FatNemo: Building a Resilient Multi-Source Multicast Fat-Tree

Stefan Birrer, Dong Lu, Fabián E. Bustamante, Yi Qiao, and Peter Dinda

Northwestern University, Evanston IL 60201, USA,
{sbirrer,donglu,fabianb,yqiao,pdinda}@cs.northwestern.edu

Abstract. FatNemo is a novel scalable peer-to-peer multi-source multicast protocol based on the idea of fat-trees. In fat-trees the available bandwidth increases as one moves up the tree, yielding a minimal mean and standard deviation of the response time. For many-to-many multicast applications, such as video conferencing, this eliminates the bottlenecks inside the overlay network.
FatNemo organizes its members into a tree of clusters. Starting at the lowest tree layer, peers can be members of multiple successive layers. Bandwidth capacity is used to decide the highest layer a peer can participate in. The size of the cluster is increased as we go up the tree. FatNemo relies on co-leaders to balance the load and to increase its resilience to path and end host failures.
We present an evaluation of our protocol using simulation, comparing its performance with that of alternative protocols (Narada, Nice and Nice-PRM). Our results show that FatNemo not only minimizes the average and the standard deviation of the response time, but also handles end host failures gracefully without suffering a performance penalty.

1 Introduction

High bandwidth multi-source multicast among widely distributed nodes is a critical capability for a wide range of important applications including audio and video conferencing, multi-party games and content distribution.

Throughout the last decade, a number of research projects have explored the use of multicast as an efficient and scalable mechanism to support such group communication. Multicast 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. However, the limited deployment of IP Multicast [19, 20], a best effort network layer multicast protocol, has led to considerable interest in alternate approaches that are implemented at the application layer, using only end-systems [17, 28, 24, 42, 13, 3, 12, 43, 56, 39, 47].

In an end-system multicast approach participating peers organize themselves into an overlay topology for data delivery. Each edge in the topology 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 in the routers of the underlying network. The goal of the multicast protocol is to construct and maintain an efficient overlay for data transmission.

Among the end-system multicast protocols that have been proposed, tree-based systems have proven to be highly scalable and efficient in terms of physical link stress, state and control overhead, and end-to-end latency. However, normal tree structures have two inherent problems: (i) *resilience:* they are highly dependent on the reliability of non-leaf nodes, and (ii) *bandwidth limitations:* they are likely to be bandwidth constrained [1] as bandwidth availability monotonically decreases as one descends into the tree.

Resilience is particularly relevant to the application-layer approach, as the trees here are composed of autonomous, unpredictable end hosts. The high degree of transiency of the hosts [2] has been pointed out as one of the main challenges for these architectures [6].

The bandwidth limitations of normal tree structures is particularly problematic for multi-source, bandwidth intensive applications. For a set of randomly placed sources in a tree, higher level paths (those nearer the root) will become the bottleneck and tend to dominate response times. Once these links become heavily loaded or overloaded, packets will start to be buffered or dropped.

We have addressed the resilience issue of tree-based systems in previous work [7] through the introduction of *co-leaders* and the reliance on *triggered negative acknowledgements* (NACKs). In this paper we address the bandwidth limitations of normal tree overlays.

Our approach capitalizes on Leiserson's seminal work on fat-trees [31]. Paraphrasing Leiserson, a fat-tree is like a real tree in that its branches become thicker the further we get from the leaves. Because its links become fatter as one moves closer the root, a fat-tree overcomes the "root bottleneck" of a regular tree. Figure 1 shows a schematic example of a binary fat-tree. We propose to organize participant end-systems in a tree that closely resembles a Leiserson fat-tree by dynamically placing higher degree nodes (nodes with higher bandwidth capacity) close to the root and increasing the cluster sizes as one ascends the tree.

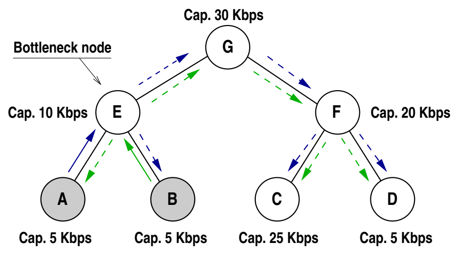
This paper makes three main contributions:

- We introduce the use of Leiserson fat-trees for application-layer multi-source multicast, overcoming the inherent bandwidth limitations of normal tree-based overlay structures.
- We describe the design and implementation of *FatNemo*, a new application-layer multicast protocol that builds on this idea.
- We evaluate the FatNemo design in simulation, illustrating the benefits of a fat tree approach compared to currently popular approaches to application-layer multicast.

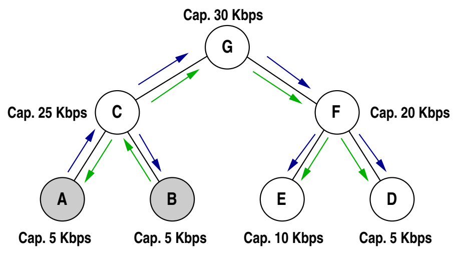
The remainder of this paper is structured as follows. We present some background and review related work in Section 2. Section 3 outlines the FatNemo's approach and presents its design and operational details. In Sections 4 and 5, we examine FatNemo's ability to build an efficient, bandwidth-optimized overlay network for application layer multi-source multicast. We conclude in Section 6 and outline some future work directions.

