INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Improving the Efficiency of a Multicast File Transfer Tool based on ALC

Vincent Roca
INRIA Rhône-Alpes, Planète project, France
vincent.roca@inrialpes.fr, http://www.inrialpes.fr/planete/

Benoit Mordelet
Activia Networks, Sophia-Antipolis, France
benoit.mordelet@activia.net, http://www.activia.net/

N° 4411

March 2002

THÈME 1

Rapport de recherche

ISSN 0249-6399

Improving the Efficiency of a Multicast File Transfer Tool based on ALC

Vincent Roca
INRIA Rhône-Alpes, Planète project, France
vincent.roca@inrialpes.fr, http://www.inrialpes.fr/planete/

Benoit Mordelet
Activia Networks, Sophia-Antipolis, France
benoit.mordelet@activia.net, http://www.activia.net/

Thème 1 — Réseaux et systèmes
Projet Planète

Rapport de recherche n° 4411 — March 2002 — 17 pages

Abstract: This work describes several techniques that we used to design a multicast file transfer tool on top of ALC, the Asynchronous Layered Coding protocol proposed by the RMT IETF working group. More specifically we analyze several object and symbol ordering schemes that improve transmission efficiency and we see how the Application Level Framing (ALF) paradigm can help to reduce memory requirements and enable processing to be hidden behind communications. Because of its popularity and availability we use a Reed-Solomon FEC code, yet most of our results can be applied to other FEC codes. A strength of this work resides in the fact that all the techniques introduced have actually been implemented and their benefits quantified.

Key-words: multicast file transfer, multi-layer multicast transmissions, Asynchronous Layered Coding (ALC), Layered Coding Transport (LCT)

Unité de recherche INRIA Rhône-Alpes
655, avenue de l'Europe, 38330 Montbonnot-St-Martin (France)
Téléphone : +33 4 76 61 52 00 — Télécopie +33 4 76 61 52 52

Amélioration de l'efficacité d'un outil de transfert de fichier multipoint basé sur ALC

Résumé : Ce travail décrit des techniques utilisées lors de la conception d'un outil de transfert de fichier multipoint basé sur le protocole ALC, ou Asynchronous Layered Coding, proposé par le groupe RMT de l'IETF. Plus précisement nous étudions plusieurs solutions d'ordonnancement d'objets et de symboles qui visent à améliorer l'efficacité de transmission, et nous montrons comment le paradigme ALF, ou Application Level Framing, peut aider à réduire les besoins de stockage en mémoire et permettre de cacher les traitements derrière les communications. En raison de sa popularité et de la disponibilité d'implémentations open-source nous utilisons un code FEC de type Reed-Solomon, cependant la plupart de nos résultats s'appliquent à d'autres codes FEC. La force de ce travail réside dans le fait que toutes les techniques proposées ont été implémentées et leurs bénéfices quantifiés expérimentalement.

Mots-clés : transfert de fichier multipoint, transmissions multipoint multi-couches, Asynchronous Layered Coding (ALC), Layered Coding Transport (LCT)

1 Introduction

1.1 Motivations for Multi-Rate and Multi-Layer Group Communications

Using several multicast groups is a scalable and efficient way of sending information to a set of highly heterogeneous receivers, either in terms of processing power or networking capabilities. In this approach the source uses a layered data coding and transmits each layer in a separate multicast group. Receivers join as many groups as possible. If a receiver experiences losses, then he leaves one or more groups in order to reduce its reception rate [16][23]. The high scalability property derives from the fact that there is no feedback information flowing back to the source.

This approach has long been used for the transmission of multi-resolution video where each layer of refinement is mapped onto a different multicast group. Here receivers may receive a different amount of data depending on the number of layers they listen.

This approach can also be used for multicast file transfers. As the same amount of data is expected by each receiver, high-end and low-end receivers differ in the time they need to get data. How the source organizes data in the various layers is discussed later in this paper. This kind of application has no real-time or ordering constraint (e.g. receiving the second file before the first one has no consequences) but it assumes reliable transmissions.

1.2 Quick Introduction to ALC

ALC (Asynchronous Layered Coding) [10], a "massively scalable reliable multicast protocol" proposed by the RMT IETF working group provides a general framework for the reliable transmission of files. It uses the LCT (Layered Coding Transport) building block [12], a Layered Congestion Control (LCC) building block [15] and a FEC (Forward Error Correction) code building block [13]. ALC can be used for instance to distribute popular files (e.g. a video-clip or the latest Linux distribution). ALC builds upon many previous works in the area of multi-layer data organization schemes [18][22], congestion control for multi-layer transmission schemes [2][16][23] and Forward Error Correction (FEC) [14][17].

Throughout this paper the official ALC terminology is used: a data message submitted by the application to ALC is called an *object*. No assumption is made on the nature and size of these objects. Each object is segmented into one or more *blocks*, of limited size, usually because of FEC codec constraints. Each block is further segmented into *data symbols*, the unit of transmission (we assume there is no IP fragmentation below ALC). Finally the FEC codec adds a certain number of redundant *FEC symbols*.

Several transmission modes are possible [12]. In this work we only consider transmissions in:

- *push mode:* all the receivers must be ready before the transmission starts (synchronous start)
- *on-demand mode:* data is sent continuously in a loop. Thus receivers can arrive at any time (asynchronous start), download the file and leave the session.

Because of reliability constraints using FEC is mandatory. [14] identifies three classes of FEC codes: small block, large block and expandable codes. Choosing one of them has many consequences:

- on efficiency: a small block code requires to split the object into multiple blocks which reduces reception efficiency [5]
- on the number of redundant FEC symbols that can be generated for each original symbol
- On the coding and decoding speed

This work relies on a small block Reed-Solomon code. If not the most efficient, this class of FEC code is currently the most popular because of the availability of high quality open-source implementations like [17].

1.3 Goal of the Work and Organization of the Paper

Our goal is to *improve the overall performances of a multicast file transfer tool built on top of ALC*, working either in a push or on-demand mode. A strength of this work resides in the fact that all the features we introduce have actually been implemented and their benefits quantified.

The rest of the paper is organized as follows: the following section introduces related work; section 3 discusses the various meanings of efficiency; we introduce our proposals in section 4; we discuss experimental results in section 6; we introduce an additional optimization in section 7; and finally we conclude.

2 Related Work

ALC raises the problem of data organization on the various layers. The problem of finding efficient cumulative layered organizations has been addressed in [1] [7] [19] [22]. A common denominator of these schemes is that they rely on a *deterministic algorithm* to decide on which layer and at what time to send each data packet. Because of this determinism, efficiency is high (e.g. in [1] the whole file can be received without any packet duplication). Yet several problems like channel desynchronization (e.g. due to different routes on the various multicast groups) or start delay in practice significantly reduce this efficiency [7]. In addition [22] adds some requirements and the number of FEC packets produced cannot be freely chosen. Finally using a congestion control protocol will lead each receiver to dynamically add and remove layers and receivers will miss many packets on higher layers.

Another class of cumulative layered organizations consists in sending data and FEC packets in a *fully random order* on the various layers [5] [12]. In addition to its simplicity, the idea is to have the same efficiency no matter how layers are added or dropped and no matter how losses occur (periodically, randomly, or in bursts). Such an approach is also required in case of non-cumulative layering [4]. Our work follows this random organization principle.

Finally [8] discusses the implementation of a multicast file transfer tool. This tool is based on a Reed-Solomon FEC codec (as us) but uses a single fixed rate layer and provides no congestion control. Because of the single fixed rate nature of the tool, some receivers frequently miss packets due to a reception rate higher than the possible disk access rate. Several strategies are analyzed to overcome this problem. Note that if this application is also called Fcast, it has no direct relationship with our own FCAST tool (section 6.1).

