A Dynamic Group Model : Real Time Synchronization Protocol and Buffers Management

Abderrahim. Benslimane and Abdelhafid Abouaissa
LaRI Sévenans –Université de Technologie de Belfort-Montbéliard
90010 Belfort cedex France
Email : {abder.benslimane, abdelhafid.abouaissa }@utbm.fr

Abstract

This paper proposes a hierarchical architecture of k-local groups, where k represents the number of local groups composing the distributed system G. Each local group is defined as a finite set of processes. The proposed architecture is defined by applying a dynamic grouping algorithm. Moreover, the paper presents a synchronization protocol using a Δ-causal ordering allowing real time delivery. We evaluate the performance of the architecture through simulation and compare it with that of a complete graph. The result shows that the hierarchical architecture reduces the buffer size significantly without decreasing the processing power of servers and delay penalty. The proposed architecture allows us to solve the local clocks drift by synchronizing the local clocks. The clocks synchronization is resolved by according to a time reference noted VMT (Virtual Master Time).

Keyword *:* Synchronization protocol, Hierarchical architecture, grouping management , performance evaluation, buffer management.

1. Introduction

The advance of distributed multimedia systems and the progress of communication networks technologies are largely contributed to the emergence of different types of data traffics, demanding different bandwidth [6, 9] and Quality of Services (QoS) [7, 12, 14], in the same real-time multimedia applications, such as : teleconferencing, tele-teaching, etc.. These systems require that the temporal relationships between media units composing the multimedia applications must be maintained in order to guarantee, in real-time, a continuous data transfer. However, the major problem of multimedia applications is the asynchronous transmission, that tends to disrupt the temporal relationships among the media unit in the network such as ATM.
Several papers have been presented in the multimedia synchronization domain [4, 5, 8, 10, 13, 15, 16, 18, 19], these protocols differ mainly by the assumptions they make on the communication patterns, the topological structure of the underlying network, and how fast the protocol is in sending and delivering media to be played back. However, many of these protocols do not take into account the total buffer size, mainly in a group communication system, where users need a large bandwidth, and consume a huge size of buffers. Thus, due to the total number of connections to users, delay jitter and the variation of the communication delay, the total buffer size increases. So, if the connection among users is not managed carefully, the multimedia applications will become more and more hard to handle.
In general, the total buffer size depends on the delay jitter, the synchronization mechanism, the number of connections in the group communication system, and the variation of the communication delay.
In this paper, we propose a new hierarchical architecture for the group communication system to reduce the buffer size at each user level. Our solution is based on a hierarchical structure of k-groups, where k represents the number of local groups that compose the distributed system. Within each local group, there is only one single server that plays the role of the local root of that group. The set of local servers forms a distributed virtual group, where one of them will play even the role of the Master server who manages eventually the clocks synchronization by sending periodically his virtual clock value. For synchronizing local clocks, we apply the synchronization procedure presented in [1] to compensate of local clock drift. This architecture can be applied to 1-n applications or to n-n application such as videoconference.

Several paper have been presented to solve the problem of the total buffer size. [17] proposes a hierarchical communication architecture and a mixing synchronization algorithm. However, it dose not take into account the use of buffers. So, when applied in wide-area network (WAN) environments, the additional delay added to the end-to-end (due to the mixing algorithm) might disturb the time-critical applications. [5] proposes a network architecture, *VuNet*. In VuNet, multimedia devices are taken out of workstation and are treated as general

peripherals. Workstations access the devices through the network. However, no explanation is given of how to solve the synchronization problems or how to assess the buffer size.

This paper proposes a hierarchical architecture of k-local groups, where k represents the number of local groups composing the distributed system G. Each local group is defined as a finite set of processes. The proposed architecture is defined by applying a dynamic grouping algorithm. Moreover, the paper presents a synchronization protocol using a Δ-causal ordering allowing real time delivery. We evaluate the performance of the architecture through simulation and compare it with that of a complete graph. The result shows that the hierarchical architecture reduces the buffer size significantly without decreasing the processing power of servers and delay penalty. The proposed architecture allows us to solve the local clocks drift by synchronizing the local clocks. The clocks synchronization is resolved by according to a time reference noted VMT (Virtual Master Time).

The rest of paper is structured as follows. In section 2, we propose a communication group model based on grouping several users in the same local group in order to reduce the buffer size, and to resolve the delay jitter effect. The section 3 presents the proposed synchronisation protocol that uses the real time Δ-causal ordering concept. In section 4, we compute users' and local servers' buffer sizes composed the system. We study in the section 5, architecture management in the case of where a user wants to join or leave a local group. In section 6, we evaluate the performance with simulations of the proposed protocol. Finally, concluding remarks are given in section 7.

2. Communication group architecture

2.1. Group concept

We consider a group communication system G, as a complete graph composing of n processes p_1, .., p_n that communicate by exchanging messages through FIFO channels, and message transfer delay is arbitrary but finite. There is neither common shared memory nor global clock. The communication is made without degradation and without duplication or loss of messages. In other words the underlying network is reliable and asynchronous.

Cooperation between processes is structured with the group concept. Each process has a distinguish identity. The information of identity will be used for electing local servers, master server, and also synchronizing local clocks in the system.

The group communication system is built upon a hierarchical architecture of k-local groups, where k represents the number of groups in the system G. Each group is defined as a set of processes, that forms a tree structure where the root is a member of that group, and is called the local server. We assume that the group communication system can detect process failures, and report them to the local server. Processes belonging to different local groups can exchange their messages only via their local servers. For that a virtual group is created as a collection of local servers in order to ensure the communication between all groups in the distributed system. Therefore, to guarantee the communication in the hierarchical structure, all data traffic must travel among local servers to be multicasted to all members of the group or be forwarded to the virtual group.

Definition 1. A group g_i is defined as a set of n_i processes $\{p_1, .., p_{ni}\}$ which possess distinct identities.

Definition 2. A group communication system G is a set of k-local groups noted : $G = \bigcup_{i=1}^{k} g_i$.

