MOB: Zero-configuration High-throughput Multicasting for Grid Applications

Mathijs den Burger and Thilo Kielmann
Dept. of Computer Science, Vrije Universiteit, Amsterdam, The Netherlands
{mathijs, kielmann}@cs.vu.nl

Abstract

A reoccurring problem for grid applications is how to distribute large amounts of data efficiently from one cluster to multiple others (multicast). Existing methods usually arrange nodes in optimized tree structures, using external network monitoring data, to cope with network heterogeneity between grid sites. This dependence on monitoring data, however, severely impacts both ease of deployment and adaptivity to dynamically changing network conditions.

In this paper, we present Multicast Optimizing Bandwidth (MOB), a high-throughput multicast approach, inspired by the BitTorrent protocol [4]. MOB does not need any external monitoring information at all. Instead, data transfers are initiated by the receivers that try to steal *data from peer clusters. Instead of planning ahead multicast trees, based on potentially outdated monitoring data, MOB automatically adapts to the currently achievable bandwidth ratios.*

Our experimental evaluation compares MOB to both the BitTorrent protocol and to our previous approach, Balanced Multicasting [11], the latter computing optimized multicast trees based on external monitoring data. Our evaluation shows that MOB outperforms the BitTorrent protocol. MOB is competitive with Balanced Multicasting as long as the network bandwidth remains stable. With dynamically changing bandwith, MOB outperforms Balanced Multicasting by wide margins.

1 Introduction

A grid consists of multiple sites, ranging from single machines to large clusters, located around the world. Contrary to more traditional computing environments like clusters or super computers, the network characteristics between grid sites are both very heterogeneous and dynamically changing. Therefore, communication libraries need to take this heterogeneity into account to maintain efficiency in a worldwide environment.

A typical communication pattern is the transfer of a substantial amount of data from one site to multiple others, also known as *multicast*. The completion time of large data transfers depends primarily on the bandwidth that an application can achieve across the interconnection network. Multicasting is usually implemented by arranging the application nodes in spanning trees over which the data are sent. This method, however, can be very inefficient in a grid environment, where the differences in bandwidth between the sites can be significant and also dynamically changing.

In previous work, we have used monitoring information about network bandwidth to construct multicast trees between grid sites [11]. This approach, however, is problematic due to three reasons: (1) It assumes network monitoring systems to be deployed ubiquitously. (2) It assumes monitored data to be both accurate and stable during a multicast operation, which might not be the case in shared networks with variable background traffic. (3) Network monitoring systems monitor the network itself; translating this data (*available bandwidth* [20]) to information that is meaningful to an application or multicasting algorithm (*achievable bandwidth* [20]) is a hard problem [21].

In this paper, we present *Multicast Optimizing Bandwidth* (MOB), a multicast approach inspired by the BitTorrent protocol [4]. With MOB, data transfers are initiated by the receivers that try to *steal* data from peer clusters. Instead of planning multicast trees based on potentially outdated monitoring data, MOB implicitly uses all network links between all clusters, while automatically adapting to the currently achievable bandwidth ratios. It does so without any monitoring data; therefore we call it a “zero configuration” multicasting protocol.

With MOB, nodes within the same cluster team up to form so-called *mobs* that, together, try to steal data from other clusters as efficiently as possible. In comparison with BitTorrent, this teamwork minimizes the amount of wide-area communication between clusters, and divides the bandwidth requirements among all participating nodes in a cluster.

We have implemented MOB (as well as the BitTorrent protocol and *Balanced Multicasting* [11]) within our Java-based Ibis system [24]. We have experimentally evaluated the three approaches on both the DAS-2 [9] and the new DAS-3 [10] multi-cluster systems installed in the Netherlands. Our experimental evaluation shows that MOB outperforms the original BitTorrent protocol due to the much better utilization of the wide-area networks. In the case of stable wide-area bandwidth, MOB automatically achieves multicast bandwidth that is competitive to *Balanced Multicasting*. In most cases MOB performs even better. As soon as bandwidth changes in the middle of a multicast operation, *Balanced Multicasting* immediately suffers while MOB automatically adapts its behavior to such changes.

The remainder of this paper is structured as follows. In Section 2, we discuss issues in multicasting in grids, as well as existing approaches. Section 3 describes the MOB algorithm and its implementation. Section 4 describes our experimental evaluation and limitations of our approach, Section 5 concludes.

2 Background and related work

In a *multicast* operation, one node, referred to as the *root*, is transmitting data to all other nodes of a given group, like the processes of an application. This is comparable to MPI's broadcast operation. For optimizing multicast, we are interested in minimizing the overall completion time, from the moment the root node starts transmitting until the last receiver has got all data. As we are interested in multicasting large data sets, we optimize for high throughput. In Section 4, we will thus report our results as achieved throughput (in MB/s).

Before presenting our new algorithm, MOB, we first discuss more traditional approaches to multicasting in grids and Internet-based environments. In this section, we also briefly summarize both our predecessor *Balanced Multicasting* and BitTorrent, and discuss their performance limitations.

2.1 Overlay multicasting

Multicasting over the Internet started with the development of IP multicast, which uses specialized routers to forward packets. Since IP multicast was never widely deployed, *overlay multicasting* became popular, in which only the end hosts play an active role. Several centralized or distributed algorithms have been proposed to find a single overlay multicast tree with maximum throughput [5, 19]. Splitting the data over multiple trees can increase the throughput even further.

A related topic is the overlay multicast of media streams, in which it is possible for hosts to only receive part of the data (which results in, for instance, lower video quality). In [7, 19], a single multicast tree is used for this purpose. SplitStream [3] uses multiple trees to do distribute streaming media in a P2P context. Depending on the bandwidth each host is willing to donate, the hosts receive a certain amount of the total data stream. The maximum throughput is thus limited to the bandwidth the stream requires. In contrast, our MOB approach tries to use the maximum amount of bandwidth the hosts and networks can deliver.

2.2 Network performance modeling

Throughout this work, we assume networks as sketched in Figure 1. Here, nodes are distributed among clusters. Within each cluster, nodes are connected via some local interconnect. Towards the WAN, each node has a network interface that is connected to a shared access link. All access links end at a gateway router (typically to the Internet). Within the WAN, we assume full connectivity among all clusters.

