end systems or peers in the underlying Internet. All multicast related functionality is implemented at the peers instead of at routers, and the goal of the multicast protocol is to construct and maintain an efficient overlay for data transmission.

Despite the undeniable advantages of this approach, a number of issues need to be addressed if it is to become a practical alternative to IP Multicast [13, 5]. First, application-layer multicast can result in higher stress on the network, as it is impossible to completely prevent multiple overlay edges from traversing the same physical link. Second, communication between peers may involve visiting other peers, possibly resulting in higher latencies. Lastly, as multicast functionality is pushed to autonomous, unpredictable peers, significant performance loss can result from the higher degree of transiency (a.k.a. *churn*) of end hosts when compared to routers.

Efficiently handling the inherent high degree of churn on peer populations may well be the primary challenge for P2P architectures [5]. Measurement studies of widely used P2P systems have reported *median session times* [1] ranging from an hour to a minute [8, 18, 29]. Achieving high delivery ratios under these conditions without sacrificing end-to-end latencies or incurring additional costs has proven to be a difficult task.

This paper introduces 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 [1], achieving, in a cost-effective manner, *performance characteristics resilient to the natural instability of its target environment.*

Simulation-based and wide-area experimentations show how Nemo can achieve high delivery ratios (up to 99.98%) and low end-to-end latency similar to those of comparable protocols, while significantly reducing the cost in terms of duplicate packets (reductions $> 85\%$) and control related traffic, making the proposed algorithm a more scalable solution to the problem.

We introduce Nemo's approach to resilient multicast and present its operational details in Section 2. Section 3 describes our experimental setup and reports the results from simulation and wide-area experiments. We discuss related work and conclude in Sections 4 and 5.

2 Nemo's Approach

Nemo follows the *implicit approach* [3, 10, 28, 36] to building a resilient overlay for multicasting: it organizes the participating peers into a control topology and implicitly defines the data delivery network based on a set of forwarding rules which we will describe shortly.

The set of communication peers are organized into clusters based on network proximity [2], where every peer is 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 the *degree*. Each of these clusters selects a *leader* that becomes a member of the immediately superior layer. In part to avoid the dependency on a single node, every

[1] *Session time* is defined as the time between when a peer joins and leaves the network.

[2] Other factors such as bandwidth [32, 13] and expected peer lifetime [8] could be easily incorporated.

cluster leader recruits a number of co-leaders with whom it form the crew. The process is repeated, with all peers in a layer being grouped into clusters from which leaders are selected to participate in the next higher layer. Hence peers can lead more than one cluster in successive layers of this logical hierarchy.[3]

Our work focuses on improving the resilience of peer-to-peer overlay multicast systems. The following 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. We introduce then the concepts of crews and co-leaders and elaborate on the algorithm for data forwarding and retransmission.

2.1 Member Join and Departure

A new peer joins the multicast group by querying a rendezvous point for the IDs of the members on the top layer. Starting there and in an iterative manner, the incoming peer continues (i) requesting the list of members at the current layer from the cluster's leader, (ii) selecting among them who to contact next based on the result from a given cost function and (iii) decreasing the layer count. When the new peer finds the bottom layer leader with minimal cost [4], it joins the associated cluster.

To deal with dynamic changes in the underlying network, every peer periodically checks the members of the next higher layer and switches clusters if another peer is closer than the current one (thresholds are used to prevent instabilities).

Members can leave Nemo in announced (graceful) or unannounced manner. Common members, without responsibilities towards other peers, can simply leave the group after informing their cluster's leader. Leader, on the other hand, must first elect replacement leaders for all clusters they own; they can then leave their top layer after informing the clusters' leaders.

To detect unannounced leaves, Nemo relies on heartbeats exchanged among the cluster's peers at regular intervals. Unreported members are given a fixed time interval, *grace period*, before being considered dead. Once a member is determined dead, a repair algorithm is initiated. If the failed peer happens to be a leader, the tree itself must be fixed: the members of the victim's cluster must elect the replacement leader from among themselves.

2.2 Planning for Node Failure

Tree-based overlay multicast protocols have proven to be highly scalable and efficient in terms of physical link stress, state and control overhead, and end-to-end latency [19, 3, 12]. As other tree-based structures, however, these proposed protocols have an inherent problem of resilience from their dependence on the reliability of non-leaf nodes [5].

[3] This is common to both Nemo and Nice [3] as well as Zigzag [31]; the degree bounds have been chosen to help reduce oscillation in clusters.

[4] In our current implementation, we use proximity as the cost function for joining.

Nemo addresses the resilience problem of tree-based overlay multicast protocols through the introduction of *co-leaders*. Every cluster leader recruits a number of co-leaders with whom it forms the *crew*. Crew lists are periodically distributed to the cluster's members. Co-leaders improve the resilience of the multicast group by avoiding dependencies on single nodes and providing alternative paths for data forwarding. In addition, crew members share the load from message forwarding, thus improving scalability. Figure 1 illustrates the logical organization of Nemo.