Definition 3. We define a virtual group g_v as a collection of local servers p_{si},(where $p_{si} \in (g_i \cap g_v)$, g_v:

$$g_v = \bigcup_{i=1}^{k} P_s$$

Definition 4. g_i and g_v (represent respectively a local and a virtual group) are called overlapped groups if there exists a process p_s (playing the role of local server), that belongs to both groups : $g_x \cap g_v = \{P_s\}$.

2.2. Distributed election algorithm

We consider a network with n processes which want to communicate with one other. In direct connection architecture, each process builds connections with other processes in the network, i.e., all processes are fully connected. Since the total number of virtual connections and the architecture given in figure 2.1 can be reduced by appropriately grouping processes. We divide the n processes into k-local groups, such that each process is a member of only one local group, as illustrated in figure 2.2. Our grouping concept puts processes within the same neighbor set called local group to allow local server, of each local group, to buffer less media units without degradation of the QoS. In this way the continuous of data transfer will not be perturbed, and the delay jitter can be compensated.
Once, the local group is known, only one process is chosen by the distributed election algorithm as local server for that group, in order to ensure real-time traffic. All local servers are also members of another group called virtual group (figure 2.2), and one of them is selected as the Master server to be the global root of the group communication system.

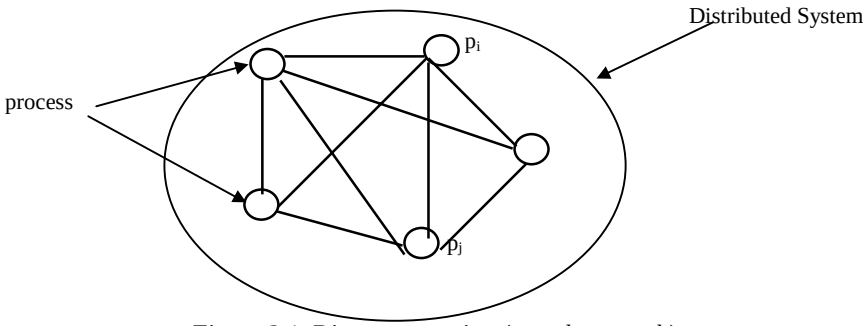

Figure 2.1. Direct connection (complete graph)

To define the so-called « neighborhood » we use some tests [11] in the first phase of distributed election algorithm. Those tests take place during the channel establishment.

i) *deterministic test*: this test consists of verifying that enough processing power is available in the process to accommodate the additional channel without impairing the guarantees given by others.
Since the packet processing time in each process is a constant parameter, the service time is the same for all channels connecting to that process. The service time is independent of the packet size because it is expressed in terms of packets.
Let :

- t_p (sec) represents the packet processing time of the process p, and
- $x_{\min,j}$ (sec) is the minimum inter-arrival time of packets to process p, and is computed on the maximum rate packet $X_{max,j}$ (packet/sec) supported by the channel j, and is defined as follows :

$$x_{\min,j} = \frac{\mathbf{1}}{X_{\max,j}}$$

Thus, if the condition in the Equation (1) is satisfied, then the channel establishment proceeds. Otherwise, a rejection message is sent to the sender :

$$\sum_{j \text{ in all channels connecting to } p} \left(t_p \Big/ x_{\min,j} \right) < 1$$

ii) *Delay bound test* is tested at every process of the channel. The delay bound is tested to ensure that no scheduler is saturated after a new channel is established. This test is performed by each process with their immediate neighbors to form its own group. Since the service time for each channel is the same as the packet processing time of the process, the delay-bound test at process p should satisfy the condition (2). If the condition is not satisfied, the channel is rejected.

$$t_p * m + t_p < \delta_{i,p} \qquad (2)$$

where :

$\delta_{i,p}$: is the delay bound of the channel i to the process p,

m : is the total number of channels already connected to p.

In order to ensure real-time data traffic, we synchronize users' local clocks. This synchronization allow us to compensate for local clock drift. We assume that each group has a local server. When each one receives a message from other groups, it broadcasts it to all processes of its group as soon as possible. Our scheme synchronizes the clocks of all users in the groups according to a time reference [8]. This value is used by all processes in the distributed system as their own local clock value. Note that each process adds a synchronization delay to the VMT value in order to have the same clock value, at a given time, with its Master server. The value of time is the value of master server clocks selected by distributed election algorithm. Our algorithm is divided in two phases :

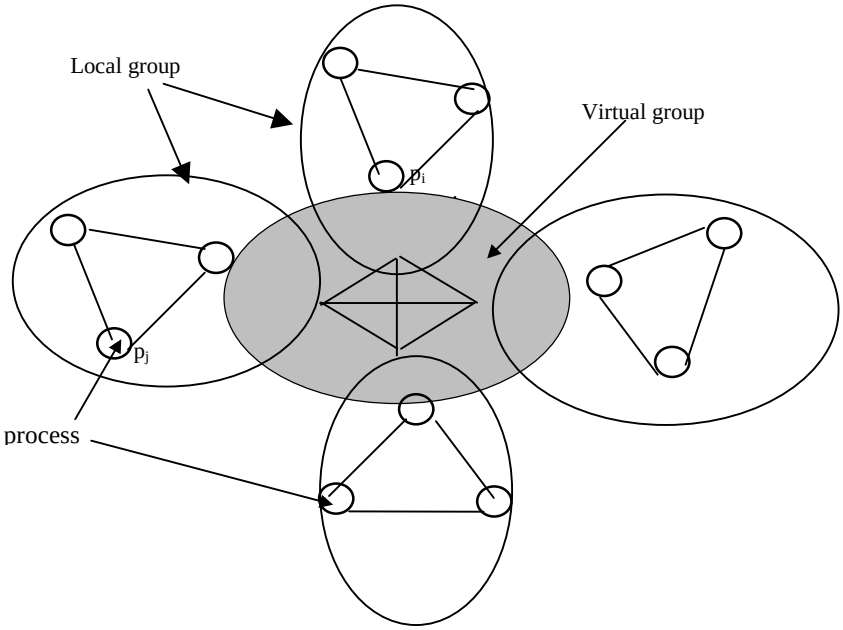

Figure 2.2 .Connection model for k- local groups

- In the first phase, all processes in the distributed system know the identities of their immediate neighbours set, and these identities are recorded in their lists in ascending order. In the first phase, each process performs the first and the second test to define its own group called the transit group (figure 2.3).