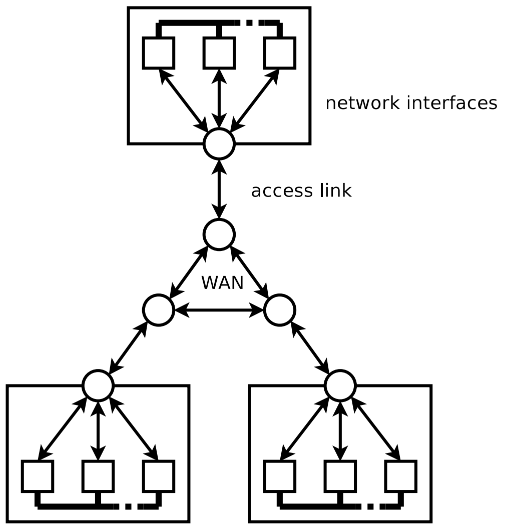


Figure 1. network model including clusters

For optimizing multicast operations, we need to efficiently use the available network bandwidth where we distinguish as outlined in [20]:

Bandwidth Capacity is the maximum amount of data per time unit that a hop or path can carry.

Achievable Bandwidth is the maximum amount of data per time unit that a hop or path can provide to an application, given the current utilization, the protocol and operating system used, and the end-host performance capability.

We are interested in maximizing the achievable bandwidth of all data streams used for a multicast operation. In multicasting, sharing effects can be observed whenever a single host is sending to and/or receiving from multiple other hosts. Here, the bandwidth capacity of the local network can become a bottleneck. This local capacity can be caused either by the network interface (e.g., a FastEthernet card, connected to a gigabit network), or by the access link to the Internet that is shared by all machines of a site.

For example, with our DAS-2 system, individual nodes use 100Mbit Ethernet adaptors to connect to the shared, Gigabit Internet uplink. Here, the capacity of each node forms a multicasting bottleneck. In Section 4 we refer to this setting as a *local bottleneck test case*, dominated by local bandwidth capacity. With our new DAS-3 system, however, this bottleneck has been resolved by attaching all nodes with Gigabit Ethernet adaptors. In Section 4 we refer to this setting as a *global bottleneck test case*, dominated by achievable bandwidth across the wide-area network.

In order to optimize multicast operations based on the given network characteristics, one has to rely on external network monitoring system like the Network Weather Service [26], REMOS [14], or Delphoi [21]. The latter uses specialized measurement tools like PathRate [13] and PathChirp [23] to measure the capacity and available bandwidth of WAN links, respectively.

Using such tools, however, has its own issues. First of all, the monitoring tools have to be deployed to measure the network between all clusters in question. Frequently, this is an administrative issue. Second, network bandwidth is measured using active probes (sending measurement traffic) which can take significant amounts of time and scales only poorly to large environments as, for N clusters, $O(N^2)$ network paths need to be measured. Consequently, measurements are run in frequencies that are too slow to properly follow dynamic bandwidth fluctuations. Finally, network monitoring tools measure the properties of the network paths themselves (like *available bandwidth* or *bandwidth capacity*) rather than the properties that are relevant to the applications themselves, like *achievable bandwidth*. Translating monitoring data to application-level terms is a hard problem [21].

2.3 Optimized multicast trees

Optimization of multicast communication has been studied extensively within the context of message passing systems and their collective operations. The most basic approach to multicasting is to ignore network information altogether and send directly from the root host to all others. MagPIe [18] used this approach by splitting a multicast into two layers: one within a cluster, and one flat tree between clusters. Such a flat tree multicast will put a high load on the outgoing local capacity of the root node, which will often become the overall bandwidth bottleneck.

As an improvement we can let certain hosts forward received data to other hosts. This allows to arrange all hosts in a directed spanning tree over which the data are sent. MPICH-G2 [16] followed this idea by building a multilayer multicast to distinguish wide-area, LAN and local communication. As a further improvement for large data sets, the data should be split to small messages that are forwarded by the intermediate hosts as soon as they are received to create a high-throughput pipeline from the root to each leaf in the tree [17].

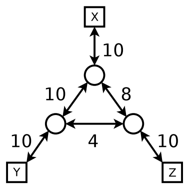
The problem with this approach is to find the optimal spanning tree. If the bandwidth between all hosts is homogeneous, we can use a fixed tree shape like a chain or binomial tree, which is often used within clusters [25]. As a first optimization for heterogeneous networks, we can take the achievable bandwidth between all hosts into account. The throughput of a multicast tree is then determined by its link with the least achievable bandwidth. Maximizing this *bottleneck bandwidth* can be done by using a variant of Prim's algorithm, which yields the *maximum bottleneck tree* [5].

However, this maximum bottleneck tree is not necessarily optimal because each host also has a certain local capacity. A forwarding host should send data to all its n children at a rate at least equal to the overall multicast throughput t. If its outgoing local capacity is less than $n * t$, it cannot fulfill this condition and the actual multicast throughput will be less than expected. Unfortunately, taking this into account generates an NP-hard problem.

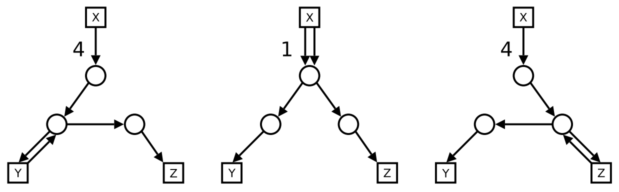
The problem of maximizing the throughput of a set of overlay multicast trees has also been explored theoretically. Finding the optimal solution can be expressed as a linear programming problem, but the number of constraints grows exponentially with the number of hosts. Although, in theory, this can be reduced to a square number of constraints, in practice finding the exact solution can be slow and expensive [6]. Any real-time applicable solution will therefore always have to rely on heuristics.

The multiple tree approach in [1] uses linear programming to determine the maximum multicast throughput given the bandwidth of links between hosts, but requires a very complicated algorithm to derive the set of multicast trees that would achieve that throughput. Therefore, the linear programming solution is only used to optimize the throughput of a single multicast tree.

The Fast Parallel File Replication (FPFR) tool [15] is implementing multiple, concurrently used multicast trees. FPFR repeatedly uses depth-first search to find a tree spanning all hosts. For each tree, its bottleneck bandwidth is "reserved" on all links used in the tree. Links with no bandwidth left can no longer be used for new trees. This search for trees continues until no more trees spanning all hosts can be found. The file is then multicast in fixed-size chunks using all trees found. FPFR does not take the local bandwidth capacity of hosts into account, leading to over subscription of links forming capacity bottlenecks. In consequence, FPFR may perform much worse than expected.



(a) Network example: x is multicasting to y and z



(b) Balanced multicast trees

Figure 2. Example of Balanced Multicasting

2.4 Balanced Multicasting

In previous work [11], we have presented *Balanced Multicasting*, a technique that improves over FPFR by also taking bandwidth capacity into account. An example is shown in Figure 2a. This network consists of three hosts, each connected to the network by their access line. Routers connect access lines with the WAN. Access lines are annotated with their local capacity, e.g. the capacity of the LAN. Wide-area connections are annotated with their achievable bandwidth. For simplicity of the example, we assume all connections to be symmetrical in both directions. (The actual units for bandwidth are not relevant here.)

