File sharing using IP-Multicast

Kai Trojahner, Peter Sobe
University of Luebeck, Germany
Institute of Computer Engineering
email: sobe@iti.uni-luebeck.de

Abstract: File sharing systems cause a huge portion of traffic in the Internet. With respect to the peer-to-peer approach, unicast delivery of content is the common case. Unfortunately, an inherent characteristic of filesharing systems employing unicast communication is that the sum of all download rates cannot exceed the sum of all upload rates. As asymmetric internet connections become increasingly popular, this restriction is clearly noticeable in form of low download rates. Together with the observation that popular files are requested by many users within a short time frame, this motivates the usage of multicast communication. This paper analyzes the effects of using IP multicast in a filesharing system.

1 Introduction

Filesharing over peer-to-peer networks is one of the most common applications in the Internet. While the exchange service *Napster* only supported the exchange of music files, quickly systems for sharing of arbitrary content came up. Nowadays, about 60 percent of the Internet's total transfer capacity is occupied by exchange of music, video and software files. As peer-to-peer services cause high costs, it is worth to check more efficient techniques for data transport over peer-to-peer networks.
One new aspect is to employ multicast of the underlying network. Another aspect that is already used in peer-to-peer networks is the aggregation of upload bandwidth. This means that a download is served by many parallel uploads of different blocks in parallel. This technique is motivated by the asymmetric nature of peers access technology, where the modem connection allows only fast download and relatively slow upload.
In this paper we analyze principles for filesharing in order to maximize the total transfer capacity of a peer-to-peer network by using data splitting and IP multicast. Furthermore these techniques help minimizing the amount of occupied bandwidth for a fixed number of data movements in a peer-to-peer network. In section 2 we motivate these techniques for a number of traffic scenarios. A prototype implementation of a multicast based peer-to-peer network called *Bazaa* then is described in section 3. Concrete transfer time measurements are reported in section 4 and compared with the predicted behavior.
In the current state of the Internet it seems to be difficult to employ multicast, because the number of world wide IP multicast addresses is limited and the way to reserve such an address is too costly for a short term dynamic usage of multicast groups for transfers in peer-to-peer networks. This may have been a reason for the absence of multicast based peer-to-peer implementations today. However, in future scenarios more multicast

addresses will be available (IP V6) and also dynamic formation of multicast groups will be supported more efficiently.

2 Case Study

For all possible cases we distinguish three representative scenarios: *One-To-Many* (a data object is distributed from a single peer to a group of peers), *Many-To-One* (many peers are storing a particular data object and a single peer is requesting that one) and *All-to-All* (all peers request all data objects that are not available locally. The other peers each own a single data object that is requested by the other peers). The latter scenario serves for a maximum exploitation of the network transfer capacity. For these representative scenarios we evaluate transfer times until all requesters have received the data object. The objective is to compare the transfer times of the following communication models:
Unicast-filesharing with store and forward: - A data object is transferred completely from a sender node to a destination node. Only when the entire file is received, other transfers may take place using the source file and the new copy of that file. Napster is one example of such a distribution technique.
Unicast-filesharing with pipelining/splitting: - Blocks of data objects are transferred independently. The order of blocks may be interchanged, a requester is allowed to receive several blocks of a data object from several sender nodes in parallel (Edonkey 2000).
Multicast-filesharing without pipelining: - Complete data objects are transmitted from a single sender to a group of receiver nodes. There is no splitting.
Multicast-filesharing with pipelining/splitting: - Blocks will be independently transferred by several sender nodes to a group of receiver nodes. The receiver nodes are able to receive the data streams of all sender nodes that are transmitting blocks of the requested data object.

For the analysis of all scenarios the following symbols are used. By T , ..., T_m we denote the m active peers of a filesharing network. R_{up} is the upload rate of each peer. As download rate R_{down} we assumed $R_{down} \geq (m-1)R_{up}$. The total transfer rate is bounded by the sum of upload rates of all nodes. In the evaluation, all data objects are of the same size. Further we assume, that the network behind does not act as a limiting factor.

2.1 Scenario One-to-Many

Peer T owns a data object of size V. There are $m-1$ peers T , ..., T_m that are requesting that data object.