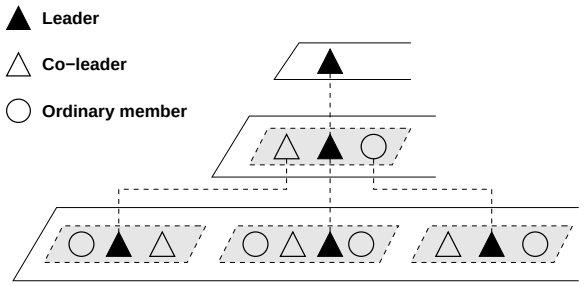


Figure 1: Nemo's logical organization. The shape illustrates only the role of a peer within a cluster: a leader of a cluster at a given layer can act as co-leader or ordinary member at the next higher layer.

2.3 Data Forwarding

As with other protocols following a similar approach [3, 10, 28, 36, 31], Nemo's data delivery topology is implicitly defined by the set of packet-forwarding rules adopted. 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 among all crew members in the cluster's leader group. The algorithm is summarized in Figure 2.

Figure 3 illustrates the data forwarding algorithm using the logical topology from Figure 1. Although we employ leaders for this example, the explanation is valid when routing instead 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 layer and the members of the second lowest layer. In the last step, the leader of the cluster on the left has to forward the packet to its members.

To illustrate Nemo's resilience to the failure of peers, Figure 4 shows an example of the forwarding algorithm in action. The forwarding responsibility is evenly shared among the leaders by alternating the message recipient among them. In case of a failed crew member, the remaining leaders can still forward their share of messages through the tree.

```
FORWARD-DATA(msg)
 1  R ← ∅
 2  if leader ∉ msg.sender_crew
 3    then R ← R ∪ leader
 4  for each child in children
 5  do if child ∉ msg.sender_crew
 6        then R ← R ∪ child
 7  SEND(msg, R, sender_crew ← crewOf(self))
 8  if isCrewMember(self) and leader ∉ msg.sender_crew
 9    then R ← ∅
10         R ← R ∪ super_leader
11         for each neighbor in neighbors
12         do R ← R ∪ neighbor
13         SEND(msg, R, sender_crew ← crewOf(leader))
```

Figure 2: Data Forwarding Algorithm: SEND transmits a packet to a list of nodes, selecting the real destination among the crew members associated with the given destination.

2.4 Data Retransmission

Similar to other protocols aiming at high resilience [26, 4], Nemo relies on sequence numbers and triggered NACKs to detect lost packets.

Every peer piggybacks a bit-mask with each data packet indicating the previously received packets. Each peer also maintains a cache of received packets and a list of missing ones. 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, and in our current is set to $2 \cdot RTT$ to the furthest known crew member. For each missing packet, a NACK is sent to a peer caching it, and a different timeout is set for retransmission.

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 5) 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 helps reduce the number of duplicate packets generated indirectly by packet losses.

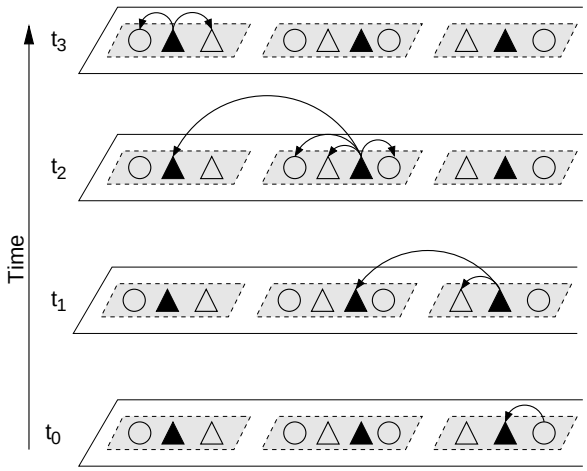


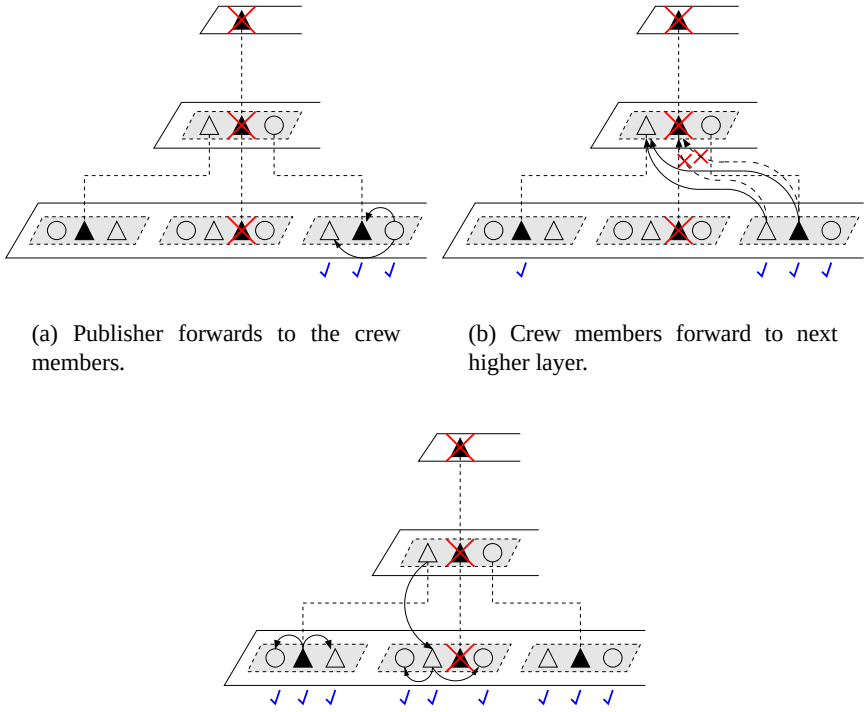
Figure 3: Basic data forwarding in Nemo. Each row corresponds to one time step.