[1] The access link of a end system becomes its bandwidth bottleneck, thus we can model the bandwidth capacity as a property of the end-system.

[2] Measurement studies of widely used application-layer/peer-to-peer systems have reported median session times ranging from an hour to a minute [9, 25, 44, 15].



(a) Normal tree.



(b) Fat-tree.

Fig. 1. Two binary trees with nodes A and B as sources, publishing at 5Kbps each. On the left, a normal binary tree where node E becomes the bottleneck, resulting on a reduced (dash line) outgoing stream quality. On the right, a fat-tree with higher capacity nodes placed higher in the tree.

2 Related Work

We first discuss the idea of fat trees in the context of parallel architectures, where they were first introduced. FatNemo builds on research in live streaming and file distribution and we discuss each in turn.

The concept of fat-trees, first introduced by Leiserson [31], has been successfully applied in massively parallel systems, such as thinking machine CM-5 [32] and Meiko CS-2 [26]. They have also been applied in high performance cluster computing, for example, Infiniband architecture [27] can support a fat-tree topology.

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 according to the sequence adopted for their construction [2, 17]. In a tree-first approach [24, 28, 42], 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 [17, 13], 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, 12, 43, 56] create only a control topology among the participant peers. Their data delivery topology is implicitly determined by the defined set of packet-forwarding rules.

Numerous protocols have been proposed to address the demand for live streaming applications, we will describe the most important of them. The first end-system multicast protocol was Narada [16], a multi-source multicast system designed for small to medium sized multicast groups. The peers are organized into a mesh with fixed out-degree. Every peer monitors all other peers to detect host failures and network partitions. The multicast tree is built on top of the mesh for every sender. The tree construction algorithm does not account for cross traffic and therefore a powerful link is likely to be used by many multicast links, which limits the efficiency of the multicast system.

In contrast, FatNemo uses crew members to share the forwarding load, thus relaxing the burden on a single high bandwidth path.

Overcast [28] is a single-source multicast system. It constructs a distribution tree by probing the available bandwidth between peers with a probe packet of 10 KB size. This probe packet is too short to accurately reflect the bandwidth of high bandwidth paths [28]. The multicast tree is constructed by moving a peer as far away from the source without reducing its available bandwidth. This is a local optimization for single-source multicast. In contrast, FatNemo is a multi-source multicast system, which uses a global optimized fat-tree construction algorithm and measures available bandwidth more accurately for high bandwidth paths.

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. FatNemo adopts the same implicit approach, and its design draws a number of ideas from Nice such as its hierarchical control topology. FatNemo introduces co-leaders to improve the resilience of the overlay and adopts a periodic probabilistic approach to reduce/avoid the cost of membership operations.

A large number of research projects have addressed reliable and resilient multicast at the network layer [41, 50, 53, 40, 34, 23]. A comparative survey of these protocols is given in [33, 46]. Like many of them, FatNemo relies on reactive techniques to recover from packet losses. STORM [51] 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 FatNemo, NACKs are used only to request missing packets from neighbors who indicated[3] to cache them locally.

In the context of overlay multicast, a number of protocols have been proposed aiming at high resilience [4, 11, 47, 39]. ZigZag [47] 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 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 FatNemo, 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, FatNemo relies on the concept of a *crew* and opts only for deterministic techniques for data forwarding. SplitStream [11] and CoopNet [39] 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. FatNemo 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 [8].

File distribution applications have gained more attention during the last few years. BitTorrent [18] was successfully introduced as the primary distribution channel for many Linux distributions. The protocol organizes its peers into a mesh for downloads. *Trackers* help the peers finding each other by returning a random list of peers. The Bit-

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

Torrent file distribution system uses tit-for-tat as a method to achieve a high resource utilization, which penalties nodes behind asymmetric links with low upstream capabilities. In contrast, FatNemo tries to optimize the system's performance without adding more penalty to low capability peers than already imposed by their access link.

FastReplica [14] addresses the problem of reliable and efficient file distribution in content distribution networks (CDN). A publisher splits a file into equal-sized portions, sends each of them to a different group member, and instructs the peers to download the missing pieces in parallel from other group members. Thus the algorithm exploits $n \times n$ potential Internet paths. In contrast, FatNemo exploits alternate paths and does adaptive load balancing as alternate path will be used to recover missing packets.

Bullet [30] is a high bandwidth data dissemination protocol for bulk data distribution. It organizes its peers into a tree and builds an overlay mesh on top of the tree structure. It uses random subsets to locate disjoint content within the system. Every node receives a *parent stream* from its parent in the tree and some number of *perpendicular streams* form chosen peers in the overlay. Similar to Bullet, FatNemo uses crew members as alternate streaming peers, but relies only on a logical tree topology to define the logical distribution tree. FatNemo reduces the effect of peers with access bandwidth bottlenecks in the tree by construction a fat tree. Bullet's random subset discovery algorithm could benefit from FatNemo's low average response time.