Unicast Filesharing with Store-and-Forward
Initially, T sends the data object to T . Subsequently, T and T are able to transfer the data object to T and T , and so on. With each step, the system is able to double the number of peers that own a copy of the requested data object (as illustrated in figure 1). The number of needed steps therefore is $\log\ (m)$ for $m \geq 2$ and $m = 2^x : x \in N$. The time for any number of requesting peers ($m \geq 2$) is given in formula 1.

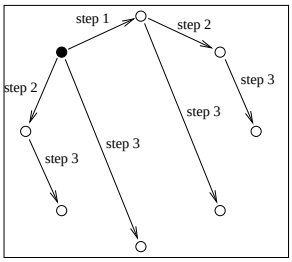

Figure 1: Distribution by unicast and the store and forward principle.

$$t = (\lfloor \log\ (m-1) \rfloor + 1)\frac{V}{R_{up}}. \tag{1}$$

Unicast-filesharing with pipelining

A first augmentation is caused by pipelined transfer. Here, a data object is split in n blocks $b\ , ..., b_n$, each of size B. Looking at the same scenario as in the store and forward case, T only sends the first block b to peer T . That pipelined transfer in a ring is continued with new blocks until the sender has sent the last block of the data object. Until that point, the transfer needed $n\frac{V}{nR_{up}}$ steps.

When this phase is completed, all nodes that have already left the transfer pipeline may send blocks to nodes that are positioned at the last positions in the ring. The rest of nodes on these last positions can be supplied with the needed blocks in fewer steps compared to a pipelined transfer. To complete the transfer in such a way takes another $\lfloor \frac{m-}{} \rfloor \frac{V}{nR_{up}}$ steps. An illustration of this process can be seen in figure 2.

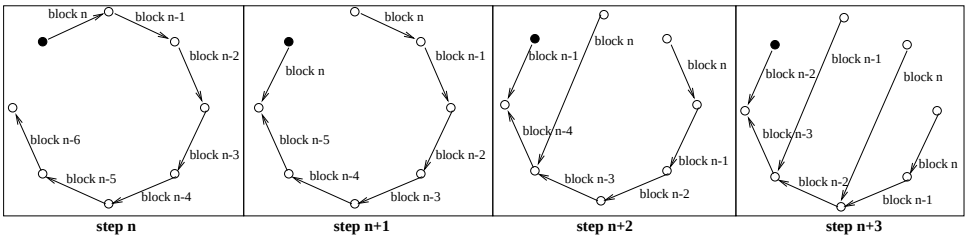

Figure 2: Broadcast of a data object by pipelined transfer

Taking into account that a step now only covers the transfer time of a block, t is obtained as:

$$t = (n + \left\lfloor \frac{m-1}{2} \right\rfloor)\frac{V}{nR_{up}} = \frac{V}{R_{up}} + \left\lfloor \frac{m-1}{2} \right\rfloor \frac{V}{nR_{up}}. \tag{2}$$

It is worth noting that the transfer time approaches $\frac{V}{R_{up}}$ for small block sizes.

Multicast filesharing
The entire data object is sent to all requesting peers via multicast. The case of pipelined transfer is not taken into account for evaluation, since there is no difference to a complete transfer of a data object. Hence, no matter how many peers are receiving the data object, we obtain:

$$t = \frac{V}{R_{up}}. \tag{3}$$

2.2 Scenario Many-to-One

In this scenario, the focus lies on a single peer requesting a data object from the peer-to-peer network while all other peers can contribute their maximum upload capacity for that delivery. In particular, $m-1$ peers T , ..., T_m locally store a data object of size V, whereby peer T is requesting that one. Having only one receiver, it is obvious that the usage of multicast has no effect. Only the usage of data splitting causes the effect that many peers can send blocks in parallel.

Unicast filesharing with Store-and-Forward
A peer T_i with $i \in [2, m]$ sends a complete data object to the requester. The needed time for the transfer is

$$t = \frac{V}{R_{up}}. \tag{4}$$

Filesharing with splitting
As in the scenario One-to-Many, the data object consists of n blocks. Several peers T_i : $i \in [2, n]$ are able to transfer one or more blocks to T . The parallel transfer of the blocks causes a reduced transfer time of

$$t = \left\lceil \frac{n}{m-1} \right\rceil \frac{V}{nR_{up}}. \tag{5}$$

2.3 Scenario All-to-All

All peers T , ..., T_m each own a local data object D_i with $i \in [1, m]$, where D_i is locally stored at T_i. In the transfer scenario, each peer T_x request all data objects D_y : $y \in [1, m], y \neq x$ at once.

