Design of a Multicast File Transfer Tool on Top of ALC

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

Benoit Mordelet
Activia Networks, Sophia-Antipolis, FRANCE
http://www.activia.net/

Abstract

This work describes several techniques that we used to design a multicast file transfer tool on top of the Asynchronous Layered Coding protocol proposed by the RMT IETF WG. Their goal is to improve the overall performances of that tool. More specifically we analyze several object and symbol ordering schemes that improve transmission efficiency and we see how the Application Level Framing paradigm can help to reduce memory requirements and enable processing to be hidden behind communications.

1 Introduction

1.1 Motivations for Multi-Rate/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 [10][16]. 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.

The same approach can 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 but assumes reliable transmissions.

1.2 Quick Introduction to ALC

The ALC (Asynchronous Layered Coding) protocol proposed by the RMT IETF WG provides a general framework for the reliable transmission of files. It uses the LCT (Layered Coding Transport) building block [7], a Layered Congestion Control building block and a FEC (Forward Error Correction) building block [8].

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. Finally the FEC codec adds a certain number of redundant *FEC symbols*.

Several transmission modes are possible [7]. In this work we only consider transmissions in *push mode* (all the receivers must be ready before the transmission starts (synchronous start)), and in *on-demand mode* (data is sent continuously in a loop, and receivers can arrive at any time (asynchronous start), download the file and leave).

Because of reliability constraints using FEC is mandatory. [9] identifies three classes of FEC codes and choosing one of them has many consequences on efficiency [2], on the maximum number of FEC symbols generated, and on the coding/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 [11].

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] [4] [15]. 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 that, efficiency is high (e.g. in [1] the whole file can be received without any duplication). Yet several problems like channel desynchronization or start delay significantly reduce this efficiency [4]. Finally using a congestion control protocol can compromize this efficiency as layers are added and removed dynamically.

Another class of cumulative layered organizations consists in sending data and FEC packets in a *fully random order* on the various layers [2]. 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). Our work follows this random organization principle.

Finally [5] 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 5.1).

3 The Various Meanings of Efficiency

Efficiency has various meanings when applied to a file transfer tool based on ALC.

- *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.

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

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

- *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.

- *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). [5]

- *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.

4 Our Proposals

4.1 ALF Applied to Multicast File Transfer

The Application Level Framing (ALF) paradigm [3] 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 a receiver, no matter the order in which it is received. 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 $b \geq 1$ blocks. 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 [5], 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 (e.g. with CPU bounded hosts).

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. during a recursive directory transmission).

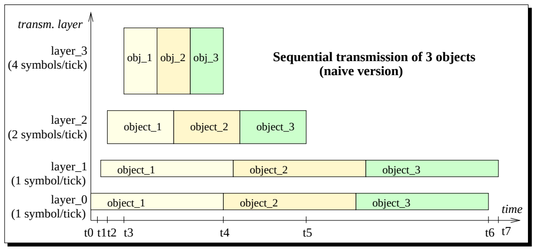


Figure 1: Multiple objects - sequential object order (m/seq).

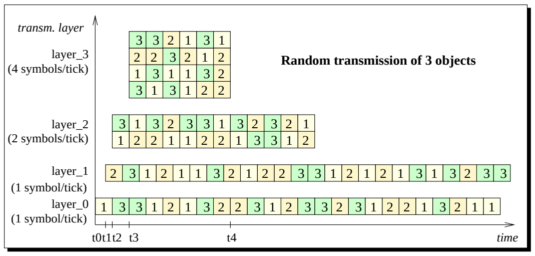


Figure 2: Multiple objects - random symbol order (m/rand) (figures identify the object to which each symbol belongs).

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 higher layers (a host receiving all four layers will get symbols of object 3 only from layer 3). This is confirmed by the experimental results of section 5.3.

4.2.2 Sending Symbols in 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: $Pr_{not\_perm}(m/p\_rand) \geq \frac{1}{total\ nb\ of\ symbols}$

4.3 What Symbols to Send in Each Layer?

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 [11]. With $k = 32$ (as suggested in [5]), there can be at most $n - k = 224$ 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 $3 * k = 96$. A symbol is by default 512 bytes long to avoid IP fragmentation 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 the *same* data symbols $\{0..k-1\}$ and FEC symbols $\{k..n-1\}$ are sent in each layer. 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".

