Analysis of Several Scheduling Algorithms for the Heterogeneous Multicast Distribution of Data Flows Generated on the Fly

Vincent Roca

Université Pierre et Marie Curie
LIP6 - CNRS, thème Réseaux et Performances
8, rue capitaine Scott; 75015 Paris; FRANCE
phone: (+33) 1.44.27.75.14; fax: (+33) 1.44.27.87.32
vincent.roca@lip6.fr; http://www-rp.lip6.fr/~roca
Work in Progress, Version 3.4 - September 28th, 1999

Abstract. This paper addresses the multicast distribution of data flows generated on the fly to a set of heterogeneous receivers, a common situation with collaborative work applications. It first discusses the existing packet scheduling algorithms that support this kind of data flow: DSS, a straightforward scheduling scheme where data is sent over separate multicast groups at different rates; GMMG, a cumulative scheduling scheme we propose and that tries to reduce packet duplications. Then detailed performance evaluations carried out with our MCL MultiCast Library show the benefits and limitations of each algorithm under various conditions. Finally an intermediate solution that mixes these algorithms is introduced and evaluated. These experiments show that significant gains are possible, in particular with the hybrid approach, when used correctly.

1 Introduction

1.1 Managing the Heterogeneity in Multicast Transmissions

Addressing the problem of multicast transmissions to a set of heterogeneous receivers is of the utmost importance. Indeed a multicast group may be composed of high-end receivers (workstations) attached to a high performance network (high-speed LAN), whereas other ones have limited processing power (PDA) or network access (low-speed modem, GSM modem). Several solutions are possible to address this heterogeneity:

- filtering techniques
- packet scheduling techniques
- transcoding techniques

The first class of solutions relies on *filtering* [7] [14]. These techniques discard packets from flows whose bandwidth exceed their fair share. The TUF proposal [7] adds a tag to each packet sent to identify the "priority level" (or drop precedence). Adding information within the packet enables the sender to keep some control on the discard process. Besides no state is kept in routers which warrants a good scalability. [14] also advocates the use of filtering within routers but without any involvement by the source and the receivers. Here per-flow state must be kept in each router. Although promising, both approaches require extensions to multicast routers (or at least some key routers) for congestion reports ([14] only) and filtering. Therefore they cannot be deployed immediately and other solutions must be used in the meantime.

The second class of solutions (in which we are particularly interested) rely on *packet scheduling within several transmission layers* (see references below). Each layer can be associated to a different multicast group and the filtering is performed by the join/leave mechanism of multicast IP. This approach is thus receiver oriented and has the advantage of being immediately deployed[1].

Finally, *transcoding* within some routers or proxy to dynamically adapt the data flow to the receiver features (e.g. to reduce an image complexity) can also be used [11][27]. But these solutions are CPU intensive, can only be used with a limited number of data flows and require the availability of adequate routers or proxies.

Hereafter we will only consider the packet scheduling approach.

1.2 Classification of Scheduling Techniques from a Data-Flow Point of View

An important way to classify these techniques consists in defining the application needs they address:

- Does the receiving application need to receive all the data (e.g. file transfer) or is a subset of the data stream sufficient (e.g. multi-quality video streams)?
- Is the data stream known in advance (e.g. video on demand) or generated on the fly (e.g. an application sharing environment)?
- In the latter case is the data stream continuous (e.g. a CBR, Constant Bit Rate, video stream) or bursty (e.g. white board)?

Many scheduling techniques only address the problem of the scalable bulk data transfer of a file ([3] [6] [9] [23]). In that case the prior complete knowledge of data greatly simplifies the scheduling algorithm.

The case of CBR data flows is always studied through video-conference applications. Using a hierarchical data encoding [4] provides an easy way to do packet scheduling, each transmission level corresponding to an increased image quality.

But as far as we know no previous work studies the case of applications generating bursty data streams on the fly. This is the goal of the current paper and of a companion one ([21]) that introduced the GMMG (Generalized Multiple Multicast Group) scheduling. Many collaborative work applications are concerned. A white board (e.g. `wb` [25], `mb` [15]) or an application sharing environment (e.g. doing X11 message multiplexing like `XTV` [28]) are two examples. The amount of data exchanged can be rather low during a certain span of time and then, because of a participant action (e.g. he launches a new X Windows application under `XTV`'s control), the tool generates on the fly large amounts of data.

[1] Using a single multicast group and relying on router filtering (when available) is also possible with a cumulative scheduling. In that case each layer must be regarded as a "priority level" and the packets be marked accordingly.