Unicast filesharing
Each peer sends its own data object to a neighbor. After that the received objects are sent to the next neighbor and so on. With such a ring-wise transfer of data objects, after $m-1$ steps each peer has been traversed by each data object. So all requests for data objects are fulfilled. With $m-1$ steps and a transfer time of $\frac{V}{R_{up}}$ per step, the total time is:

$$t = (m-1)\frac{V}{R_{up}}. \tag{6}$$

Unicast filesharing with pipelining
The pipelined variant is based on a ring-wise transfer too. On the ring, data objects are transfered blockwise in a pipelined manner. Compared to the store and forward scenario, we need $(m-1) \times n$ steps. This is a larger number of steps but a step only covers the transfer of a block. This is expressed in formula 7. Pipelining does not cause an improvement for all-to-all transfer.

$$t = (m-1)n\frac{V}{nR_{up}} = (m-1)\frac{V}{R_{up}}. \tag{7}$$

Filesharing using IP multicast
All peers request the missing data objects in parallel. So, each peer receives $m-1$ requests that fit to the file it owns and starts to send out the data object by multicast. When a peer uploads the data object with a rate of R_{up}, the incoming data streams accumulate to a rate of $R_{down} = (m-1)R_{up}$ on each peer. A particular data object is received in time $t = \frac{V}{R_{up}}$. Because all transfers are done in parallel, the same time applies for all data objects. Independently from whether pipelining is used or not, the total time needed to transfer all data objects is given by:

$$t = \frac{V}{R_{up}}. \tag{8}$$

2.4 Comparison

Transfer pipelining causes benefits for one-to-many and many-to-one scenarios. Using multicast speeds up the scenarios one-to-many and all-to-all. Activating both techniques, in all scenarios will cause the best time behavior, regardless that in some particular cases a single technique does not cause any effect. Table 1 summarizes complexity order for transfer duration for the three scenarios assuming a constant file size.

	One-to-Many	Many-to-One	All-To-All
Unicast	O($log\ (m)$)	O(1)	O(m)
Pipelining/Splitting	O(1+$\frac{m}{n}$)	O($\frac{}{m}$)	O(m)
Pipelining/Splitting and multicast	O(1)	O($\frac{}{m}$)	O(1)

Table 1: Transfer time complexities related to a system size of m nodes and splitting over n blocks

3 Bazaa - A multicast based filesharing system

The case study tells that multicast is beneficial in the One-to-Many and the Many-to-Many scenario. Solely, if only a single peer requests a data object, multicast has no effect. This motivates an implementation of a peer-to-peer filesharing system based on IP multicast. Bazaa is a Java-based implementation of such a peer-to-peer system. As described later,

the data delivery strategy of the peers give multicast scenarios a preference. According to a classification of peer-to-peer systems given in [An02], Bazaa (i) is purely decentralized and (ii) is based on an unstructured network. There is no relation of node addresses or a place within an overlay topology with the placement of data objects.

3.1 Identification of data objects

To identify multiple copies of a data object, even when the users stored them under different filenames, each data object is indexed by a non-ambiguous key. As keys we use Ed2k-URLs (uniform resource locator) adopted from Edonkey 2000 with the following structure: `ed2k://|file|<name>|<size>|<Blocked-MD4-Hash>|`
Also the field `Blocked-MD4-Hash` is chosen similar to the Edonkey 2000 block structure. A data object consist of blocks of 9500 kByte, the last block is allowed to be shorter. For calculation of the `Blocked-MD4-Hash`, each block's MD4-Hash [Ri91] is calculated, separately. If the data object consists of a single block, this value is used finally. Otherwise a MD4-Hash over all MD4-Hashes of the blocks is calculated that forms the final `Blocked-MD4-Hash`.