3 The Various Meanings of Efficiency

Efficiency has various meanings when applied to a file transfer tool based on ALC. We list them and give an indication of how we propose to address them:

- *Efficient transmissions:* The number of duplicated symbols (e.g. the same symbol received on various layers, or symbols received after the decoding of their block) must be kept as low as possible.

  ⇒ Section 4 introduces several object and symbol organization schemes to improve transmission efficiency.

- *Efficient behavior in front of losses:* Transmissions must be robust in front of all forms of packet losses.

  ⇒ The random nature of our symbol organization and the presence of a large number of FEC symbols warrant a good behavior in front of losses.

- *Efficient CPU usage:* It concerns either the source or the receiver and can be a limiting factor on lightweight hosts.

  ⇒ Section 6.4.1 defines several profiles to accommodate CPU bounded receivers, or on the opposite to enable powerful workstations to hide computations behind communications.

- *Efficient memory usage at a receiver:* This is the amount of physical memory required to receive the objects. With large objects it can quickly become a limiting factor on lightweight hosts.

  ⇒ Section 4 introduces several schemes (ALF approach and "m/p_rand" organization) to reduce this requirement.

- *Efficient disk usage at a receiver:* Because of the random nature of transmissions, a receiver may quickly be limited by the non-sequential disk access speed (e.g. if he tries to store symbols at their final location). [8] describes several strategies to optimize disk I/Os at a receiver.

⇒ Section 4 introduces several schemes (ALF approach and "m/p_rand" organization) that postpone the moment when, by lack of physical memory, a receiver needs to store symbols on disk. Otherwise a receiver always stores symbols in sequence on disk once its physical reception cache turns full.

- *Efficient disk usage at the source:* With huge files whose size exceeds the physical memory size of the source, storing data symbols on disk is unavoidable. Besides, because of the processing cost of generating FEC symbols, these latter cannot be produced just-in-time. Instead FEC symbols are pre-calculated which still increases the storage requirements. The problem is that the random nature of transmissions quickly limits the transmission rate with the non-sequential disk I/O speed.

  ⇒ This aspect is out of the scope of the present document and will be addressed in future work.

4 Our Proposals

This section introduces our proposals. To the best of our knowledge, these problems have never been addressed before.

4.1 ALF Applied to Multicast File Transfer

The Application Level Framing (ALF) paradigm [6] says (1) that the control and transmission units must be the same for optimal efficiency and (2) that the application is the best location to define this unit. Applied to our case, the application can choose to cut a large file into multiple independent fragments. Each fragment contains all the information required to enable its processing at the receiver, no matter the order in which it is received. An example is given in figure 8. Note that with a small block FEC code a large object is anyway split into several blocks. In that case a fragment can be composed of b blocks, where $b \geq 1$ is a small integer. This approach has several benefits:

- *a reduced memory consumption at the receiver:* A receiver no longer needs to keep a copy of the whole file in memory until the last missing symbol arrives. Instead, memory can be freed as soon as a fragment is completed.
- *it postpones the moment when, by lack of physical memory, disk storage of incoming symbols is required.* This possibility may dramatically increase the effective reception rate. Indeed, in the absence of elaborated schemes like [8], symbols are stored sequentially on disk. But the high randomization of the symbol transmission order leads to inefficient random disk I/Os when recovering each object from the scattered symbols.
- *processing can be hidden behind communications:* Each fragment can be processed by the application as soon as it is received, instead of waiting the end of reception of the whole file. This is more comfortable for a user as the file is almost immediately available on receiving the last missing symbol.

Yet the ALF approach *assumes that the network is indeed the limiting factor* which may not be true in all situations (section 6.4.3). Another constraint is the necessity to add information (meta-data) to each fragment (section 6.1). The associated overhead is anyway very limited, around 0.2% with 64 kilobyte file fragments.

4.2 How to Deal with Multiple Objects?

The following issue is how to manage multiple objects? These objects can be either the fragments of a given file (section 4.1) or each of them be a separate file (e.g. with a recursive directory transmission).

4.2.1 Sending Objects in Sequential Order

A *first scheme*, called "m/seq" (for Multiple objects, SEQuential object order), consists in sending objects in a sequential manner, i.e. all the symbols of object $i-1$ are sent before those of object i, even if within each object symbols are sent in a random order. This solution is obviously inefficient. Figure 1 shows that as time goes, lower layers become more and more late compared to

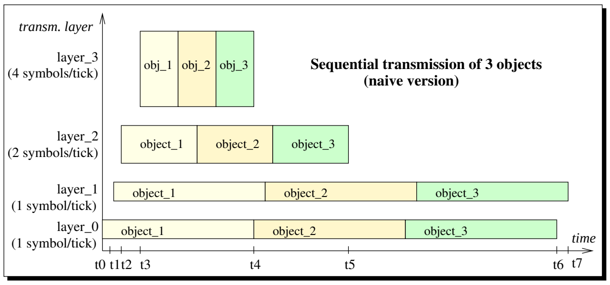


Figure 1: Multiple objects - sequential object order (m/seq).

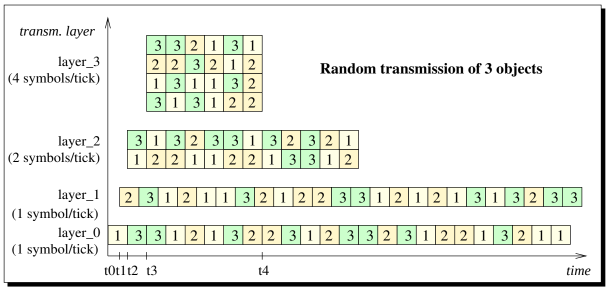


Figure 2: Multiple objects - random symbol order (m/rand) (figures identify the object to which each symbol belongs).

higher layers. A host receiving all four layers will get symbols of object 3 only from layer 3 and cannot finish before time t_4. This is confirmed by the experimental results of section 6.3.

4.2.2 Sending Symbols in Fully or Partially Random Order

A *second scheme*, called "m/rand" (for Multiple objects, RANDom symbol order), consists in mixing all the symbols of all the objects and sending them in a different random order in each layer (figure 2). Here a host receiving all layers still benefits from all of them at any time and reception finishes before time t_4.

A *variant of this second solution*, called "m/p_rand" (for Multiple objects, Partially RANDom symbol order), consists in using a partially random permutation of symbols. In that case, the probability that a symbol s is not permuted is:

$$Pr_{not\_perm}(m/p\_rand) \geq Pr_{not\_perm}(m/rand)$$

with:

$$Pr_{not\_perm}(m/rand) = \frac{1}{total\ nb\ of\ symbols}$$

A very simple algorithm to calculate pseudo-random permutations is given in Annex A.

4.2.3 The KEEP/PUSH Functions of the API

The last two schemes raise an issue: the scheduling function of ALC must be informed of all the objects the application wants to transmit before being able to start. Therefore two functions are required in an API (Application Programming Interface) built on top of ALC (e.g. section 6.1): before submitting the first object, the application issues a `KEEP_DATA` which informs ALC that several objects are expected; once all the objects are submitted, the application issues a final `PUSH_DATA` which triggers the scheduling of all the symbols of all the objects.