The following rules are necessary for the validity of the transit group :

- $\forall p_i : |g_{tri}| \geq 2$, *each transit group must be constituted of at last two processes*
- $\forall p_i , \forall p_j : \ p_j \in g_{tri} \Rightarrow p_i \in g_{trj}$

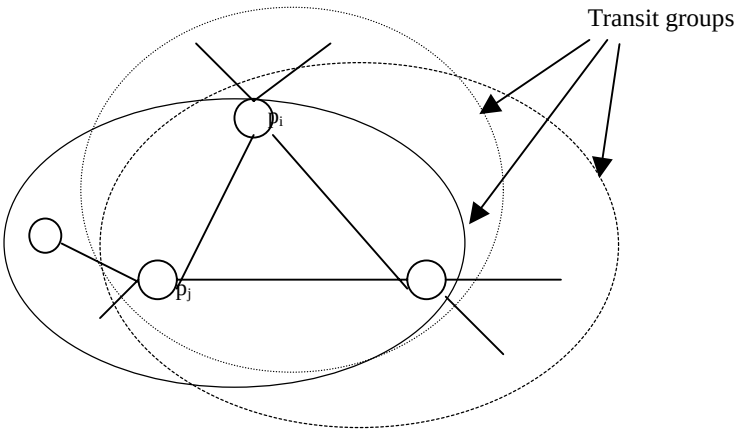

figure 2.3. transit group for each node presented in the complete graph

Once the transit groups have been defined, a mechanism must breaks the unnecessary connections to create the local group. Now, all processes have their own lists that are composed of their immediate neighborhoods set, however, many processes can belong to many transit groups. To solve this problem, we proceed as follows, each process p_i sends a Req_set(g_{tri}) message to inform the participants in its transit group of the contents of its own set g_{tri}. After recording all received sets, each process selects the maximum of processes existing simultaneously in all received sets. In the case of overlapping group, the process will belong to the group which has the lowest identity or the successor has the lowest identity if the lowest belongs to the overlapping group. So in this way a local group is specified by breaking the unnecessary links. Thus, the communication group system will be represented by k-local groups, where each process is belonging only to one local group. Once, the local group has been defined, the process generating the lowest identity is selected as local server. Once the local server p_s has been elected, it sends a «Establish(g_s)» message to confirm the connection and to build its hierarchical group model (figure 2.4). The main goal of breaking those unnecessary links is to ensure, efficiently, that the new successor of the local server belongs only to an one local group.

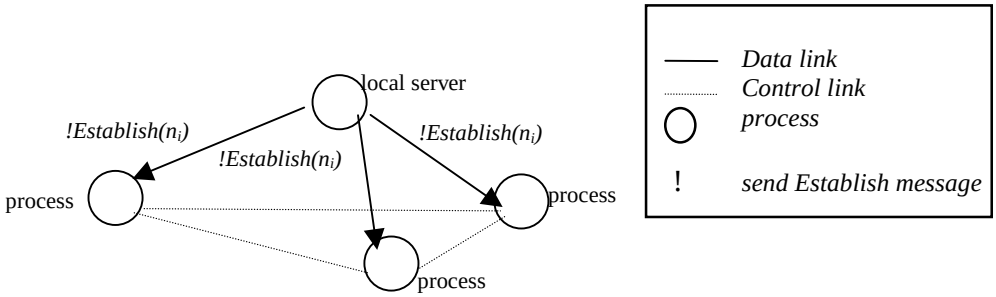

Figure 2.4. Hierarchical architecture in a local group

- In the second phase, when the local servers of the local groups have been selected, they constitute a virtual group. We suppose that the virtual group is a complete graph allowing distributed communication between the local servers. Let g_v this virtual group, then the process having the lowest identity is elected as the master server p_{mast} as illustrated in figure 2.5. Once the Master server has been elected, it sends a «Connect(VMT)» message to all local servers belonging to g_v, where VMT represents the Virtual Master Time value, of the master server p_{master}, that will be used by all server for synchronizing their local clocks. Therefore, the processes in all local groups takes the VMT value via their local servers as their own local clocks. Thus, all participants in groups have the same time value at a given time.
  The role of p_{mast} is to broadcast periodically his virtual local clock VMT to all members of g_v allowing then the local clocks synchronization.

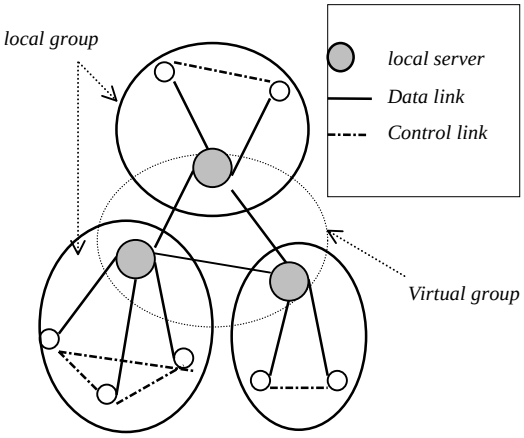

Figure 2.5. Hierarchical architecture in a distributed system

Algorithm of process p_i **:**
begin

let $g_{tr_i} \in G$ ***a transit group of process*** p_i ***such as, constructed by performing tests 1) and 2),***

Let $n_i = |g_{tri}|$ ***,***

$\forall p_j \in g_{tr_i}$ → ***send Req_set(*** g_{tri}***) to*** p_j ***;***

$g_i = \emptyset$***;***

Repeat

When receiving Req_set(g) from p_j

$g_i = g_i \cap g$***;***

$n_i := n_i - 1$ ***;***

Until $n_i - 1 = 0$ ***;***

If p_i = ***min(*** g_i***) then*** $\forall p_j \in g_i$ → ***send Establish(*** g_i***) to*** p_j ***/*** $p_j \neq p_i$ ****/ ;***

VP(); */* call of the VP procedure by the local server */*

/ The process which has the lowest identity value in each g is selected as local server for that local group */*

When receiving Establish(g) from p_j

Mark p_j ***as my local server and g as my local group;***

end.

The correction and the verification of the proposed algorithm have been studied in our paper [2].