3.2 Search groups

Many filesharing systems, such as Napster or Edonkey 2000 are based on one or more central directory servers. These centralized instances keep track of peers and data objects and so can respond to search queries. Only the transfer itself is delegated to the peers.
Even though centralized servers are practically not a bottleneck in terms of scalability with the number of peers, the existence of any centralized instance is subject to attacks or censorship (e.g. Napster). The first filesharing network with a decentralized infrastructure came up with Gnutella, which distributed the search for data objects. The peers are embedded in a loosely coupled overlay network. Requests are forwarded in that network from peer to peer until a location is found that is able to deliver the requested data. A time-to-live value is used to bound the number of hops and to avoid flooding of that overlay network.
If a filesharing system can employ IP multicast, another solution for handling requests is reasonable. In *Bazaa*, all peers join a particular multicast group on which the requests are propagated. A search for a particular Ed2k-URL is driven by sending the search keyword to the search group via multicast. Each receiver tries to resolve the query and possibly sends back a result by unicast to the requester.

3.3 Transfer Requests

A client that knows the Ed2k-URL of a data object, may request them over the peer-to-peer network. In a conventional filesharing system such a request would lead to the identification of the peers owning the data object in a more or less narrow distance. After that a single peer would be asked to initiate the transmission. As *Bazaa* tries to maximize

the benefit of multicasting, it needs to apply a different protocol:
Each request sent to a search group contains information about the needed blocks of the requested data object. This is done with a bit vector in which each bit represents a particular block. A peer that receives a transfer request uses this bit vector to build a priority queue for all requested blocks. The priority for a block that is requested to transfer is calculated by

$$Priority = \frac{NumberRequests}{Availability}. \tag{9}$$

With each request in which the particular bit for a block is set, the number of requests is incremented. In contrast, when a block is not required, its availability is incremented.

3.4 Transfer invocation

A peer with available upload capacity selects the upper entry in the priority queue and starts a transfer to as many receivers as possible. Therefore, the sender chooses a multicast group for the transfer. Before the transfer starts, the requesters are asked again via unicast, whether the block is still needed or not. If there is at least one requester still needing the block, the multicast transfer is initiated by a protocol described in 3.5. As each receiver only agrees to receive a block once, only very few block transfers are redundant.

3.5 Transfer Protocol

Unfortunately, there is no equivalent to TCP for safe communication over multicast. *Bazaa* therefore implements its own protocol based on UDP messages for reliable transmission of data. Similar to RMTP [SY97], the protocol transmits a large block of data in multiple passes. After each pass the receivers are asked which parts of the block they did not receive yet. These will then be retransmitted until all the receivers have acknowledged the whole block. While this protocol is simple and very scalable it does not implement any kind of flow or congestion control.

4 Experimental Performance Evaluation

The experiments have been done on a cluster of double processor machines, connected by a 100 Mbit/s switched Ethernet. The results express transfer times, measured from begin of the first transfer until the end of the last transfer for the selected transfer pattern. For all experiments, the upload capacity of the sender has been bounded to $256 \frac{kByte}{s}$ in order to do not exceed the total network bandwidth or the disk access rates.

4.1 Scenario One-to-Many

To evaluate the One-to-Many scenario, we uploaded a data object of 40 MByte size to a number of receiver peers in parallel, whereby the number of receivers has been varied.

num. requester	transfer time	num. requester	transfer time
1	166 s	9	165 s
2	171 s	10	166 s
3	165 s	11	180 s
4	165 s	12	165 s
5	164 s	13	171 s
6	169 s	14	170 s
7	166 s	15	184 s
8	165 s	16	172 s

Table 2: Transfer time for a data object of a size of 40 MByte.

num. contributors	transfer time
1	269 s
2	153 s
3	115 s
4	77 s
5	77 s
6	77 s
7	39 s

Table 3: Download time from the perspective of a single peer.

The transfer times are shown in table 2.
The transfer time remains constant with an increasing number of receiver peers. A predicted time of 160 seconds is very closely reached, leaving a little overhead time caused by the protocol. The main reason for that overhead are the FIN packets that are send for one second in order to terminate a transfer, even in the case of packet loss. Further experiments with upload rates bounded to 2048, 4192 and 8192 kByte/s show a similar good scalability, whereby the absolute transfer time was shorter and the overhead caused by FIN packets has been more visible.

4.2 Scenario Many-to-One

For the Many-to-One scenario we measured the time that is needed by a varied number of sender peers, to supply a single peer with a data object of size 65 MByte (see table 3). The measured transfer times confirm the predictions for the Many-to-One scenario using multicast. The overhead mentioned in 4.1 here is sinking with increasing number of sender peers.

num. peers	transfer time
2	170 s
3	170 s
4	195 s
5	195 s
6	200 s
7	230 s
8	245 s

Table 4: Data objects of size 40 MByte (5 blocks), exchanged in an all-to-all scenario.