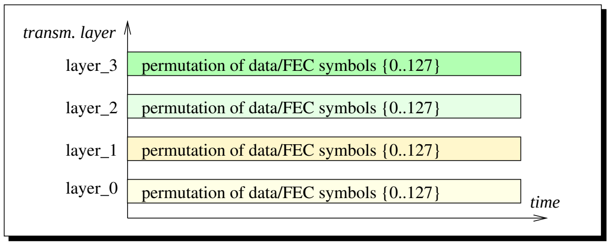
4.3 What Symbols to Send in Each Layer?

The previous section explained how to transmit multiple objects; this section discusses a complementary issue, namely what symbols to send in each layer.

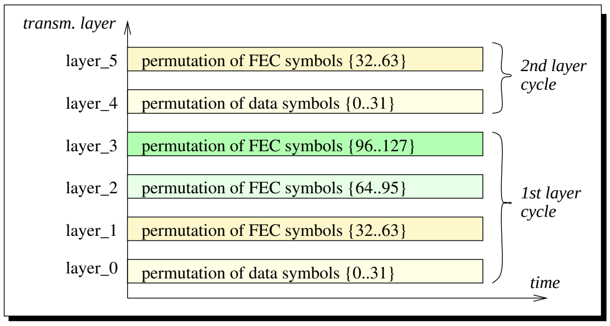
Let k be the number of data symbols per block. Let n be the total number of symbols per block (data plus FEC). Using a Reed-Solomon FEC codec like over a Galois Field GF(2^8) (default) limits the n parameter to 256 and k to a small value for computational reasons [17]. We choose $k = 32$ as suggested in [8]. There can be at most $n - k = 256 - 32 = 224$ FEC symbols for each data block. In practice, because of the need to pre-calculate and store them, we limit the maximum number of FEC symbols to $3 * k = 96$. Finally, a symbol is by default 512 bytes long to avoid IP fragmentation (path MTU discovery is difficult with multicast). Therefore a large object is segmented into $32 * 512 = 16$ kbyte blocks. We identify two strategies to assign symbols to layers.

4.3.1 Symbol Organization 1

For each block of an object, the *same* data symbols $\{0..k-1\}$ and FEC symbols $\{k..n-1\}$ are sent in each layer (figure 3 (a)). Then, for each layer, the final symbol transmission order is controlled by one of the three scheduling schemes previously defined: "m/seq", "m/p_rand", "m/rand".



(a) Organization 1: n symbols per layer



(b) Organization 2: k symbols per layer

Figure 3: The two symbol organizations, single block, k=32, n=128.

4.3.2 Symbol Organization 2

With organization 2 the n/k ratio is necessarily an integer. With $n/k = 4$, data symbols $\{0..k-1\}$ are affected to layer 0, FEC symbols $\{k..2k-1\}$ are affected to layer 1, FEC symbols $\{2k..3k-1\}$ to layer 2 and FEC symbols $\{3k..n-1\}$ to layer 3. We say in that case that there are "three FEC layers" and each group of four layers forms a "layer cycle". More generally layer i contains the symbols affected to layer $i \, modulo \, 4$ (figure 3 (b)). Here also for each layer the final transmission ordering is controlled by the scheduling scheme previously defined.

The obvious asset of this organization is that no matter how layers are subscribed, no matter the loss pattern, a receiver experiences no packet duplication *among* each layer cycle[1]. This scheme implicitly assumes that a low-end receiver can receive at least the first two layers as no FEC packet is sent on the base layer. An alternative is to merge layers 0 and 1.

[1]Other sources of duplication are still possible, for instance when receiving additional packets of an already completed block.

5 Mathematical Models

This section introduces two models giving an approximation of the average end of reception time of a block for the two symbol organizations defined in section 4.3. These models are too much simplified to realistically reflect the reality. Yet they are sufficient to *fairly compare* the two symbol organizations.

5.1 Notations and Hypothesis

We use the following notations:

- n_l: total number of layers a receiver uses.
- l: index of the layer considered: $l \in \{0..n_l - 1\}$.
- t: time expressed in ticks. Transmissions on the various layers occur at each time tick. The number of ticks/s is an internal parameter that controls the transmission granularity.
- r_i: transmission rate in symbols/tick for layer i
- R: cumulative transmission rate in symbols/tick over all the layers: $R = \sum_{i=0}^{n_l-1} r_i$
- k: number of data symbols per block.
- n: total number of symbols per block (data plus FEC).
- b: number of blocks in the object: $b = \lceil \frac{object\_size}{k*symbol\_size} \rceil \geq 1$

We assume that:

- there is no loss: the models only give an optimistic bound that is nonetheless sufficient for comparisons.
- a receiver gets data from n_l layers immediately (i.e. there is no congestion control): this is the asymptotic behavior with large objects where the layer addition time is low in front of the total reception time. This assumption is of course wrong with small files.

- n/k is an integer to enable a fair comparison of the two symbol organizations.
- we only consider the case of a single object whose symbols are transmitted in a fully random order, using either symbol organization 1 or 2.

Because there is no symbol loss, there is no symbol duplication on each layer either (i.e. reception is finished quickly enough so that the source does not have time to enter a new transmission cycle on any of its layer). The only possibility of duplication is the reception of the same symbol on different layers.

5.2 Symbol Organization 1

To calculate the number of new unduplicated symbols brought by each layer we use the same approach as in [3]. In [3] symbols arrive from multiple mirror sites whereas in our case they come from multiple independent layers. A difference is the necessity in our case to distinguish several blocks (we do not restrict ourselves to large block FEC codecs).

Let $z_i(t)$ be the average number of unduplicated symbols received on layers $\{0..i\}$ at time t. Then from the average $\frac{t*r_{i+1}}{b}$ symbols of a block received on layer $i+1$ up to time t, only a $(1-\frac{z_i(t)}{n})$ fraction are unduplicated. Therefore:

$$\left|\begin{array}{l} z_0(t) \simeq \frac{t*r_0}{b} \\ z_1(t) \simeq z_0(t) + \frac{t*r_1}{b}(1-\frac{z_0(t)}{n}) \\ \dots \\ z_{n_l-1}(t) \simeq z_{n_l-2}(t) + \frac{t*r_{n_l-1}}{b}(1-\frac{z_{n_l-2}(t)}{n}) \end{array}\right. \quad (1)$$

Note that this is only an optimistic estimation as it assumes that the number of symbols received on each layer actually equals its expected value (in fact there cannot be fractions of symbols).

For the reception to finish, at least k symbols must be received for each block. The average end of reception time for any block, t_{end_rx}, is the solution of an equation of degree n_l: $z_{n_l-1}(t) = k$. This is not the end of reception time of the whole object (in particular this reception is subject to the so-called "coupon collector's problem", i.e. waiting for the last block to fill [5]), but it is sufficient to fairly compare both organizations.