Having a virtual hierarchical structure, the links of the network are divided into two types (figure 2.4 and 2.5) :

(i) data links are a part of the hierarchical structure, their role is to transfer multimedia traffic to their destinations ;

(ii) control links are all the rest, their role is to inform the other processes in groups in the case of a process or a link failure by transporting control information like the Master server's group content.

NB.

- The data transmission is accomplished in distributed manner between the servers of the virtual group. However, it is in hierarchical manner in the local group.
- For maintaining this synchronization the Master server broadcasts periodically its VMT to inform all participants in the group communication system of the current time taking by itself. By this manner, we solve the local clock drift problem.

VP- procedure :

Once a local server is elected, it visits each nodes in its group members set and assigns a VP number at its incoming port (figure 2.6). The main role of the VP numbers is to maintain the hierarchical structure in life in case of a node or a server failure. The objective is to obtain disjoint sub-tree for the connect nodes to ensure continuous communication among them. For assuring its successor, the local server closes every node failure by deleting its VP numbers from its set.

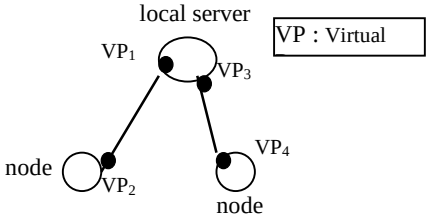

Figure 2.6. VP-procedure in a local group

If a process wants to leave the group it is removed from the group members set by all other processes. If a process wants to join the group it is added to this set by all processes.

Termination procedure is guaranteed since eventually all the processes in the group are labelled. Correctness follows since each VP of each process is labelled exactly once, and the number VP is incremented by one to create a unique VP identifier in the local group.

Text of the VP Procedure :
begin
Let n_s ***be the local server (root) of the group***
starting from n_s ***;***
let DF be the Depth-First order ;
let VP the virtual port number ;
VP $\leftarrow$ ***0 ;***
traverse of the group hierarchical structure
For each unlabelled input n_i ***visited by DF***
Do
VP = VP +1 ;
Assign VP to n_i ***;***
Mark n_i ***as labelled in LIST$_s$.***
EndDo
end.

3. A New delivery condition for the group

The lifetime, Δ, of a message is the time duration during which such a message can be used by its destination process. For a message m the instant *sending time of m* + Δ is actually the new time duration value after which the QoS of that message is degraded, and the message is useless for the destination process.
A message that arrives, to destination, to early is delayed until its delivery condition becomes true. This allows reordering of messages since the execution of this message would have invalidate the rest of messages that would have made the deadline had the other message not arrived so early.
In multimedia applications, the lifetime of a message is the maximum transmission delay that an application can tolerate before delivering a message before the quality of its service degrades. This delay is strictly connected to the nature of the multimedia applications. For example, in teleconferencing systems, the maximum lifetime for audio and video messages is 250 msec. [22] (an acceptable one is 100 msec. [23]). Thus, to ensure a real-time causal order in a group communication system the time duration of a message must be less or equal Δ.
In all multimedia systems, the lifetime of messages composing the applications do not indicate their delivery time periods at which all destinations start to playback simultaneously the same message in a synchronous fashion. To fulfill the synchronization requirement in the multicast configuration, we must solve the following problems : local clock drift, and the delay jitter.

3.1. Local clock drift

In order to enforce the real-time delivery constraints, processes must synchronize their local clocks. Without loss of generality, we use the synchronization mechanism proposed in [1], where the time reference is a virtual clock noted VMT (Virtual Master Time) maintained by a initiator in the group communication system. Thus each process use this VMT time value to synchronize its own local clock by adding a synchronization delay to it. Consequently, at a given time all processes will have the same time value.
Once all local clocks have been synchronized, we must keep this synchronization among the communication exchange to insure the real-time relationships. If that constraint is relaxed, the clocks are no more than ε time units out of synchronization with each other. To solve that local clock drift without changing to the algorithm, it is suitable to update only the definition of Δ. For that, the initiator sends periodically its time value VMT (Virtual Master Time) to all members in the group system, in order to keep clocks synchronized within 5-10 milliseconds [20, 21].
If $\varepsilon = H_{new}-H_{old}$, (Where H_{new} = VMT+ τ_i is the time value of a local clock according to the initiator clock (VMT) and τ_i is the synchronization delay of the local clock), is non-zero, then Δ is defined to be the time data in a message with the maximum clock drift : $\Delta_{new} = \Delta_{old} + \varepsilon$
This change does not have an influence upon the functionality and the validity of the protocol, because Δ defines only the lifetime of a message.

3.2. Delay jitter

the delay jitter problem between a sender process p_i and a receiver process p_j can be solved by prefetched buffering with buffer size B_j. This is the amount of time required for process p_j to buffer the media units from p_i

after p_i starts to collect them. In this way, the delay jitter effect can be compensated. Given by the following formula [16], the buffer size can be defined as : $B_j = \left\lceil \frac{J_{i,j}}{\mu_j} \right\rceil$ (1)

where $J_{i,j}$ is the delay jitter between p_i and p_j and is computed by : $J_{i,j} = D_{i,j} - d_{i,j}$. (2)
With $D_{i,j}$ is the maximum end-to-end delay between p_i and p_j; $d_{i,j}$ is the minimum end-to-end delay between p_i and p_j, including the propagation delay, minimum collection delay and minimum delivery delay (figure 3.1). We note that $d_{i,j,}$ can obtained by taking the minimum value over several end-to-end delay measurements. Further, the collection delay is the time needed for the process sender to collect media units and prepare them for transmission. The delivery delay is the time needed for the receiver process to process the received media units and prepare them for playback; μ_j is the playback period of a medium at receiver process p_j, and $\lceil x \rceil$ is the smallest integer greater than x.
Note that B_j is measured in terms of media units. The size of media units may be constant or various depending on the characteristics of the medium and the encoding scheme. By this way we can prevent the asynchrony caused by delay jitter. If $J_{i,j}$ can be kept small, then the required buffer size B_j will decrease as can be seen in equation 1. Since the values of $D_{i,j}$, $d_{i,j}$ and μ_j can be determined in the connection phase for multimedia communication between processes, the delay jitter effect can be solved by selecting an appropriate size of pre-fetched buffer at the receiver process as given in equation1.