In this example, *Balanced Multicasting* creates the three multicast trees shown in Figure 2b, with a total achievable bandwidth of 9. Together, these trees maximize the multicast throughput while not over subscribing individual link capacities. Please note that the individual trees may have different bandwidth, in the example 4, 1, and 4. These different data rates are enforced by traffic shaping at the sender side. This process of balancing the bandwidth shares gave the name to the approach. If the sender would not balance the shares of the three trees, then the middle tree (using the LAN at x twice) would consume bandwidth that was intended for the other trees, resulting in a total bandwidth of $3 \times 10/4 = 7.5$, instead of the anticipated 9.

This example shows balanced multicast trees as they are computed by our algorithm published in [11]. In this paper, we omit the details how these trees are constructed. It is worth mentioning, however, that finding the optimal set of balanced multicast trees is an NP-hard problem. For this reason, our implementation is using heuristics to find solutions which we have shown in [11] to be close to the optimum.

When evaluating MOB, we compare to *Balanced Multicasting* as a (close-to) optimal solution that can be found with complete network performance information. We would like to emphasize that Balanced Multicasting, as well as all other spanning-tree based multicasting strategies, is computing its optimized spanning trees based on the monitoring data available at the time when the multicast is started. Later changes in the network will *not* be taken into account.

2.5 BitTorrent

As explained so far, deriving optimized multicast trees is a hard problem. Especially in the case of dynamically changing network performance, carefully computed multicast trees can easily turn out to be inefficient. In this paper, we propose a radically different approach, namely receiver-initiated communication as inspired by the BitTorrent protocol.

BitTorrent [4] is a peer-to-peer file-sharing application, designed to distribute large files efficiently. The BitTorrent protocol leaves some room for choice, and nowadays a lot of different implementations exist. We will describe only the basic concepts of the protocol, and use the implementation choices of the original BitTorrent client.

The goal of BitTorrent is to distribute a large amount of data, consisting of one or more files. The data is logically split into P equally-sized pieces, usually a few hundred kilobytes each. The file distribution process is bootstrapped via a so-called *tracker* that serves as the central component through which users can find other users that are downloading the same data. The tracker is published via a *.torrent* file that also contains SHA1 hash values for all pieces. The user who wants to distribute the data has to run a BitTorrent client himself, and make it accessible via the tracker. Since this client already possesses all data, it is called a *seed*.

A user who wants to download the data first downloads the .torrent file into his BitTorrent client. His client then contacts the tracker, which returns a random list of C users already downloading the file (the set of all nodes downloading the same data is called a *swarm*). The client now initiates connections with those C clients. Later on, other nodes joining the swarm may also initiate connections, and the set of peers of a client will therefore consist of nodes he initiated a connection with, and nodes that initiated a connection with him.

After connection setup, peers first exchange the indices of the pieces they already possess in a *'bitfield'* message. Whenever a node n has obtained a new piece i, it informs all its peers about it with a *'have(i)'* message. If a peer p

of n does not have this piece yet, it sends an *'interested'* message to n. Whether n will send p any pieces depends on the so-called *choking* algorithm.

The choking algorithm determines which peers to send data to. By default, all peers are choked and will not receive any pieces. If a node n is willing to send data to a peer p, it sends it an *'unchoke'* message. From that moment on, p can request a piece i from n from by sending it a *'request(i)'* message, to which n replies with a *'piece(i)'* message containing the actual data of piece i. Each node keeps track of the set of pieces each peer already possesses, which is updated whenever a *'bitfield'* or *'have'* message is received. The piece to request from a peer is chosen randomly out of this set. Each node always has R outstanding requests (we use $R = 5$) to get the 'pipelining' effect described in [4].

A node can have N unchoked peers at the same time (we use $N = 5$). Which peers will be unchoked is decided on a *'tit-for-tat'* basis: Each node keeps track of the throughput each peer provides. Among the peers that are interested, the four with the highest download rate will be unchoked. Seed nodes, which have nothing to download, unchoke the four peers to which the upload rate is the highest. The decision to choke or unchoke peers is made every 10 seconds. The fifth peer is chosen using a different mechanism called *optimistic unchoke*: every 30 seconds, a peer is chosen randomly, irrespective of its download rate. Newcomer nodes that do not yet possess any pieces at all are helped into the swarm by a three times higher selection probability for unchoking, compared to nodes that have already downloaded some pieces.

With only two users, BitTorrent is equivalent to a client-server setting. With more users, clients will start downloading pieces from each other instead of the central seed node. This shifts the bandwidth demand from the seed to the other users, which greatly improves throughput and avoids overloading a single node in case of flash crowds. Even better, peers will adapt to changes in available bandwidth in the network; peers behind slow connections are likely to be choked and replaced by others, and peers behind fast connections will be kept.

The tit-for-tat choking strategy counters clients that only download pieces and never upload anything: it is in a client's own interest to return as many pieces as possible to other peers, since this affects its popularity among its peers in the future. The optimistic unchoking provides a way to discover better peers and higher bandwidth routes.

BitTorrent's ability to both automatically discover suitable multicasting topologies and to adapt to changing network conditions makes it interesting for multicasting between multiple clusters in a grid. In Section 3, we will present our MOB algorithm which has been developed starting from BitTorrent's ideas. For comparison with MOB, we have also implemented the BitTorrent protocol within our Java-based Ibis system [24].

BitTorrent has been designed for un-cooperative peer-to-peer environments of individual users, with possible failures and peers joining and leaving in the middle of a transmission. This requires some features in the BitTorrent protocol that are not needed in stable and cooperative multi-cluster grid environments. In order to provide a fair comparison, we have left out these features from our implementation, as they would only cause unnecessary runtime overheads on the original BitTorrent protocol: We do not use SHA1 hash values for the pieces as we consider transmissions to be reliable. We also kick-start the choking algorithm by immediately unchoking five random peers, regardless of whether they are interested or not. Normally, only interested peers are unchoked, but the root node may execute its choking algorithm just before the *'interested'* messages of its peers arrive. In that case, the root's peers will only be unchoked in the next execution of the choking algorithm, 10 seconds later; our kick-start prevents this from happening. Finally, we assume peers to stay online during the whole multicast operation to provide a fair comparison with other multicast methods in which nodes can contribute during the whole multicast operation. This eliminated the need to implement joining and leaving peers, which both simplified our implementation and eliminated the overhead incurred by the code handling these cases.

