Asynchronous Multicast Push: AMP

Jörg Nonnenmacher Ernst W. Biersack
Institut EURECOM
06904, Sophia Antipolis Cedex, FRANCE
{nonnen,erbi}@eurecom.fr

September 9, 1997

Abstract

We propose a new transport paradigm, called asynchronous multicast push (AMP) that leads to much more efficient use of network bandwidth, to higher throughput, and to reduced server load. AMP decouples request and delivery for frequently requested data and uses multicast delivery. We derive quantitatively by how much the server load and the network load is reduced by introducing an asynchronism between data request and data delivery.

1 Introduction

The World Wide Web currently uses the **pull paradigm**, where clients request data from a server and then wait for the immediate response from the server sending the requested data. The tremendous popularity of the Internet, and the Web in particular, often causes network and server congestion resulting in high response times that discourage an interactive usage. To avoid that the Internet becomes the victim of its own success, we propose for the retrieval of popular data a new paradigm called **asynchronous multicast push** that replaces the current synchronous *browse and pull* by *browse*, *subscribe* and *asynchronous push*. Browsing itself will continue to be a synchronous activity that requires the immediate transfer of small amounts of data. Whenever a popular document comprising a larger amount of data is requested, this request however will be satisfied asynchronously.

The advantages of our Push approach are multiple: Instead of transmitting (pulling) popular data multiple times, the sender–initiated push ensures that data will be delivered to multiple clients only *once*, namely when the data is available, or when the data has changed. AMP results in a more efficient use of the network bandwidth and assures the delivery of data right when it is available. Browsing and searching for information will still be possible, even with much better response times than before.

AMP exploits the fact that pushing data can be done using efficient multicast delivery from the sender to several receivers, as opposed to unicast pulling by distinct receivers via unicast.

The gain of Multicast delivery depends on the number of receivers that are delivered at the same time. Dependent on the number of accesses AMP switches between different delivery modes:

- **Unicast Pull:** The request is immediately satisfied. This delivery method is used for unpopular data.

- **Asynchronous Multicast Push:** Short requests for the same data are accumulated over an interval in time and and data is transfered by multicast, satisfying all accumulated requests simultaneously with low resources and network cost. This delivery method is used for popular data.

- **Continuous Multicast Push:** Continuous cyclic transmission on the same multicast channel. This kind of delivery is used for very popular data with high access rates and also for data that changes frequently. Interested clients with knowledge of the multicast channel that carries the information will not contact the server anymore, but immediately join the channel. This results in a low re-

sponse time and in a server load that is independent of the number of accesses.

Using Multicast Push changes the requirements of the multicast transport protocol that ensures the reliable delivery to every receiver. The multicast transport protocol underlying AMP was designed for high requirements of scalability to very large numbers of receivers, TCP-like congestion control and high efficiency.

The rest of the paper is organized as follows. In section 2 the gain of Multicast Push is quantified. Section 3 describes the AMP server and AMP client. Section 4 discusses the multicast transport protocol that is used by AMP. Section 5 concludes the paper.

2 The Advantages of Asynchronous Multicast Push

Asynchronous Multicast Push leads to several gains due to (i) multicast delivery, (ii) the asynchronism between request and delivery, (iii) and sender–initiated push.

1. **Multicast**: Multicast delivery requires less network bandwidth, compared to unicast. Also the server load for sending is reduced, since less packets are emitted by the server.

2. **Asynchronism**: Increases the gain of multicast delivery further, since requests of receivers are accumulated over time and delivered once to *more* receivers at a later point in time. Asynchronism further allows for efficient transmission scheduling by the sender.

3. **Push**: Pushing data from the sender to the receivers allows to deliver the data, just when the information is available, as e.g. for a daily newspaper, a new software update, or a changed Web page. Receivers will immediately get the latest information. Push further allows to distribute the sender load, since the time of the transmission is determined by the sender and not by the receivers.

We will first quantify the network bandwidth gain achieved, when data is delivered by multicast, instead of unicast from one sender to R receivers. The gain is measured in terms of packets on the network, for multicast and unicast.

- **Multicast:** Data is emitted once towards R receivers into the network, follows the multicast distribution tree, and is copied in the network whenever the route fork off.

- **Unicast:** Data is emitted R times at the sender, once for every receiver.

Besides the fact that unicast to R receivers at the *same* time may be impossible due to sender overload (R simultaneous connections), unicast also uses much more network bandwidth than does multicast. This fact is generally well known and presented as the advantage of multicast distribution.

However, there is a second advantage due to multicast that is rarely realized and which is due to extending multicast into the $time$–dimension: In AMP, a receiver registers for a data transmission that takes place at a later point in time. This asynchronism between request and push allows to collect multiple requests for the same data that are later multicast to a larger number of receivers. Receivers interested in the same data are therefore grouped in the $time$–dimension.

The asynchronism introduced by AMP further provides for more flexibility in scheduling the transmissions. Based on measurements of previous transfers and the current server load transfers are scheduled in order to optimally distribute the server load.

Section 2.1 quantifies the bandwidth gain achieved by grouping *spatially* distributed receivers over the network and delivering by multicast. Section 2.2 quantifies the bandwidth gain by grouping receivers also in the *time* domain, introducing asynchronism.

