Using Reliable Multicast for Scalable Information Dissemination

Stefan Dresler, Markus Hofmann[1], Markus Roth
Institute of Telematics, University of Karlsruhe
Zirkel 2, 76128 Karlsruhe, Germany
E-Mail: [dresler, hofmann, roth]@telematik.informatik.uni-karlsruhe.de

Abstract. Widespread deployment of multicast networking technologies has not only become an enabler of new forthcoming applications, such as video conferencing or webcasting. Equally important, it also enables well-known applications to scale and to service many users without overloading network and server resources. However, the use of multicast techniques in certain application environments is not always straightforward. This paper presents the design of a forthcoming push-like application based on multicast transmissions, and it discusses an approach to enable efficient mail delivery via multicast. Both applications have been implemented on top of the *Local Group based Multicast Protocol (LGMP)* [Hof98], which provides scalable and reliable multicast data transfer. For mail distribution, the main goals were transparency to the user and smooth integration into the existing mailing infrastructure. It turns out that using underlying multicast network services does not only improve scalability. Furthermore, it makes application development less complex, because data replication no longer need to be realized at application level.

1 Introduction

Today's Internet is mainly organized for "pull" delivery of information. The user employs a browser to search the Internet and individually requests the desired information on demand. A typical example for "pull" based information delivery is accessing web pages. However, it is also desirable to have something similar to a subscription service, in which a user subscribes to content and has it delivered automatically. This approach is called "push". Application examples are subscription to and delivery of stock quotes, customized news headlines and news articles, or cafeteria menus and conference announcements. More application examples can be found in [Mil98]. Existing "push" services use the architecture of defining content-based *channels* allowing customers to tune in to channels according to their interests and preferences [Bac99, Mar99, Poi99].

Another very popular "push" architecture is e-mail. Although the "push" principle might not be obvious, information is sent directly from the source without explicit request through receivers. People simply subscribe to mailing lists and automatically receive information according to the outline of the mailing list chosen. However, there is a significant difference between the "push" architecture based on channels and pushing via e-mails. With the channel-approach, clients simply tune in to the channel and, most times, the sender does not know the receiver set. There is no absolute delivery guarantee. In the case of mailing lists, the distributor exactly knows all the receivers and directly forwards mails to them using unicast transfers. If there are any problems with mail transmission, the distributor is notified and a retransmission will be rescheduled.

It is obvious that both pushing architectures will benefit from underlying multicast network services. However, usage of multicasting might not be straightforward, and unreliable IP multicast service needs

[1] Markus Hofmann is currently working at the Networking Software Research Department at Bell Laboratories in Holmdel, New Jersey, USA.

to be extended to provide some sort of reliability. Section 2 briefly describes a reliable and scalable multicast protocol that is suitable for all kinds of push-based applications. It has been used to implement a channel-based ticker application as described in Section 3 and a system for multicast-based mail delivery as described in Section 4.

2 Reliable Multicast Transfer using LGMP

The *Local Group based Multicast Protocol (LGMP)* [Hof98] supports reliable and semi-reliable transfer of both continuous media and data files. It is based on the principle of subgrouping for local error recovery and local acknowledgment processing, as defined by the *Local Group Concept (LGC)* [Hof96]. Receivers dynamically organize themselves into subgroups, which are called *Local Groups*. They dynamically select a *Group Controller* to coordinate local retransmissions and to handle status reports. The selection of appropriate receivers as Group Controllers is based on the current state of the network and of the receivers themselves. In contrast to other reliable multicast protocols, no manual or administrative intervention is necessary. LGC subgroups are self-organizing and self-adapting, thus, improving fault-tolerance and system reliability. However, the selection of Group Controllers is not a task of a data transfer protocol like LGMP. Instead, a separate protocol has been defined and implemented, which is named *Local Group Configuration Protocol (LGCP)* [Hof98]. LGCP provides mechanisms for automated self-configuration of Local Groups and for dynamic reconfiguration in accordance with the current network load and group membership.

In LGMP, packet errors are firstly recovered inside Local Groups using a receiver-initiated approach. Missing data units are requested from the sender or a higher level Group Controller only if not even a single member of the Local Group holds a copy of the missing data unit. Otherwise, errors will be recovered by local retransmissions. This mechanism is different from tree-based protocols using strong hierarchical error recovery [YGS95, LiP96]. In strong hierarchical error recovery, Group Controllers always request missing data packets from their parent, even if the data packet has been successfully received by a member of their subgroup. LGMP ensures full reliability and efficient buffer utilization by using a novel, three-state acknowledgment scheme. Significant performance benefits of LGMP/LGCP have been proven in worldwide MBone experiments including up to 50 machines in 16 different countries [Hof98].

LGMP as well as LGCP have been implemented and tested on a variety of different platforms, such as Linux, Solaris, Digital Unix, SCO UnixWare, Windows 95 and Windows NT. There is also a Java-API available for LGMP/LGCP. The source code of both protocols is available free of charge on the World Wide Web [LGC99].

3 Two-Channel-Based Information Delivery

Users subscribe to an information delivery service to get up-to-the-minute information on topics of their specific interests and to receive future updates on this information. Services might provide stock quotes, news headlines or any other kind of information. It is essential that such services deliver the most recent version of requested information as fast as possible and that updates are sent as soon as they are available. Users will not accept outdated information nor are they willing to wait several minutes to get the requested information. Furthermore, they do not want to receive any unwanted or redundant data.