1.3 Fast Discussion of Related Aspects

Many other properties can be provided in addition to the heterogeneity support. Reliability is a common requirement. Two classes of solutions exist: the first one rely on ACK/NAK feedbacks ([10] and many others). But when heterogeneous receivers are considered, FEC (Forward Error Correction) ([19] [20] [12] and many others) is preferred. It enables each receiver to recover from losses independently to one another. Note that a hybrid solution is also possible ([13]) either to improve the reliability level of FEC or improve the scalability of ACK/NAK schemes. In particular a hybrid FEC/NAK is well suited to the scheduling schemes considered in this paper.

Congestion control is also essential. We have already mentioned the join/leave mechanism of multicast IP. [16] describes the RLM (Receiver-driven Layered Multicast) protocol that enables receivers to find the fixed (this is a premise) but unknown available rate. [24] adds refinements to this protocol to react to congestions. Yet this reaction may take time. Router-based filtering seems promising from this point of view.

Finally the management of *late arrivals* (e.g. a new user joins an existing white-board session) or *asynchronous starts* (e.g. a new user contacts a video-on-demand server) is often required. This is easily supported by the scheduling approaches for data flows known in advance ([9] [23]) thanks to cyclic transmissions, but this is no longer possible with data flows generated on the fly. In that case late-arrival is managed at application level (e.g. the white-board session master sends the current state to the newcomer).

These three aspects (reliability, congestion control and late-arrival) are out of the scope of this paper even if they are closely related to the scheduling algorithm.

The remainder of the paper is organized as follows: next section discusses the transmission of data flows generated on the fly. It proposes a description of the data flow and introduces some scheduling algorithms for their distribution. Section 3 presents the results of experimental tests carried out. At the light of these results, section 4 proposes a hybrid solution and discusses its benefits.

2 Managing Data Flows Generated on the fly

As seen in the introduction, this kind of data flow has received little attention (except for the particular case of video stream). Therefore we first briefly explain how to handle them.

2.1 The General Processing Model

The data processing line is composed of three key elements:

The application: This latter submits data as a collection of messages (called ADUs, or Application Data Units in this paper).

The multicast control layer: This is the core of the system and it is usually implemented as a library (section 3.1). The scheduling process consists in *segmenting* each ADU into several packets and *scheduling* these packets appropriately on the available transmission layers. Each layer is then served *periodically* and a given number of packets is sent at each tick. In section 3 we use these tick as a virtual time scale.

The underlying communication stack: the multicast control layer relies on the UDP/multicast IP protocol stack for the transmission of its packets.

The service provided by the multicast control layer is the multicast distribution of *bursty* data flows generated *on the fly*. It also assumes that each receiver needs to *get a copy of all the data* (as opposed to a solution based on a multi-definition data encoding [4]). This kind of data flow is typical of a white-board or an application sharing environment. On the opposite, it is *not suited* to the transmission of streaming media (like video).

The delivery *preserves ADU boundaries* (as with UDP). We try to avoid as much as possible IP fragmentation which is considered inefficient [18]. Therefore each packet has a small size (512 byte payload or less).

Finally, even if we use the term of ADU, the service provided is not necessarily ALF compliant [5]. Depending on the application, a receiver can ask either to receive ADUs in order (which may delay their delivery) or in arrival order. Besides ALF suggests that the unit of control (ADU) be equal to the unit of transmission (datagram). But small ADUs can also prevent an efficient use of scheduling algorithms (section 3.6).

2.2 The DSS scheduling algorithm

With the DSS (Destination Set Splitting, also called Simulcast) algorithm the whole data is sent over separate multicast groups at different rates ([1] [2]). A receiver chooses *the* appropriate group according to its capabilities. If this algorithm is straightforward, unfortunately it creates a high load on the network close to the sender since each packet is sent as many times as there are layers (this is not a cumulative scheme).

2.3 The GMMG scheduling algorithm

The GMMG (Generalized Multiple Multicast Group) algorithm has been designed to handle bursty flows generated on the fly. [21] introduced this technique using an analytical approach. In this paper we use a new algorithmic approach. This algorithm is both simpler and more general since it accommodates *any scale of transmission rates*[2].

It uses a transmission matrix (figure 1). Each cell (or transmission slot) contains the packet sequence number that will be sent at a given time (column) on a given

[2] E.g. one can decide to send 2 packets on layer 0 at each time tick, 5 on layer 1, and 20 on layer 2. Figure 1 uses an exponential scale and 1, 1, 2, and 4 packets are respectively sent each time.

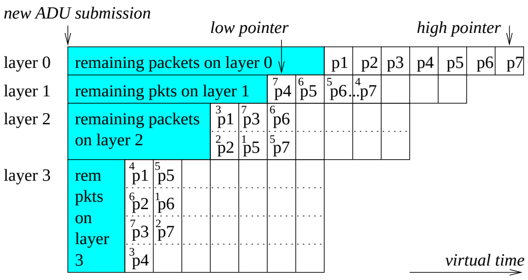


Fig. 1. GMMG scheduling in presence of remaining packets (ADU segmented into 7 packets, 4 layers).