2.1 Gain of Multicast Transmission

Sending one packet from the sender to R geographically dispersed receivers requires for unicast delivery that a packet is sent $|R|$ times via paths from the sender to the receivers. Multicast delivery happens via a tree rooted at the sender. The packet sent from the sender to all receivers traverses only one time every link of the tree - assumed loss free conditions. See figure 1 for unicast and multicast delivery to four receivers on the same network.

In order to quantify the gain of network bandwidth 10 random networks with $N = 200$ nodes and an average nodal outdegree of 3.0 were constructed following Waxman [Wax88], with the modification of Doar in [DL93] that avoids the influence of the number of

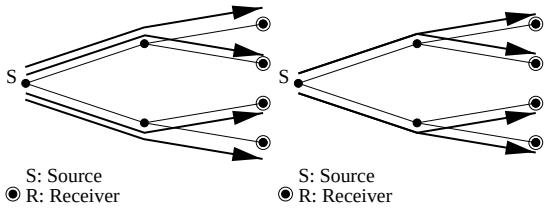

Figure 1: Unicast delivery (left) and Multicast delivery (right) via a network with $N = 7$ nodes from one sender to $R = 4$ receivers.

nodes on the average nodal outdegree. The method of Waxman is commonly used by the Multicast Routing community [WE94, DL93, Wax88, Kad94] to compare the performance of different Multicast Routing Algorithms on random networks.

On each of the 10 random nets, 100 multicast groups with varying group sizes ($R = 5, \ldots, 140$) and random receiver locations had been routed by the Shortest Path Tree Algorithm, analyzed by Doar [DL93]. The **shortest path tree** (SPT) connects every receiver to the sender via the shortest path.

Let c be the cost of the delivery from one packet via one link (node) of the delivery tree, then is the cost for the delivery of one packet from the sender to the receivers the product of c and the number of links the packet traverses.

Let the **cost of unicast delivery** be C_{UC} and the **cost of multicast delivery** be C_{MC}. We compare both and show the percentual cost reduction by multicast delivery via the network. Figure 2 shows that the network cost for unicast delivery increases linearly with the number of receivers, while the cost for multicast delivery increases much slower with the number of receivers.

The gain in terms of bandwidth can therefore be defined as the percentual cost reduction of using a multicast tree compared to R single unicast connections, (one to each receiver):

$$gain = \frac{C_{UC} - C_{MC}}{C_{UC}}$$

Figure 3 shows an increasing gain as the number of receivers increases – only 5% receivers at all network nodes are sufficient to reduce the bandwidth by 40% due to multicast delivery.

In order to get results also for larger networks with more than $N = 200$ nodes we approximate the gain given as a function of the ratio $x = \frac{R}{N}$.

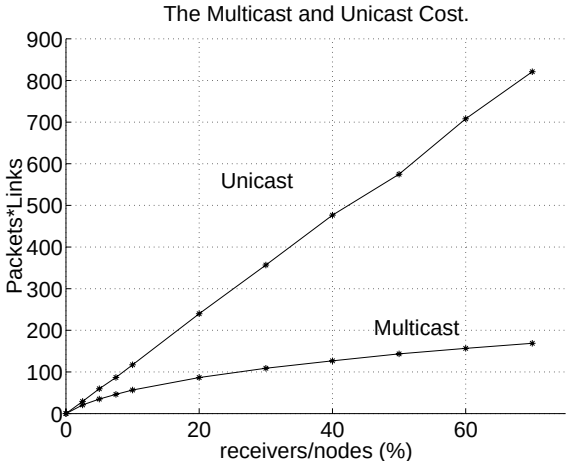

Figure 2: A cost comparison between unicast and multicast transmission in terms of packets on the network.

The approximation $g(x)$ for the multicast gain dependent on the ratio x of receivers to network nodes is:

$$g(x) = 19.3 \cdot \log(x) - 0.2x + 10$$

For a network with a fixed number N of nodes shows the approximation that the bandwidth gain due to multicast is mostly determined by the logarithm of the number R of receivers and a small linear component. The measured gain and the approximation are shown in figure 3: The approximation is tight for our networks with average outdegree 3.0. In the following we use the approximation $g(x)$. This allows to evaluate the multicast gain also for networks with more than 140 nodes, since the parameter x expresses the multicast group size R relative to the size of the network in nodes N.

2.2 Gain due to Asynchronism

Receivers that request data at different times can be accumulated in a time interval and can all be delivered at the same time. The bandwidth gain for the network by grouping receivers in the $time$-domain will be assessed by modeling requests to a highly requested data over the day time: The number of requests at different day times is modeled by a modified Gauss–function with the granularity of accesses per day minute. Arguments to the function is the minute m of the day:

$$A(m) = (A_{max} - A_{min}) \cdot e^{-\left(\frac{m - m_{mid}}{s}\right)^6} + A_{min}$$

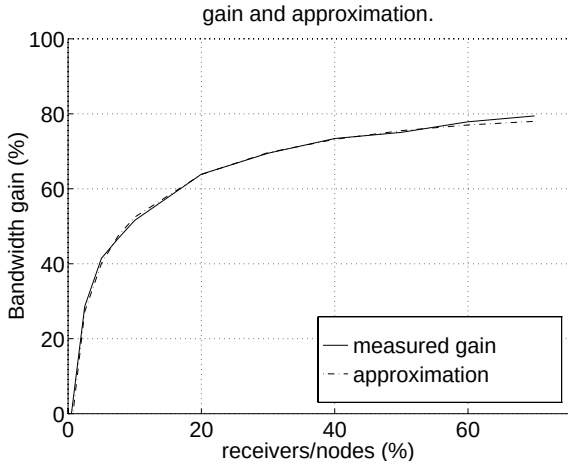

Figure 3: The bandwidth gain of multicast and the approximation $g(x)$.