5.3 Symbol Organization 2

We use two additional notations here:

- n_{lc}: number of layers in a cycle. Each cycle consists in a single data layer and $\frac{n}{k}-1$ FEC layers: $n_{lc} = \frac{n}{k}$
- c: number of layer cycles. We assume that there is an integral number of cycles: $n_l = c * n_{lc}$

As long as $n_l \leq n_{lc}$, there cannot be any symbol duplication between the layers. With more than one layer cycle, some symbols sent on layer $l = l_1 * n_{lc} + l_2 \geq n_{lc}$ may have already been sent on layers $l_2 + i * n_{lc}$, $\forall i \in \{0..l_1-1\}$ (figure 3 (b)). The same method as above gives:

$$\left|\begin{array}{l} first\ layer\ cycle: \\ z_0(t) \simeq \frac{t*r_0}{b} \\ \dots \\ z_{n_{lc}-1}(t) \simeq \frac{t*r_{n_{lc}-1}}{b} \\ \\ second\ layer\ cycle: \\ z_{n_{lc}}(t) \simeq z_0(t) + \frac{t*r_{n_{lc}}}{b}(1-\frac{z_0(t)}{k}) \\ \dots \\ z_{2*n_{lc}-1}(t) \simeq z_{n_{lc}-1}(t) + \frac{t*r_{2n_{lc}-1}}{b}(1-\frac{z_{n_{lc}-1}(t)}{k}) \\ \\ layer\ cycle\ c-1: \\ z_{(c-1)*n_{lc}}(t) \simeq z_{(c-2)*n_{lc}}(t) + \\ \qquad \frac{t*r_{(c-1)*n_{lc}}}{b}(1-\frac{z_{(c-2)*n_{lc}}(t)}{k}) \\ \dots \\ z_{c*n_{lc}-1}(t) \simeq z_{(c-1)*n_{lc}-1}(t) + \\ \qquad \frac{t*r_{c*n_{lc}-1}}{b}(1-\frac{z_{(c-1)*n_{lc}-1}(t)}{k}) \end{array}\right. \quad (2)$$

For the reception to finish, at least k symbols must be received for each block. The average end of reception time for any block, t_{end_rx}, is the solution of an equation of degree c: $\sum_{i=0}^{n_{lc}-1} z_{(c-1)*n_{lc}+i}(t) = k$.

5.4 Partial Conclusions

We solved these equations numerically. The t_{end_rx} values of each organization for a given number of layers and FEC symbols are presented in figure 4 and the corresponding ratio: $\frac{t_{end\_rx}(org\,2)}{t_{end\_rx}(org\,1)}$ in figure 5. It shows that *organization 2 always enables a 6.8 to 14.7% faster object* *reception than organization 1.* This is all the more true as the number of layer cycles is low (there will be fewer or even no duplication at all for organization 2) and the number of FEC symbols generated low (the probability of symbol duplication for organization 1 is higher).

Receiving the object faster also means receiving less duplicated symbols. Figure 6 shows the duplication ratio experienced by a receiver: $dup\_ratio = \frac{number\,of\,duplicated\,symbols\,received}{total\,number\,of\,symbols\,received}$, for a given number of layers and FEC symbols. The optimal value is of course 0. From that point of view, *organization 2 enables significant savings compared to organization 1.* For instance with a single layer cycle, there is no duplication for organization 2, whereas this duplication ratio is between 14.6 and 8.6% for organization 1. With two or three layer cycles organization 2 remains efficient with more than 6% savings compared to organization 1.

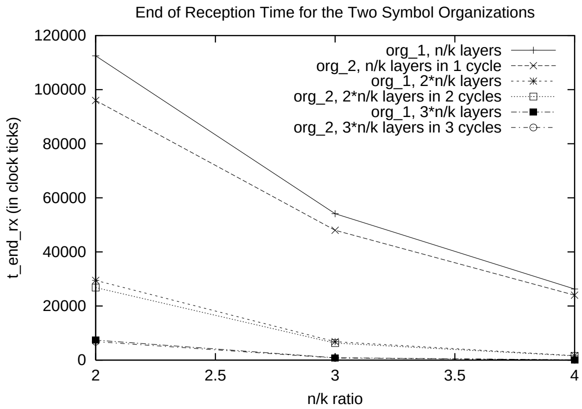


Figure 4: “End of Reception” time for the two symbol organizations (100 MB file, k=32).

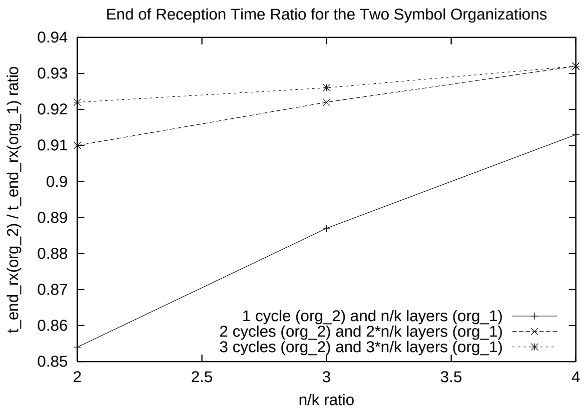


Figure 5: “End of Reception ratio” for the two symbol organizations (100 MB file, k=32).

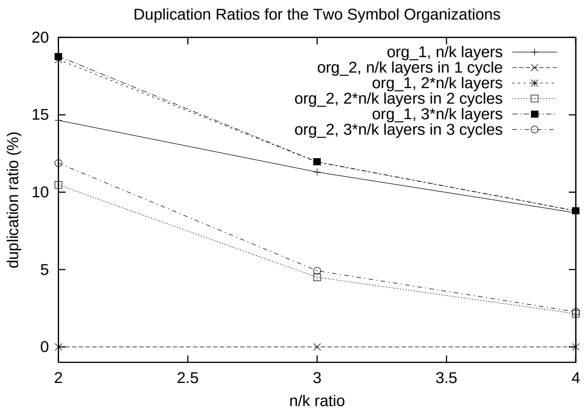


Figure 6: “duplication ratio” for the two symbol organizations (100 MB file, k=32).

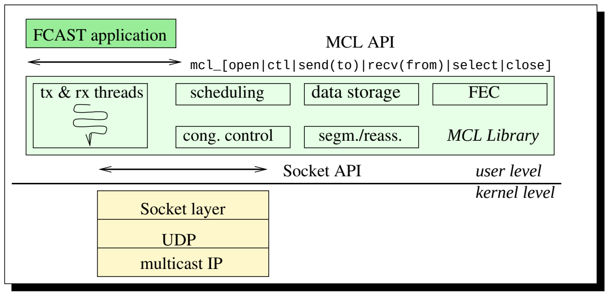


Figure 7: The MCL library and the FCAST application.

6 Experimental Results

All the proposals introduced so far have been implemented. This section gives an account of several experiments we carried out and the various transmission profiles we defined based on these results.

6.1 The MCL Library and the FCAST Application

We implemented the ALC, LCT and RLC protocols within a library, MCL (MultiCast Library) [20], built on top of UDP/multicast IPv4 (figure 7). We also implemented several applications on top of MCL, in particular FCAST, a recursive multicast file transfer tool inspired from [11]. We integrated all the schemes of section 4 in FCAST. For instance when splitting a large file, several meta-data are appended to each fragment to make

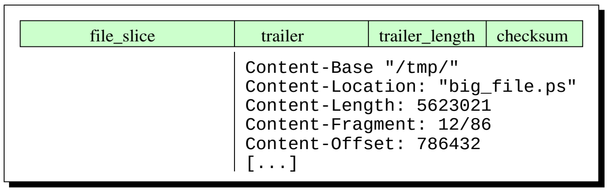


Figure 8: The FCAST encapsulation format of a file fragment (12th slice of file /tmp/big_file.ps). If the file is sent as a single object, the "content-fragment/offset" meta-data are removed.

them autonomous (figure 8). FCAST also includes an application-level checksum to check the fragment integrity. The whole trailer size is typically around 140 to 170 bytes long, which is reasonable (e.g. with the default 64 kilobyte fragment size it represents a 0.2% overhead).

6.2 Tests Methodology

Table 1 summarizes the tests performed, showing the various combinations between the transmission scenario and symbol organization. We evaluate several performance metrics:

- the end of reception time of each object,
- the redundancy experienced by a receiver: $dup\_ratio = \frac{number\ of\ duplicated\ symbols\ received}{total\ number\ of\ symbols\ received}$ (we consider here the two kinds of packet duplications: inter-layer duplicates and extra packets received after the completion of a block)
- the maximum amount of memory required by a receiver,
- and the CPU load at the source and at a receiver.