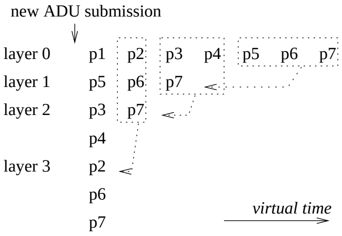


Fig. 2. GMMG scheduling of an isolated ADU (ADU segmented into 7 packets, 4 layers).

layer (row). The ADU of the example has been split into 7 packets called `p1` to `p7`. Because in the general case there can be some packets from previous ADUs waiting to be sent, the rows are logically shifted to the right as shown in the figure. Then the algorithm fills in this matrix. During the first step it copies packet numbers from pointer "high" to pointer "low" (small numbers on the top of each cell). Then it decrements "high", increments "low" and starts again until both pointers are in the same column. Because the packet sequence numbers copied to this row may no longer be increasing, the second step consists in reodering them (large numbers). Then the algorithm starts all over again for the next layer. At the end, the cells corresponding to unused transmission slots will remain empty.

From figure 1(a) we see that layers 1 and 2 take advantage of transmissions on previous layers. This is no longer the case on layer 3 because of the packets waiting to be sent. Therefore all the packets are sent on layer 3 thereby creating duplications.

Of course this scheduling algorithm *performs the best with isolated ADUs* (i.e. when there are no waiting packets). This is illustrated in figure 2. In that case a total of 16 packets are sent over all layers instead of 24 in figure 1. Note that GMMG is very close to the MMG scheduling ([3]) when dealing with isolated ADUs. Therefore MMG will therefore be regarded as a *particular case of GMMG* in section 3.

3 Experimental Results

This section discusses the experiments carried out with DSS and GMMG. For comparison purposes MMG is also considered. We first introduce the experimental setup, then we analyze the algorithm behaviors under various conditions, finally we discuss the results.

3.1 A Versatile Multicast Library

These experiments have been carried out with the MCL MultiCast Library ([22]) that we designed following the Building Block approach ([26]). This library has two goals: first of all it enables the experimentation of new mechanisms and protocols (e.g. for packet scheduling).

Secondly it enables application developers to use very easily advanced multicast services. To achieve this goal, the upper API (Application Programming Interface) has been kept as simple as possible and is very close to the Socket API. A consequence of this choice is that the application and the MCL library are *completely dissociated*. On the opposite using the ILP (Integrated Layer Processing [5]) paradigm would enable several optimizations at the expense of a more complex design and API. This issue is discussed in section 3.6.

3.2 Parameters of a bursty data flow

Several parameters of the data flow submitted by the application will impact the scheduling algorithm (figure 3):

ADU size: Layered transmission schemes (like GMMG) rely on the availability of large data chunks that can be segmented and scheduled appropriately. This is not a problem with a file transfer application since the ADU can be the whole file. This is quite different in our case since the application may use only small ADUs (e.g. whose size is close or even inferior to the MTU).

Average ADU submission rate: This submission rate must be compared to the transmission rate of layer 0 (the bottleneck). Indeed, the MCL library can be regarded as a FIFO which is filled by the application and emptied by layer 0. Having a submission rate higher than the transmission rate creates packet accumulations. At some time the FIFO turns full and the application is blocked. The same kind of phenomenon takes place if a burst has a size larger than the FIFO (i.e. local buffering) capabilities.

Idle period: This is the application inactivity period (no ADU submission) between two consecutive bursts. Saying that the average submission rate is lower than the layer 0's transmission rate means that this idle period is long enough so that all the packets accumulated during a burst are sent when the next burst starts.

Burst size: This is the number of bytes that compose a burst of ADUs.

3.3 Testing conditions

In the following tests we study only the algorithm behaviors in front of a *single burst*. This is sufficient if we assume a periodic and bursty data flow whose submission rate

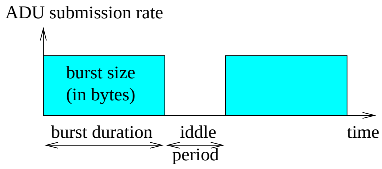


Fig. 3. Relationships of the parameters during an ADU submission burst.

is inferior to the transmission rate of layer 0. The maximum packet payload size is set to 512 bytes. For a given submission rate (expressed in packets/tick rather than ADUs/tick) we use several ADU sizes, starting from 256 bytes (half a full-sized packet) up to 65536 bytes (segmented into 128 packets). The ADU submission rate is calculated so that its value expressed in *packets/tick is constant* for all ADU sizes[3]. Five rates are used:

- 2 packets/tick to simulate an application generating a low-rate stream,
- 4, 8 and 16 packets/tick for an intermediate rate, and
- an infinite submission rate where all the ADUs are submitted immediately.