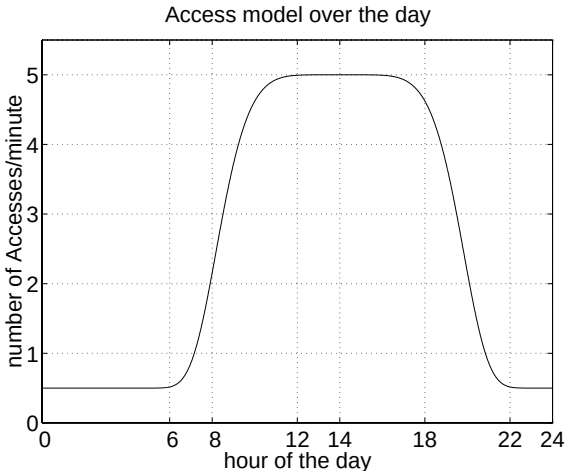

Figure 4: The server accesses per minute for $A_{max} = 5\ \frac{accesses}{min}$.

The choice of the other fixed parameters was done with the following motivation. The accesses should be distributed over the day time and model a working day. Measurements [Bar96] of accesses to servers had a shape as modeled with a mean around day time 2 pm, thus the $m_{mid} = 840$th minute of the day. The standard deviation of the accesses over the day is chosen as $s = 6h = 300$ min. The maximum number of accesses per minute is A_{max}. In order to model also a certain access rate the night the minimum number of accesses was chosen as $A_{min} = A_{max}/10$. The hull curve of the accesses per minute is shown in figure 4.

Access/Request accumulation is done over an interval $I = [a, b]$, where a and b denote minutes of the day. To allow for accumulation over midnight, a can be larger than b. The maximum accumulation time will be $24\,h$.

Let l denote the size of the accumulation interval, then:

$$l(I) = \begin{cases} b - a & , b \geq a \\ (b + 24 * 60) - a & , b < a \end{cases}$$

Different accumulation interval sizes will be investigated. Dependent on day time and interval length $l(I)$, the accumulation yields a different number of receivers that can be served at the end of the accumulation with one multicast transmission. The number $Cum(I, A)$ of accesses in the interval $I = [a, b]$, where A is the access function $A(m)$ of the day, is given by:

$$Cum(I, A) = \begin{cases} \sum_{m=a}^{b} A(m) & , b \geq a \\ \sum_{m=a}^{23\cdot 60+59} A(m) + \sum_{m=0}^{b} A(m) & , b < a \end{cases}$$

We are interested in the number of accesses accumulated at the end of the accumulation interval $I = [a, b]$ dependent on the interval length $l(I)$ and the minute b of the day the accumulation is finished and the transmission takes place.

We evaluate the bandwidth gain for multicast transmission after accumulation over an interval I for a network of $N = 5000$ nodes, using the approximation $g(x)$:

$$gain = g\left(\frac{Cum(I, A)}{N}\right) \quad (1)$$

This gain is shown is shown in figure 5 for low accessed data with a maximal access rate of $A_{max} = 0.05\ \frac{accesses}{minute} = 3\ \frac{accesses}{hour}$ and highly accessed data with a maximal access rate of $A_{max} = 5\ \frac{accesses}{minute}$ in figure 6.

It can be seen that there is nearly no gain by accumulating requests for data with low access. On the other hand, it can be seen that there is always a gain by accumulating requests for highly requested data.

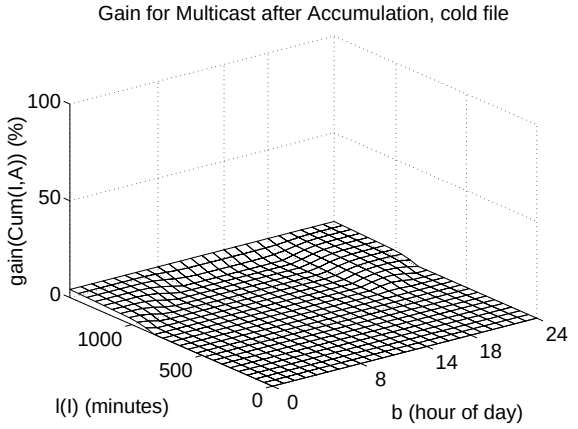

Figure 5: The bandwidth gain of a multicast transmission at the end b of the accumulation and the accumulation interval length $l(I)$ for rarely requested data ($A_{max} = \frac{1}{20}\frac{accesses}{minute}$).