2.6 Clustering additions to BitTorrent

Other work has already recognized that grouping BitTorrent nodes into clusters can increase the overall throughput. *Biased neighbor selection* [2] proposes grouping of BitTorrent nodes by ISP (Internet Service Provider) to reduce the amount of costly traffic between ISP's. MOB is doing a similar grouping by cluster, but also adds teamwork among the nodes of a cluster to further improve multicast performance. In uncooperative peer-to-peer environments, this improvement would not be possible.

Another approach is followed by Tribler [22], a BitTorrent client that groups users into social clusters of friends. Users can tag each other as a friend, indicating they are willing to donate upload bandwidth to each other by searching each other's pieces. The amount of trust between BitTorrent nodes is thereby increased using existing relations between people. MOB is essentially an automation of this technique applied to grid clusters, with all nodes in the same cluster being friends.

3 Multicast Optimizing Bandwidth (MOB)

In this section we present *Multicast Optimizing Bandwidth (MOB)*. First we will describe the algorithm by which

MOB nodes connect to and communicate with each other. Second, we will outline our implementation.

3.1 Algorithm

Similar to BitTorrent, MOB distributes data over a random mesh by letting nodes 'steal' pieces from other nodes. In addition, nodes in the same cluster cooperate by forming a *'mob'* together. Each member of a mob is responsible for stealing only a part of the total data from nodes in other clusters; we we call this part a node's *mob share*. The other pieces will be stolen by other members of the mob. Every stolen piece is exchanged locally between members of the same mob, such that, at the end, all nodes will have received all data.

Which nodes are located in which clusters is assumed to be globally known, and often provided by the runtime system (in our case, Ibis). For each node n, we will call nodes located in the same cluster *local* nodes of n, and nodes located in other clusters *global* nodes of n. The number of nodes located in cluster x is called its size $s(x)$. Each node n in cluster x has a cluster rank $r(x, n)$, ranging from 0 to $s(x) - 1$.

The algorithm to multicast data using MOB can be split in four phases:

1. Each node n initiates a connection to five randomly selected local nodes. Incoming connections from other local nodes are always accepted, and the node will be added to node n's *local peers*. Once node n has found five local peers this way, it stops connecting to other local nodes itself, while still accepting incoming connections from them.

2. After enough local peers are found, node n in cluster x creates a set of possible *global peers*. For each cluster $y \neq x$, this set will contain the nodes p in cluster y with cluster rank

$$r(y, p) = r(x, n) \mod s(y)$$

   Five peers from this set are then randomly selected as the global peers of n, to which it initiates connections in the same manner as in phase 1). This selection of global peers ensures that the connections between nodes in different clusters are well spread out over all clusters and the nodes in those clusters.

3. After all connections are set up, data can be transferred. Like with BitTorrent, the data is logically split into P equally-sized pieces, numbered 0 to $P-1$. Each node n in cluster x steals those pieces i from its *global peers* that are part of its *mob share*:

$$\left\lfloor r(x, n) \cdot \frac{P}{s(x)} \right\rfloor \leq i < \left\lfloor (r(x, n) + 1) \cdot \frac{P}{s(x)} \right\rfloor$$

   All other pieces are stolen from the node's *local peers*. By using this scheme, MOB transfers each piece to each cluster exactly once.

4. As soon as a node has received all pieces, it joins a final synchronization phase. Here, all nodes synchronize by sending an acknowledgement over a binary tree spanning all nodes. This ensures that each node keeps serving requests from its peers until all nodes have all the data.

In phase 3), nodes communicate with their peers using a simplified version of the BitTorrent protocol, only consisting of *bitfield, $have$*, *request* and *piece* messages. The semantics of those messages slightly differ from those in the BitTorrent protocol. The *bitfield* message is sent only once by the root node to all its peers (both local and global) to inform them it has all pieces. Nodes use a *request(i)* message to instruct a peer node to return piece i, which is *always* honored by the peer node. Whenever a node has received a new piece i, it informs all its local peers about it by sending a *have(i)* message. Global peers are only informed if piece i is part of their mob share.

Splitting the data in mob shares alleviates the load on the bandwidth bottleneck in case of of two common network scenarios, which we will call *'global bandwidth bottleneck'* and *'local bandwidth bottleneck'*. (Recall the network model depicted in Figure 1.) In the global bandwidth bottleneck scenario, the achievable bandwidth of the wide-area links between clusters is the overall bandwidth bottleneck. In this case, it is necessary to minimize the amount of wide-area communication. Using MOB, each node will only inform a global peer of a received piece if it is part of that peer's mob share. This means that only one *'have'* message per piece will be sent out to each cluster, and each piece will only be transferred once into each cluster. Together, this greatly relieves the load on the wide-area network.

In the local bandwidth bottleneck scenario, the local capacity of the wide-area network interface of the nodes in a cluster is the overall bandwidth bottleneck. Since, in MOB, each node only communicates the data in its mob share through this wide-area network interface, the bandwidth demand is automatically spread over all wide-area network interfaces in the same cluster. This was already observed in balanced multicasting [11], but the setup there depended on network monitoring information; MOB achieves the same result automatically.

3.2 Implementation

We have implemented MOB, BitTorrent, and Balanced Multicasting on top of Ibis [24], our Java-based Grid programming environment. We used the SmartSockets library (part of Ibis) to take advantage of multiple network interfaces within hosts, e.g. using both Myrinet within a cluster for very high throughput and FastEthernet between clusters. The SmartSockets library also enabled us to emulate different clusters inside a single cluster by providing configurable custom routing of data, which we used in the test case with emulated clusters as described in Section 4.1.

In Ibis, nodes locate each other through the Ibis *name server,* which serves as a central directory service for node lookups. Ibis also provides all nodes with a Pool object containing the names and ranks of all other nodes and the names of their clusters, which is all the input data MOB needs. Balanced Multicasting also needs network monitoring data information, which is abstracted in a Gauge object. For the experiments in Section 4.1, we used a gauge that reads the same data file as used as input for the emulation itself.

The communication primitive provided by Ibis is a unidirectional pipe, in which messages are sent from a *send port* to a *receive port*. Connections between nodes in MOB and BitTorrent are bidirectional, and hence use two send/receive port pairs, one in each direction. On top of Ibis, we have implemented different MulticastChannel objects, one each for MOB, BitTorrent, and Balanced Multicasting. Providing the identical interface allows us to compare the three multicast strategies within the same framework.

All received messages are processed in asynchronous upcalls provided by Ibis. The data to multicast is abstracted in a Storage object. To avoid comparing disk access bottlenecks instead of protocol behavior, we only used a MemoryStorage implementation that stores data in a byte array.