Initially, the peer sets the forwarding direction to *upward* and *downward* (line 1). It then checks all recently received packets inside a *for*-loop (lines 2-7). If the packets belong to the same stream, the peer uses the packet's sender crew to reduce 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 needs only be transmitted to the peers up in the tree. Alternatively, when the packet's sender was an ancestor in the tree, the packet needs only be relay downwards.

3 Evaluation

We analyze the performance of Nemo using detailed simulation and wide-area experimentation. We compare Nemo's performance to that of three other protocols – Narada [13], Nice [3] and Nice-PRM [4] – both in terms of application performance and protocol overhead. Application performance is captured by delivery ratio and end-to-end latency, while overhead is evaluated in terms of number of duplicate packets.

- *Delivery Ratio:* Ratio of subscribers which have received a packet within a fixed time window. Disabled receivers are not accounted for.
- *End-to-End Latency:* End-to-end delay (including retransmission time) from the source to the receivers, as seen by the application. This includes path latencies along the overlay hops, as well as queueing delay and processing overhead at peers along the path.
- *Duplicate Packets:* Number of duplicate packets for all receivers counted per sequence number, reflecting an unnecessary burden on the network. Packets arrived outside of the delivery



(a) Publisher forwards to the crew members.

(b) Crew members forward to next higher layer.

(c) Co-leader forwards to all other clusters, including its own.

Figure 4: Data forwarding in Nemo with a failed node: All nodes are able to receive the forwarded data despite a node failure. Note how a sender alternates the packet destination among the crew members.

window are accounted for as duplicates, since the receiver already assumed them as lost.

It is worth noticing that the implementation of all evaluated protocols is shared between the wide-area and the simulation experiments. We achieve this by abstracting the protocols' logic from the environment-dependent functionality through well-defined interfaces. Thus, for any given protocol, the difference in the code-base of the wide-area and the simulator implementations is limited to how the communication between two agents is realized.

```
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 ∩ ¬ UPWARD
6             if isLeader(packet.sender_crew)
7                then d ← d ∩ ¬ DOWNWARD
8  FORWARD(msg, direction ← d)
```

Figure 5: Data Recovery Algorithm: The function FORWARD forwards a packet as specified by a logical direction (UPWARD and/or DOWNWARD).

3.1 Details on Protocol Implementations

For each of the three alternative protocols, the values for the available parameters were obtained from the corresponding literature [13, 3, 4].

For Narada [13], we employ the latency-only scheme for constructing the overlay. For Nice [3] and Nice-PRM [4], the cluster degree, k, is set to 3. We used PRM-(3,0.01), PRM-(3,0.02) and PRM-(3,0.03) with three random peers chosen by each node, and with one, two, and three percent forwarding probability. We employ a grace period of 15 seconds.

For Nemo, the cluster degree k and the crew size are set to 3. [5] The grace period is set to 15 seconds.

For the wide-area implementation, we use UDP with retransmissions: ten attempts for heartbeats and five for all other control traffic. Data communication does not rely on retransmissions.

3.2 Experimental Setup

We performed our evaluation through detailed simulation using a locally written, packet-level, event-based simulator and wide-area experimentation on PlanetLab, a world-wide deployed set of nodes for experimentation with distributed systems.

We run our simulations using Inet topologies [20] with 3,072 nodes and a multicast group of 512 members. [6] 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 sec. of the simulation. A warm-up time of 300 sec. is omitted from the figures. Starting at 600 sec. and lasting for about 1200 sec., each simulation has a phase with rapid membership

[5] We are exploring the tradeoffs for different cluster sizes.

[6] Comparable results were obtained using alternative topologies including Transit-Stub and AS Waxman, and different group sizes [6].

changes. During this time each protocol is exercised under two different failure rates derived from Xu et al. [33] study on networked system failure and related research on resilient multicast [4, 33]. Under a *high-failure rate*, nodes fail independently at a time sampled from an exponential distribution with mean, *Mean Time To Failure*, equal to 5 min., to rejoin shortly after (time sampled from an exponential distribution with mean, *Mean Time To Repair*, equal to 2 min.). The same set of simulations is also run with a *low-failure* rate given defined by MTTF of 60 min. and a MTTR of 10 min. The means for each failure rate are chosen asymmetrically to allow, on average, 5/7 of all members to be up during this phase.