We also use several burst sizes, from 65536 bytes (128 full-sized packets) up to 1048576 bytes (2048 full-sized packets). To make these tests reflect the scheduling algorithms only, no packet is lost. Five layers are used and we only consider the case of a host receiving all of them. Because of space limitations, we do not consider the impacts of the number of layers received on the algorithm behavior in this paper.

3.4 Study of the End of Reception Times

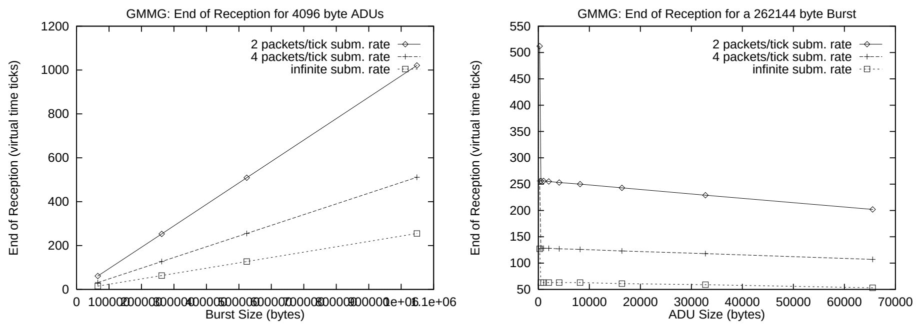
In this section we study the curve:

$$end\_of\_reception = f(submission\_rate, ADU\_size, burst\_size)$$

Case of the GMMG Scheduling: In the GMMG tests, we use five layers with an exponential transmission rate distribution: 1, 1, 2, 4 and 8 packets/tick are sent on each layer (for a cumulative rate of 1, 2, 4, 8 and 16 packets/tick according to the number of layers received). Figure 4(a) shows the end of reception time when the application uses 4096 byte ADUs, and figure 4(b) in case of a burst of 262144 bytes.

The more data is available (large ADUs and high submission rates), the better. Yet an analysis (not shown) of the behavior of each layer proves that the benefits come more and more from the higher layers because of packet accumulation problems. It means that this is also *to the detriment of the duplication ratio* (section 3.5). These figures also show that the algorithm performs better when the ADU size is a multiple of a full-sized packet (otherwise some packets are partially filled). Finally

[3] E.g. for a 2 packets/tick submission rate using 2048 byte ADUs (segmented into 4 packets), a single ADU is submitted every other time tick



(a) As a function of the burst size and submission rate, 4096 byte ADUs.

(b) As a function of the ADU size and submission rate, 262144 byte burst.

Fig. 4. End of Reception with GMMG.

the end of reception time is more sensitive to the burst size and submission rate than to the ADU size (if multiple of the full-sized packet).

Case of the DSS Scheduling: The experimental parameters are the same as before. Here 1, 2, 4, 8 and 16 packets/tick are respectively sent on each of the five layers and figure 5 gives the end or reception times for a 262144 byte burst.

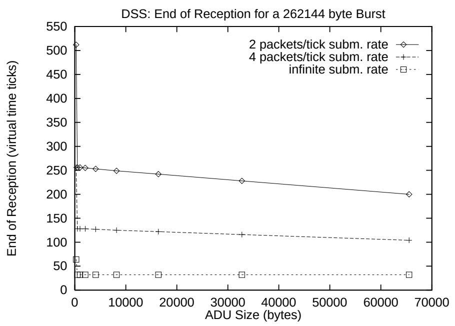


Fig. 5. End of reception as a function of the ADU size and submission rate with DSS, 262144 byte burst.

The end of reception times are (almost) the *same as those of the GMMG scheduling* up to a submission rate of 16 packets/tick (not shown). *Then* for an infinite submission rate that the reception time is roughly *reduced by half*. Indeed, the accumulation problems encountered by GMMG make it behave like DSS (i.e. each packet is sent on each layer) but with a maximum rate of 2^4 packets/tick on layer 5 instead of 2^5. This is less true with large ADUs. In this case packets are better scheduled and the accumulation phenomenon is less important.

Case of the MMG Scheduling: The MMG scheduling requires that data is known in advance. Therefore the "burst" consists of a single ADU. Table 1 shows that the reception times are close to that of DSS with an infinite submission rate. If this is rather good, this result is in fact the *asymptotic behavior of GMMG* when the ADU size and submission rate both increase.

Table 1. End of reception as a function of the burst size with MMG.