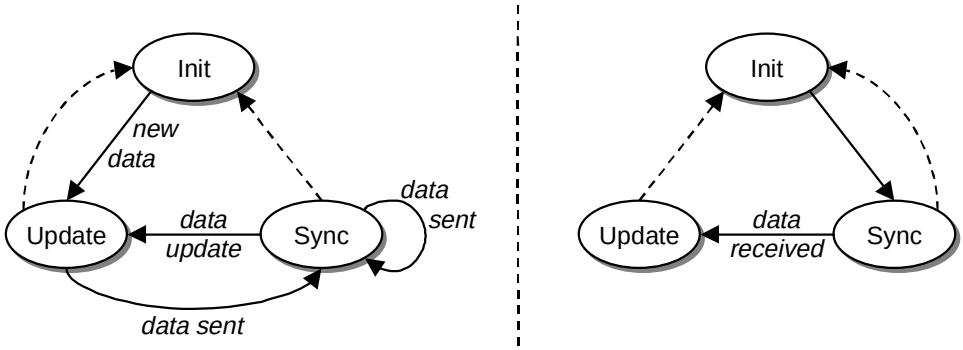
A naïve approach would use unicast transmissions for implementing such a delivery service. Users register themselves at the service provider and immediately get the current information via unicast transfer. When new information is available, the service provider will transmit the updated information to all registered users using multiple unicast transmissions. It is obvious that this approach does not scale.

A straightforward extension would deliver updated information using a single multicast transfer. Interested users simply join the multicast group and wait for the next update to be sent. By staying in the multicast group, they automatically receive future updates, too. While this approach scales very well, it is feasible only if the information is updated in very short time intervals. Otherwise, new subscribers would have to wait relatively long to receive any data. News headlines or stock quotes, for example, might be modified every few hours with no updates in between. In this case, users that subscribe to the service right after an update has been sent would have to wait several hours to receive any information.

Periodic and frequent re-broadcasting of the most recent data solves the problem of synchronizing new subscribers. It allows receiving current data within a short time interval after joining the multicast group, independent from the update interval. However, periodic re-broadcasting results in transmission of redundant data. Once a subscriber has received the most recent version, she no longer needs to get the same data over and over again. Only updates of the information should be transmitted to those subscribers. In order to solve these problems, we propose a two-channel-based information delivery scheme.

3.1 Update-Channels and Sync-Channels

As stated above, periodic re-broadcasting of data is necessary to allow new subscribers immediate reception of current data. In order to avoid redundant data transmissions, we introduce an additional distribution channel called *Update-Channel* (Channel 1). This channel will be used to distribute data updates utilizing a reliable multicast protocol. Periodic re-broadcasts of current data are sent on a separate channel named *Sync-Channel* (Channel 2). The Sync-Channel provides means for synchronizing new subscribers. Both channels are represented by different multicast addresses. Figure 1 illustrates the state machines of a sender and a receiver and explains the usage of both channels.



(a) State Machine of Sender (b) State Machine of Receiver

Figure 1: State Machines of Two-Channel-Based Sender and Receiver

Sender and receiver are operating on three different states, namely `Init`, `Update`, and `Sync`. The first time a user issues a data transmission, the sender changes from `Init` to `Update` (Figure 1a). In this state, a sender starts transmitting data on the Update-Channel. Once the data has been sent successfully, the sender changes from `Update` to `Sync` and periodically re-broadcasts data on the Sync-Channel using a different multicast address. When the user updates data, the sender changes back to `Update` and transmits the modified data on the Update-Channel. After successful transmission, he switches back to `Sync` and keeps on re-broadcasting the updated data on the Sync-Channel.

As shown in Figure 1b, new subscribers go from `Init` to `Sync` and first in to the Sync-Channel. After receiving the current re-broadcast, they change to `Update`, thus leaving the Sync-Channel and tuning into the Update-Channel. From now on, subscribers will no longer receive any re-broadcasts; instead, data updates will be received on the Update-Channel. For further optimization, new subscribers

can tune in to both channels in `Sync` state and change to `Update` as soon as any data has been received on either of both channels.

The effect of using two separate channels for re-broadcasting and updating is illustrated in Figure 2.