All the hosts are attached to the same LAN to focus on our proposal performances without being disturbed by those of the multicast routing infrastructure. We assume that $n/k = 4$, i.e. 3 times more FEC symbols than data symbols. With symbol organization 2 we say in that case that there are 3 FEC layers.

6.3 Comparison of the Various Transmission Schemes

This section compares the various transmission schemes of section 4 during the transmission of a single 1042142 byte (approximately 1 MB) file. The eight possibilities are summarized in figure 9. In these tests, the sending rate is voluntarily low. We also use in a first step a simplified version of FCAST which includes a single meta data, the fragment offset in the object.

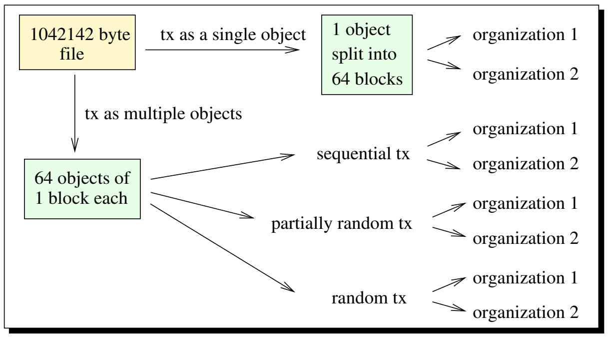


Figure 9: Experiments performed (∼1 MB file).

6.3.1 Loss-Free Transmissions

We first assume that *no loss occurs* in order to have a fair comparison in an optimal situation. We repeat each test 40 times and plot the minimum/average/maximum transmission duration, duplication ratio and maximum buffer space (figures 10 to 12).

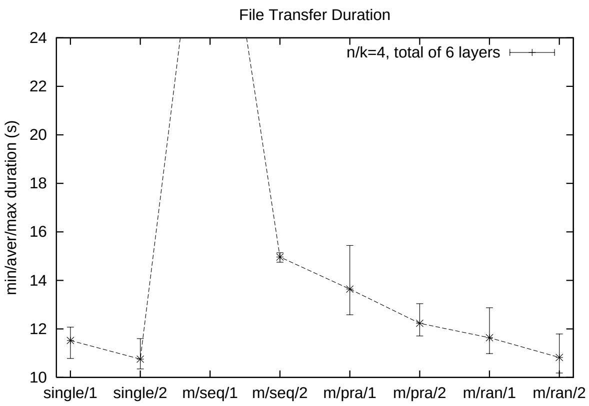


Figure 10: Transfer duration with various transmission schemes (no loss, ∼1 MB file).

6.3.2 Lossy Transmissions

We now introduce random bursty losses according to a Guilbert loss model [9]:

- loss probability when previous symbol is received $= p_{ok} = 0.01$,

Table 1: Test matrix showing the various combinations.

scenario description	name	org_1	org_2
transmit as a single object	single	yes	yes
split in mult. obj., sequential order	m/seq	yes	yes
split in mult. objects, partially random order, imm. delivery upon rx	m/p_rand	yes	yes
split in mult. objects, random order with immediate delivery upon rx	m/rand	yes	yes

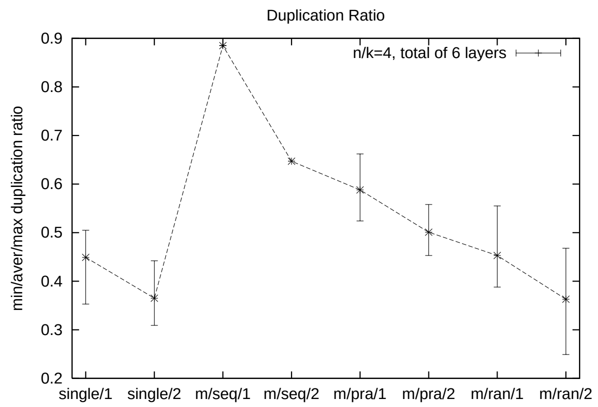


Figure 11: Duplication ratio with various transmission schemes (no loss, ~1 MB file).

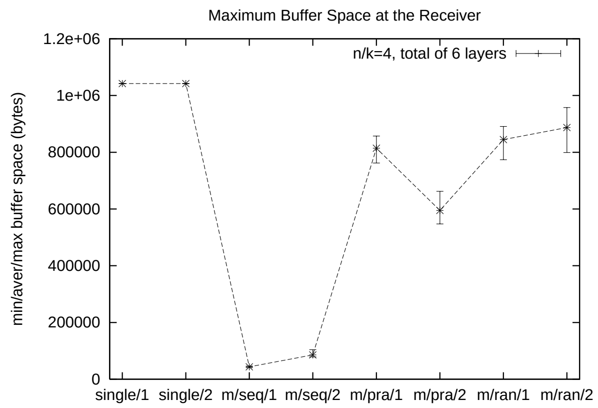


Figure 12: Maximum buffer space at the receiver with various transmission schemes (no loss, ~1 MB file).

- loss probability when previous symbol is lost $= p_{bad} = 0.75$.

The stationary probability for a symbol to be lost is:

$$proba_{lost} = \frac{p_{ok}}{1 - p_{bad} + p_{ok}} = 0.0385$$

the average number of consecutive losses:

$$consecutive\_losses = \frac{1}{1 - p_{bad}} = 4 \ symbols$$

and the average number of consecutive received symbols:

$$consecutive\_non\_lost = \frac{1}{p_{ok}} = 100 \ symbols$$

Because of the presence of a congestion control scheme, high loss ratios are not expected to be frequent under normal conditions. We perform the same experiments as previously, repeating each test 40 times, and plot the minimum/average/maximum values (figures 13 to 15). Results are summarized in table 2.

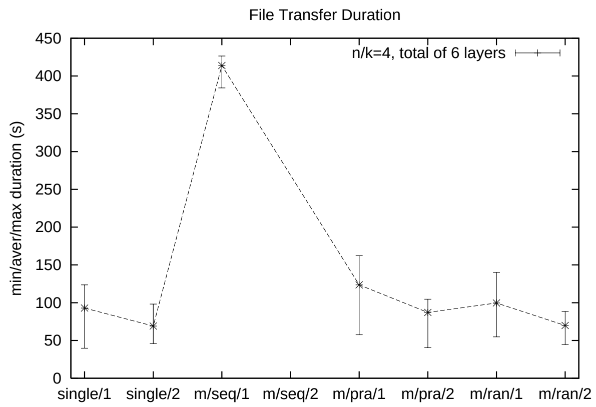


Figure 13: Transfer duration with various transmission schemes (bursty random 1%/75% losses, ~1 MB file).

The relatively high loss rate prevented the receiver to subscribe to more than three layers in all cases. In case of the "m/seq/2" scheme, the receiver never managed to complete all objects (between 48 to 61 objects out of 64 have been completed).