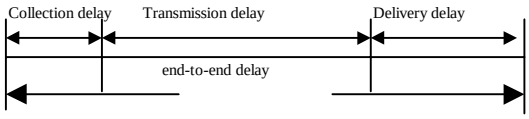

Figure 3.1.End-to-end delay model

3.3. Global delivery time for the group

Once the local clock drift problem and delay jitter effect have been solved, we show now the derivation of the global delivery time φ_g, that allows destination processes to deliver the same media unit simultaneously. Without loss of generality, we assume that the sending time of media units from sender process p_i to all other processes is the same as specified in [4]. This time is equivalent to the collection time at which a process sends its media units to the destination process (it is the send event).

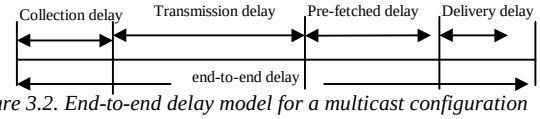

Figure 3.2. End-to-end delay model for a multicast configuration

One approach to maintain synchronization is to timestamp messages as it leaves the sources and, in parallel, to calculate the maximum delay it will take for message to travel from source to destination. Once this maximum delay is agreed upon by the destinations, they delay the message they receive until a time equal to the timestamp of the massage plus this maximum delay. In this manner, the destinations will consume the message simultaneously to be played back in the same time. Our approach is based on the model of data delay in figure 3.2. It shows four components : a sending time that represents the event of sent message, it is the time delay at which the message is prepared to be sent ; a transmission delay to reach the protocol peer across the network; a pre-fetched time to buffer a message within its deadline until its delivery condition becomes true; and a delivery delay from the moment the protocol releases the message to the moment this message is presented to the application. By assuming the maximum and minimum end-to-end delay, we can calculate the pre-fetched buffer sizes $B_j(m)$ for any medium of destination process p_j using equations (1) and (2). For connections between p_i and p_j, the time that p_j could deliver the message is : $\gamma_{i,j} = \max_m \left\{ \delta_j(m) + D_{i,j}(m) \right\}$ (3)

where, $\delta_j(m)$ is the time spent by a media unit in the buffer after which it is available to deliver it, and is expressed as follows : $\delta_j(m) = B_j(m)\sigma_i(m)$ (4)

where B_j is the pre-fetched buffer size and $\sigma_i(m)$ is the value of the inter-collection time between media units at destination pj. This can be defined as: $\sigma_i(m) = sending\_time_m + \psi(k+1) - \psi(k)$ (5)

where the delaystamp $\psi(k)$ is the sum of all times spent by the media unit k in the nodes (switches) of the network plus the collection delay at p_i and the delivery delay at p_j. If this time is constant, so the second term of the equation (5) will equal to zero, and $D_{i,j}(m)$ is the maximum end-to-end delay between p_i and p_j.

Thus, based on the equation (3), we see that if the receiver p_j starts the deliver media units before $\gamma_{i,j}$, then discontinuity may occur due to the delay jitter and insufficient pre-fetched buffer size. On other hand, if p_j starts to deliver media units from p_i after the time $\gamma_{i,j}$, then p_j must buffer more media than it needs, a wastes of network resources. However if we want to synchronize the delivery time for each process in the system denoted the global delivery time φ_g, we must select a large delivery time $\gamma_{i,j}$ of media units from one process at all other in the system, and this time must be the largest one among all processes in this system. So, we can select the global delivery time of the processes as : $\varphi_g = \text{sending\_time}_m + \max_i \left\{ \max_j \left(\gamma_{i,j} \right) \right\}$ (6)

where φ_g represents the maximum delivery time $\gamma_{i,j}$ among the connections associated with a process p_i, that is the largest one among the processes in the system. Thus, each process can select this time value as its own delivery time by using the equation (6).

3.4. Δ *-causal delivery for group configuration*

the Δ-causal protocol for multicast configuration can be implemented over the original underlying system. Due to the new delivery constraints imposed by Δ*-causality* ordering in group communication system, a received message within its deadline can be delayed until the delivery condition for group becomes true. Thus, this new abstraction is associated with each multicast message m in such a way that m is delivered to all group members as soon as possible, i.e., when all multicast messages that causally precede m have either been delivered or have been lost or they are still in transit when their deadlines expire. Basing on the clock synchronization, described in section 3.1, each process has the *current_time* value that is continuously updated by the local clock. Even, each one of them manages a vector *sent*, where $sent_i[j]$ that represents the knowledge of process p_i about sending time of the last message sent by p_i to p_j and a vector *del*, where $del_i[j]$ represents the sending time of the last message sent by p_j and delivered to p_i. Each message m piggybacks the following control information : VT_m is a copy of the vector *sent* of the sending process at m's sending time. Note that from the definition of the vector *sent*, there is a difference between the $send\_time_m$ value and VT_m. The first one is defined as the send event, and the second one represents the timestamp of the sending time of message. Thus the following relation must be hold :

$\forall p_i \in G : send\_time_m > VT_m[i]$ (rule 1)

Also, to achieve the delivery condition for group configuration, the lifetime, Δ, that represents the duration of message must be larger or equal to the global delivery time value φ_g, otherwise no process in the distributed system does not playback its media units. Thus, the QoS of media will be degraded. So, to synchronize the delivery event of all processes the following relation must be respected :

$\forall m : \Delta + send\_time_m \geq \varphi_g$ (rule 2)

A message m, sent by a process p_i to p_j is discarded if $(send\_time_m + \Delta > current\_time) or (VT_m[i] > del_j[i])$. By definition, all messages arriving at their destination within their deadline must be delivered when their condition becomes true. More formally, the delivery condition of a message m sent to p_j is expressed as DC(m):

$\forall i : (VT_m[i] < current\_time - \Delta) \wedge (\varphi_g \leq current\_time \leq \Delta)$ Thus, by this way all synchronization requirements are solved and all group members can playback simultaneously their media units in a synchronization fashion.