For the wide-area experiments we restrict our comparison to Nice, Nice-PRM and Nemo with between 60 and 200 members distributed across $\sim$90 sites. The number of members per site varies from 1 to $\sim$12, with most sites having two members. We inject failures at the *high-failure* defined used for simulation ($MTTF = 5\ min$ and $MTTR = 2\ min$). Each protocol runs for 30 min. with a 10-min. period with failures. The order of the three protocols is randomly chosen.

To estimate the end-to-end delay, we make use of a global time server. Every peer estimates the difference of its local time to the time at the server. The algorithm is inspired by [23] and leads to sufficient accuracy for our application.

All experiments were run with a payload of 100B . We opted for this relatively small packet size 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 at a rate of 10 packets per second. This corresponds to the usage of a 3.2-second buffer, which is a realistic scenario for applications such as multimedia streaming.

3.3 Experimental Results

The remainder of this section presents and discusses results from simulation and wide-area evaluations.

3.3.1 Simulation Results

For all simulation results, each data point is the mean of 25 independent runs.

The delivery ratio during the group-membership-change phase is shown in Figure 6. The first graph shows the delivery ratio of Narada, followed by Nice, Nice-PRM(3,0.01) and Nemo. As the figure illustrates, Nemo has a higher delivery ratio than the alternative protocols. The alternate data paths in Nemo explain these improvements, as they enable the early detection and retransmission of lost packets within the allowed timeout interval.

The summarized results for both failure rates (high and low) are presented in Table 1 and 2, respectively. Nice-PRM(3,0.02), Nice-PRM(3,0.03) and Nemo achieve comparably high delivery ratio, significantly better than Nice and Narada.

The cost of a resilient multicast protocol can be measured in terms of duplicate packets per sequence number. The second set of columns in Tables 1 and 2 show this overhead for the com-

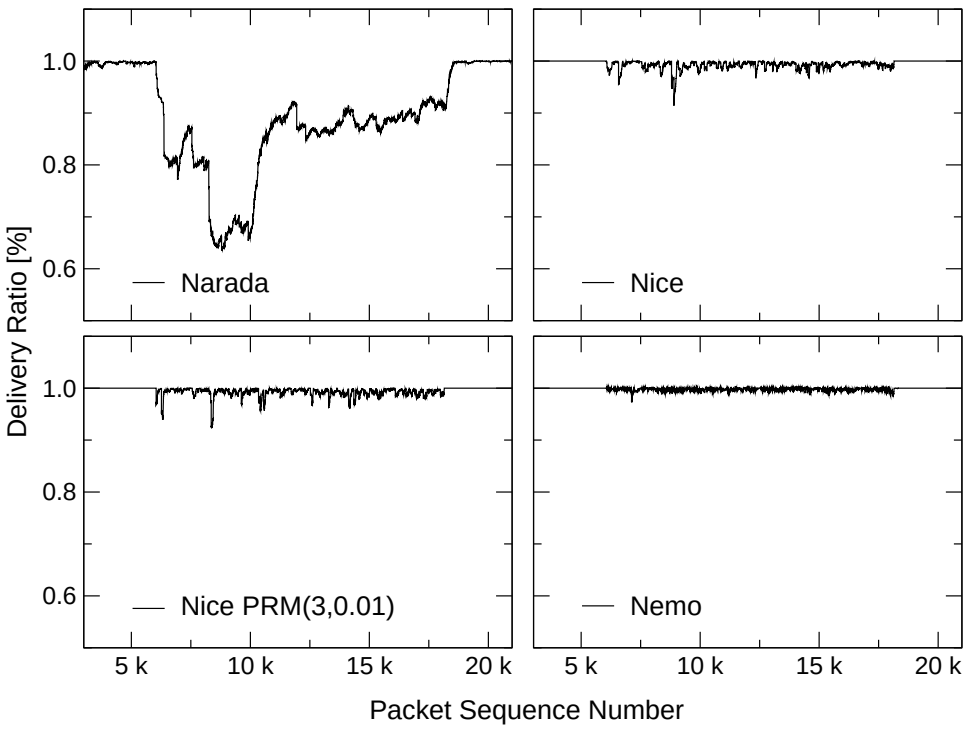


Figure 6: Delivery ratio (512 end hosts, high failure rate).

pared protocols. In both cases, Nemo's approach results in comparably high delivery ratios with significantly lower cost in terms of duplicates.

Under both failure rates, Nice-PRM incurs a higher number of duplicate packets than Nemo and Nice as a result of PRM's proactive randomized forwarding. As can be seen from the alternative PRM configurations, the number of duplicate packets correlates well with its delivery ratio.