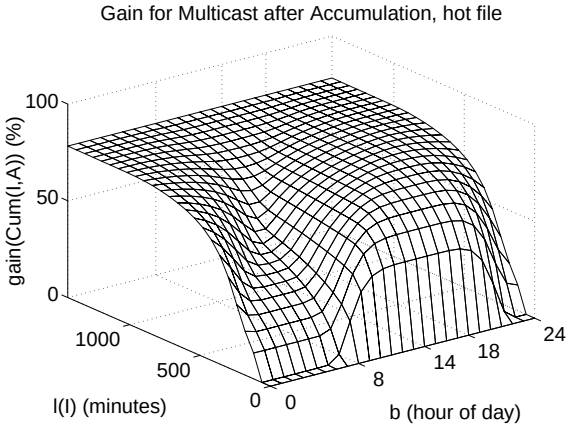

Figure 6: The bandwidth gain of a multicast transmission at the end b of the accumulation and the accumulation interval length $l(I)$ for highly requested data ($A_{max} = 5\frac{accesses}{minute}$).

3 AMP framework

In order to asses the gains shown in the previous sections AMP was implemented on top of IP multicast with respect to the following design goals:

- **Reliability:** AMP transfers *all* data reliably to *all* receivers.
- **Scalability:** AMP was designed to scale to very large numbers of receivers (up to 10^6). AMP scales to the popularity of the content – it adapts to changes in the access rate by switching the transfer mode.
- **Efficiency:** A major goal was to avoid unnecessary receptions at receivers. This happens, when a receiver lost a packet that is received by other receivers. The multicast retransmission is unnecessary for all receivers that already received the packet.
- **End – to – End:** All mechanisms of AMP work on a pure end–to–end basis between sender and receivers without any support from the network, except for data delivery. This allows to employ AMP over nearly any kind of network, including satellites.
- **Heterogeneity:** AMP allows to deliver data *only once* to heterogeneous receivers, i.e. receivers with different maximal reception rate. AMP is further completely implemented in Java and therefore runs on heterogeneous platforms and operating systems.

First a brief overview over AMP client and AMP server is given. In section 4 is then pointed out how the design goals are achieved also for the reliable multicast protocol by parity transmission. Parity packets are packets derived from the original data packets using results from coding theory. For instance a single parity packet can be obtained by a bitwise XOR operation over all data packets.

AMP consists of three major parts, a Client module, a Server module and a Scheduling module, see figure 7.

AMP works as follows:

1. A client requests data at the corresponding AMP Server via a TCP connection.
2. The server module communicates with the scheduling module to determine the time of the next transmission of the data. The scheduling module schedules transmissions dependent on an access statistic.
3. The server returns the multicast address, the data volume, the relative time of the transmission and an internal registration number to the client. The

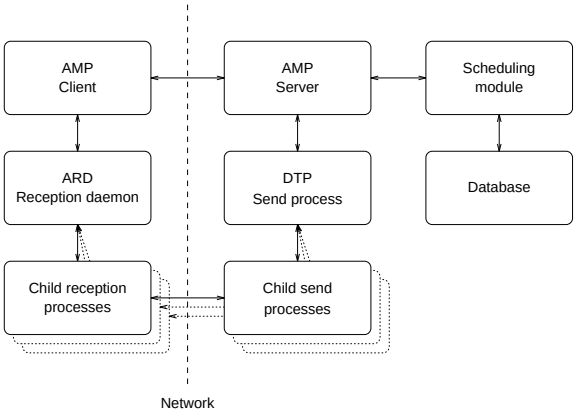

Figure 7: AMP modules.

registration number is used for identification purposes and to keep the ability for cancelation of a scheduled data transmission to the client.

4. The client reserves the space needed for the data. It then registers for the data with the reception time in his internal reception table.

5. At the announced time the client ARD reception daemon forks thread for the reception on the given multicast channel.

6. After the data transfer from the DTP send process the server updates the transfer statistic to further optimize the scheduling of subsequent transmissions.

3.1 AMP Client

The client is composed of two programs, the *AMP client* which contains the user interface and the *ARD reception daemon* which receives the data at the scheduled time. A modular approach has been chosen to allow easy future evolution towards security and different transport protocols. The client offers two functionalities, order data and cancel a previously registered order. The cancel procedure sends the registration number to the reception daemon, which checks if such an order exists in the list and if this is the case, sends the IP address back to the client. On each order, the client reserves the needed space announced by the server. In the case the memory is not available a NACK is sent to the server.

The reception daemon manages a list which contains the data for all requested data. When a user wants to order data, the reception daemon is called by the AMP client. The daemon opens the announced port and gives the port number back to the AMP client. The port stays open until the end of the data reception to assure that the port will be available at the time the transfer takes place.

After the completion of a request, the client hands all needed data over to the daemon which creates a new entry in the list. If the new entry is at the top of the list, the timer (alarm) is set to the reception time of the first data.

At the scheduled time, a reception process is forked which receives the data on the previously reserved port. The corresponding list entry is moved to another list to assure that the alarm is immediately set to the reception time of the next data.

3.2 AMP Server

The AMP server has to handle the following major tasks:

- Treat requests.
- Make transfers.

Both tasks are split to different processes in order to cope with high request rates. The *AMP Server* treats requests, responds to commands issued by the clients and communicates with the scheduling module.

The *DTP Send Process*' sole task is transferring requested data on the server to AMP clients. After each transmission, this process updates the database by giving some statistic informations like the duration of the transmission.

A TCP/IP connection is established between a client and the AMP server for the request. The connection for the final transfer between the DTP and the client ARD uses the reliable multicast transport protocol described in section 4 over UDP/IP.