In the context of structured peer-to-peer overlay networks, [37] proposes to dynamically adjust protocol parameters, such as heartbeat intervals and grace periods, based on the operating conditions. Similar to FatNemo, it tries to reduce the maintenance cost without failures while still providing high resilience. FatNemo's approach differs in the way that it uses a static low cost algorithm, which handles an increased level of transiency with comparatively low cost. FatNemo's refinement algorithm could potentially 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, thus reducing the total control overhead.

3 FatNemo's Approach

FatNemo follows the *implicit approach* to building an overlay for multicasting: participating peers are organized in a control topology and the data delivery network is implicitly defined based on a set of forwarding rules which we will describe in the following paragraphs.

FatNemo organizes the set of communication peers into clusters, where every peer is a member of a cluster at the lowest layer. Each of these clusters selects a *leader* that becomes a member of the immediate superior layer. The process is thus repeated, with all peers in a layer being grouped into clusters from where 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.

The formation of clusters at each level of the hierarchy is currently based on available bandwidth [48, 16], although other factors such as latency [16, 3, 7] and expected peer lifetime [9] can be easily incorporated. Clusters vary in size between k and $2k+2$, where k is a constant known as *degree*.

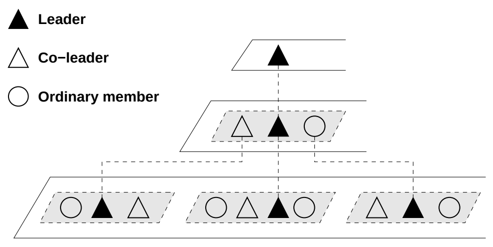


Fig. 2. FatNemo's logical organization. The shape illustrates only the role of a peer within a cluster: a leader of a cluster at a given layer can act as co-leader or ordinary member at the next higher layer.

Initially the peers join the tree based on proximity. The fat-tree optimization algorithm dynamically transforms the tree into a bandwidth optimized fat-tree at run time as we will describe in the following paragraphs.

To improve the resilience of the multicast group and for load sharing, FatNemo builds on Nemo's concept 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 2 illustrates the logical organization of FatNemo.

3.1 Fat-Tree Optimization Algorithm

To build the fat-tree, FatNemo relies on three techniques: (1) higher degree nodes should be placed higher up in the tree, (2) the size of clusters should increase exponentially as one ascends the tree, and (3) all peers must serve as crew members in order to maximize load balancing.

In the following paragraphs we explain the dynamics of our protocol, such as the joining and departure of peers, and the set of rules employed for forwarding and retransmission. We additionally describe how co-leaders help increase the resilience of multicast and discuss FatNemo's probabilistic approach to overlay maintenance. This algorithms are inherited from Nemo and, thus, we only provide a summarized description here.˜footnoteFor the complete details please see [7].

3.2 More Protocol Details

We assume the existence of a well-known special host, the *rendezvous point (RP)* [17]. A new peer joins the multicast group by querying the RP 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 from among them who to contact next based on the result from a given cost function, and (iii) moving into the next layer. When the new peer finds the leader with minimal cost at the bottom layer, it joins the associated cluster.

Member peers can leave FatNemo in a graceful (e.g. user disconnects) or ungraceful manner (unannounced, e.g. when the host node crashes). For graceful departures, since a common member has no responsibilities towards other peers, it can simply leave the group after informing its cluster's leader. On the other hand, a leader must first elect replacement leaders for all clusters it owns before it leaves the session.

To detect unannounced leaves, FatNemo relies on heartbeats exchanged among the cluster's peers. Unreported members are given a fixed time interval, or *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.

To deal with dynamic changes in the underlying network, every peer periodically checks the leaders of the next higher layers and switches clusters if another leader has a bigger available bandwidth than the current one (thresholds are used to prevent oscillation). Additionally, every leader checks its highest owned cluster for better suited leaders and transfers leadership if such a peer exits. This continuing process is called *refinement*.

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 split and merge operations are carried on by the cluster's leader.

FatNemo'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 Fig. 3.

Figure 4 shows an example of the forwarding algorithm in action and illustrates FatNemo's resilience under different failure scenarios. 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 in the tree. As other protocols aiming at high resilience [41, 4], FatNemo 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. In addition, each peer 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.