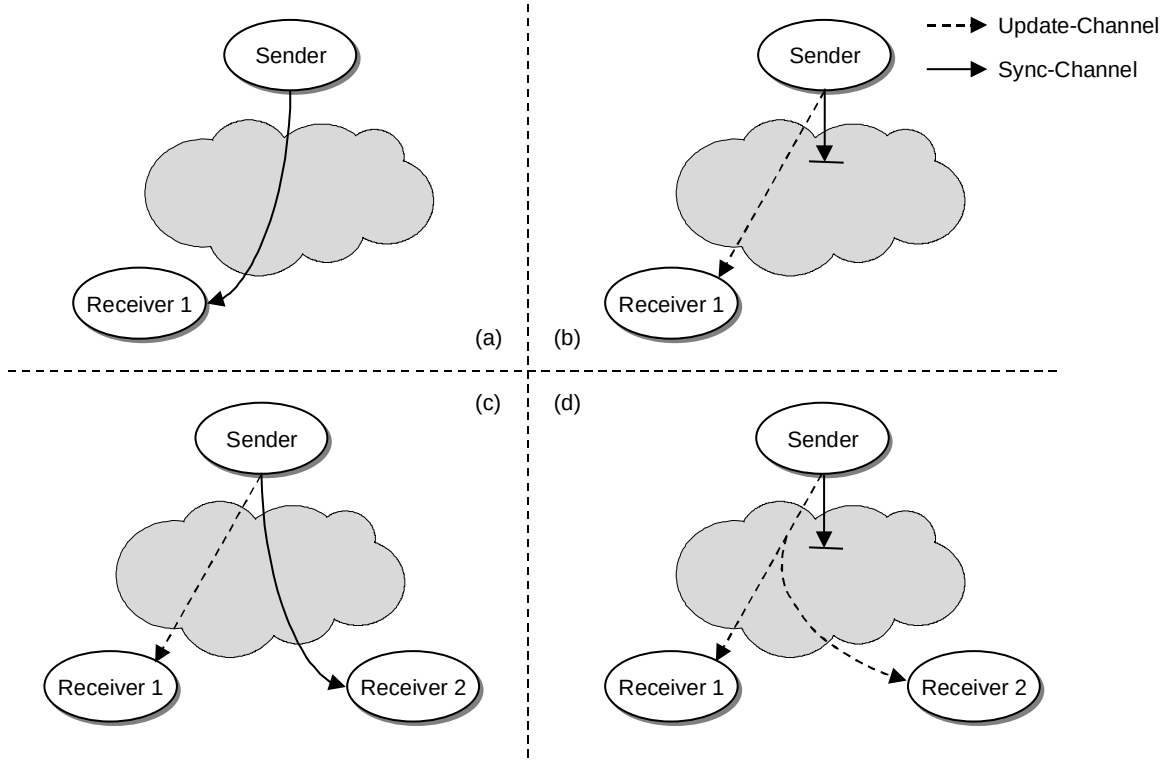


Figure 2: Network Effect on Using Separate Channels

As shown in Figure 2a, a new subscriber "Receiver 1" tunes in to the Sync-Channel, thus receiving periodic re-broadcasts. In Figure 2b, "Receiver 1" left the Sync-Channel and joined the Update-Channel. In this situation, multicast routing prevents forwarding of re-broadcasts into the network, because there is no receiver subscribed to the Sync-Channel. Therefore, re-broadcasts will not consume any network bandwidth[2] and will not be forwarded to any subscriber. However, "Receiver 1" is still able to receive data updates, because she has tuned in to the Update-Channel. In Figure 2c, "Receiver 2" joins the Sync-Channel, thus causing the multicast protocol to forward re-broadcasts to herself. After "Receiver 2" left the Sync-Channel and tuned in to the Update-Channel, re-broadcasts will no longer be forwarded into the network (Figure 2d). Consequently, receivers get only the data that they are interested in, while new subscribers are still able to synchronize immediately.

3.2 A Two-Channel-Based Ticker Application

Based on the idea of two-channel information delivery, we have implemented a so-called *Ticker Application*. The application allows users to specify arbitrary messages (e.g. news headlines, stock quotes, etc.) that are transferred to all subscribers. At the receiving sites, the messages are displayed in a marquee on screen. Figure 3 shows a screenshot of the Ticker-Manager, which allows users to define new and to edit existing ticker services. The Ticker Manager is also used to launch separate windows for displaying received messages and for updating the message that is sent via the ticker. Although the text message is edited manually, the application can easily be modified to read data from disk or to receive it from another application. It would also be possible to further process received data instead of simply displaying it on the screen.

[2] Here, we do not consider the network load that might be caused in the subnet of the sender.

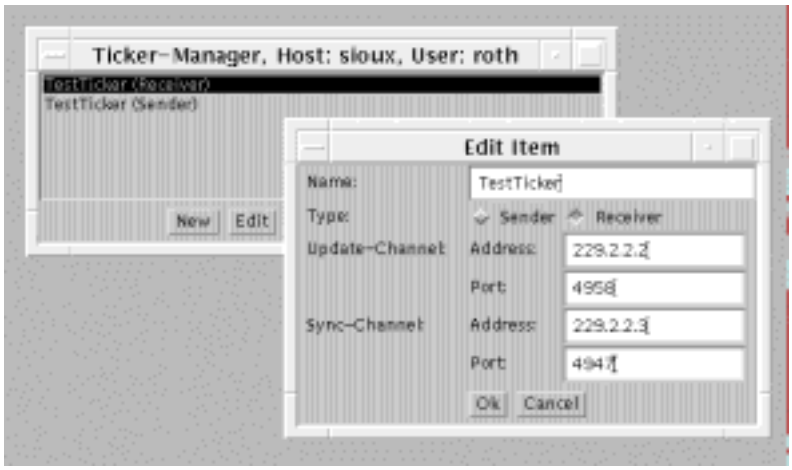


Figure 3: Screenshot of Ticker-Manager

In the Edit menu, users have to specify a name for the ticker service and the multicast addresses of the Sync-Channel and the Update-Channel, respectively. If the user selects type “Sender”, he will become the data source of the ticker service. On pushing the “Start” button of the Ticker-Manager, a new window “Ticker-Sender” pops up allowing the user to enter or to modify the ticker message (Figure 4). The message will be transmitted upon hitting the “Send” button. If the user has selected type “Receiver”, another window “Ticker-Receiver” pops up showing received ticker messages in a marquee (Figure 4).