In order to keep consistency between the transmission database and the DTP, the AMP server informs the Data Transfer Process by signaling about the insertion of a new command and the destruction of an existing entry in the database.

In sum, the main task of the AMP server implementation is:

- transfer requested data to clients
- update the database when a transmission is over with statistic information

- receive sent signals from AMP server and handle an associated action

In order to measure the performance of our server, we made several data transfers between E.P.F.L in Switzerland and Eurecom. For the same data, during the night, the transmission can take only 2 to 3 seconds to complete, but during the day, sometimes 300 seconds are needed (5 minutes).

This example shows that our implementation has satisfied one of the main objectives. The AMP data transfers can be scheduled during a period where the throughput is highest.

Dependent on the access rate to data AMP switches between the three different delivery modes **Unicast Pull**, **Asynchronous Multicast Push** and **Continuous Multicast Push**. The switch at the AMP server to a different delivery strategy is controlled via thresholds for the access rate to the data.

When the current delivery mode for data is **Unicast Pull** or **Asynchronous Multicast Push** is the access rate known, since the access is controlled via a request preceding the transmission. For the case of **Continuous Multicast Push** the data is multicast continuously over and over again. The current number of receivers of the data is not accessible by counting the requests for the data: receivers can tune directly into the multicast channel, without reaching the AMP server. This results in a server load independent of the potentially very high number of receivers.

The access rate to the data in the **Continuous Multicast Push** mode is estimated by polling the multicast channel periodically for feedback via a probabilistic feedback method. The feedback allows to give an estimate on the number of receivers tuned into the multicast delivery channel at this time and allows therefore to assess the access rate.

The major challenge for feedback in multicast communication is the *feedback implosion* problem that occurs, if several receivers attempt to send feedback at the same time to the sender. The problem gets very stringent for a very high number of receivers. The high amount of feedback leads to a high traffic concentration at the sender, wasted bandwidth and high processing requirements at the sender. The amount of potential feedback increases linearly with the number of receivers and imposes high requirements to the mechanism for *feedback implosion* avoidance. Several solutions for implosion avoidance exist based on hierarchies [YGS95, PSLB97, Hof96, BP97, DO97], sampling [BTW94], tokens [JN93] and timers [SDW92, FJL+96, SEFJ97, Gro97].

AMP uses a probabilistic feedback method based on timers that is shown to avoid feedback implosion for up to 10^6 receivers and is shown to be robust against loss and different delays between sender and receivers. In contrast to other feedback mechanisms does the one of AMP not need topological information, nor delay estimates between multicast group members.

The feedback mechanism in AMP is described in detail in [NB98] and works as follows: On the reception of a poll message emitted by the AMP server every receiver chooses a random timer t in the interval $[0, T]$ via a truncated exponential distribution with parameter λ. The parameter λ and the interval size T are given by the sender in the poll message. A receiver multicasts its feedback, if and only if no feedback is received from another receiver before.

A receiver sending feedback includes its timer $t \in [0, T]$ in the feedback message. Due to the feedback messages received and the knowledge of the timer settings the AMP server can give an estimate of the number of receivers: $\hat{R}$.

Due to this estimate the interval size T and the parameter λ are chosen carefully in order to avoid *feedback implosion* and to minimize the delay for feedback for the next feedback round.

4 Reliable Multicast Transport

After discussing the signaling between AMP client and AMP server now the reliable multicast transport protocol used for the delivery is presented. AMP is based on a protocol using parity transmissions and TCP-like congestion control over several multicast layers.

4.1 Multicast Loss Repair by Parities

Current multicast transport protocols suffer from the fact that for every lost packet one retransmission of the same packet is required. We show how parity transmission can greatly improve the throughput and the network bandwidth used for reliable multicast transmission.

Loss for a multicast of k packets from the sender to R receivers can be described as a loss matrix $Z \in \{0, 1\}^{R,k}$, where

$$Z_{i,j} = 1$$

describes the loss of packet j at receiver i, where $i = 1, \ldots, R$ and $j = 1, \ldots, k$. An example loss matrix for $R = 5$ receivers and $k = 4$ packets is given below:

$$Z = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

$$Z' = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$Z'' = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

In this example receiver 3 sees a burst loss of length 2, from packet 2 to packet 3. A shared loss happened for packet 2, which is lost at receivers 3 and 5. Retransmitting one parity packet to the receivers allows for the repair of any lost one packet at all receivers in the case the parity packet is not lost. The resulting loss matrix is Z'. Losses at all receivers except for receiver 3 are repaired. Receiver 3 needs another, different parity packet in order to repair its two losses by decoding (Z'') with the received 2 original packets and the received 2 parity packets.

This simple example demonstrates the repair efficiency of retransmitted parity packets:

- A single parity packet can be used to repair the loss of *any* one of the k data packets. This means that a *single parity packet* can repair the loss of *different data packets at different receivers.*

More than one parity can be coded the following way: A Reed Solomon Erasure correcting code (RSE code), such as the one described by McAuley [McA90], is used to generate the redundant data. Suppose we have a set $\{d_1, d_2, ..., d_k\}$ of k data packets each of which is P bits long. The RSE encoder takes $\{d_1, d_2, ..., d_k\}$ and produces a set $\{p_1, p_2, ..., p_{n-k}\}$ of packets each P-bit long that are called **parities**. We also use the parameter h to denote the number $n - k$ of parities. For the purpose of coding, we consider the data packets $\{d_1, d_2, ..., d_k\}$ as elements of the Galois field $GF(2^P)$ [LC83] and define the polynomial $F(X)$ as

$$F(X) = d_1 + d_2 X^1 + ... + d_k X^{k-1} \qquad (2)$$

If α is the primitive element of $GF(2^P)$, the RSE encoder computes p_j as $p_j = F(\alpha^{j-1})$ for $j \in \{1, .., n-k\}$.