Burst size (bytes)	65536	262144	524288	1048576
End of rx (virtual time ticks)	8	32	64	128

3.5 Packet Duplication

This section studies the curve:

$$DupR = f(submission\_rate, ADU\_size, burst\_size)$$

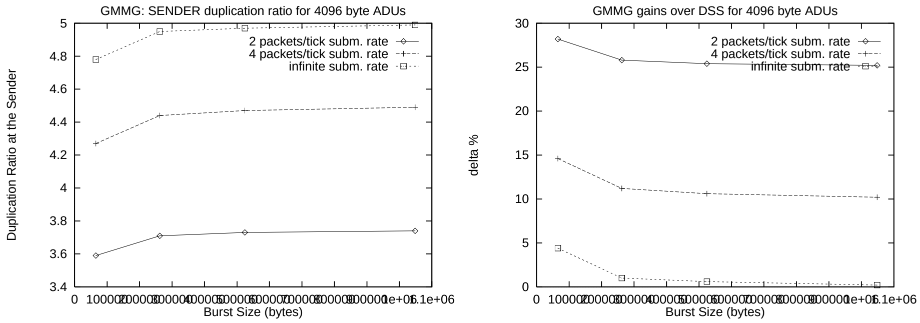
where $DupR$ is the packet duplication ratio and is defined by:

$$DupR = \frac{total\_number\_of\_packets\_received}{number\_of\_non\_duplicated\_packets}$$

For instance a 3.7 ratio means that 3.7 packets are received for each valuable packet, among which 2.7 are duplicated. This ratio can be measured either at the sender ($TxDupR$) or at the receiver(s) ($RxDupR$). Sender-based measures highlight the load this flow creates on the first few networks and (multicast) routers. On the contrary receiver-based measures reflect the load on the networks and routers close to a given receiver.

Packet Duplication at the Sender ($TxDupR$): The $TxDupR$ for GMMG is given by figures 6 and 7. Under the same conditions, DSS sends each packet 5 times (one copy for each of the 5 layers).

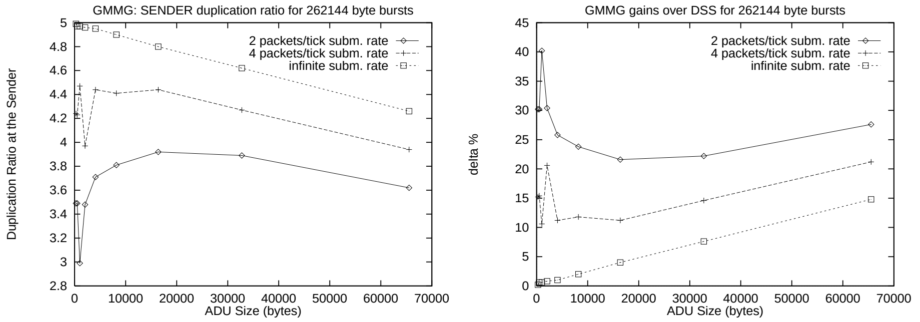
These tests show that *using GMMG rather than DSS can lead to significant savings at the sender*, from 0 to 40% depending on the exact conditions. In case of MMG, the $TxDupR$ ratio is constant for all burst sizes and equal to 3.0, which is a 40% improvement over DSS. The GMMG scheduling achieves this excellent ratio only with a low ADU submission rate since there are no (few?) packet accumulation then. Another result is that with GMMG the $TxDupR$ ratio is more sensitive to the ADU size and submission rate than to the burst size.



(a) Sender duplication ratio

(b) GMMG gains over DSS

Fig. 6. TxDupR with GMMG as a function of the burst size and submission rate, 4096 byte ADUs.



(a) Sender duplication ratio

(b) GMMG gains over DSS

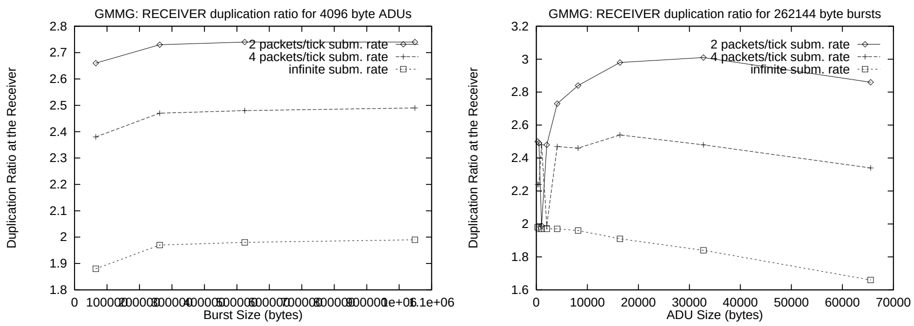
Fig. 7. TxDupR with GMMG as a function of the ADU size and submission rate, 262144 byte bursts.