Table 2: Summary of the average results normalized by the "single/org_2" case (bursty random 1%/75% losses, ~1 MB file, 6 layers, $n/k = 4$).

	dup ratio	*space (bytes)*	*duration*
single/org_1	1.59	1.00	1.35
single/org_2	*1.00*	*1.00*	*1.00*
mult/seq/org_1	3.35	0.09	6.00
mult/seq/org_2	failed	failed	failed
mult/p_rand/org_1	2.18	0.60	1.79
mult/p_rand/org_2	1.49	0.53	1.26
mult/rand/org_1	1.69	0.70	1.44
mult/rand/org_2	1.01	0.74	1.01

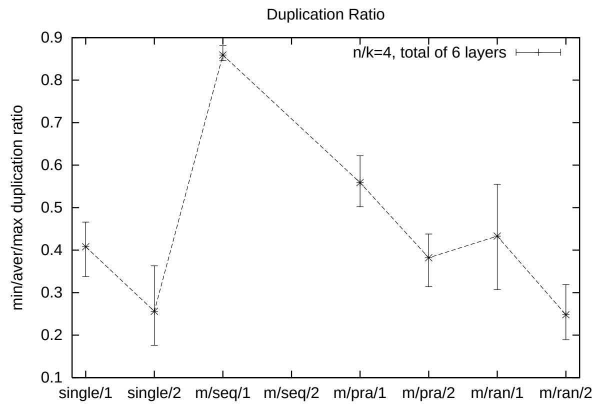


Figure 14: Duplication ratio with various transmission schemes (bursty random 1%/75% losses, ~1 MB file).

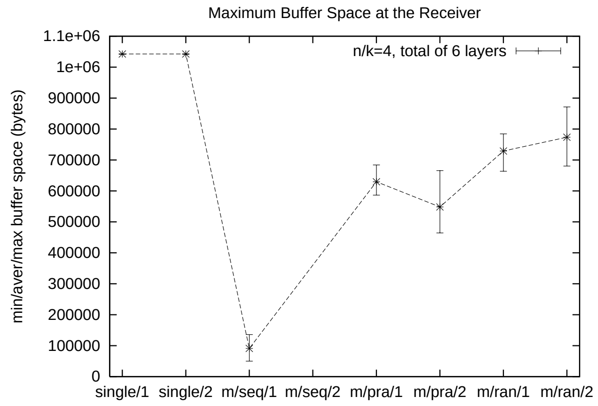


Figure 15: Maximum buffer space at the receiver with various transmission schemes (bursty random 1%/75% losses, ~1 MB file).

6.3.3 Partial Conclusions

In all cases, the organization 1 versions have a higher duplication rate than the corresponding organization 2 versions which confirms the theoretical results of section 5. Therefore we will only consider organization 2 in the rest of this paper.

The "single/org_2/3_FEC_layers" and "m/rand/org_2/3_FEC_layers" have very similar performances. For instance, the latter has a 0.7% (no loss) to 1% (with losses) higher average reception duration. The difference is more important when considering memory requirements. Indeed, the "m/rand/org_2/3_FEC_layers" approach enables between 14.9% (no loss) and 25.8% (with losses) memory savings at the receiver.

In situations where receivers are highly memory-limited, using "m/p_rand/org_2/-3_FEC_layers" scheme is surely the best solution. It requires between 42.9% (no loss) to 47.4% (with losses) less memory than with the "single/org_2/3_FEC_layers" approach. It is in fact an intermediate solution between the "m/seq" (neither efficient nor robust in front of losses) and the "m/rand" extremes.

Finally, sending objects in sequence is definitively a bad strategy.

6.4 Application to FCAST

6.4.1 Definition of the FCAST Profiles

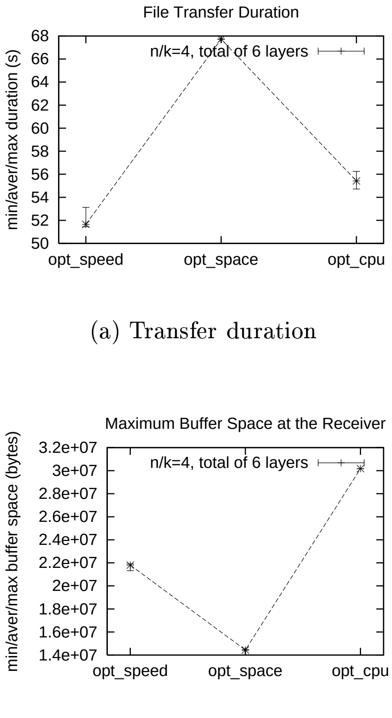
Previous results lead us to define three profiles:

- `opt_speed` to hide processing behind communications while optimizing reception speed. It is equivalent to "m/rand/org_2".
- `opt_space` to reduce the maximum memory requirements, hide processing behind communications, and spread the CPU load at the receiver. The price to pay is a slightly more

important reception time. It is equivalent to "m/p_rand/org_2"

- `opt_cpu` in situations where processing is the limiting factor. Without this profile, symbol losses may occur and would reduce the reception speed (section 6.4.3). It is equivalent to "single/org_2" with delayed FEC decoding (i.e. once all the required symbols of all blocks have been received) and delayed object delivery to the receiving application.

These profiles concern both the source (e.g. when defining the object/symbol ordering) and the receiver (e.g. to postpone FEC decoding). Therefore this option must be agreed using an out-of-band mechanism before the transmission starts, otherwise the desired feature may not be achieved (but without compromising the transfer). A solution if receivers have different desires is for the source to create three sessions, one per FCAST profile, and to let each receiver choose the most appropriate one.



(a) Transfer duration

(b) Maximum buffer space

Figure 16: Comparison of the FCAST profiles (30MB file).

6.4.2 Effectiveness of the Profiles With Powerful Hosts

In order to assess the effectiveness of the profiles with the full featured FCAST tool, we transmit a large (30MB) file between two powerful hosts (PIII-1GHz/Linux PCs). We repeat each experiment 10 times and plot the minimum/average/maximum values in figure 16. We can see that the results are fully compliant with the theory. In particular the `opt_speed` profile reduces reception duration by spreading processing (i.e. FEC decoding, data copy and application level checksum) during the whole reception time. The corresponding reception rate (including coding, transmission and decoding times) is 4.6Mbps. Memory requirements are also reduced significantly (-27%).

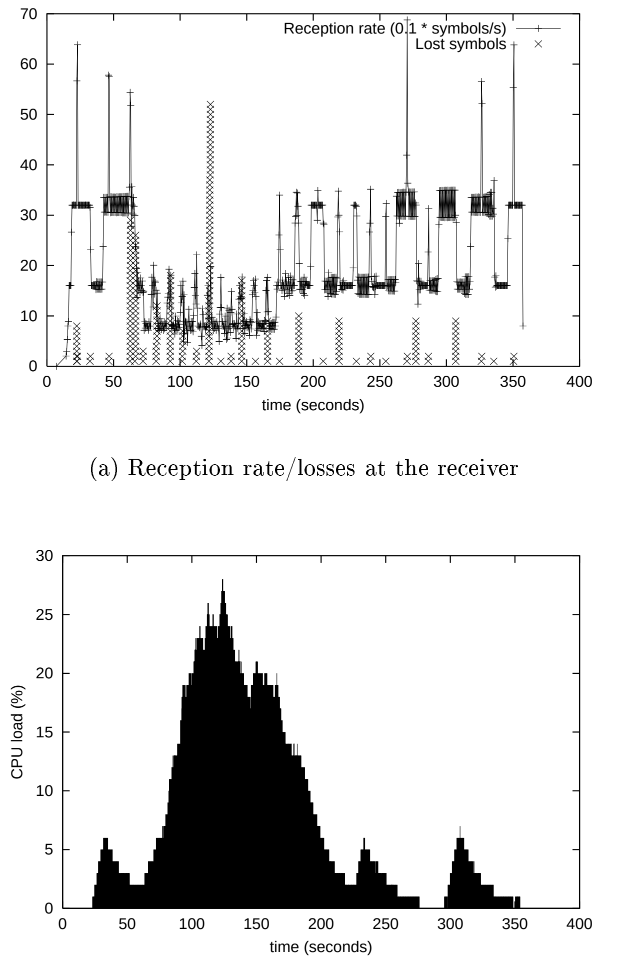
6.4.3 The Case of CPU Bounded Hosts

We now compare the `opt_speed` and `opt_cpu` profiles on a CPU bounded host. We send a 10 MB file, using 6 layers and 3 FEC layers. Experiments show that a Sun/Ultra10 running a full featured FCAST tool, that uses a Reed-Solomon FEC codec with $k = 64$ (to increase the processing load), and where the transmission rate is $\sim$1.5 Mbps, can indeed be limited by its processing power. This is visible in figure 17 where the high loss period $[60s; 180s]$ matches the high CPU load period (FEC decoding, data copy to the application buffer, FCAST checksum verification and final copy to disk)[2].