4 Evaluation

We have evaluated MOB using two test cases: emulating a heterogeneous wide-area network within one cluster of the Distributed ASCI Supercomputer 3 (DAS-3), and using four clusters of the Distributed ASCI Supercomputer 2 (DAS-2). The former compares MOB with Balanced Multicasting and BitTorrent in various *'global bottleneck'* environments with varying dynamics. The latter compares MOB with Balanced Multicasting and BitTorrent in a *'local bottleneck'* scenario, to show that the optimized throughput between clusters in Balanced Multicasting is matched by MOB without needing any network monitoring information.

4.1 Global bottleneck test case (DAS-3)

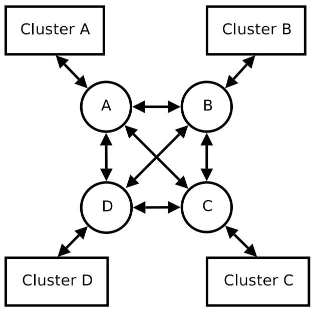
We evaluated the performance of MOB under various emulated WAN scenarios. To ensure a fair comparison and reproducible results, we chose to emulate the network performance of WAN links between nodes of the DAS-3. This enabled us to precisely control the environment and subject each multicast method to exactly the same network conditions without any interfering background traffic.

Each node in the DAS-3 is equipped with two 2.4 GHz AMD Opterons and 10Gbit Myrinet network card for fast local communication. Figure 3a shows our setup of four emulated clusters. Nodes in the same emulated cluster communicate directly, but traffic between nodes in different emulated clusters is routed over special *'hub'* nodes using the SmartSockets library. For example, data sent from a node in cluster A to a node in cluster B is first routed to hub node A, then forwarded to hub node B and finally delivered at the destination node in cluster B. Besides routing inter-cluster traffic, the hub nodes also emulate the wide-area bandwidth between clusters using the Linux Traffic Control (LTC [8]) kernel module to slow down their Myrinet interface: all traffic from a hub node X to another hub node Y is sent through a HTB queueing discipline [12] that limits the throughput to precisely the emulated bandwidth from cluster X to cluster Y. We would like to emphasize that the emulation only concerns the WAN performance. All application nodes run the real application code (Ibis and one of the three multicast protocols on top).

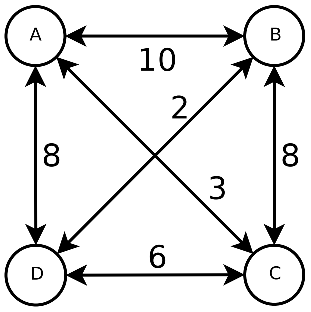
We used our testbed to emulate five scenarios:

1. **fast links**: the WAN bandwidth on all links is stable and set according to Figure 3b (all links are annotated with their emulated bandwidth, in MB/s).
2. **slow links**: like scenario 1, but with the bandwidth of links $A \leftrightarrow D$ and $B \leftrightarrow C$ set to 0.8 MB/s (indicated by the dotted lines in Figure 3c).
3. **fast** $\rightarrow$ **slow**: like scenario 1 for 30 seconds, then like scenario 2, emulating a drop in throughput on two WAN links
4. **slow** $\rightarrow$ **fast**: like scenario 2 for 30 seconds, then like scenario 1, emulating an increase in throughput on two WAN links
5. **mayhem**: like scenario 1, but every 5 seconds all links change their bandwidth to random values between 10% and 100% of their nominal bandwidth, emulating heavy background traffic. The random generator is always initialized with the same seed, so the fluctuations are the identical every time this scenario is used.

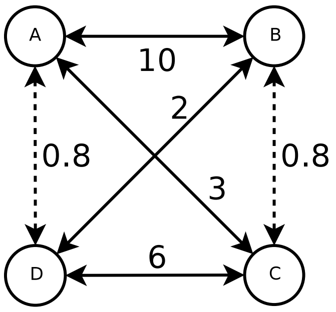
In the first experiment, we emulated four clusters of 16 nodes each. One root node in cluster A is multicasting 600 MB to all other nodes, using three different multicast methods (Balanced Multicasting, BitTorrent and MOB) in the five scenario's described above. In each scenario, Balanced Multicasting used the exact *initial* emulated bandwidth values as input for its algorithm.



(a) wide-area emulation setup: each cluster has a separate gateway node (A, B, C, and D) that routes traffic and applies traffic control



(b) scenario 1: A-D and B-C are fast links



(c) scenario 2: A-D and B-C are slow links

Figure 3. Emulation setup with four clusters (A, B, C, and D) and the two static WAN scenario's; the wide-area links are annotated with their emulated bandwidth in MB/s.

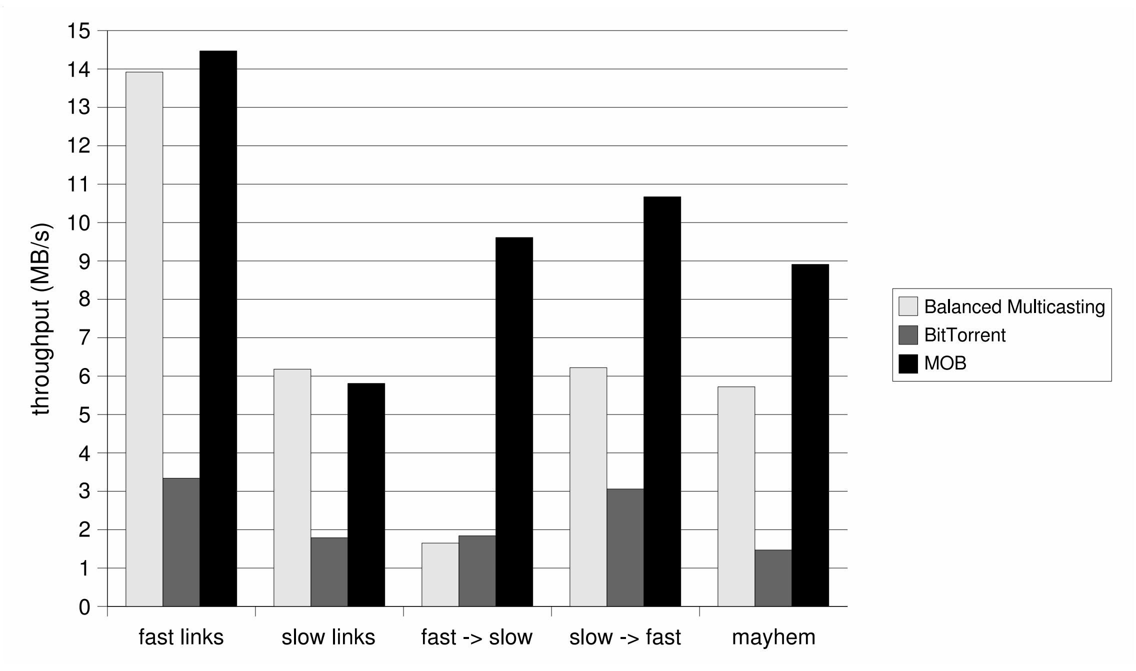


Figure 4. Multicast throughput between four clusters of 16 nodes each connected by emulated WAN links; the root node multicasts 600 MB to all others using three different methods.