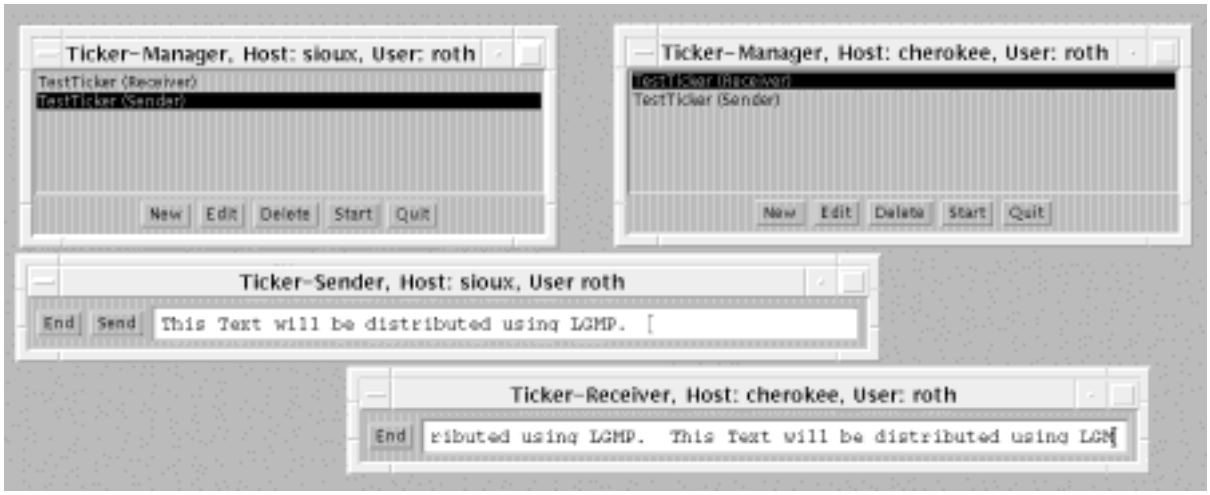


Figure 4: Screenshot of Ticker Marquee and Ticker Input Line

The ticker has been implemented in Java. For interaction with LGMP, a Java-based protocol interface has been written. The interface allows Java applications to transparently access the entire LGMP functionality. Although the current ticker implementation offers a very simple and restricted service, it can easily be extend to provide more complex distribution services.

4 Mailcasting – An Architecture for Multicast Mail Distribution

Mailing lists are common in the Internet and account for a certain bandwidth portion in the network. Their goal is to forward every mail addressed to the list to all receivers. If the mail sender is in Germany and there are several receivers in the USA, multiple copies of the mail are sent across the ocean. Although there are mechanisms on the application layer to reduce the number of copies sent to one host, the gain is not significant as shown in Section 4.6. The mechanisms mentioned ensure that if there are

several receivers on a list located in the same domain, only one copy of the mail is sent to the receiving mail server. The latter will duplicate the mail and deliver it to all local receivers [RFC821, p. 3]. By using IP multicast functionality and deploying a reliable multicast protocol like LGMP (in combination with LGCP), multicast is provided at a lower protocol layer within the network, so that the application is relieved from copying a mail message several times. Compared to the application-level multicast overlay network realized by SMTP, the total amount of data sent is reduced, and often the average mail delivery latency is reduced.

4.1 Multicast Mail Distribution – Overview

For the reasons mentioned, the *Multicast Mail Distribution (MMD)* architecture, depicted in Figure 5, has been developed. Deployment of the architecture is transparent to the users, meaning that mails addressed to the list are still being sent and received by regular mail applications. No modification of any client software is necessary, nor do the users have to install any additional software. The MMD architecture can be deployed transparently by ISPs or company computing centers to offer a value-added service to their customers.

As usual with mailing lists, the mails are – controlled by an appropriate configuration entry in the sendmail alias file – forwarded to a majordomo [Maj99] (or any other list server). Besides forwarding the mail to subscribed receivers not using the MMD service, majordomo hands mails to the *MMD-Stub*, which in turn forwards the mail to an *MMD-Sender* process. The MMD-Sender then distributes the mail using LGMP. In every part of the network where MMD is being deployed, an *MMD-Receiver* is running. These multicast receivers pick the mail sent from the multicast channel and forward it to the receiving mail accounts using sendmail, which in turn stores it until the user accesses it. This way, the receiving users can fetch their mail the way they are used to, e.g. using the Post Office Protocol (POP) [RFC1939] or the Internet Message Access Protocol (IMAP) [RFC2060], or by directly accessing a mail file.