Proactive recovery, where a system tries to react immediately to membership changes, adds additional stress to an already-stressed network [44]. FatNemo relies on a set of periodic algorithms for overlay maintenance in order to avoid congestion collapse, but adopts a probabilistic approach to reduce the load on a possible stressed system. Some of the most costly maintenance operations, such as splitting, merging and refinement, are only executed with some probability or, alternatively, deferred to

```
FORWARD-DATA($msg$)
 1  $R \leftarrow \emptyset$
 2  if $leader \notin msg.sender\_crew$
 3    then $R \leftarrow R \cup leader$
 4  for each $child$ in $children$
 5  do if $child \notin msg.sender\_crew$
 6        then $R \leftarrow R \cup child$
 7  SEND($msg, R, sender\_crew \leftarrow crewOf(self)$)
 8  if $isCrewMember(self)$ and $leader \notin msg.sender\_crew$
 9    then $R \leftarrow \emptyset$
10          $R \leftarrow R \cup super\_leader$
11          for each $neighbor$ in $neighbors$
12          do $R \leftarrow R \cup neighbor$
13          SEND($msg, R, sender\_crew \leftarrow crewOf(leader)$)
```

Fig. 3. 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.

the next interval. We refer to them as *periodic probabilistic operations*. 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.

4 Evaluation

We analyze the performance of FatNemo through simulation and compare it to that of three other protocols – Narada [17], Nice [3] and Nice-PRM [4]. We evaluate the effectiveness of the alternative protocols in both terms of performance improvements to the application and protocol's overhead, as captured by the following metrics:

Response Time: 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. A lower mean response time indicates a higher system responsiveness, while a smaller standard deviation implies better synchronization among the receivers.

Delivered Packets: The number of packet successful delivered to all subscribers within a fixed time window. This metrics intents to illustrates the protocols ability in preventing bottlenecks in the delivery tree.

Delivery Ratio: Ratio of subscribers which have received a packet within a fixed time window. Disabled receivers are not accounted for.

Duplicate Packets: Number of duplicate packets per sequence number, for all enabled receivers, reflecting an unnecessary burden on the network. Late packets are accounted as duplicates, since the receiver already assumed them as not received.

Control-Related Traffic: Total control traffic in the system, in mega bits per second (Mbps), during the observation interval is one part of the system's overhead.

The remainder of this section discusses implementation details of the compared protocols and describes our evaluation setup. Section 5 presents our evaluation results.

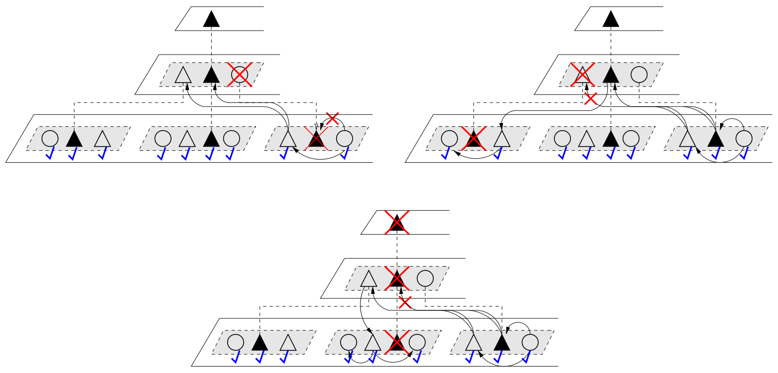


Fig. 4. Data forwarding in FatNemo: All nodes are able to receive the forwarded data under different failure scenarios. Note how a sender alternates the packet destination among the crew members.

4.1 Details on Protocol Implementations

For each of the three alternative protocols, the values for the available parameters were obtained from the corresponding literature.

For Narada [17], the number of directly connected peers (fanout) is set to six and may approach 12 for a short period of time. We employ the bandwidth-only scheme [16] for constructing the overlay, as this will result in the maximal throughput. The distance vectors, the set of the maximal bandwidth to all other peers, are exchanged in 10-sec. intervals. The timeout for detecting dead members is set to 60 sec., and the one for mesh partition repairs to 50 sec.

For Nice [3], heartbeats are sent at 10-sec. intervals. The cluster degree, k, is set to 3. The grace period for dead neighbor detection is set to 15 sec.

Nice-PRM is implemented as described in [4, 5]. We used PRM-(3,0.02) with three random peers chosen by each node, and with two percent forwarding probability. Discover messages to locate random overlay nodes are sent with 5-sec. intervals and a time-to-live (TTL) of 5 hops.

For FatNemo, the cluster degree at the lowest layer is set to three. It grows exponentially with every layer, at the second lowest layer it is nine, at the third lowest layer it is 27, and so on. The crew size is set to the size of the cluster, thus making all peers member of the crew. The grace period is set to 15 sec. and the mean time between crew list exchanges is set to 30 sec.

Our implementations of the alternative protocols closely follow the descriptions from the 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 FatNemo were made available to all protocols in our evaluations. The benefits from these algorithms help explain the performance improvements of the different protocols with respect to their original publications [17, 3, 4]. We have opted for this approach to isolate the contribution of PRM and co-leaders to the overall resilience of the multicast protocols.

For Nice, Nice-PRM and FatNemo, we check the clusters every second, but limit the minimal time between maintenance operations. Assuming that the triggering conditions are satisfied (an undersized cluster, for example) and that there has not been a maintenance operation within the last 5 sec., a merge operation is executed with 1% probability, a split operation with 100% probability, and a refinement operation with 1% probability. When the cluster's cardinality falls bellow its lower bound, we reduce the probability of execution of the refinement operation to 0.1% in an additional attempt to avoid an expensive merge.