In the case of loss, the RSE **decoder** at the receiver side can reconstruct the data packets $\{d_1, d_2, ..., d_k\}$, whenever **any** k out of the n packets $\{d_1, d_2, ..., d_k, p_1, p_2, ..., p_{n-k}\}$ are received. The k data packets will also be referred to as **transmission group** or **TG**. The n packets $\{d_1, d_2, ..., d_k, p_1, p_2, ..., p_{n-k}\}$ will be referred to as an **FEC block**. Sending the original data as the first k packets of the FEC block simplifies decoding:

- If all the k data packets are received, no decoding at all is necessary at the receiver.
- If $l < n - k$ out of the k data packets are lost, the decoding overhead is proportional to l.

There are multiple benefits of using the parity packets for loss recovery instead of retransmitting the lost packets:

- Improved transmission efficiency: A single parity packet can be used to repair the loss of *any* one of the n data packets. This means that a *single parity packet* can repair the loss of *different data packets at different receivers*.
- Improved scalability in terms of group size: When a lost packet is retransmitted via multicast, the packet will be received more than once by all the receivers that have already successfully received the packet. Such duplicate packets waste transmission bandwidth and processing capacity. The number of duplicate packets received is reduced nearly down to zero with parity transmission.

A software version of another RSE coder was recently put into public domain by L. Rizzo [Riz97] that codes at a rate of multiple Megabits/s. Given such high

performance and the fact that the bandwidth requirements of many current multicast applications are typically less than 100 KBytes, suggests that coding will not affect the packet sending rate and, hence, loss recovery using parity data is feasible.

4.2 Congestion Control

The objective of congestion control is to avoid a collapse in the network due to uncontrolled access and transmission rates. Traffic that crosses a network is generated at arbitrary points in time.

For the case of the Internet everyone is able to transmit data through the Internet. The available bandwidth in a network node is ideally equally shared among all connections crossing this network node. Data gets lost in the case, where the amount of traffic through a node gets larger than its capacity. In the case of a reliable transmission retransmissions are needed in order to complete reception of all data. If every sender continues sending retransmissions with the same rate that caused the congestion a collapse occurs.

Congestion control is the mean to avoid this collapse by adjusting the amount of traffic emitted through this overloaded node, henceforth referred to as bottleneck. The congestion control algorithm of TCP handles congestion for *unicast* traffic over Internet bottlenecks.

A straightforward solution of porting congestion control from unicast to multicast is feedback from all receivers to the sender. The sender then adjusts equivalent to TCP its sending rate to the receiver with the lowest possible reception rate.

Clearly the disadvantage is that the delivery speed to all receivers is determined by the weakest – the one with the smallest capacity on the path from the sender to itself.

Our solution allows to deliver receivers with different maximal reception rates with the same data at the same time. Before pointing out our congestion control mechanism we state general requirements for multicast congestion control.

Receiver-initiated. Receiver-initiated mechanisms result in much better scalability than do sender-initiated ones.

Fairness between Unicast and Multicast traffic. When unicast and multicast share the same bandwidth congestion control should allocate the bandwidth fair between unicast and multicast traffic. Multicast leads, from a global network point of view, to a more efficient usage of network bandwidth, since packets are copied in the network. Therefore multicast traffic should be prioritized over unicast traffic.

Fairness among Multicast traffic. Three major types of multicast traffic can be distinguished: video, audio and reliable data transfer. In the case where all three share the same bandwidth the same congestion control algorithm should be used for all of them.

Fast Congestion Control. A small control delay allows for better control. Congestion control mechanisms are more effective if the reaction on congestion is fast. We define the *control delay* d_c as the time between the moment congestion occurs and the time the congestion control gets active (the time the decreased rate arrives at the bottleneck)

Let us assume the bottleneck B located halfway between the sender S and a receiver R in the delay metric. The *control delay* of TCP congestion control corresponds to one round trip time (RTT): The output buffer in the bottleneck is full and the bottleneck router B drops a packet (congestion indication). The receiver sees the loss after $d(B, R) = 1/4\, RTT$ (indication B $\rightarrow$ R). The receiver R reports congestion to the sender S, wasting $d(R, S) = 1/2\, RTT$. The sender S reacts by decreasing the sending rate and the new rate propagates within $d(S, B) = 1/4\, RTT$ to the bottleneck B. It can be seen that the control delay corresponds to:

$$d_c = 1/4\, RTT + 1/2\, RTT + 1/4\, RTT = 1\, RTT$$

4.2.1 Receiver-driven Layered Multicast (RLM)

Is a technique, where each individual receiver controls the rate of delivery to itself. The whole data is transfered via g different multicast groups/layers. Control over the delivery rate is done via subscribing and dropping of layers. While demonstrated for video [MJV96, MVJ97] and audio [TPB97] a first step towards congestion control via layered transmission for reliable multicast data transmission can be found in [RV97]. This is a first step towards a unique congestion control mechanism for all multicast traffic.