4.3 Scenario All-to-All

Each peer is the owner of a 40 MByte data object and all other peers are requesting all data objects. As result of the transfer, each peer will have a copy of each data object. As shown in table 4, *Bazaa* does not scale as good as predicted. The time needed for an all to all transfer among eight peers is by 50 percent longer than the time needed for a data exchange among two peers. Looking more into the details, peers are overloaded when receiving many streams and reading a block (9500 kByte) from the disk in parallel. The access path to the disk and to the network are concurrently used by many data streams. This problem is getting more visible with higher upload rates. But for all that, the transfer capacity is much higher than in systems using unicast. Using the same upload rate, a unicast based system would need about 19 minutes, which is nearly by the factor of five slower than the transfer time achieved by *Bazaa*.

5 Related Work

Multicast and Overlay Networks
An example for a system that directly is built upon IP multicast has not been found by the authors. Instead, many peer-to-peer networks build their own multicast functionality along the logical connection network of peers. It is common to use peer-to-peer networks as an overlay to the physical network structure and use application based multicast (e.g. Tapestry [Zh03] as multicast platform for Internet wide storage networks). Multicast is used on a higher level for instance to inform peers about updates of data objects.
Although, multicast is not a scalable and widely available function in the Internet hardware and protocols today, it is worth to study multicast transfer for peer-to-peer networks. On the one hand, multicast may be supported in an overlay network that is responsible for message routing in the network. On the other hand, future Internet hardware should be able to carry the traffic caused by relatively expensive group management protocols.
Especially if large multimedia files are transfered over such a peer-to-peer network, multicast will become an essential feature. An example for a media streaming peer-to-peer architecture using multicast trees is given by Zigzag [THD03].

Data Locations and Indexing
A detail that was not in the focus of Bazaa until now is the structure of the network. At the moment a file is stored at all locations that have downloaded them. There is no relation between the topological position and placement of data objects. Systems such as Freenet [Cl02] assign data objects locations in a topology, where they are likely to find based on a search key. These structures do not conflict with multicasting.

6 Conclusion

Based on a comparison of several communication strategies, it could be shown that pipelining of data transfer in peer-to-peer networks leads to benefits for one-to-many and many-to-one scenarios. Using multicast of an underlying network speeds up the scenarios one-to-many and all-to-all. Thus, a combination of pipelining and multicast is promising for new peer-to-peer filesharing systems. Bazaa is a prototype implementation of a filesharing peer-to-peer system that is based on these principles. The usage of IP multicast is a distinct feature that has been in the focus of this paper. It could be shown that in a network with bounded upload rate the total transfer rate is scaling much better with the number of peers in the system compared to unicast delivery. In contrast to common filesharing systems, the invocation of transfers is triggered by the owner of files who calculate a priority for sending particular files and blocks of them. This way of transfer invocation is advantageous to exploit multicast in applicable scenarios.

References

[Ri91] Rivest, R.L.: The MD4 message digest algorithm. In Advances in Cryptology – Crypto'90, pages 303-311, New York, 1991. Springer-Verlag

[Cl02] Clarke, I., Miller, S.G., Hong, H.W., Sandberg, Wiley, B.: Protecting Free Expression Online with Freenet. IEEE Internet Computing, January-February 2002, IEEE Computer Society, 40-49

[Zh03] Zhao, B.Y., Huang, L., Stribling, J., Rhea, S.C., Joseph, A.D., Kubiatowicz, J.D.: Tapestry: A Global-scale Overlay for Rapid Service Deployment. In: IEEE Journal on Selected Areas in Communications, 2003

[An02] Androutsellis-Theotokis, S.: A Survey of Peer-to-Peer File Sharing Technologies. White Paper, ELTRUN Athens University, Greece, 2002

[SY97] Shiroshita, T., Yamanouchi, N.: Reliable Multicast Transport Protocol(RMTP) and its Scalability. IETF, TSA, Reliable Multicast Group, Memphis, TN, April 1997

[THD03] Tran, D.A., Hua, K.A. Do, K.K.: A Peer-to-Peer Architecture for Media Streaming. To appear in IEEE Journal on Selected Areas in Communications, Special Issue on Advances in Service Overlay Networks, 2003