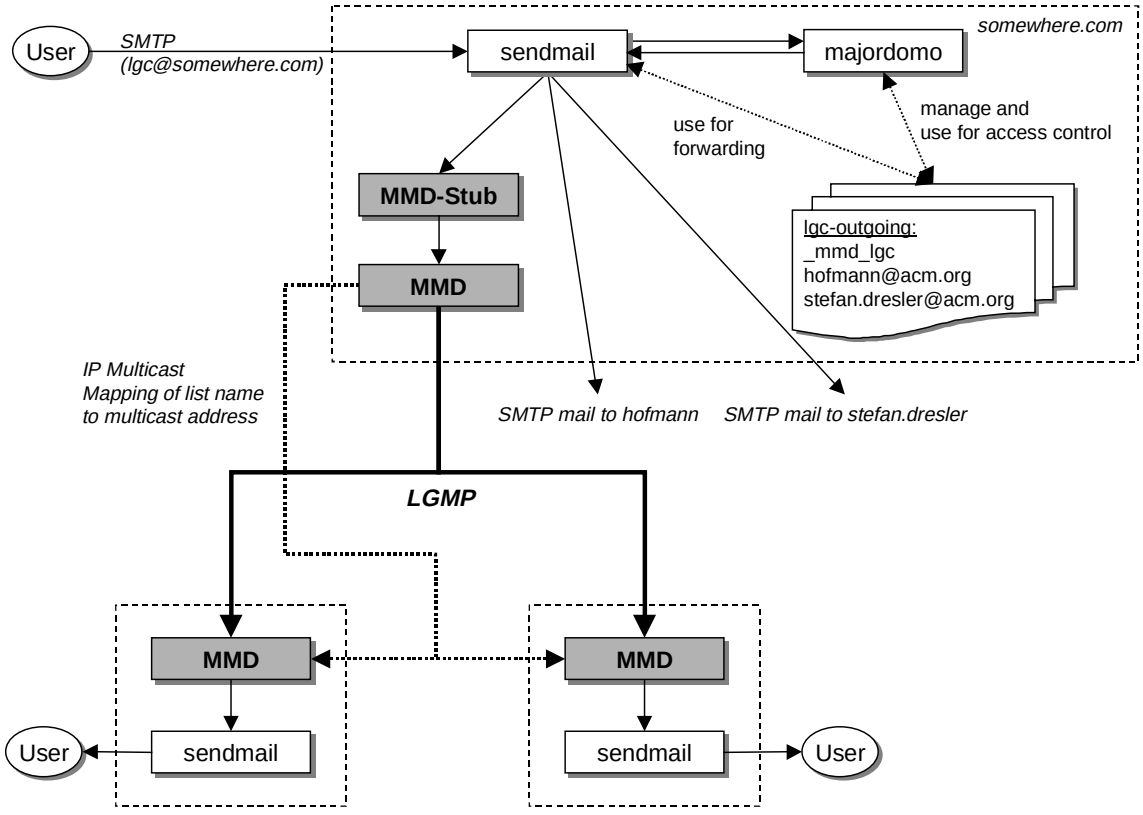


Figure 5: The Multicast Mail Distribution Architecture

It is important to notice that mails will be delivered as usual to receivers not using the MMD service. In Figure 5, for example, users “hofmann@acm.org” and “Stefan.Dresler@acm.org” do not participate in mailcasting. They subscribed to the “lgc” mailing list the regular way creating resultant entries in the majordomo list file. On receiving a mail addressed to the “lgc” list, majordomo will not only handle the mail to the MMD Stub. In addition, he will forward the mail to “hofmann@acm.org” and “Stefan.Dresler@acm.org” using SMTP. This way, the MMD architecture can be deployed stepwise by adding one MMD system after the other. It is not necessary to switch to mailcasting at once.

4.2 Multicast Mail Distribution — Details

As known from the majordomo model, the user sends a mail to be distributed over the mailing list to the mailing list address, in our example `lgc@somewhere.com`. The mail server (in general the `sendmail` program) finds a corresponding entry in its alias file indicating that this mail is to be forwarded ('piped') to the `wrapper` program of the majordomo package. The `wrapper` checks if the sender is allowed to send messages to the mailing list; if yes, it forwards the mail to `sendmail` again, this time with the name of the alias containing all subscribers on the list as address (in our example `lgc-outgoing`). `Sendmail` now forwards the mail to all subscribers.

Up to this point, the regular majordomo scheme has been followed. Additionally, an alias `_mmd_lgc` has been subscribed to the list in our example. This alias points to a program referred to as *MMD-Stub*, for example by an alias entry similar to '`_mmd_lgc : "| /mmd/stub lgc"`', including the list concerned, “lgc”. The MMD-Stub connects to the MMD-Sender using socket communication and hands over the mail. In the simplest case, the MMD-Stub can be realized by the `nc` [Net99] or `socket` [Nic99] program, which simply forward the data to a choosable socket. The separation of accepting a mail from the sendmail and sending it to the group is necessary because distributing a mail may take some time, and a multicast channel (i.e., the corresponding multicast group) can then be re-used for many mails, without the need to coordinate several sending processes.

The MMD-Sender now determines the multicast address corresponding to the mailing list (see Section 4.3) and transmits the mail using LGMP. LGMP was chosen because it scales very well and it provides a reliable multicast service. It is also available on a variety of different platforms. For reasons of resource reuse, multicast channels can be shared between several mailing lists, if they originate from the same host, e.g. a major list server. Therefore, some control information is added to the mail indicating the mailing list it was sent to. Notice that channel reuse is an optional feature. Furthermore, MMD-Sender and Receiver may annotate the mail in the mail header similar to what sendmail does when forwarding a mail.

Each MMD-Receiver checks an incoming mail if there are any local users registered on the corresponding mailing list (see Section 4.4). If there are registered users, the MMD-Receiver forwards the mail to these users via SMTP. If several mailing lists are distributed using the same multicast address, the MMD-Receiver ignores mails received for lists without local subscribers.

What remains to be done is to ensure that all receiving users served by this architecture reliably receive all mails sent by the MMD-Sender. LGMP in combination with LGCP ensures that all active receivers do receive all messages correctly. LGMP will not detect an MMD-Receiver that has not been up and running. Thus, there has to be some protocol at the MMD level that takes care that all MMD-Receivers obtain a copy of the mail. As soon as the mail is transferred from the MMD-Receiver to the local mail server, reliability is ensured by the reliability model of sendmail which stores mails on disk as long as they have not been forwarded to the next hop.

To ensure mail delivery on the multicast part of the architecture even after long-lasting connectivity interrupts or process crashes, it is thus necessary to put an MMD level protocol on top of LGMP. Its

concrete nature depends on whether the MMD-Sender knows its MMD-Receivers or not. These two alternatives are similar to the situation any reliable multicast protocol faces:

- If the MMD-Sender knows its MMD-Receivers, it can learn from missing confirmations that some receiver has not received a certain mail. It can then retransmit the lost mail to the concerned MMD-Receiver. As soon as it knows that all receivers have obtained a mail, it can discard its local copy. The advantage of this procedure is that no unnecessary mail will be kept. The disadvantage is the overhead to manage all MMD-Receivers. Notice that confirmations are for application data units rather than for protocol data units. In our case, confirmations are for entire mails, thus lessening the effect of sender implosion.