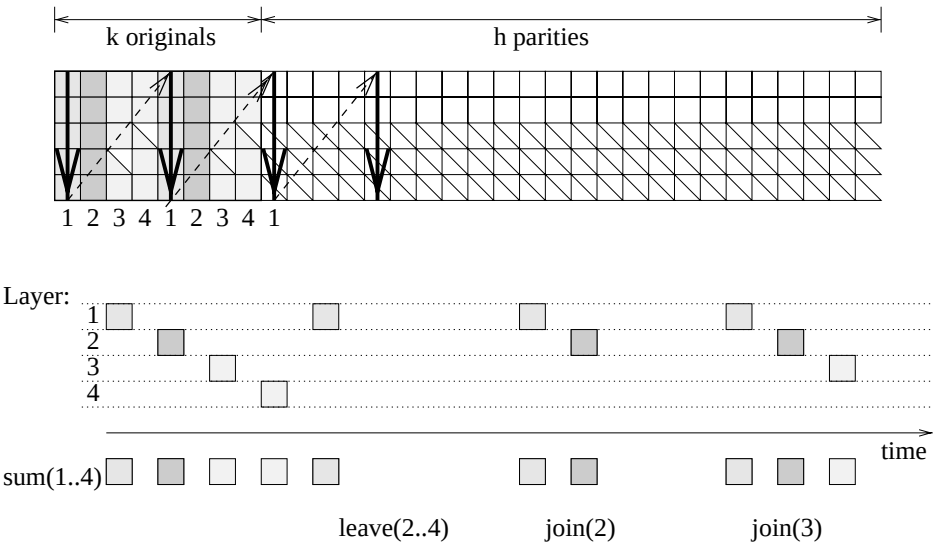

Figure 8: Layered multicast transmission and congestion control for four layers.

4.2.2 Multicast Congestion Control using Parities

Parity transmissions are shown to lead to highly efficient reliable multicast [NBT97]. Another, yet unrecognized, advantage of parities consists in the context of multicast congestion control.

The following describes the foundation of multicast congestion control using parities.

Let k be the number of original data packets and h be the number of parity packets coded from the original k packets. Let h be much larger than k: $h >> k$. Then the property that **any** k out of the $k + h$ data packets received are sufficient to decode the original k can be used to serve slow and fast receivers at the same time using RLM:

Consider the $k + h$ packets to be numbered from $1, \ldots, n$ and two multicast layers. One slow receiver is subscribed just to layer 1, another, fast receiver is subscribed to both layers: 1 and 2.

Consider the sender transmits all odd numbered packets on layer 1 and all even numbered packets on layer 2. Then the fast receiver will be done after k packets under loss free conditions, the slow receiver will also be done after k packets, but has two wait two times longer.

All packets delivered on layer 1 are useful for *both* receivers!

This simple example demonstrates:

- *Parities allow to multicast the same data to heterogeneous receivers without doing unnecessary transmissions.*

The simple example extended to several layers allows receiver-driven congestion control by adding/dropping layers. Figure 8 shows $g = 4$ layers and a data block of 5 rows each consisting of $k = 8$ originals. In the upper part of figure 8 just the transmission order of packets on layer 1 is shown. The lower part of figure 8 shows the transmission on the four layers and a leave and two join actions.

Whenever a receiver sees a loss it decreases its rate by dropping layers and reduces thereby its reception rate. The control delay d_c for this kind of congestion control is determined by the speed a router is able to process a *leave message* for a multicast group. Neglecting the processing cost in routers results in a control delay d_c that corresponds to a bit more than half a round trip time: The receiver sees the loss after $d(B, R) = 1/4\,RTT$ (indication B $\rightarrow$ R). The receiver R leaves one or several multicast groups by sending a *leave message* towards the source. The traffic through the bottleneck is decreased, when the leave message is processed at the first router after the bottleneck on its way towards the source ($d(R, B) + \delta = 1/4\,RTT + \delta$). Then the control is active and the traffic through the bottleneck decreased. The control delay is therefore lower than in the case of TCP:

$$d_c = 1/4\,RTT + 1/4\,RTT + \delta < 1\,RTT$$

4.2.3 Requirements for Layered Congestion Control

In order to allow this kind of layered congestion control for reliable multicast the following requirements exist:

- A fast processing and propagation of join/leave for multicast groups is needed.
- Layers have to take the same routes. Layered congestion control is based on the assumption that the multicast trees for the different layers, all belonging to the same delivery, take the same paths. This can be supported by introducing hierarchical multicast addresses, where the last x bits are used to address the number of layers used. An advantage that would come for free is: leaving and joining layers could be done by a compound action: *Join(15)* means join all layers up to layer 15. *Leave(3)* means leave all layers down to layer 3 and stay in layers $1, 2, 3$. By a compound join/leave and hierarchical multicast addresses the state in routers is reduced due to the aggregation of multicast addresses.

5 Conclusion

We presented a new communication paradigm - asynchronous multicast push and showed that the asynchronism between request and delivery combined with multicast delivery reduces the network bandwidth required for highly popular data by up to 80%.