Algorithm for each process :

We suppose that there is neither process nor link failure during the communication, the maximum end-to-end delay is known, and the channels are reliable and FIFO.

begin
$send\_time_m$:= current_time ;
$\forall p_j \in S$:
    $VT_m := sent_P$;
    send(m,VT_m, $send\_time_m$) to p_j ;
    $sent_i[i] := send\_time_m$;

upon receiving (m,es, st) by P_j
    if (st+ Δ < current_time) or (es> $del_j[i]$)
    then discard(m) ;
    else wait $(es < current\_time - \Delta) \wedge (\varphi_g \leq current\_time \leq \Delta)$
    $del_i[j] := st$;

deliver(m) ;
end.

To prove the Δ -causal protocol for multicast configuration, we show that all messages arrived within their deadlines will be delivered in the same time to all processes within their deadlines (*liveness property).* In other hand, we prove that all delivery events respect the causal order. For more details concerning the proof, the reader can consult our article[24]. One can find also a performance evaluation with Coloured Petri Nets.

4. Buffer sizes

In reality the machine buffer size is not fully used for supporting a real-time multimedia application. The buffer size required to run such application, at each user, is based on the delay jitter, the nature of the application, etc… For example, in a teleconference application, the buffer is used to compensate of delay jitter caused by the asynchrony of the communication channels and their propagation delays.
As data is stored and processed on computers in discrete values (bits and bytes), *continuous* data streams like audio, and video have to be digitized and quantized. So, we can represent a multimedia application as a set of streams, where each one of them is defined as a set of media units (MUs). One media unit contains the atomic information of a media stream that can be displayed. This is ,e.g, an audio sample or a video frame. The duration of one MU in data streams is media dependent and can vary a lot. In this work, we assume that the MUs are independent from each other and can be processed independently.

4.1. Buffer sizes for clients belonging to a local group

Once a local group is defined, the local server p_i of a local group g_i sends media units of a multimedia application to its group member p_j (j=1..n_i) with a maximum peak rate $\tau_{ij}^{\max}(m)$ and a maximum delay jitter $J_{i,j}(m)$ of the connection carrying medium m. As known, the one way to compensate for the effect of delay jitter is to estimate the amount of time $T_{i,j}$ required for user p_j to buffer and to start playback the media unit sent by the local server p_i. So, to insure that all group members play out simultaneously the same media unit, we must estimate the time required for each user j to buffer media unit after the moment the local server starts to collect it :

$$T_{i,j} = \max_{m} \{ D_{i,j}(m) + J_{i,j}(m) \} \qquad (8)$$

where :

- $D_{i,j}(m)$ is the end-to-end maximum delay supported by the medium m carried by the channel connecting the user p_j to its local server p_i,
- $J_{i,j}(m)$ is the maximum delay jitter supported by the medium m from p_i to p_j, and is expressed as follows : $J_{i,j}(m) = D_{i,j}(m) - d_{i,j}(m)$, where $d_{i,j}$ is the minimum end-to-end delay including the minimum collection delay, propagation delay and the minimum delivery delay, and is computed as follows :

$$d_{i,j}(m) = \delta(m) + ppt_i \qquad (9)$$

where $\delta(m)$ is the minimum end-to-end delay including the minimum collection delay, and minimum delivery delay; and ppt_i is the packet-processing time of each local server. The buffer is to ensure that each user can buffer the first media unite MU_0, and is given by the following formula.

$$\beta_{i,j}(m) = \frac{J_{i,j}(m)}{\omega(m)} \qquad (10)$$

where $\omega(m)$ is the presentation period of that media unit in the medium m.

The relation (8) allows to guarantee that each receiver can buffer the first media unit MU_0, because the presentation time of this media unit is unknown.

Until now, the group members in the local group g_i can not playback in the same time the same media unit. For that, each user p_j must still wait until all users in the local group complete buffering the media. Thus the time required at which all participants start playback the same media units from the local server p_i is :

$$T_i = \max_{j=1}^{n_i} \{ T_{i,j} \} \qquad (11)$$

If we take into account this constraint, then the group members must keep buffering the media units from local server of the local group i until the time T_i is reached. Thus, the buffer size B_{ij} at each user p_j belonging to the same local group is : $B_{ij} = \sum_m (T_i - d_{i,j}(m)) \tau^{max}(m)$ (12)

where $\tau^{max}(m)$ is the maximum peak rate supported by the medium m, and is expressed as follows :

$$\tau^{max}(m) = \sum_{j=1}^{j=n_i} \tau_{ij}^{max}(m) + \sum_{l=1}^{k-1} \tau_{li}^{max}(m) \qquad (13)$$

where :

- $\tau_{ij}^{max}(m)$ is the maximal peak rate of each medium m between process p_j and its local server p_i,
- $\tau_{li}^{max}(m)$ is the maximum peak rate of medium m from two local servers p_l and p_i in the virtual group,
- n_i is the number of users in the local group g_i, and
- k is the number of local servers composing the virtual group.

4.2. Buffer sizes for local servers

Once all users synchronize their local clocks, each local server p_i can receive and buffer the temporally related media units before sending them to its local and/or virtual groups members. After transmission, the temporally media unit must stay in the local server until that the playback time is reached. To do that, the local server of the group g_i must send media units from its own local group synchronously. However, since the end-to-end delay is different over each medium, the local servers will not send the media units in a synchronous fashion. To ensure a synchronous transmission, we provide a mechanism that makes possible to buffer temporally the media units.
Let :

- S_i a time needed for a local server p_i to synchronize between its local group members in order to allow them to present their media units in the same time.

$$S_i = \max_{j=1}^{n_i} \left\{ \max_m \{ D_{j,i}(m) + J_{j,i}(m) \} \right\} \qquad (14)$$

Also, at time S_i , the local server starts to transmit the media units into the virtual group, and

- $S_s = \max_{i=1}^{k} \left\{ \max_m \{ D_{s,i}(m) + J_{s,i}(m) \} \right\}$ is the necessary time needed to a local server to buffer a media unit sent from the others servers in the virtual group.