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). 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 a receiver experiences no packet duplication *within* each layer cycle (other sources of duplication are still possible though, e.g. when receiving additional packets of an already completed block). 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.

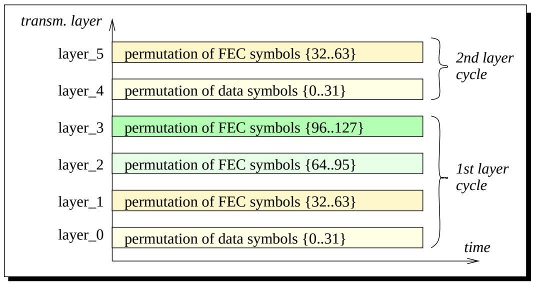

transm. layer
layer_5
permutation of FEC symbols {32..63}
layer_4
permutation of data symbols {0..31}
2nd layer cycle
layer_3
permutation of FEC symbols {96..127}
layer_2
permutation of FEC symbols {64..95}
layer_1
permutation of FEC symbols {32..63}
layer_0
permutation of data symbols {0..31}
1st layer cycle
time

Figure 3: The symbol organization 2, k symbols per layer, single block, k=32, n=128.

5 Experimental Results

5.1 The MCL Library and FCAST Application

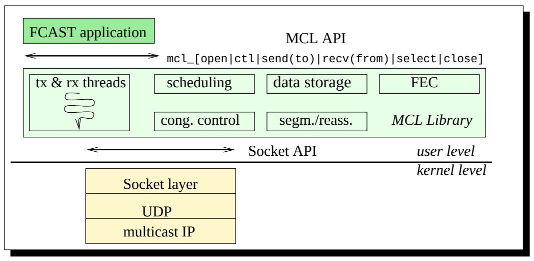
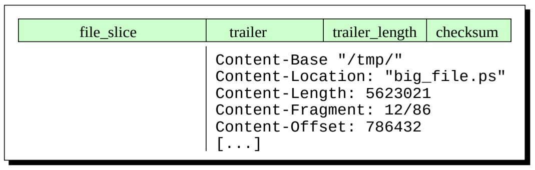


Figure 4: The MCL library and the FCAST application.



```
Content-Base "/tmp/"
Content-Location: "big_file.ps"
Content-Length: 5623021
Content-Fragment: 12/86
Content-Offset: 786432
[...]
```

Figure 5: The FCAST encapsulation format of the 12th slice of file /tmp/big_file.ps.

We implemented the ALC, LCT and RLC protocols within a library, MCL (MultiCast Library) [12], built on top of UDP/multicast IPv4 (figure 4). We also implemented FCAST on top of MCL, a recursive multicast file transfer tool inspired from [6]. 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 them autonomous (figure 5). 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. a 0.2% overhead with 64 kilobyte fragments).

5.2 Tests Methodology

Table 1: Test matrix showing the various combinations.

scenario description	name	org_1	org_2
tx. as a single object	single	yes	yes
split in mult. objects, 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 imm. delivery upon rx	m/rand	yes	yes

Table 1 summarizes the tests performed, showing the various combinations between the transmission scenario and symbol organization. We evaluate several performance metrics: (1) the *end of reception time of each object*; (2) 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), and (3) the *maximum memory required by a receiver*.

5.3 Comparison of the Various Transmission Schemes

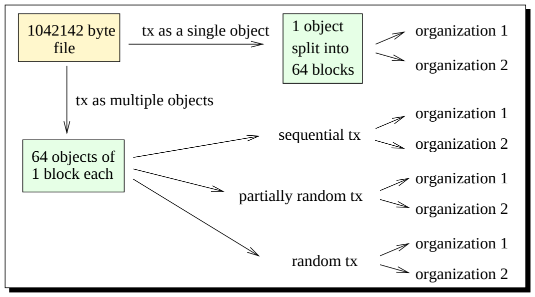


Figure 6: Experiments performed (∼1 MB file).