We presented and implemented AMP (see http://www.eurecom.fr/AMP), which allows to achieve this gain.

We showed that further reduction of the required network bandwidth for the transport is achieved by our multicast transport protocol based on parity transmission for loss recovery.

Scalability to the number of receivers is achieved by a pure end-to-end feedback mechanism and receiver-initiated congestion control of parity transmissions over several multicast layers.

AMP also scales to the popularity of data by switching the delivery mode.

References

[Bar96] D. Barnes. An analysis of world-wide web proxy cache performance and its application to the modelling and simulation of network traffic. Technical Report 10-96, University of Kent, Computing Laboratory, Kent, UK, 1996.

[BP97] J.M. Bonnin and J.J. Pansiot. A scalable collect. In *HIPPARCH Workshop*, Sweden, June 1997.

[BTW94] J.C. Bolot, T. Turletti, and I. Wakeman. Scalable Feedback Control for Multicast Video Distribution in the Internet. In *Proceedings of SigComm*. ACM, September 1994.

[DL93] M. Doar and I. Leslie. How bad is naïve multicast routing. In *Proceedings of INFOCOM'93*, volume 1, pages 82–89. IEEE, 1993.

[DO97] Dante DeLucia and Katia Obraczka. Multicast feedback suppression using representatives. In *IEEE Infocom97*, 1997.

[FJL+96] S. Floyd, V. Jacobson, C. Liu, S. McCanne, and L. Zhang. A reliable multicast framework for light-weight sessions and application level framing. *Submitted to IEEE/ACM Transactions on Networking*, 1996.

[Gro97] Matthias Grossglauser. Optimal deterministic timeouts for reliable scalable multicast. *IEEE Journal on Selected Area in Communications*, 15(3):422–433, April 1997.

[Hof96] Markus Hofmann. A generic concept for large scale multicast. In *International Zürich Seminar*, Zürich, Switzerland, February 1996.

[JN93] J.M.Chang and N.F.Maxemchuk. A broadcast protocol for broadcast networks. In *Proceedings of GLOBECOM*, Dec. 1993.

[Kad94] James Kadirire. Minimising packet copies in multicast routing by exploiting geographic spread. *ACM Computer Communication Review*, 24(3):47–62, July 1994.

[LC83] S. Lin and D. J. Costello. *Error Correcting Coding: Fundamentals and Applications*. Prentice Hall, Englewood Cliffs, NJ, 1983.

[McA90] A. J. McAuley. Reliable broadband communications using a burst erasure correcting code. In *Proc. ACM SIGCOMM 90*, pages 287–306. Philadelphia, PA, September 1990.

[MJV96] S. McCanne, V. Jacobson, and M. Vetterli. Receiver-driven layered multicast. In *SIGCOMM 96*, pages 117–130, Stanford, CA, August 1996.

[MVJ97] S. McCanne, M. Vetterli, and V. Jacobson. Low-complexity video coding for receiver-driven layered multicast. Technical Report SSC/1997/001, EPFL, Lausanne, Switzerland, January 1997.

[NB98] Jörg Nonnenmacher and Ernst Biersack. Optimal multicast feedback. Technical report, Institute EURECOM, BP 193, 06904 Sophia Antipolis cedex, FRANCE, July 1998.

[NBT97] J. Nonnenmacher, E. W. Biersack, and Don Towsley. Parity-based loss recovery for reliable multicast transmission. In *To appear in SIGCOMM '97*, Cannes, France, September 1997.

[PSLB97] S. Paul, K. K. Sabnani, J. C. Lin, and S. Bhattacharyya. Reliable multicast transport protocol (rmtp). *IEEE Journal on Selected Areas in Communications, special issue on Network Support for Multipoint Communication*, 15(3):407 – 421, April 1997.

[Riz97] L. Rizzo. On the feasibility of software fec. Technical report, Univ. di Pisa, Italy, January 1997.

[RV97] Luigi Rizzo and Lorenzo Vicisano. A reliable multicast data distribution protocol based on software fec techniques (rmdp). In *Proceedings of HPCS'97 Workshop*, Chalkidiki, Grece, June 1997. IEEE.

[SDW92] Timothy W. Strayer, Bert J. Dempsey, and Alfred C. Weaver. *XTP - THE XPRESS TRANSFER PROTOCOL*. Addison-Wesley, 1992.

[SEFJ97] P. Sharma, D. Estrin, S. Floyd, and V. Jacobson. Scalable timers for soft state protocols. In *Proc. INFOCOMM 97*, April 1997.

[TPB97] Thierry Turletti, Sascha Fosse Parisis, and Jean Bolot. Multicast transmission of layered audio over the internet. INRIA, B.P.93, Sophia-Antipolis Cedex, France, February 1997.

[Wax88] B. M. Waxman. Routing of multipoint connections. *IEEE JSAC*, 6(9):1617–1622, December 1988.

[WE94] Liming Wei and Deborah Estrin. The trade-offs of multicast trees and algorithms. In *Proceedings of ICCCN'94*, San Francisco, CA, USA, Sept 1994.

[YGS95] Rajendra Yavatkar, James Griffoen, and Madhu Sudan. A reliable dissemination protocol for interactive collaborative applications. In *Proceedings of ACM Multimedia*, pages 333–344, San Francisco, CA USA, 1995. ACM.