- If the MMD-Sender does not know its receivers, it faces the same situation as LGMP. In contrast to LGMP, however, the MMD-Receiver must be able to detect and to recover from complete communication loss or system crashes. This could be realized by checking sequence numbers sent with every mail. Furthermore, the MMD-Receiver must be able to request missing mails over a period in the order of some days from the MMD-Sender. The advantage of this approach is that the MMD-Sender does not have to manage its MMD-Receivers and their confirmations. The disadvantage is that the MMD-Sender has to keep the mails for a sufficiently long time. It cannot find out whether or when all receivers have obtained a mail. Furthermore, if mails are not posted very often on the mailing list, the MMD-Sender must regularly (e.g. several times a day) send the last sequence number. This is important if an MMD-Receiver misses the last mail sent. If the next mail arrives two weeks later, a mail that has been lost during transmission to a receiver has already been deleted at the MMD-Sender. Mail storage is no problem if all mails are kept in an archive anyway.

In either case it is possible that an MMD-Receiver does not receive all mails. In the first case of a sender knowing all its receivers, this will happen if the sender receives no confirmation from a certain MMD-Receiver for a long period (e.g. several days). In this case it deletes the receiver from its list of receivers and sends a message to the list administrator. The time until the MMD-Receiver is being unsubscribed depends, for example, on the amount of disk space an administrator is willing to spent. In the case of a sender not knowing all its receivers, non-repairable mail loss occurs if the mail is deleted too early from the MMD-Sender. Both cases only occur in situations where the receiver is not up for a long period of time (in the order of several days). This extreme situation, however, would also prevent the classical delivery only using SMTP. Consequently, with respect to tolerance of connectivity interrupts or receiver crashes, the MMD service is not worse or even better than the classical mail distribution.

In order to keep the MMD-Sender as simple as possible, the second alternative has been chosen for our prototypical implementation. If an MMD-Receiver detects a gap in the sequence numbers, it requests the missing mail from the MMD-Sender using a unicast TCP connection (Figure 6). Depending on the number of requests, the MMD-Sender will either use LGMP or TCP for retransmissions.

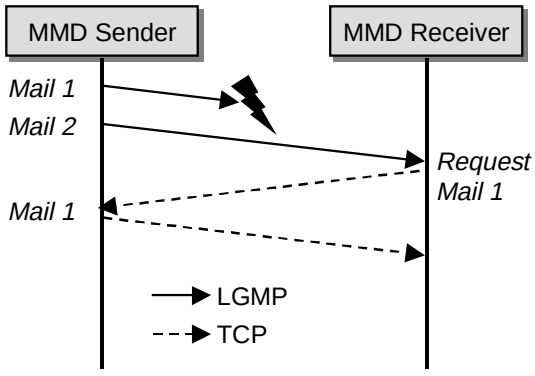


Figure 6: Using TCP to Request Missing Mails

4.3 Mapping List Names to Multicast Addresses

In order to receive mails for a certain mailing list via multicast, MMD-Receivers have to join the correct multicast group. As a prerequisite, they have to map list names to the correct multicast addresses. The mapping information is obtained using a mechanism similar to the one used by the Session Directory (SDR) [Mil98]. All MMD-Senders periodically announce the mailing lists that they are distributing together with the corresponding multicast addresses on a well-known multicast group (Figure 7). Announcements are sent every 15 minutes, so that the imposed network load is relatively low. MMD-Receivers tune in to the announcement channel. Based on the received information, they create a table including available mailing lists and their corresponding multicast addresses.

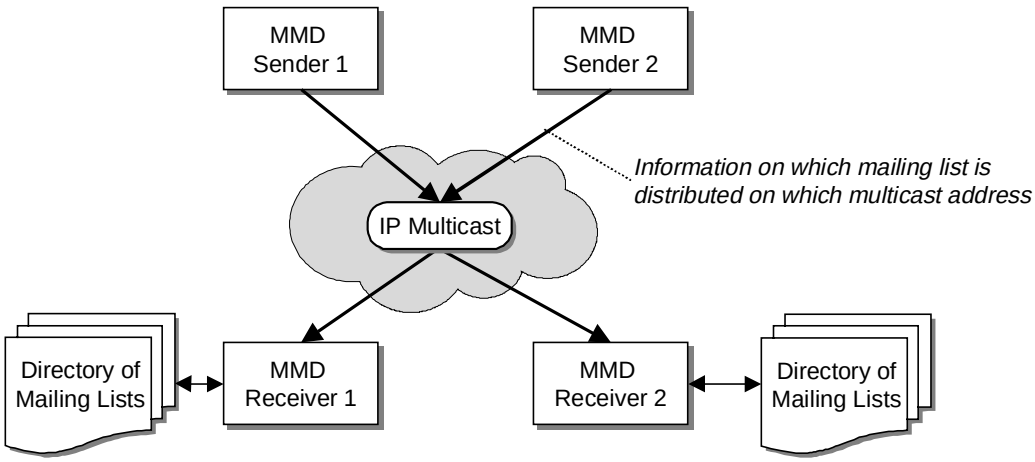


Figure 7: Distribution of Mapping Information

The soft-state-based directory service uses timeouts to remove the entries of deleted mailing lists from the table. If an MMD-Receiver does not receive any announcement for a certain mailing list within a time interval *S*, a notification will be sent to the administrator of the MMD-Receiver. This allows the administrator to check the system for possible problems or to manually verify deletion of the mailing list. If no problems were found or the administrator does not react within another time interval *T*, the list will automatically be removed from the table. Subscribers of the removed mailing list will be notified via e-mail. The probability of removing a valid mailing list by mistake is very low. Assuming an announcement frequency of 4 announcements per hour and a timeout value of $S = 3$ days, the probability for removing an entry by mistake is close to zero even for packet loss rates higher than 50%.