This section compares the various transmission schemes of section 4 during the transmission of a single 1042142 byte file (figure 6). In these tests, the sending rate is voluntarily low. 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.

5.3.1 Loss-Free and Lossy Transmissions

We first assume that *no loss occurs* in order to have a fair comparison in an optimal situation. Because of size limitations, this paper does not include the figures (see [14]).

We then introduce random bursty losses according to a Guilbert loss model: loss probability when previous symbol is received $= p_{ok} = 0.01$, and loss probability when previous symbol is lost $= p_{bad} = 0.75$. The stationary probability for a symbol to be lost is: $\frac{p_{ok}}{1-p_{bad}+p_{ok}} = 0.0385$; the average number of consecutive losses is: $\frac{1}{1-p_{bad}} = 4\ symbols$; and the average number of consecutive received symbols: $\frac{1}{p_{ok}} = 100\ symbols$.

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 min/average/max values (figures 7 to 9).

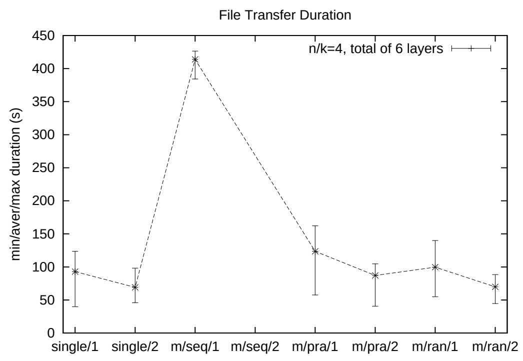


Figure 7: Transfer duration with various transmission schemes (bursty random 1%/75% losses, ~1 MB file).

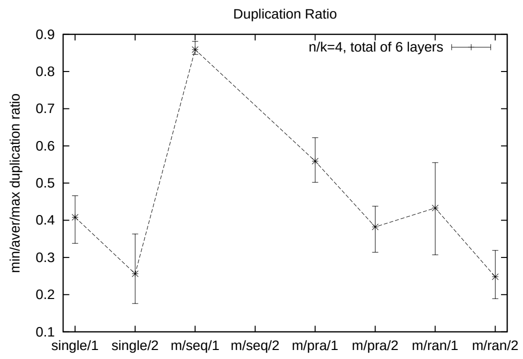


Figure 8: Duplication ratio with various transmission schemes (bursty random 1%/75% losses, ~1 MB file).

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).

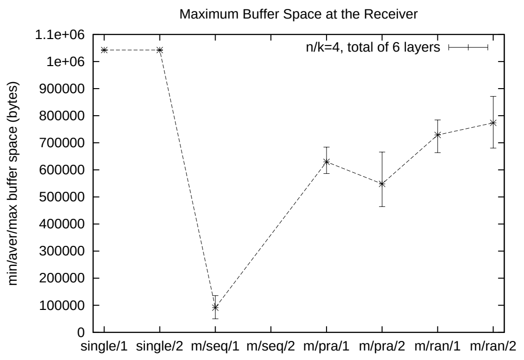


Figure 9: Maximum buffer space at the receiver with various transmission schemes (bursty random 1%/75% losses, ~1 MB file).

5.3.2 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 [14]. 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.

5.4 Application to FCAST

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 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 (to define the object/symbol ordering) and the receiver (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 to create three sessions, one per FCAST profile, and to let each receiver choose the most appropriate one.

Because of size limitations, this paper does not include the experiments carried out to assess the efficiency of the FCAST profiles. The results are rather positive yet. The interested reader is invited to see [14].

6 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.

All of these schemes have been implemented and experiments carried out. 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; and the third one is dedicated to CPU bounded hosts.

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 [13].

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

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

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

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

[6] 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.

[7] 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>.

[8] 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>.

[9] 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>.

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

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

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

[13] 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/.

[14] V. Roca and B. Mordelet. Improving the efficiency of a multicast file transfer tool based on alc. Research Report 4411, INRIA, March 2002.

[15] 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.

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