Then, the buffer space required for the local server of a local group p_i is :

$$B_i = \sum_{j=1}^{n_i} \sum_m \{ (S_i - d_{j,i}(m)) . \tau_{ji}^{max}(m) \} + \sum_{s=1}^{k-1} \sum_m \{ (S_s - d_{s,i}(m)) . \tau_{li}^{max}(m) \} \qquad (15)$$

The first term on right-hand side of Equation 15 refers to the buffer size required to store the media units from users in the same local group and the second term is the buffer size required to store media units received from the others servers.

5. Model management

Once the hierarchical architecture is built, a mechanism must be provided to handle the entering and leaving of users and local servers. We introduce a reconfiguration algorithm that solves the problem of users entering and leaving the architecture. If a user belonging to a local group wants to leave the group, it is removed from the list by all other processes in that local group via their local server. However, if a user P_{new} wants to join the group communication model, it multicasts a message request to its immediat neihgboors p_j, requiring the server address and the group belonging to of each one of them. Since the election algorithm is distributed, each user has full knowledge of the grouping information. Thus, user p_j can give the addresse of its local server to the new user P_{new}. The new user P_{new}, after getting the addresses of the local servers, sends a connecting message to the local servers. When the local server receives this message, it performs the first, and the second tests, to ensure that the local server has enough processing power. If this new user P_{new} is accepted by the local server, it receives a acknowledgement. In the case of it receives more than one acknowledgement the new user P_{new} makes connection

withe local server which has the lowest identity. During this phase the new user negotiates the maximum end-to-end delays with each local server separately. Once, the connection with the local server is established all group membrs adds the new user to its group members set. If the maximum end-to-end delay D_{new} between the new user and its local server p_i is higher than $\max_{j=1}^{n_i}\{D_{ij}\}$ then all group members update their buffer sizes Equation (12).

In the extreme case, if the new user is not accepted by any local group, so the new user becomes a local server and communicates with others directly.
However, if a local server wants to leave the communication, then it informs all local group members, and the virtual group members. As each member of that local group has the same group members set contents, each one of them makes a connection with the local server's successor who have the lowest identity. After the connections are established, processes can get messages through the old local server, or the new local server, but they send their messages to the new local server. In other hand, the new local server informs the users that they can receive its messages at a time stamp T, this time is the time at which the old local server sends all messages stored in its buffer.

6. Performance evaluation and simulation

In this section, we investigate the performance of the hierarchical architecture. Our simulation model consists of n users (i.e. processes). Each user wants to communicate with other users by sending and receiving two media stream, video and audio. The playback period ω for video is 40 ms and for voice, it is 15 ms. The maximum peak rate of each media is approximately 2Mb/s and voice 64Kb/s. The maximum delay jitter of a connection is between 20% to 65% of the minimum end-to-end delay of the connection. The simulation results given in Table 1-2.
We divide the group communication model into several local groups and calculate the buffer size for each user, each local server, and the master server with Equations 12 and 15.

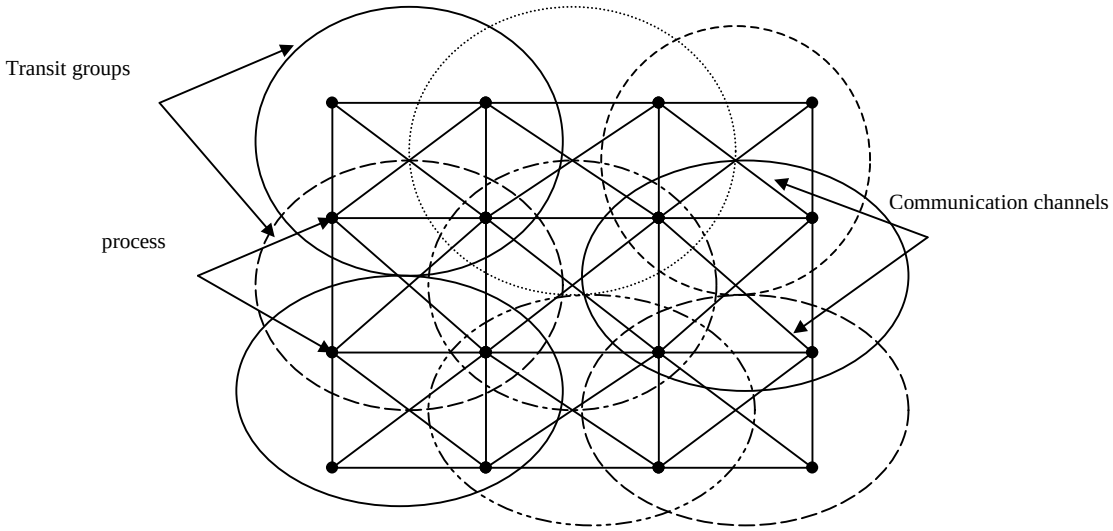

Figure 6.1. Model of repartition of processes in distributed systems

The results of simulation are presented in the figures 6.2, 6.3, 6.4. We used the dynamic algorithm of grouping to obtain local groups from the transit groups as shows it figure 1. the number of process composing the distributed system varies between 5 and 50. In addition, we built the hierarchical structure in each local group.