Nemo's higher resilience with low overhead comes at no cost in terms of latency, as can be observed in Figure 7. The graph shows the cumulative distribution function (CDF) of latency for all received packets with a buffer of 32 packets, which corresponds to 3.2-second delay.

Remember that 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. However, the advantage of delivering more packets outweighs this disadvantage, and Nemo suffers no noticeable additional delays for packets delivered without retransmission as seen on the left side of the plot in Figure 7.

Protocol	Delivery ratio		Duplicate packets		
	Mean	Std	Mean	Max	Std
Nemo	0.998	0.89E-3	3.16	3.77	0.29
Nice-PRM(3,0.01)	0.993	1.25E-3	12.47	14.43	1.04
Nice-PRM(3,0.02)	0.994	1.23E-3	18.20	19.75	0.77
Nice-PRM(3,0.03)	0.994	1.01E-3	24.22	28.14	1.82
Nice	0.992	1.95E-3	7.10	8.32	0.71
Narada	0.852	60.3E-3	0.00	0.00	0.00

Table 1: High-Failure Rate (512 end hosts).

Protocol	Delivery ratio		Duplicate packets		
	Mean	Std	Mean	Max	Std
Nemo	1.000	0.12E-3	0.34	0.59	0.09
Nice-PRM(3,0.01)	0.999	0.56E-3	6.42	6.98	0.38
Nice-PRM(3,0.02)	0.999	0.36E-3	12.00	13.43	0.66
Nice-PRM(3,0.03)	0.999	0.27E-3	16.74	18.42	0.65
Nice	0.999	0.52E-3	1.29	1.92	0.40
Narada	0.950	38.3E-3	0.00	0.00	0.00

Table 2: Low Failure Rate (512 end hosts).

3.3.2 Wide-Area Results

The results presented here are based on twenty-five runs, each a set of one experiment per protocol. We took several measurements at different times of the day and present representative graphs for Nemo, Nice and Nice-PRM(3,0.02). We chose 2% forwarding probability for Nice-PRM, since this offers the best tradeoff between high delivery ratio and low number of duplicate packets.

Figure 8 shows the delivery ratio of one run each; the packet losses observed during the warm-up interval are due to the non-deterministic influence of the environment. The graphs confirm the simulation results.

Protocol	Delivery ratio		Duplicate packets		
	Mean	Std	Mean	Max	Std
Nemo	0.979	0.010	1.27	2.53	0.56
Nice-PRM(3,0.02)	0.953	0.024	2.02	3.00	0.57
Nice	0.939	0.032	1.06	1.83	0.47

Table 3: Wide-Area Results with High Failure Rate (PlanetLab, $\sim$72 end hosts). The statistics include the packets with sequence numbers from 6,000 to 12,000.

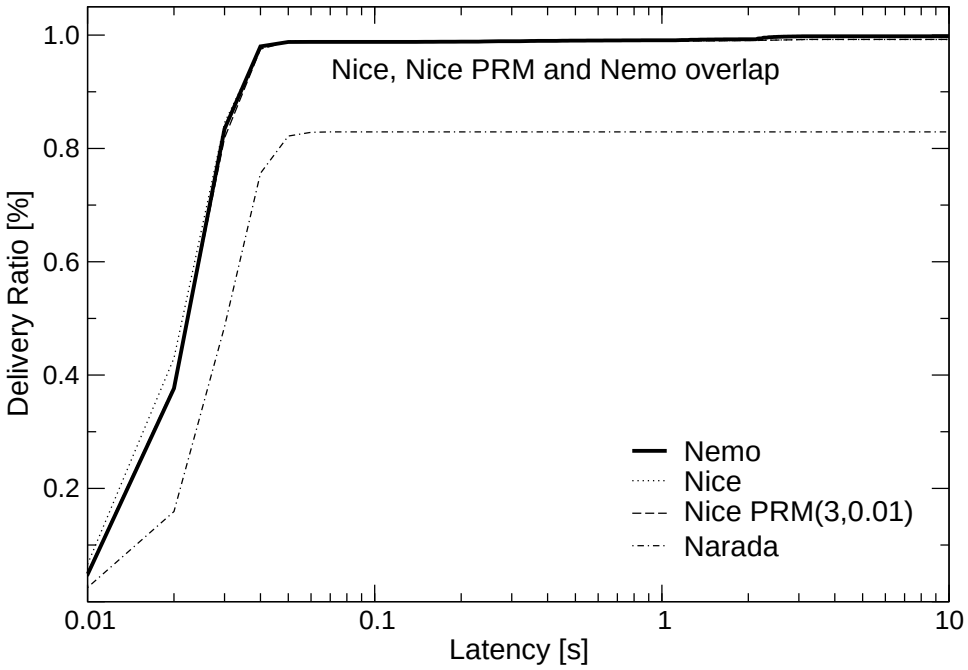


Figure 7: Latency CDF (512 end hosts, high failure rate).