4.2 Experimental Setup

We performed our evaluations through detailed simulation using SPANS, a locally written, packet-level, event-based simulator. We ran our simulations using GridG [35, 36] topologies with 5510, 6312 and 8115 nodes, and a multicast group of 256 members. GridG leverages Tiers [21, 10] to generate a three-tier hierarchical network structure, before it applies a power law enforcing algorithm while retaining the hierarchical structure.

Members are randomly attached to end systems, and a random delay of 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. We configure GridG to have different bandwidth distributions for different link types [30]. We assume that the core of the Internet has higher bandwidth capacities than the edge, as shown in Fig. 5. In all three scenarios, the bandwidth has a uniform distribution with ranges shown in Fig. 5.

Fig. 5. Three simulation scenarios. The bandwidth is expressed in Kbps.

Scenario	Routers	End systems	Links	Client-Stub	Stub-Stub	Transit-Stub	Transit-Transit
1	510	5000	11240	400-6000	3000-8000	4000-10000	10000-20000
2	312	6000	12730	800-8000	4000-10000	6000-15000	15000-30000
3	615	7500	16450	1000-15000	10000-30000	10000-50000	50000-100000

Each simulation experiment lasts for 500 sec. (simulation time). All peers join the multicast group by contacting the rendezvous point at uniformly distributed, random times within the first 100 sec. of the simulation. A warm-up time of 200 sec. is omitted from the figures. The publisher join the network and start publishing at the beginning of the simulation. Starting at 200 sec. and lasting for about 300 sec., each simulation has a phase with membership changes. We exercise each protocol with and without host failures during this phase. Failure rates are obtained from a published report of field failures for networked systems [49]. Nodes fail independently at a time sampled from an exponential distribution (with mean, *Mean Time To Failure*, equal to 60 min.) to rejoin shortly after (time 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, 5/7 of all members to be up during this phase. The failure event sequence

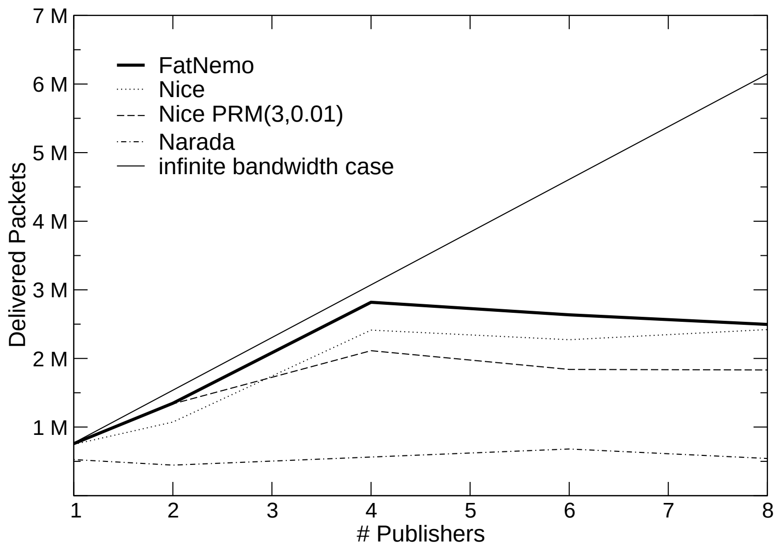


Fig. 6. Delivered packets (256 end hosts, Scenario 1).

is generated a priori based on the above distribution and used for all protocols and all runs.

In all experiments, we model multi-source multicast streams to a group. Each source sends constant bit rate (CBR) traffic of 1000 B payload at a rate of 10 packets per second. The buffer size is set to 16 packets, which corresponds to the usage of a 1.6-second buffer, a realistic scenario for applications such as video conferencing.

5 Experimental Results

This section presents and discusses results from our evaluation. Here we report results of five runs per protocol obtained with the different GridG topologies and the scenario 1. Similar results were obtained with scenario 2 and scenario 3.

Figure 6 shows the average number of delivered packets of all runs with no host failures. As we increase the number of publishers, the protocol's data delivery topology collapse. This happens first for Narada, which cannot even stream the one publisher case at the full rate. Nice and Nice PRM deliver substantially fewer packets compared to Fat-Nemo with an increased number of publishers. FatNemo is best at avoiding bottlenecks in the delivery tree, thus it delivers the most packets when the network is overloaded, as seen with 8 publishers.

The performance of a multi-source multicast system can be measured in terms of mean and standard deviation of the response time. Table 1 shows these two metrics for the evaluated protocols with one publisher. FatNemo outperforms Nice, Nice PRM and Narada in terms of mean and standard deviation of response time. With an increased number of publishers the relative number of delivered packets for Nice, Nice PRM and

Table 1. Response Time (1 Publisher, 256 end hosts, Scenario 1).