Packet Duplication at the Receiver ($RxDupR$): Two situations must be considered:

– the burst belongs to a longer data flow. In that case $RxDupR$ at the receiver is equal to $TxDupR$ of section 3.5 with GMMG.
– nothing else needs to be received after this burst. In that case the receiver unsubscribes to all layers as soon as it has received all the required packets. Then $RxDupR$ is much lower.

Let us suppose we are in this second case. Then $RxDupR$ is given by figure 8 for GMMG. They show that unlike $TxDupR$, here the faster data is submitted the better. The reason is that it enables the receiver to unsubscribe faster and then avoid the reception of many duplicated packets.

On the contrary with DSS $RxDupR$ is always equal to 1.0 (no duplication) since a receiver gets a single data flow. With MMG too $RxDupR$ is constant and equal to 1.0 because the prior knowledge of data to send enables an "optimal" scheduling.



(a) As a function of the burst size and submission rate, 4096 byte ADUs

(b) As a function of the ADU size and submission rate, 262144 byte burst

Fig. 8. $RxDupR$ with GMMG scheduling.

3.6 Discussion of the Results

Several conclusions can be drawn from these experiments. First of all the scheduling efficiency greatly depends on the exact data flow generated by the application: ADU size, submission rate, burst size. It also depends on the receiver topology. Therefore no single conclusion can be drawn, only tendencies.

Secondly, DSS is more appropriate than GMMG in case of a high ADU submission rate (compared to layer 0 rate). Because of packet accumulation problems, GMMG losses its cumulative scheduling asset, hence higher duplication rates.

Thirdly DSS creates a heavier load at the sender ($TxDupR$) than GMMG. With low or medium submission rates, GMMG can lead to significant packet savings compared to DSS (up to 40%). At the opposite GMMG creates a higher load at the receiver ($RxDupR$). It means that GMMG spreads the load more evenly on the network than DSS which creates a single hot spot (around the sender). Note that these experiments have only considered the case of a high-end receiver. More limited hosts (e.g. a PDA) only receive layer 0 and do not experience any duplication!

Fourthly an algorithm limited to applications where the data streams is known in advance largely outperforms those for the general case of flows generated on the fly. This is not surprising because of the strong hypothesis required.

Finally, layered scheduling schemes like GMMG rely on the availability of large ADUs that can be segmented and scheduled appropriately. Three solutions are possible in case the application generates only small ADUs:

- modify the application in order to use larger ADUs,
- use smaller packet sizes, or
- accumulate ADUs and treat them as a “super-ADU”.

The first solution is efficient but is also contrary to our goal of easy integration of the MCL library. The second approach may be sufficient in many situations. The drawback of using smaller packets is the increased load on the router (more packets to forward). With the third approach, ADUs are accumulated in an incoming ADU list instead of being processed immediately. Every N virtual time ticks all the ADUs currently on the list are aggregated in a "super-ADU". This latter is then processed as would a standard ADU except that each receiver dismantles this super-ADU once it is completely received. The choice of a value to N is of course a balance between the efficiency and real-time considerations.

4 A Hybrid DSS/GMMG Scheduling

Section 3 has shown the limitations of the DSS and GMMG solutions. In this section we propose a hybrid solution and discuss its field of application.

4.1 Rational

Let us suppose that the group is composed of very slow receivers (e.g. using a 33kbps modem) and of high-end receivers (e.g. using a 100Mbps LAN). In that case the transmission rate spectrum is too large for a single GMMG scheduling to work efficiently (packets would accumulate). But GMMG may be interesting if two separated schedulings are used, one for the slow rates and the other one for the high rates.

More generally whenever the scale of transmission rates includes large gaps it may be interesting to *use several dissociated sets of transmission layers*. For each set one chooses either DSS (in case of a single layer) or GMMG (in case of several layers with close rates) as shown in figure 9.

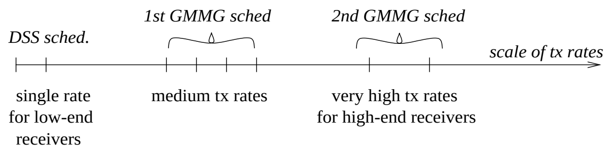


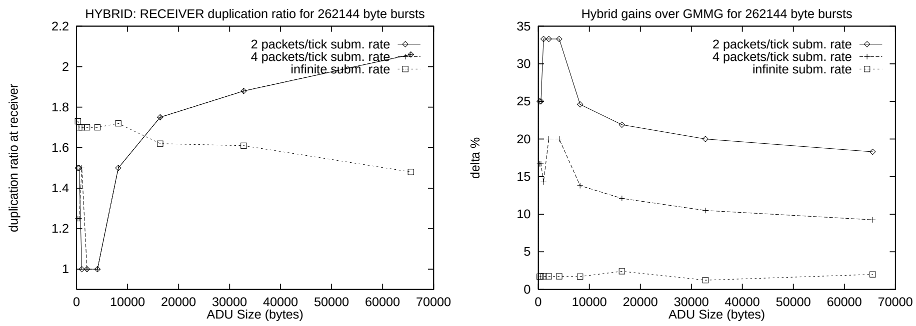
Fig. 9. Example of use of mixed scheduling in case of a broad range of transmission rates.