In Table 3 we summarize the runs of the wide area experiment on PlanetLab. Nemo has a substantially higher delivery ratio than Nice-PRM, while incurring less duplicate packets.

We show the latency distribution achieved in one wide area experiment in Figure 9. The data sets correspond to the plots shown in Figure 8. The three graphs confirm the data gathered through simulation, although the experienced latencies are slightly higher in the wide area experiment (due in part to user-level processing time where limited CPU allocation decreases throughput). Nemo outperforms Nice on the latency of packets requiring retransmission, as Nemo's alternate data paths translate into earlier packet-loss detection and faster recovery. We see that the slope of the plot starting at 0.2 seconds latency is steeper to the right for Nemo, which is partially due to recovered packets.

4 Related Work

All peer-to-peer or application-layer multicast protocols organize the participating peers in two topologies: a control topology for group membership related tasks, and a delivery tree for data forwarding. Available protocols can be classified based on the sequence adopted for their construc-

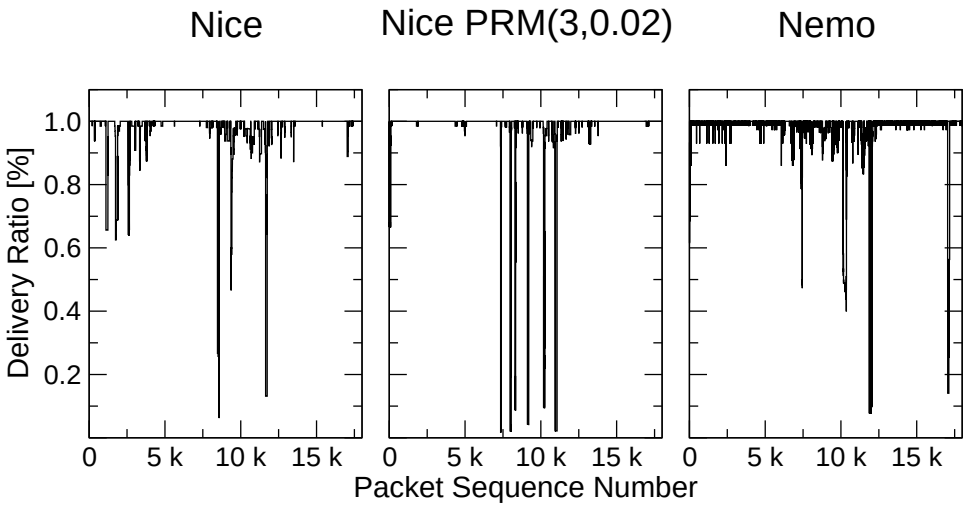


Figure 8: Delivery ratio (PlanetLab, ∼72 end hosts, high failure rate).

tion [2, 13]. In a tree-first approach [17, 19, 27], peers directly construct the data-delivery tree by selecting their parents from among known peers. Additional links are later added to define, in combination with the data-delivery tree, the control topology. With a mesh-first approach [13, 11], peers build a more densely connected graph (mesh) over which (reverse) shortest path spanning trees, rooted at any peer, can be constructed. Protocols adopting an implicit approach [3, 10, 28, 36] 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. [3] introduce Nice and demonstrate the effectiveness of overlay multicast across large scale networks. The authors also 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 such as its hierarchical control topology. Nemo introduces co-leaders to improve the resilience of the overlay.

A large number of research projects have addressed reliable and resilient multicast at the network layer [26, 34, 35, 25, 22, 16]. A comparative survey of these protocols is given in [21, 30]. Like many of them, Nemo relies on reactive techniques to recover from packet losses. STORM [34] uses hierarchical NACKs for recovery: NACKs are sent to parents (obtained from a parent list) until the packet is successfully recovered or deemed obsolete. In the case of Nemo, NACKs are used only to request missing packets from neighbors who indicated[7] to cache them locally.

In the context of overlay multicast, a number of protocols have been proposed to improve resilience [4, 9, 31, 24]. ZigZag [31] is a single source P2P streaming protocol. Resilience is achieved by separating the control and data delivery trees at every level, with one peer being held responsible

[7]Peers distribute the local cache state with every data packet they send.