Protocol	Mean	Std
FatNemo	0.158	0.073
Nice	0.183	0.082
Nice-PRM(3,0.02)	0.195	0.086
Narada	0.770	0.464

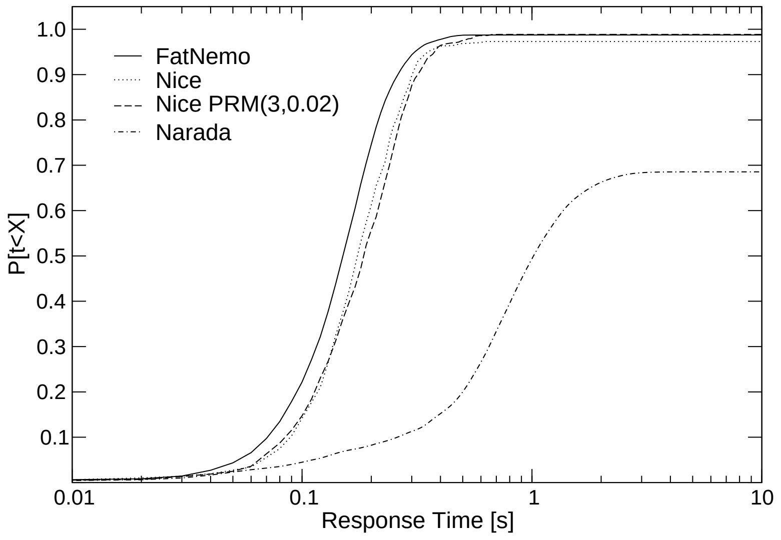


Fig. 7. Response Time CDF (1 Publisher, 256 end hosts, Scenario 1).

Narada decreases compared to FatNemo, which makes it impossible to fairly compare the response time for more than one publisher with only one number. This is due the fact, that protocols with less delivery ratio drop the packet with a high response time more likely than others, thus giving them an unfair advantage in terms of the mean and standard deviation of the response time in a overloaded network.

The mean response time illustrates the average performance. The distribution of the response time serves as a per packet based performance metric. Figure 7 shows the CDF of the response time per packet for one publisher. The y-axis is normalized to the infinite bandwidth case, that is when all receivers receive all possible packets. FatNemo, Nice and Nice PRM perform well, but FatNemo has a substantial advantage as it delivers the packets earlier. Narada only delivers a fraction of all possible packets and even these packets are substantially delayed. As we increase the number of publishers, the protocols face more and more bottlenecks. Despite these harder conditions, FatNemo outperforms the evaluated protocols in terms of packet delivery times as illustrated in Fig. 8 and Fig. 9. We see that even though the mean response time for Narada is lower

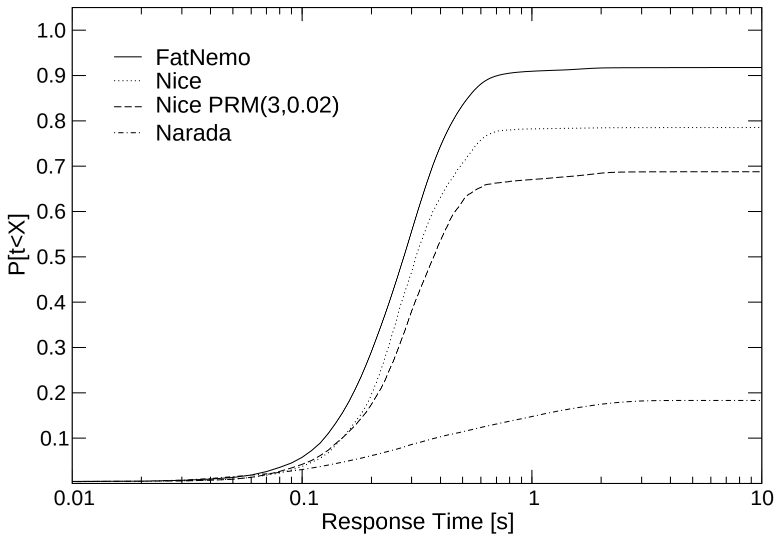


Fig. 8. Response Time CDF (4 Publisher, 256 end hosts, Scenario 1).

Table 2. Delivery Ratio (1 Publisher, 256 end hosts, Scenario 1).

Protocol	No Failures	With Failures
FatNemo	0.987	0.966
Nice	0.973	0.956
Nice-PRM(3,0.02)	0.989	0.970
Narada	0.685	0.648

than for FatNemo, FatNemo's response time per packet still outperforms Narada clearly, as seen with eight publishers.

Table 2 shows the delivery ratio using one publisher with and without end host failures. We see that FatNemo performs as well as Nice PRM under the low failure scenario with only a a drop of 2.1% in delivery ratio. Nice has a slightly lower delivery ratio for this scenario, while Narada suffers already from a collapsed delivery tree with only about 70% delivery ratio. In general, the delivery ratio will decrease with a increase in the number of publishers, as the protocol's data delivery topology slowly collapses.