When such a phenomenon occurs it is wiser to postpone FEC decoding and object delivery to the application. This is visible in figure 18 where the CPU activity remains low except after the completion of all the object blocks, at time 231s. Thanks to the `opt_cpu` profile, the total reception time as seen by the user has been reduced, from 357.5 to 272.5 seconds (-23%).

In practice a user may know that its host is not powerful enough for a high speed session and may decide beforehand to join an FCAST session optimized for CPU. Otherwise the user may decide changing mode midway through a transfer.

[2]Note that the tool used to calculate CPU usage largely underestimates it; e.g. decoding effectively uses 100% of the CPU.



(a) Reception rate/losses at the receiver

(b) CPU load at the receiver

Figure 17: Impacts of FCAST with the opt_speed profile on a CPU bounded receiver.

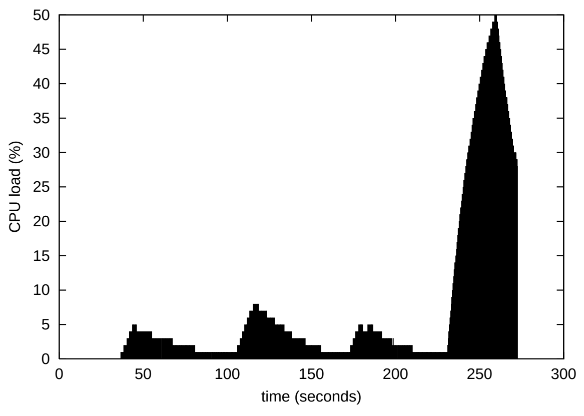


Figure 18: Impacts of FCAST with the opt_cpu profile on a CPU bounded receiver.

7 The Transmission Anticipation Optimization for PUSH Sessions

This section introduces an additional optimization that provides refinements on the various object and symbol organization schemes already discussed.

7.1 Principles

So far a source had to calculate all the FEC symbols – a CPU intensive process – and an appropriate object and symbol organization before being able to start transmissions. An optimization consists in starting the transmission of data symbols immediately, in sequence, only once, and on a predefined fixed number of layers, n_{al}, without waiting the end of the FEC calculation and symbol organization processes (figure 19). This is an *optimistic approach* where the source bets that receivers will be able to take advantage of this transmission to recover many – if not all – objects rapidly, before entering the standard transmission step that follows.

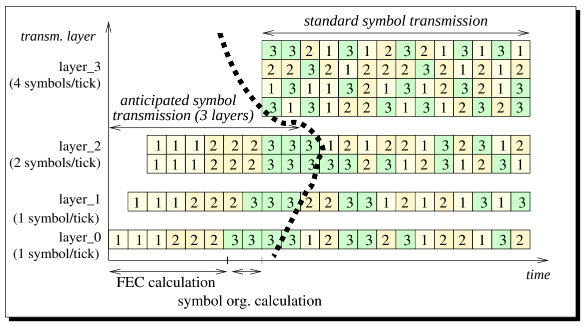


Figure 19: Symbol organization with the anticipated transmission optimization.

This optimization has many benefits:

- it enables transmissions to *start earlier*, thereby reducing the total reception time
- it favors the reception of *in sequence data symbols*, instead of receiving data and FEC symbols in a random order according to the FCAST profile
- as the probability of quickly receiving and processing the various objects in sequence is

higher, the *CPU load is more evenly spread* and the *memory requirements lower*

This optimization does not compromise the FCAST robustness in front of the various loss models (periodic, random or in burst). Even if losses can affect the anticipated transmissions, these losses will anyway be recovered afterwards as anticipated transmission are followed by the standard symbol organization.

Of course this optimization cannot be applied to transmissions in on-demand mode as clients will arrive at any time while our optimization only applies at the very start of a session.

7.2 Experimental Results, loss-free transmissions

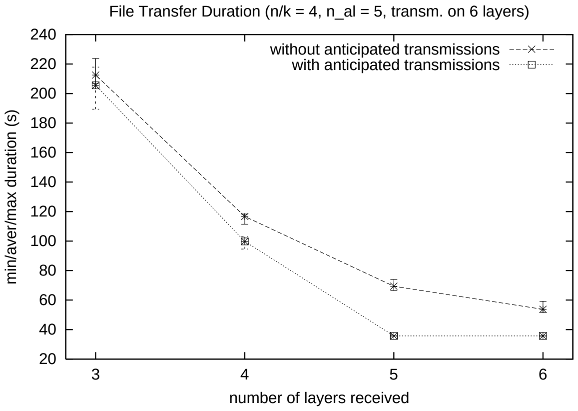
We carried out several experiments in which a large (30MB) file is transmitted between two powerful hosts using the `opt_speed` profile of FCAST and $n_{al} = 5$. Each experiment is repeated 10 times and we report the minimum/average/maximum transmission duration and buffering requirements in fig 20.

These figures show the *high efficiency of this optimization* on hosts capable of receiving at least n_{al} layers. With 6 layers it results in a 88,5% buffering space improvement and 33,6% reception time improvement.

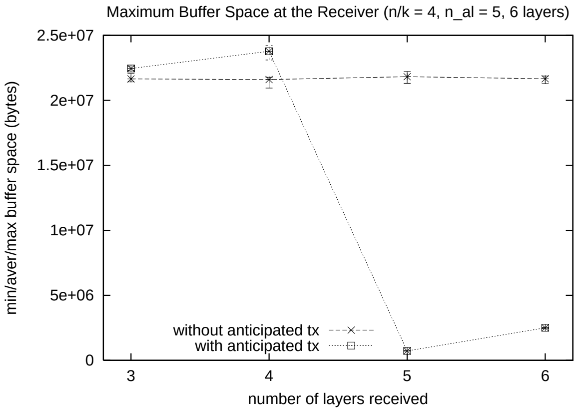
The *efficiency is maximal for hosts receiving exactly n_{al} layers* as it is equivalent to the sequential transmission of all objects. In that case:

- the reception time is close to the optimum: $file\ size/cumulative\ tx\ rate\ over\ n_{al}\ layers$ and represents a 48.4% improvement
- the buffering space is close to the object size – instead of being close to the file size –. The average maximum buffering space is 757 kbytes, a 96.7% improvement compared to the 21.8 MB required without this optimization

On hosts receiving fewer than n_{al} layers, the efficiency is lower as these hosts will miss some of the data symbols in all objects transmitted in advance. But it still *enables a reduction of the reception time* as these hosts anyway need fewer FEC symbols (and thus FEC decoding) and start reception earlier. A drawback is the need for (slightly) more buffering space as all the objects are received in parallel.



(a) Transfer duration



(b) Maximum buffer space

Figure 20: Benefits of the anticipated transmission optimization (30 MB file, FCAST in push mode, opt_speed profile, no loss).

7.3 Experimental Results, lossy transmissions

In a second step we introduced random 1%/75% bursty losses and compared the performances with and without the anticipated transmission optimization.