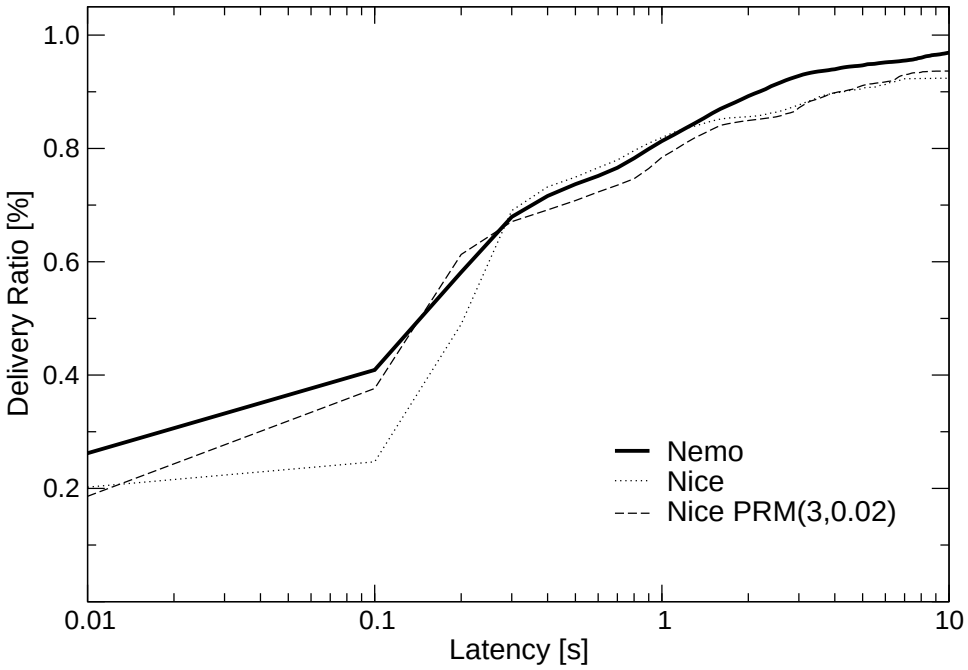


Figure 9: Latency CDF (PlanetLab, ∼72 end hosts, high failure rate).

for the organization of the sub-tree and a second one dealing with data forwarding. In the presence of failures, both peers share repair responsibilities. In Nemo, the forwarding responsibility of a peer is shared among its crew members and its repair algorithm is fully distributed among cluster members. PRM [4] uses randomized forwarding and NACK-based retransmission to improve resilience. In contrast, Nemo relies on the concept of a *crew* and opts only for deterministic techniques for data forwarding. SplitStream [9] and CoopNet [24] improve resilience by building several disjoint trees. In addition, CoopNet adopts a centralized organization protocol and relies on Multiple Description Coding (MDC) 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 use of leaders and co-leaders. We are exploring the use of data redundancy using forward error correction (FEC) encoding [7].

5 Conclusions

We present Nemo, a new overlay multicast protocol designed for high resiliency from the ground up. Through the introduction of co-leaders to minimize dependencies and the use of triggered NACKs to detect lost packets, Nemo is able to achieve high delivery ratios under high stress at a lower

cost in terms of duplicate messages than alternative protocols and without penalties in terms of additional delays. Detailed simulation and wide-area experimentation results presented have shown the advantages of this approach.

Acknowledgments

We would like to thank Karsten Schwan and Peter Dinda, who kindly loaned us their equipment for some of our experiments. We are also grateful to Jeanine M. Casler, Yi Qiao, Jason Skicewicz, Ananth Sundararaj and Dong Lu for their helpful comments on early drafts of this paper.

References

[1] ANDERSON, T., SHENKER, S., SOTICA, I., AND WETHERALL, D. Design guidelines for robust Internet protocols. In *Proc. of HotNets-I* (October 2002).

[2] BANERJEE, S., AND BHATTACHARJEE, B. A comparative study of application layer multicast protocols, 2002. Submitted for review.

[3] BANERJEE, S., BHATTACHARJEE, B., AND KOMMAREDDY, C. Scalable application layer multicast. In *Proc. of ACM SIGCOMM* (August 2002).

[4] BANERJEE, S., LEE, S., BHATTACHARJEE, B., AND SRINIVASAN, A. Resilient multicast using overlays. In *Proc. of ACM SIGMETRICS* (June 2003).

[5] BAWA, M., DESHPANDE, H., AND GARCIA-MOLINA, H. Transience of peers & streaming media. In *Proc. of HotNets-I* (October 2002).

[6] BIRRER, S., AND BUSTAMANTE, F. E. Resilient overlay multicast from ground up. Tech. Report NWU-CS-03-22, Northwestern U., July 2003.

[7] BLAHUT, R. E. *Theory and Practice of Error Control Codes*. Addison Wesley, 1994.

[8] BUSTAMANTE, F. E., AND QIAO, Y. Friendships that last: Peer lifespan and its role in P2P protocols. Tech. Report NWU-CS-03-21, Northwestern U., June 2003.

[9] CASTRO, M., DRUSCHEL, P., KERMARREC, A.-M., NANDI, A., ROWSTRON, A., AND SINGH, A. Splitstream: High-bandwidth multicast in cooperative environments. In *Proc. of the 19th ACM SOSP* (October 2003).

[10] CASTRO, M., ROWSTRON, A., KERMARREC, A.-M., AND DRUSCHEL, P. SCRIBE: A large-scale and decentralised application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communication 20*, 8 (2002).

[11] CHAWATHE, Y. *Scattercast: an architecture for Internet broadcast distribution as an infrastructure service*. Ph.D. Thesis, U. of California, Berkeley, CA, Fall 2000.