The overhead of a protocol can be measured in terms of duplicate packets. We show this metric in the second column of Table 3. Despite the high delivery ratio, FatNemo has in average only 0.367 duplicate packets per sequence number, while Nice PRM suffers from 5.168 duplicate packets per sequence number generated by its probabilistic forwarding algorithm. Nice and Narada generate approximately no duplicates, but also deliver less packets as shown in Table 2. FatNemo's control related traffic is higher than for Nice and Nice-PRM. FatNemo increases the cluster cardinality as one moves up the three, thus creating a small additional overhead. The control traffic is accounted at

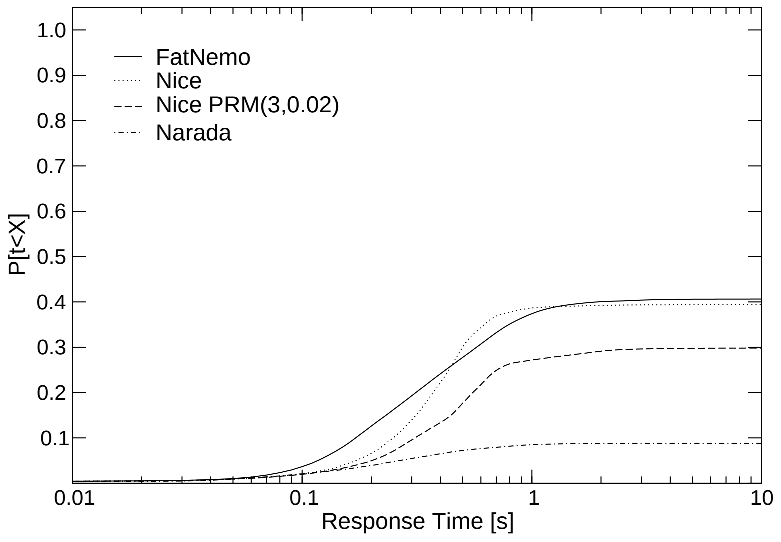


Fig. 9. Response Time CDF (8 Publisher, 256 end hosts, Scenario 1).

Table 3. Overhead (1 Publisher, 256 end hosts, Scenario 1).

Protocol	Duplicate packets	Control Traffic [Mbps]
FatNemo	0.367	17.99
Nice	0.000	9.211
Nice-PRM(3,0.02)	5.168	11.48
Narada	0.006	157.3

router level, thus the choice of neighbors in FatNemo adds additional overhead, since it chooses not the closest, but the peer with the highest bandwidth as its neighbor.

6 Conclusions and Further Work

In this paper we introduced the parallel architecture concept of fat trees to overlay multicast protocols. We have described FatNemo, a novel scalable peer-to-peer multicast protocol that incorporates this idea to build data delivery topologies with minimized mean and standard deviation of the response time. The resulting protocol is capable of attaining high delivery ratio under heavy load and host failures, while incurring neglectable cost in terms of control-related traffic. We have demonstrated the effectiveness of our approach through simulation under different stress scenarios. We are currently validating our findings through wide-area experimentation.

We showed that FatNemo can achieve much higher delivery ratios when compared to alternate protocols (an increase of up to 360% under high load), while reducing the mean (up to 80%) and standard deviation (up to 84%) of the response time in the non-overloaded case.

References

1. D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris. Resilient overlay networks. In *Proc. of the 18th ACM SOSP*, October 2001.
2. S. Banerjee and B. Bhattacharjee. A comparative study of application layer multicast protocols, 2002. Submitted for review.
3. S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable application layer multicast. In *Proc. of ACM SIGCOMM*, August 2002.
4. S. Banerjee, S. Lee, B. Bhattacharjee, and A. Srinivasan. Resilient multicast using overlays. In *Proc. of ACM SIGMETRICS*, June 2003.
5. S. Banerjee, S. Lee, R. Braud, B. Bhattacharjee, and A. Srinivasan. Scalable resilient media streaming. Tech. Report UMIACS TR 2003-51, U. of Maryland, 2003.
6. M. Bawa, H. Deshpande, and H. Garcia-Molina. Transience of peers & streaming media. In *Proc. of HotNets-I*, October 2002.
7. S. Birrer and F. E. Bustamante. Nemo - resilient peer-to-peer multicast without the cost. Submitted for publication, April 2004.
8. R. E. Blahut. *Theory and Practice of Error Control Codes*. Addison Wesley, 1994.
9. F. E. Bustamante and Y. Qiao. Friendships that last: Peer lifespan and its role in P2P protocols. In *Proc. of IWCW*, October 2003.
10. K. L. Calvert, M. B. Doar, and E. W. Zegura. Modeling internet topology. *IEEE Communications Magazine*, 35(6):160–163, June 1997.
11. 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.
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), 2002.
13. Y. Chawathe. *Scattercast: an architecture for Internet broadcast distribution as an infrastructure service*. Ph.D. Thesis, U. of California, Berkeley, CA, Fall 2000.
14. L. Cherkasova and J. Lee. FastReplica: Efficient large file distribution within content delivery networks. In *Proc. of USENIX ITS*, 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. 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.
17. Y.-H. Chu, S. G. Rao, and H. Zhang. A case for end system multicast. In *Proc. of ACM SIGMETRICS*, June 2000.
18. B. Cohen. BitTorrent. bitconjurer.org/BitTorrent/, 2001. File distribution.
19. S. E. Deering. Multicast routing in internetworks and extended LANs. In *Proc. of ACM SIGCOMM*, August 1988.
20. 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.
21. M. B. Doar. A better model for generating test networks. In *Proc. of Globecom*, November 1996.