Figure 4 shows for each scenario the throughput of each multicast method, calculated as 600 MB divided by the time passed between the moments when the root started the multicast and when the last node has received all data. It can be seen that BitTorrent always performs worst, which is caused by the overhead it creates by sending duplicate WAN messages. MOB performs similar to Balanced Multicasting in the static scenarios, and outperforms it in all three dynamic ones. Only in the static *'slow links'* scenario MOB is slightly slower than Balanced Multicasting, which is caused by a minor slowdown of sending threads by the slow connections between nodes in cluster A and D. In the first dynamic scenario *'fast $\rightarrow$ slow'* , Balanced Multicasting overuses the links that became slow and does not adapt, for which it is heavily penalized. In contrast, MOB adapts and uses the other WAN links to distribute the data, resulting in much higher throughput. In the second dynamic scenario *'slow $\rightarrow$ fast'*, Balanced Multicasting does not use the extra achievable bandwidth that became available on two WAN links due to its sender-side traffic shaping. It therefore achieves the same throughput as in the *'slow links'* scenario, whereas MOB greedily uses the extra bandwidth that became available. In the last dynamic scenario *'mayhem'* , the average throughput of all links is 55% of that in the *'fast links'* scanerio. Balanced Multicasting actually achieves 42% of its throughput in the *'fast links'* scenario, since it continuously uses the same WAN links and the throughput of the bottleneck link in each of its multicast trees determines its overall throughput. MOB on the other hand is much more adaptive and far better in using all the bandwidth that becomes available on the WAN links, achieving 62% of its throughput in the *'fast links'* scenario.

The second experiment used the same WAN scenarios and multicast methods as the first one, but with clusters of different sizes. Clusters A, B, C, and D now consisted of 4, 12, 16, and 32 nodes, respectively. As can be seen from Figure 5, the general performance of MOB is now a little worse than in the previous experiment due to the added asymmetry in the environment. Still, MOB outperforms Balanced Multicasting in all cases, showing good robustness even in the case of disproportionally sized clusters.

In the emulation experiments reported here, the round-trip latency between two remote processes has been 20 ms, which roughly corresponds to communication across central Europe. This latency is caused by the forwarding along two SmartSockets hub nodes. We would have liked to also experiment with higher latencies, by adding emulated latency to the Linux Traffic Control kernel module. Unfortunately, in the case of emulated latency, the combination of LTC with the Ibis SmartSockets did not result in a reliable communication environment. As a consequence, we could not investigate the effects of higher latency on the three multicast protocols in a reliable and reproducible manner.

4.2 Local bottleneck test case (DAS-2)

Our second test case involves multicasting between multiple clusters of the DAS-2. Those clusters are connected by SURFnet's high-speed backbone of 10 Gb/s. Each compute node is equipped with a Myrinet card for fast local communication and a 100 Mbit FastEthernet card for wide-area communication.

In the experiment, we used four clusters located at four Dutch universities: Vrije Universiteit, and the universities of Amsterdam and Leiden, and the Technical University Delft. We use Balanced Multicasting, BitTorrent and MOB to multicast 600 MB from a root node in the VU cluster to all others, using up to 16 nodes per cluster.

Figure 6 shows that both MOB and Balanced Multicasting manage to split the data over multiple network interfaces in parallel, thereby overcoming the throughput of a single wide-area network interface of the nodes. With two nodes per cluster, BitTorrent also mimics this behavior. In this case, the chance the two local nodes connecting to each other is relatively high, causing each cluster to spontaneously behave as a two-node mob achieving similar throughput. With more nodes per cluster, however, the change of this happening decreases quickly, and the amount of duplicate pieces transferred into the same cluster slows down BitTorrent's throughput significantly.

It turned out that the ultimate bandwidth bottleneck in this experiment was the speed with which data could be transferred between nodes inside the same cluster. Balanced Multicasting arranges local nodes in chains over which data is forwarded, which limits the throughput to about 26 MB/s. MOB reaches 29 MB/s by using piece exchanges instead. However, the overhead of this approach increases with the number of nodes per cluster, which is the reason the throughput of MOB decreases a little when 8 or 16 nodes per cluster are used.

5 Conclusions

The completion time of large-data multicast transfers depends primarily on the bandwidth that an application can achieve across the interconnection network. Traditionally, multicasting is implemented by arranging the application nodes in spanning trees over which the data are sent. This method, however, can be very inefficient in a grid environment, where the bandwidth between the sites can be significantly different and also dynamically changing.

In previous work, we have used monitoring information about network bandwidth to construct multicast trees between grid sites [11]. This approach, however, is problematic because it assumes network monitoring systems to be deployed ubiquitously, and because it assumes monitored