4.2 Experimental Results

In this experiment we use four layers with the following rates: 32kbps (in case of a modem access), and 256, 512 and 1024 kbps (in case of a LAN access)[4]. The goal is to compare three possible solutions: (1) pure DSS, (2) a single GMMG scheduling over the four layers, and (3) a hybrid scheme (DSS for layer 0 plus GMMG for the upper three layers).

Let us compare the GMMG and the hybrid schemes first. There is *no difference in the end of reception times* of a high-end receiver since in both cases the base layer

[4] I.e. 1, 8, 16 and 32 packets/tick are respectively sent on the layers.

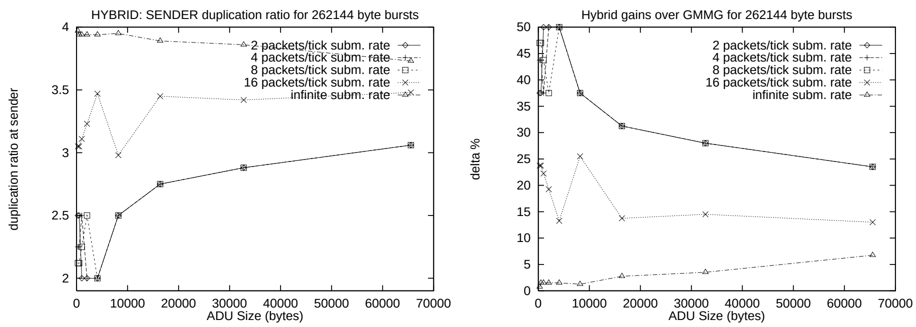


(a) Receiver duplication ratio

(b) Gains of Hybrid vs. GMMG

Fig. 10. Duplication at the receiver with the Hybrid scheduling, 262144 byte bursts.

is useless for him. There is *no difference in $TxDupR$* either since in both cases all the packets are sent on layer 0. But *the difference is the $RxDupR$ ratio.* Figure 10(a) shows that $RxDupR$ ranges between 1.0 and 2.0. For comparaison $RxDupR$ for a pure GMMG scheduling ranges between 1.5 and 2.5. This is better illustrated by figure 10(b). We see that the *gains of the hybrid scheme versus pure GMMG can be significant* with low to medium submission rates (up to 33%).



(a) Sender duplication ratio

(b) Gains of Hybrid vs. DSS

Fig. 11. Duplication at the sender with the Hybrid scheduling, 262144 byte bursts.

Now let us consider the DSS scheme (figure 11). If a pure DSS scheme still has the advantage of creating no duplication at the receiver, the $TxDupR$ ratio is constant and equal to 4.0. On the opposite $TxDupR$ for the hybrid (or pure GMMG) scheme ranges between 2.0 and 3.0 up to 8 packets/tick submission rates. Therfore the corresponding *relative improvements over DSS is very significant* (between 23% and 50%). Even with a 16 packets/tick rate the improvement never falls below 13%.

These results prove that *a hybrid scheme is an efficient way to use the various scheduling algorithms closer to their optimum.* It results in significant reduction of the number of duplicated packets, either at the sending or receiving sides.

5 Conclusions

This papers has described several algorithms for the multicast distribution of bursty data flows generated on the fly to a set of heterogeneous receivers. We have shown what are the main parameters that affect the behavior of these algorithms. These parameters have large impacts on the algorithm efficiency (i.e. end of reception time, packet duplication at the sender and receiver). According to them, several recommendations are made on the use of each algorithm. We have shown in particular that a hybrid approach can be very benefic since it makes the DSS and GMMG scheduling schemes work closer to their optimum.

This work has been made possible by the MCL MultiCast Library. It provides a basis for the experimentation of several mechanisms like (but not limited to) scheduling algorithms. The upper interface has also been made as simple as possible in order to facilitate its use by existing applications. This library is freely distributed (with full source code and documentation) in the author home page.

Acknowledgments

The author thanks Anca Dracinschi for her comments and Christophe Cavrot who contributed to the design of the library.