22. M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the internet topology. In *Proc. of ACM SIGCOMM*, September 1999.
23. 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(6), December 1997.
24. P. Francis. Yoid: Extending the Internet multicast architecture. http://www.aciri.org/yoid, April 2000.
25. 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.
26. M. Homewood and M. McLaren. Meiko CS-2 interconnect elan – elite design. In *IEEE Hot Interconnects Symposium*, August 1993.
27. InfiniBand Trade Association. Infiniband architecture specification (1.0.a). www.infinibandta.com, June 2001.
28. 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.
29. C. Jin, Q. Chen, and S. Jamin. Inet: Internet topology generator. Technical Report CSE-TR-433-00, U. of Michigan, Ann Arbor, MI, 2000.
30. D. Kostić, A. R. adn Jeannie Albrecht, and A. Vahdat. Bullet: High bandwidth data dissemination using an overlay mesh. In *Proc. of the 19th ACM SOSP*, October 2003.
31. C. E. Leiserson. Fat-trees: Universal networks for hardware-efficient supercomputing. *IEEE Transactions on Computers*, 34(10):892–901, October 1985.
32. C. E. Leiserson, Z. S. Abuhamdeh, D. C. Douglas, C. R. Feynman, M. N. Ganmukhi, J. V. Hill, W. D. Hillis, B. C. Kuszmaul, M. A. S. Pierre, D. S. Wells, M. C. Wong-Chan, S.-W. Yang, and R. Zak. The network architecture of the Connection Machine CM-5. *Journal of Parallel and Distributed Computing*, 33(2):145–158, 1996.
33. B. N. Levine and J. Garcia-Luna-Aceves. A comparison of reliable multicast protocols. *Multimedia Systems Journal*, 6(5), August 1998.
34. 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.
35. D. Lu and P. A. Dinda. GridG: Generating realistic computational grids. *ACM Sigmetrics Performance Evaluation Review*, 30(4):33–41, March 2003.
36. D. Lu and P. A. Dinda. Synthesizing realistic computational grids. In *Proc. of SC2003*, November 2003.
37. R. Mahajan, M. Castro, and A. Rowstron. Controlling the cost of reliability in peer-to-peer overlays. In *Proc. of IPTPS*, February 2003.
38. A. Medina, A. Lakhina, I. Matta, and J. Byers. BRITE: Universal topology generation from a user's perspective. Technical Report 2001-003, Boston University, 2001.
39. V. N. Padmanabhan, H. J. Wang, and P. A. Chou. Resilient peer-to-peer streaming. In *Proc. of IEEE ICNP*, 2003.
40. C. Papadopoulos, G. M. Parulkar, and G. Varghese. An error control scheme for large-scale multicast applications. In *Proc. of IEEE INFOCOM*, March 1998.
41. 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(3), April 1997.
42. D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel. ALMI: An application level multicast infrastructure. In *Proc. of USENIX USITS*, March 2001.
43. S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Application-level multicast using content-addressable networks. In *Proc. of NGC*, November 2001.
44. S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz. Handling churn in a DHT. In *Proc. of USENIX ATC*, December 2004.

45. H. Tangmunarunkit, R. Govindan, S. Jamin, S. Shenker, and W. Willinger. Network topology generators: Degree-based vs. structural. In *Proc. of ACM SIGCOMM*, August 2002.
46. 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(3), April 1997.
47. 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.
48. Z. Wang and J. Crowcroft. Bandwidth-delay based routing algorithms. In *Proc. of IEEE GlobeCom*, November 1995.
49. J. Xu, Z. Kalbarczyk, and R. K. Iyer. Networked Windows NT system field failure data analysis. In *Proc. of PRDC*, December 1999.
50. X. Xu, A. Myers, H. Zhang, and R. Yavatkar. Resilient multicast support for continuous-media applications. In *Proc. of NOSSDAV*, May 1997.
51. X. R. Xu, A. C. Myers, H. Zhang, and R. Yavatkar. Resilient multicast support for continuous-media applications. In *Proc. of NOSSDAV*, May 1997.
52. J. Yang. Deliver multimedia streams with flexible qos via a multicast dag. In *Proc. of ICDCS*, May 2003.
53. R. Yavatkar, J. Griffoen, and M. Sudan. A reliable dissemination protocol for interactive collaborative applications. In *ACM Multimedia*, November 1995.
54. B. L. Yi Cui and K. Nahrstedt. oStream: Asynchronous streaming multicast in application-layer overlay networks. *IEEE Journal on Selected Areas in Communication*, 22(1), January 2004.
55. E. W. Zegura, K. Calvert, and S. Bhattacharjee. How to model an internetwork. In *Proc. of IEEE INFOCOM*, March 1996.
56. 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.