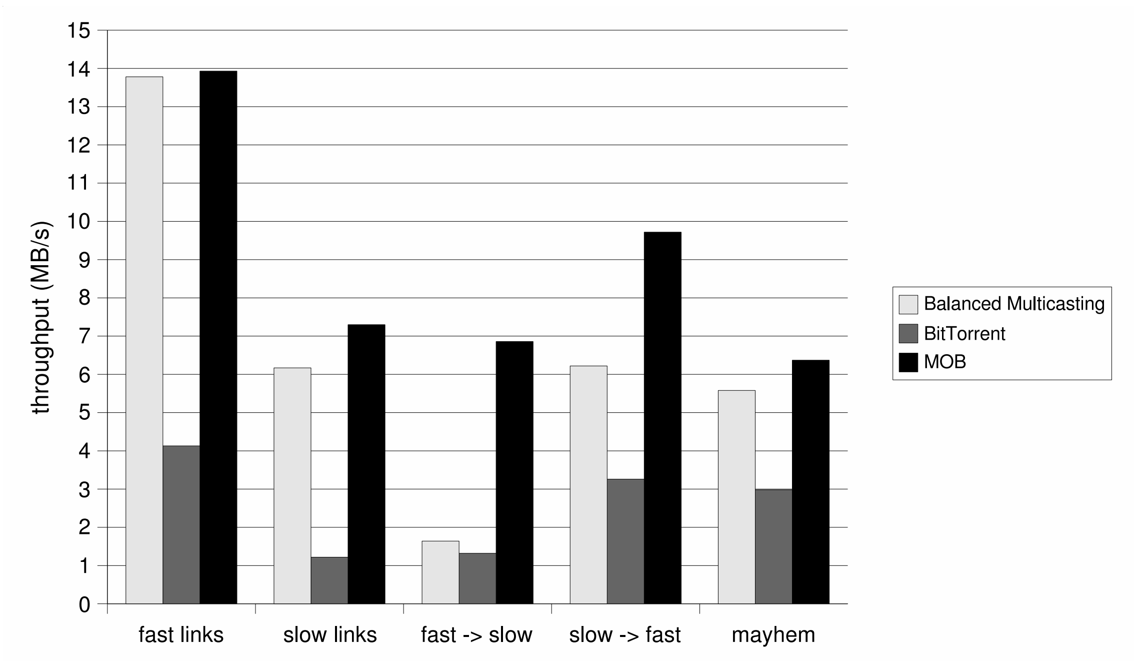


Figure 5. Multicast throughput between four emulated clusters of 4, 12, 16 and 32 nodes, respectively; the root node multicasts 600MB to all others using three different methods.

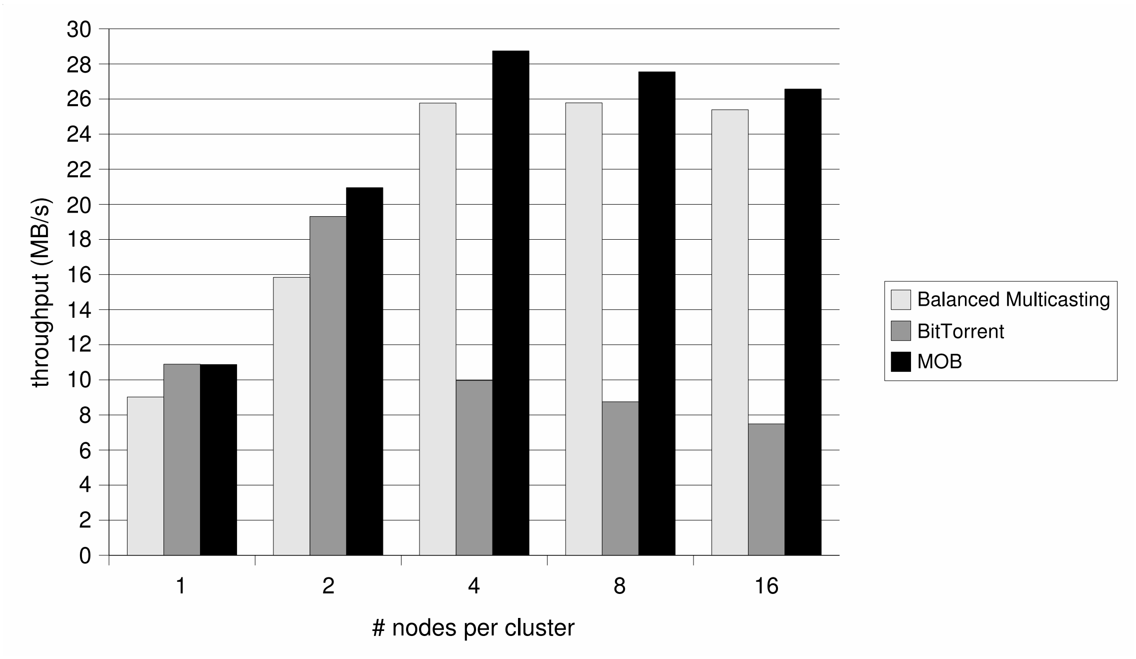


Figure 6. Multicast throughput between four DAS-2 clusters, using 1 to 16 nodes per cluster.

data to be both accurate and stable during a multicast operation, which might not be the case in shared networks with variable background traffic.

In this paper, we have presented *Multicast Optimizing Bandwidth* (MOB), a new multicast approach, inspired by the BitTorrent protocol [4]. With MOB, data transfers are initiated by the receivers that try to *steal* data from peer clusters. Instead of planning multicast trees based on potentially outdated monitoring data, MOB implicitly uses all network links between all clusters, while automatically adapting to the currently achievable bandwidth ratios.

With MOB, nodes within the same cluster team up to form so-called *mobs* that, together, try to steal data from other clusters as efficiently as possible. This teamwork guarantees that all data is sent to each cluster exactly once. In comparison with BitTorrent, this significantly reduces the amount of wide-area communication between clusters. It also divides the bandwidth requirements among all participating nodes in a cluster.

We have implemented MOB (as well as the BitTorrent protocol and Balanced Multicasting [11]) within our Java-based Ibis system [24]. We have experimentally evaluated the three approaches on both the DAS-2 and the new DAS-3 multi-cluster systems installed in the Netherlands. Our experimental evaluation shows that MOB outperforms the original BitTorrent protocol by wide margins due to the much better utilization of the wide-area networks.

We have compared MOB to *Balanced Multicasting* as a (close-to) optimal solution using spanning trees that can be found with complete network performance information. Dynamically changing network performance, however, can not be taken into account by Balanced Multicasting. In the case of stable wide-area bandwidth, MOB automatically achieves multicast bandwidth that is competitive to *Balanced Multicasting*. In most cases, MOB achieves higher bandwidth, actually. This holds for the bandwidth bottleneck being either in the WAN (the global bottleneck case) or in the LAN (the local bottleneck case).

For the global bottleneck case, we have investigated various scenarios with emulated WAN connections inside a single cluster of the DAS-3 system. Besides two static cases (fast and slow), we have investigated three different cases of dynamically changing bandwidth. In all cases, Balanced Multicasting is using the performance information that is valid at the start of a multicast operation. In the first case, we degrade two WAN links to 10 % of their initial bandwidth. As expected, Balanced Multicasting drastically suffers from this situation as its pre-computed spanning trees no longer match the actual network conditions. MOB automatically adapts to the new situation. In the second case, we improve the bandwidth of two WAN links. While Balanced Multicasting cannot use this added bandwidth, MOB does. Finally, our third case randomly changes all bandwidth values every five seconds, emulating heavy background traffic. While Balanced Multicasting only achieves 42% of its throughput in the case of all links being permanently at their maximum bandwidth, MOB reaches 62%.

To summarize, Multicast Optimizing Bandwidth (MOB) is significantly improving achievable multicast bandwidth in multi-cluster grid environments while not using any network performance monitoring data ("zero configuration"). As such, it lends itself as an ideal strategy for multicasting in grid applications. While we are currently seeking a suitable platform for evaluating MOB and its competitors in high-latency environments, we are confident that our findings from this paper will also hold qualitatively in the case of larger latency.