References

1. M. Ammar, L. Wu, "Improving the Performance of Point to Multi-Point ARQ Protocols through Destination Set Splitting", *Proceedings of IEEE INFOCOM'92, May 1992.*
2. M. Ammar, D. Towsley, "Flow control for multicast applications", *Multicast tutorial during SIGCOMM'97, September 1997.*
3. S. Bhattacharyya, J. Kurose, D. Towsley, R. Nagarajan, "Efficient multicast flow control using multiple multicast groups", *Proceedings of IEEE INFOCOM'98, February 1998.*
4. J. Bolot, T. Turletti, "Scalable feedback control for multicast video distribution in the Internet", *Proceedings SIGCOMM'94, September 1994.*
5. D. Clark, D. Tennenhouse, "Architectural considerations for a new generation of protocols", *Proc. ACM SIGCOMM'90, September 1990.*
6. A. Clerget, W. Dabbous, "Organizing data transmission for reliable multicast over satellite links", *Proceeding of 5th Conf. on Computer Communications, Africom'98 CCDC'98 Internet and Global Networking, 1998.*
7. A. Clerget, "A Tag-based UDP Multicast Flow Control Protocol", INRIA Technical Report 3728, July 1999.
8. C. Diot, W. Dabbous, J. Crowcroft, "Multipoint communication: a survey of protocols, functions, and mechanisms", *IEEE Journal on Selected Areas in Communications, Vol. 15, No. 3, April 1997.*
9. M. Donahoo, M. Ammar, E. Zegura, "Multiple-Channel Multicast scheduling for scalable bulk-data transport", *Proceedings of IEEE INFOCOM'99, March 1999.*

10. S. Floyd, V. Jacobson, S. McCanne, C. Liu, L. Zhang, "A reliable multicast framework for light-weight sessions and application level framing", *Proceedings ACM SIGCOMM'95, 1995.*
11. A. Fox, S. Gribble, Y. Chawathe, E. Brewer, "Adapting to network and client variation using infrastructural proxies: lessons and perspective", *IEEE Personal Communications, Vol. 5 No. 4, August 1998.*
12. C. Huitema, "The case for packet level FEC", *Proceedings Protocols for High Speed Networks (PfHSN'96), October 1996.*
13. M. Jung, J. Nonnenmacher, E. Biersack, "Reliable Multicast via Satellite: uni-directional vs. bi-directional communication", *Proceedings of KiVS'99, Germany, March 1999.*
14. M. Luby, L. Vicisano, T. Speakman, "Heterogeneous multicast congestion control based on router packet filtering", *Work in progress, presented at RMRG meeting, Pisa, June 1999.*
15. "The MASH project", *available at URL http://www-mash.cs.berkeley.edu/mash/*
16. S. McCanne, V. Jacobson, M. Vetterli, "Receiver-driven layered multicast", *Proceedings ACM SIGCOMM'96, October 1996.*
17. J. Nonnenmacher, E. Biersack, D. Towsley, "Parity-based loss recovery for reliable multicast transmissions", *Proceedings ACM SIGCOMM'97, September 1997 (also in IEEE Transactions on Networking, 1998).*
18. J. Mogul, S. Deering, "Path MTU discovery", *Request for Comments, RFC1191, November 1990.*
19. L. Rizzo, L. Vicisano, "Effective erasure codes for reliable computer communication protocols", *ACM Computer Communication Review, V.27, N.2, April 1997.*
20. L. Rizzo, L. Vicisano, "RMDP: a FEC-based reliable multicast protocol for wireless environments", *ACM Mobile Computing and Communication Review, V.2, N.2, April 1998.*
21. V. Roca, "Packet scheduling for heterogeneous multicast transmissions", *Protocols for High Speed Networks (PfHSN'99), August 1999.*
22. V. Roca, "A library for heterogeneous multicast distributions: concepts, architecture and use", *Work in progress (available on the author's home page), September 1999.*
23. L. Vicisano, "Notes on a cumulative layered organisation of data packets across multiple streams with different rates", *Work in progress (available on the author's home page), May 1998.*
24. L. Vicisano, L. Rizzo, J. Crowcroft, "TCP-like congestion control for layered multicast data transfer", *Proceedings of IEEE INFOCOM'98, February 1998.*
25. "wb - LBNL Whiteboard Tool", *available at URL http://www-nrg.ee.lbl.gov/wb/*
26. B. Whetten, L. Vicisano, R. Kermode, M. Handley, S. Floyd, "Reliable multicast transport building blocks for one-to-many bulk data tranfer", *Work in Progress, draft-ietf-rmt-buildingblocks-00.txt, June 1999.*
27. R. Wittmann, M. Zitterbart, "Active multicasting for heterogeneous groups", *Proceeding of 4th IFIP Broadband Communications, BC'98, April 1998.*
28. Hussein Abdel-Wahab, Mark Feit, "XTV: A Framework for Sharing X Window Clients in Remote Synchronous Collaboration", *Proceedings, IEEE TriComm '91: Communications for Distributed Applications and Systems, April 1991.*