Assigning a multicast address to a mailing list is the responsibility of the server managing the respective mailing list. Servers obtain unused multicast addresses as described in [HTE99]. After assigning the multicast address, list servers start to announce newly created mailing lists multiple times within short time intervals. This allows fast detection of new lists. After some time, announcements are sent with lower frequency.

4.4 Subscribing to Mailing Lists

The classical subscription method known from majordomo of sending a mail to the list server can be adapted to include subscribers to the MMD service. It was decided, however, to present a more comfortable subscription method to the user. Based on received announcements, every MMD-Receiver generates a web page with all available mailing lists. Browsing this page, the user can subscribe to one or more mailing lists with a simple mouse click. If the host on which the MMD-Receiver runs is assigned a mnemonic name such as `mmd` (similar to `www`, `ftp`, `mailhost`, etc.) in the domain name

system (DNS), and if the MMD-Receivers always uses the same HTTP port, it is possible to use a fixed URL (e.g. http://mmd:2512/) to access the local MMD-Receiver. The regular HTTP port (port 80) should not be used because it might conflict with a regular web server on the same host.

On the web page generated by the MMD-Receiver, the user specifies her email address and chooses the mailing lists she is interested in. The MMD-Receiver registers this subscription in its database and determines the corresponding multicast addresses for the mailing lists chosen. If there is no LGMP receiver for any of the multicast addresses yet (as there is no subscribed user for the mailing lists on this multicast channel), the MMD-Receiver starts a new LGMP receiver (in general just a new thread) for the relevant multicast address. From now on, the MMD-Receiver forwards all mails arriving from that mailing list to the newly subscribed user via SMTP.

Based on the presented set of web pages, it is possible to offer more comfortable features for the subscription process. For example, it is possible to implement a keyword search, which returns the names of mailing lists related to a specific keyword. For this purpose, keywords have to be defined for every mailing list at list creation and they are included in periodic list announcements. As can be seen, the new MMD architecture offers possibilities to overcome the old-fashioned subscription process known from majordomo and to provide a more comfortable user interface.

4.5 Support for Closed Mailing Lists

The majordomo model allows restricting transmission of mail messages to a list to users subscribed to this list. Similarly, operations like `who`, which request a list of subscribers, will often only be processed if sent by a subscriber. If a user is subscribed to a mailing list via mailcasting, she is not listed as a majordomo subscriber, but only on some local MMD-Receiver's list, and could therefore be denied the right to send mails to the list or administrative requests to the majordomo system. Furthermore, users subscribed via mailcasting are not included in the result of a `who` request.

To overcome these problems, the MMD-Sender could be placed in front of the `wrapper` call, intercepting any regular mail message, and also in front of administrative requests send to the `majordomo` alias. That way, regular mail messages could be sent even to formerly closed groups. As at the sender side there is no information on the set of actual human subscribers, an effective access control requires enhanced list management mechanisms. For example, an MMD-Sender can obtain the identity of all subscribers to a list by sending a `subscriber request` message to the corresponding multicast address via LGMP. MMD-Receivers will answer the request by transmitting the identity of local subscribers back to the MMD-Sender via TCP. Alternatively, administrative requests like the `who` command could be modified to return a merged list of regular majordomo subscribers and a list of domains served by mailcast.

4.6 Analysis of Existing Mailing Lists

The benefits of deploying the MMD architecture were approximated by analyzing statistical data of existing mailing lists. For this purpose, we gathered and evaluated information on various mailing lists within the IETF and the ATM Forum. The information was used to calculate the average data traffic on these lists and to determine possible cost saving by using multicast mail distribution.

A summary of analyzed mailing lists is given in Table 1. The number of subscribers per list varied between 2 and 5770, whereby some subscribers seem to represent other (local) mailing lists (“list-rsvp@…”). As we pointed out earlier, SMTP ensures that if there are several receivers on a list located in the same domain, only one copy of the mail is sent to the receiving mail server. The latter will duplicate the mail and deliver it to all local receivers [RFC821, p. 3]. Therefore, we also determined the number of subscribers located in different domains, which corresponds to the number of mails sent via

SMTP without using multicast. It is interesting to see that transmitting only one copy per domain decreases the number of mails to be sent by about 19%.

mailing lists	receivers	domains	domains/ receivers
IETF			
int-serv	1364	858	62.9%
rsvp	1413	970	68.6%
rsvp-test	358	264	73.7%
end2end-interest	1134	671	59.2%
end2end-tf	22	20	90.9%
ATM-Forum			
af-xadsl	362	272	75.1%
af-xmpoa	258	219	84.9%
af-xpnni	230	209	90.9%
af-xsecurity	176	164	93.2%
af-xwatm	54	46	85.2%
af-xlane	28	27	96.4%
af-xpress	2	2	100.0%
other lists			
linux-net	1387	1271	91.6%
linux-kernel	3379	2895	85.7%
firewalls	5770	4975	86.2%
total	**15937**	**12863**	**80.7%**

Table 1: Subscriber Data for Sample Mailing Lists

Obviously, the number of outgoing copies can be reduced down to one using multicast. As a result, traffic load on the outgoing link of the mail server will be reduced dramatically. In order to find out the quantitative benefits, we first determined the average daily data volume for each of the mailing lists by analyzing their mail archives. The results are given in Table 2.

list	total bytes	# mails	from	until	# days	mails/day	bytes/mail	bytes/day
rsvp	810504	268	1/7/99	4/20/99	104	2.58	3024	7793
int-serv	1817632	561	10/23/97	12/30/97	69	8.13	3240	26342
end2end-interest	5074535	1469	1/1/98	12/18/98	352	4.17	3454	14416
firewalls	2420881	869	2/1/98	2/28/98	28	31.04	2786	86460
linux-kernel	4284312	1107	4/16/99	4/23/99	8	138.38	3870	535539
debian-user (1)		3907	12/1/98	12/31/98	31	126.03	2200	277271
linux-net (2)		795	4/1/99	4/30/99	30	26.50	3096	82044

(1) bytes/mail detected using spot checks (2) bytes/mail defined as average of the other lists

Table 2: Data Volume

There were no archives available for the lists “debian-user” and “linux-net”. Therefore, we determined the averaged mail size for “debian-user” by spot checks. We examined a subset of all mails and estimated the average mail size. For “linux-net” we defined the average mail size to be the average over the mail sizes of all the other lists.