Acknowledgements

This work has been partially funded by the Dutch National Science Foundation (NWO), grant 631.000.003, "*Network-Robust Grid Applications*." The authors would like to thank Jason Maassen for his SmartSockets library that allowed us to emulate wide-area networks.

References

[1] O. Beaumont, L. Marchal, and Y. Robert. Broadcast Trees for Heterogeneous Platforms. In *19th International Parallel and Distributed Processing Symposium (IPDPS'05)*, Denver, Colorado, April 3-8 2005.

[2] Ruchir Bindal, Pei Cao, William Chan, Jan Medval, George Suwala, Tony Bates, and Amy Zhang. Improving traffic locality in BitTorrent via biased neighbor selection. In *The 26th International Conference on Distributed Computing Systems (ICDCS 2006)*, Lisboa, Portugal, July 4-7 2006.

[3] M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. SplitStream: High-bandwidth multicast in cooperative environments. In *ACM Symposium on Operating System Principles (SOSP)*, Lake Bolton, New York, October 2003.

[4] Bram Cohen. Incentives build robustness in BitTorrent. In *Proceedings of Workshop on Economics of Peer-to-Peer Systems*, Berkeley, CA, May 2003. http://www.bittorrent.org/bittorrentecon.pdf.

[5] R. Cohen and G. Kaempfer. A Unicast-based Approach for Streaming Multicast. In *20th Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE INFOCOM 2001)*, pages 440–448, Anchorage, Alaska, April 22-26 2001.

[6] Y. Cui, B. Li, and K. Nahrstedt. On Achieving Optimized Capacity Utilization in Application Overlay Networks with Multiple Competing Sessions. In *16th annual ACM symposium on parallelism in algorithms and architectures (SPAA '04)*, pages 160–169, Barcelona, Spain, June 27-30 2004. ACM Press.

[7] Y. Cui, Y. Xue, and K. Nahrstedt. Max-min Overlay Multicast: Rate Allocation and Tree Construction. In *12th IEEE International Workshop on Quality of Service (IwQoS '04)*, Montreal, Canada, June 7-9 2004.

[8] Linux Advanced Routing and Traffic Control. http://lartc.org/.

[9] The Distributed ASCI Supercomputer 2. http://www.cs.vu.nl/das2/, 2002.

[10] The Distributed ASCI Supercomputer 3. http://www.cs.vu.nl/das3/, 2006.

[11] Mathijs den Burger, Thilo Kielmann, and Henri E. Bal. Balanced multicasting: High-throughput communication for grid applications. In *Supercomputing 2005 (SC05)*, Seattle, WA, USA, November 12-18 2005.

[12] Martin Devera. HTB Linux queuing discipline manual - user guide. http://luxik.cdi.cz/~devik/qos/htb/userg.pdf.

[13] C. Dovrolis, P. Ramanathan, and D. Moore. What Do Packet Dispersion Techniques Measure? In *20th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2001)*, Anchorage, Alaska, April 22-26 2001.

[14] T. Gross, B. Lowekamp, R. Karrer, N. Miller, and P. Steenkiste. Design, Implementation and Evaluation of the Remos Network. *Journal of Grid Computing*, 1(1):75–93, May 2003.

[15] R. Izmailov, S. Ganguly, and N. Tu. Fast Parallel File Replication in Data Grid. In *Future of Grid Data Environments workshop (GGF-10)*, Berlin, Germany, March 2004.

[16] N. T. Karonis, B. R. de Supinski, I. Foster, W. Gropp, E. Lusk, and J. Bresnahan. Exploiting Hierarchy in Parallel Computer Networks to Optimize Collective Operation Performance. In *14th International Parallel and Distributed Processing Symposium (IPDPS '00)*, pages 377–384, Cancun, Mexico, May 1-5 2000.

[17] T. Kielmann, H.E. Bal, S. Gorlatch, K. Verstoep, and R.F.H. Hofman. Network Performance-aware Collective Communication for Clustered Wide Area Systems. *Parallel Computing*, 27(11):1431–1456, 2001.

[18] Thilo Kielmann, Rutger F.H. Hofman, Henri E. Bal, Aske Plaat, and Raoul A.F. Bhoedjang. MagPIe: MPI's Collective Communication Operations for Clustered Wide Area Systems. *ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP)*, pages 131–140, May 1999.

[19] M.S. Kim, S.S. Lam, and D.Y. Lee. Optimal Distribution Tree for Internet Streaming Media. In *23rd International Conference on Distributed Computing Systems (ICDCS '03)*, Providence, Rhode Island, May 19-22 2003.

[20] B. Lowekamp, B. Tierney, L. Cottrell, R. Hughes-Jones, T. Kielmann, and M. Swany. A Hierarchy of Network Performance Characteristics for Grid Applications and Services. Proposed Recommendation GFD-R-P.023, Global Grid Forum, 2004.

[21] J. Maassen, R.V. Nieuwpoort, T. Kielmann, and K. Verstoep. Middleware Adaptation with the Delphoi Service. In *AGridM 2004, Workshop on Adaptive Grid Middleware*, Antibes Juan-les-Pins, France, September 2004.

[22] J.A. Pouwelse, P. Garbacki, J. Wangand A. Bakker, J. Yang, A. Iosup, D. Epema, M.Reinders, M.R. van Steen, and H.J. Sips. Tribler: A social-based based peer to peer system. In *5th Int'l Workshop on Peer-to-Peer Systems (IPTPS)*, Santa Barbara, CA, USA, February 27-28 2006.

[23] V. Ribeiro, R. Reidi, R Baraniuk, J. Navratil, and L. Cottrel. PathChirp: Efficient Available Bandwidth Estimation for Network Paths. In *Passive and Active Measurement workshop (PAM 2003)*, La Jolla, California, April 6-8 2003.

[24] R.V. van Nieuwpoort, J. Maassen, G. Wrzesinska, R. Hofman, C. Jacobs, T. Kielmann, and H.E. Bal. Ibis: A Flexible and Efficient Java-based Grid Programming Environment. *Concurrency & Computation: Practice & Experience*, 17(7-8):1079–1107, June-July 2005.

[25] K. Verstoep, K. Langendoen, and H.E. Bal. Efficient Reliable Multicast on Myrinet. In *International Conference on Parallel Processing*, volume 3, pages 156–165, Bloomingdale, IL, August 1996.

[26] R. Wolski. Experiences with Predicting Resource Performance On-line in Computational Grid Settings. *ACM SIGMETRICS Performance Evaluation Review*, 30(4):41–49, March 2003.