[12] CHU, Y.-H., GANJAM, A., NG, T. S. E., RAO, S. G., SRIPANIDKULCHAI, K., ZHAN, J., AND ZHANG, H. Early experience with an Internet broadcast system based on overlay multicast. In *Proc. of USENIX Annual Technical Conference* (June 2004).

[13] CHU, Y.-H., RAO, S. G., SESHAN, S., AND ZHANG, H. A case for end system multicast. *IEEE Journal on Selected Areas in Communication 20*, 8 (October 2002).

[14] DEERING, S. E. Multicast routing in internetworks and extended LANs. In *Proc. of ACM SIGCOMM* (August 1988).

[15] DIOT, C., LEVINE, B. N., LYLES, B., KASSAN, H., AND BALENSIEFEN, D. Deployment issues for the IP multicast service and architecture. In *IEEE Networks special issue on multicasting* (2000).

[16] FLOYD, S., JACOBSON, V., LIU, C.-G., MCCANNE, S., AND ZHANG, L. A reliable multicast framework for light-weight sessions and application level framing. *IEEE/ACM Transactions on Networking 5*, 6 (December 1997).

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

[18] GUMMADI, K. P., DUNN, R. J., SAROIU, S., GRIBBLE, S. D., LEVY, H. M., AND ZAHORJAN, J. Measurement, modeling and analysis of a peer-to-peer file-sharing workload. In *Proc. of ACM SOSP* (December 2003).

[19] JANNOTTI, J., GIFFORD, D. K., JOHNSON, K. L., KAASHOEK, M. F., AND O'TOOLE JR, J. W. Overcast: Reliable multicasting with and overlay network. In *Proc. of the 4th USENIX OSDI* (October 2000).

[20] JIN, C., CHEN, Q., AND JAMIN, S. Inet: Internet topology generator. Technical Report CSE-TR-433-00, U. of Michigan, Ann Arbor, MI, 2000.

[21] LEVINE, B. N., AND GARCIA-LUNA-ACEVES, J. A comparison of reliable multicast protocols. *Multimedia Systems Journal 6*, 5 (August 1998).

[22] LEVINE, B. N., LAVO, D. B., AND GARCIA-LUNA-ACEVES, J. J. The case for reliable concurrent multicasting using shared ack trees. In *ACM Multimedia* (November 1996).

[23] MILLS, D. L. Improving algorithms for synchronizing computer network clocks. In *Proc. of ACM SIGCOMM* (August 1994).

[24] PADMANABHAN, V. N., WANG, H. J., AND CHOU, P. A. Resilient peer-to-peer streaming. In *Proc. of IEEE ICNP* (2003).

[25] PAPADOPOULOS, C., PARULKAR, G. M., AND VARGHESE, G. An error control scheme for large-scale multicast applications. In *Proc. of IEEE INFOCOM* (March 1998).

[26] PAUL, S., SABNANI, K. K., LIN, J. C.-H., AND BHATTACHARYYA, S. Reliable multicast transport protocol (RMTP). *IEEE Journal on Selected Areas in Communication 15*, 3 (April 1997).

[27] PENDARAKIS, D., SHI, S., VERMA, D., AND WALDVOGEL, M. ALMI: An application level multicast infrastructure. In *Proc. of USENIX USITS* (March 2001).

[28] RATNASAMY, S., HANDLEY, M., KARP, R., AND SHENKER, S. Application-level multicast using content-addressable networks. In *Proc. of NGC* (November 2001).

[29] RHEA, S., GEELS, D., ROSCOE, T., AND KUBIATOWICZ, J. Handling churn in a DHT. Tech. Rep. UCB/CSD-03-1299, Computer Science Division, U. of California, Berkeley, 2003.

[30] TOWSLEY, D., KUROSE, J. F., AND PINGALI, S. A comparison of sender-initiated and receiver-initiated reliable multicast protocols. *IEEE Journal on Selected Areas in Communication 15*, 3 (April 1997).

[31] TRAN, D. A., HUA, K. A., AND DO, T. ZIGZAG: An efficient peer-to-peer scheme for media streaming. In *Proc. of IEEE INFOCOM* (April 2003).

[32] WANG, Z., AND CROWCROFT, J. Bandwidth-delay based routing algorithms. In *Proc. of IEEE GlobeCom* (November 1995).

[33] XU, J., KALBARCZYK, Z., AND IYER, R. K. Networked Windows NT system field failure data analysis. In *Proc. of PRDC* (December 1999).

[34] XU, X. R., MYERS, A. C., ZHANG, H., AND YAVATKAR, R. Resilient multicast support for continuous-media applications. In *Proc. of NOSSDAV* (May 1997).

[35] YAVATKAR, R., GRIFFOEN, J., AND SUDAN, M. A reliable dissemination protocol for interactive collaborative applications. In *ACM Multimedia* (November 1995).

[36] ZHUANG, S. Q., ZHAO, B. Y., JOSEPH, A. D., KATZ, R. H., AND KUBIATOWICZ, J. D. Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination. In *Proc. of NOSSDAV* (June 2001).