Table 3 shows that the reception time is almost identical (0.33% penalty) with a slight (-9.2%) reduction of the maximum buffering size. It confirms that *this optimization does not affect the global robustness in front of losses.*

Table 3: Impacts of the anticipated transmission optimization on robustness (30 MB file, FCAST in push mode, opt_speed profile, bursty random 1%/75% losses).

	min/aver/max space (Mbytes)	*min/aver/max duration (s)*
without anticipated tx	22.22 / 22.58 / 23.00	333.83 / 387.88 / 438.35
with anticipated tx	19.64 / 20.50 / 22.05	334.52 / 389.18 / 445.52

8 Conclusions

This paper describes several schemes that we used to design an efficient multicast file transfer tool, FCAST, on top of ALC. We show how the Application Level Framing (ALF) paradigm can be used in this context to reduce the memory requirements at a receiver and to hide computation behind communications.

We also discuss the problem of object and symbol scheduling on the various layers, assuming there is no real-time constraint. We show that the transmission of several objects (e.g. resulting from an ALF version of FCAST) can be made several orders of magnitude more efficient when using an appropriate scheduling that randomly mixes all the symbols of all the objects.

We also introduce a new way to assign symbols to the various layers which improves reception efficiency by reducing the probability of duplication.

All of these schemes have been implemented and experiments carried out. A mathematical model of the two symbol organizations is also presented. Our results lead us to define three profiles to FCAST: one of them improves the reception speed by hiding computation behind communications; another one reduces the maximum amount of memory required by a receiver (we experienced 43 to 47% savings); and the third one is dedicated to CPU bounded hosts (we experienced a 23% speedup in that case).

Finally we introduce an efficient optimization for sessions in "push" mode where the source starts sending symbols immediately, without waiting the end of FEC calculation. The speedup and memory savings can be respectively as high as 48.4% and 96.7% in optimal cases.

Last but not least the MCL MultiCast Library implementing ALC and the FCAST tool are both distributed under an Open Source / GNU GPL license and are available on the author's home page [21].

Acknowledgments

The authors want to acknowledge Julien Labouré who contributed to the design of FCAST and MCL, Imad Aad for his help, and anonymous reviewers for their useful comments.

References

[1] S. Bhattacharyya and J. Kurose. Efficient multicast flow control using multiple multicast groups. In *IEEE INFOCOM'98*, February 1998.

[2] J. Byers, M. Frumin, G. Horn, M. Luby, M. Mitzenmacher, A. Roetter, and W. Shaver. Flid-dl: Congestion control for layered multicast. In *2nd Workshop on Networked Group Communication (NGC2000)*, November 2000.

[3] J. Byers, M. Luby, and M. Mitzenmacher. Accessing multiple mirror sites in parallel: Using tornado codes to speed up downloads. In *IEEE INFOCOM'99*, March 1999.

[4] J. Byers, M. Luby, and M. Mitzenmacher. Fine-grained layered multicast. In *IEEE INFOCOM'01*, April 2001.

[5] J. Byers, M. Luby, M. Mitzenmacher, and A. Rege. A digital fountain approach to reliable distribution of bulk data. In *ACM SIGCOMM'98*, August 1998.

[6] D. Clark and D. Tennenhouse. Architectural considerations for a new generation of protocols. In *IEEE SIGCOMM'90*, September 1990.

[7] M. Donahoo, M. Ammar, and E. Zegura. Multiple-channel multicast scheduling for scalable bulk-data transport. In *IEEE INFOCOM'99*, March 1999.

[8] J. Gemmell, E. Schooler, and J. Gray. Fcast multicast file distribution. *IEEE Network*, 14(1), January 2000.

[9] E.N. Guilbert. *Capacity of a burst-noise channel.* Bell Systems technical journal, September 1960.

[10] M. Luby, J. Gemmell, L. Vicisano, L. Rizzo, and J. Crowcroft. *Asynchronous Layered Coding (ALC) protocol instantiation*, February 2002. Work in Progress: <draft-ietf-rmt-pi-alc-06.txt>.

[11] M. Luby, J. Gemmell, L. Vicisano, L. Rizzo, J. Crowcroft, and B. Lueckenhoff. *Asynchronous Layered Coding (ALC): a massively scalable reliable multicast protocol*, July 2000. Work in Progress: <draft-ietf-rmt-pi-alc-01.txt>, now obsoleted.

[12] M. Luby, J. Gemmell, L. Vicisano, L. Rizzo, M. Handley, and J. Crowcroft. *Layered Coding Transport (LCT) building block*, February 2002. Work in Progress: <draft-ietf-rmt-bb-lct-04.txt>.

[13] M. Luby, L. Vicisano, J. Gemmell, L. Rizzo, M. Handley, and J. Crowcroft. *Forward Error Correction (FEC) building block*, February 2002. Work in Progress: <draft-ietf-rmt-bb-fec-06.txt>.

[14] M. Luby, L. Vicisano, J. Gemmell, L. Rizzo, M. Handley, and J. Crowcroft. *The use of Forward Error Correction (FEC) in reliable multicast*, February 2002. Work in Progress: <draft-ietf-rmt-info-fec-02.txt>.

[15] M. Luby, L. Vicisano, and A. Haken. *Reliable multicast transport building block: Layered Congestion Control*, November 2000. Work in Progress: <draft-ietf-rmt-bb-lcc-00.txt>.

[16] S. McCanne, V. Jacobson, and M. Vetterli. Receiver-driven layered multicast. In *ACM SIGCOMM'96*, October 1996.

[17] L. Rizzo and L. Vicisano. Effective erasure codes for reliable computer communication protocols. *ACM Computer Communication Review*, 27(2), April 1997.

[18] L. Rizzo and L. Vicisano. Reliable multicast data distribution protocol based on software fec techniques. In *Fourth IEEE Workshop on the Architecture and Implementation of High Performance Communcation Systems (HPCS'97), Greece*, June 1997.

[19] V. Roca. *Analysis of Several Scheduling Algorithms for the Heterogeneous Multicast Distribution of Data Flows Generated on the Fly*, October 1999. unpublished work in progress, http://www.inrialpes.fr/planete/people/roca/.

[20] V. Roca. *The MCL Multicast Library: Concepts, Architecture and Use*, March 2001. Work in Progress, http://www.inrialpes.fr/planete/people/roca/.

[21] V. Roca and J. Laboure. *The MCL library: an implementation of the ALC/LCT protocols for scalable multicast distribution*. http://www.inrialpes.fr/planete/people/roca/mcl/.

[22] L. Vicisano. Notes on a cumulative layered organisation of data packets across multiple streams with different rates. Research Note Note RN/98/25, University College London (UCL), May 1998.

[23] L. Vicisano, L. Rizzo, and J. Crowcroft. Tcp-like congestion control for layered multicast data transfer. In *IEEE INFOCOM'98*, February 1998.

A A Simple Algorithm for Pseudo-Random Permutations

```
// create a partially random permutation of
// (1..n)
// the probability for an element not to be
// permuted is controlled by parameter l
// (i.e. permute one element out of l).
// NB: calling this function with l==1
// creates a fully random permutation.
void
pseudorandom_permutation
              (int *a,    // in/out array
               int n,     // #elt in array
               int l)
{
        int     i, j, temp;

        for (i = 0; i < n; i++)
                a[i] = i;
        for (i = 0; i < n; i += l) {
                j = random() % n;
                temp = a[i];
                a[i] = a[j];
                a[j] = temp;
        }
}
```

Unité de recherche INRIA Rhône-Alpes
655, avenue de l'Europe - 38330 Montbonnot-St-Martin (France)

Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399