The values from Table 2 have been used to calculate the average traffic volume on the outgoing link caused by each of the mailing lists. The results are shown in Table 3. The “linux-kernel” list, for example, transmits ~1550 MB per day with SMTP replication at the receiver sites. If the mail server would have to sent a single copy per receiver (i.e. without SMTP replication at the receiver sites), the data volume would increase to ~1810 MB per day. This relatively high data volume is one of the reasons why most mailing lists restrict the maximum size per mail to several KB.

list	# subscribers	# domains	bytes / receiver	total data volume (without SMTP replication) [MB]	total data volume (with SMTP replication) [MB]
rsvp	1413	970	7793	11.0	7.6
int-serv	1364	858	26342	35.9	22.6
end2end-interest	1134	671	14416	16.3	9.7
firewalls	5770	4975	86460	498.9	430.1
linux-kernel	3379	2895	535539	1809.6	1550.4
debian-user	unknown		277271		
linux-net	1387	1271	55228	76.6	70.2

Table 3: Daily Data Volume Handled by the Sender

Using multicast-based mail distribution, network load can be reduced significantly. For example, only a single copy of each mail has to traverse the outgoing server link. In case of the "rsvp" list, daily data traffic on the outgoing link will be reduced from ~7.6 MB per day down to 7793 Bytes per day. Obviously, the load on the outgoing link is independent from the number of receivers in case of multicast mail distribution.

5 Conclusion

We have sketched two applications for reliable multicast protocols. The push-based ticker application can be used to efficiently distribute information that changes over time, such as stock quotes or news headlines. The second application described, the Multicast Mail Distribution architecture, integrates into the existing architecture based on sendmail and list servers and reduces the overall bandwidth consumed by mail distribution to a large number of receivers. The architecture deploys the transport layer multicast service of LGMP in a way transparent to the user, who can continue using the common mail tools. The architecture nicely cooperates with the existing infrastructure for mail distribution. Deployment of the architecture saves bandwidth and processing cost especially at the sender side, which in turn leads to decreased average delivery latency of mails at the receivers. Further experiments using our LGMP-based implementations will be done to evaluate the benefits of information dissemination based on reliable multicast.

6 Bibliography

[Bac99] BackWeb: BackWeb – Enterprise Push. Information at http://www.backweb.com/, 1999.

[Hof96] M. Hofmann: A Generic Concept for Large-Scale Multicast. International Zurich Seminar on Digital Communication, February 21-23, 1996, Zurich, Switzerland, Ed.: B. Plattner: Broadband Communications, Lecture Notes in Computer Science, No. 1044, pp. 95-106, Springer Verlag, 1996.

[Hof98] M. Hofmann: Scalable Multicast Communication in Wide Area Networks (in German). Ph.D. Thesis at University of Karlsruhe, Germany, published by Infix Publishing Company, DISDBIS, Volume 42, ISBN 3-89601-442-0, February 1998.

[LGC99] M. Hofmann: The Local Group Concept. WWW Homepage at http://www.telematik.informatik.uni-karlsruhe.de/~hofmann/lgc/, 1999.

[LiP96] J.C. Lin, S. Paul: RMTP: A Reliable Multicast Transport Protocol; Proc. of IEEE INFOCOM'96, San Francisco, CA., USA, March 1996.

[Maj99] Majordomo: *Majordomo Homepage*. http://www.inf.utfsm.cl/~marcos/majordomo/, 1999.

[Mar99] Marimba: Castanet – How Software gets down the Business. Information at http://www.marimba. com/, 1999.

[Mil98] K. Miller: Multicast – Networking and Applications. Addison Wesley, ISBN 0-201-30979-3, 1998.

[Net99] Netcat: *Netcat Homepage*. Information at http://www.l0pht.com/~weld/netcat/.

[Nic99] J. Nickelsen: *The 'socket' Man Page*. Berlin Germany, 1999.

[Poi99] Pointcast: The PointCast Network. Information at http://www.pointcast.com/, 1999.

[HTE99] M. Handley, D. Thaler, D. Estrin: The Internet Multicast Address Allocation Architecture. Work in progress, draft-ietf-malloc-arch-01.txt, April 1999.

[RFC821] J. Postel: Simple Mail Transfer Protocol. Information at http://www.ietf.org/rfc/rfc0821.txt, August 1982

[RFC1939] M. Crispin: Post Office Protocol - Version 3. Information at http://www.ietf.org/rfc/rfc1939.txt, May 1996

[RFC2060] M. Crispin: Internet Message Access Protocol - Version 4rev1. Information at http://www.ietf.org/rfc/rfc2060.txt, December 1996.

[YGS95] R. Yavatkar, J. Griffioen, M. Sudan: A Reliable Dissemination Protocol for Interactive Collaborative Applications; Proc. of ACM Multimedia '95 Conference, November 1995.