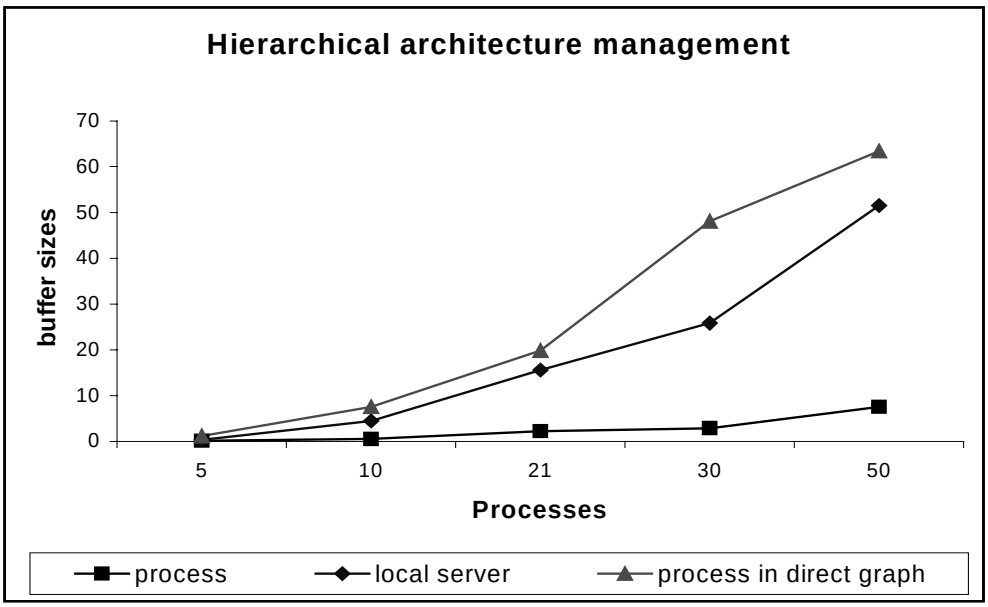

Figure 6.2. Buffer sizes in the hierarchical architecture

Moreover, we determined buffer sizes of processes and local servers composing this hierarchical architecture.

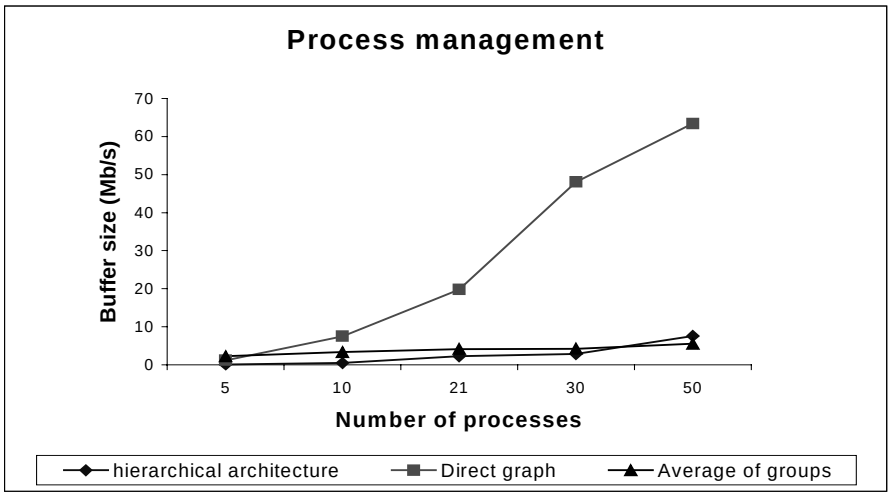

Figure 6.3. Comparison of buffer size between direct graph and hierarchical structure

In figure 6.2, we show the buffer size of each process belonging to the group structure. Moreover, we present in figure 6.3, a comparison of the buffers size in the case of a direct graph and the presented hierarchical architecture. We note that with the increase in the number of participants in the distributed system, the buffers size of the processes increases proportionally in the related graph. However, in the hierarchical structure the processes consume less memory in the presence of a great number of processes. This phenomenon can be explained by the number of local groups constituting the hierarchical architecture.

Moreover, the buffers size on local servers are larger than those of processes in local groups (figure 6.4). That means that processes stock only the necessary information units to ensure the playback phase of audio and video each process of distributed systems.

The fact that the local servers need enough memory capacities, that proves that the processes of the local groups can share these spaces and reduce their own buffers thereafter.

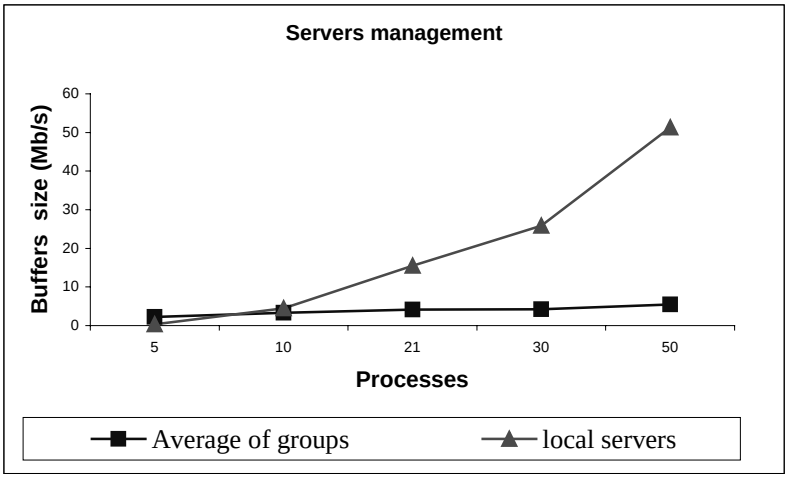

Figure 6.4. Relationships between the groups number and Buffers

In addition, according to figure 6.4, we note that when the number of processes increases, the total number of the local groups existing in the hierarchical structure does not increase significantly. Indeed, the local groups size depend initially on the variation of the initial grouping model, i.e. according to the existing number of connections between the immediate neighbors and the processes of the distributed system. Thus, the average size of these servers is always in relation to the grouping model.
Consequently, the average buffers size varies. Moreover, this variation becomes also valid if a local group plays the role of a transmitter. In this case, this group must store more messages to await the site the most remote sender in the system of groups, in order to synchronize the playback phase. However, the average size of the buffers of the processes remain very small, compared to the number of participants in the distributed system.

7. Conclusion

In this paper, we were interested in new mechanism of management of the buffers having for vocation to ensure on the one hand, the continuous transfer of the data and on the other hand, the playback phase at each process level. The dynamic management suggested in this paper, is based on a hierarchical architecture made up of k-groups. This management makes it possible to reduce the buffers size of the destination without deteriorating or adding penalties during the transmission or the playback phase.
In addition, the buffers sizes of the local servers composing the architecture have almost the same sizes which can have the processes on a direct graph. In other terms, this mechanism of buffers management does not introduce other memories sizes. Thus, the buffers management model in the communication systems makes it possible to ensure the delivery of the media units in same time to guarantee the playback phase without the users not absorbing very large buffers sizes

References

[1] A. Abouaissa, A. Benslimane, M. Naimi, ”A Group Communication Model for Distributed Real-time Causal Delivery”, the 7th IEEE ICCCN’98, 12-15 October 1998, Lafayette, Louisiana, 1998.

[2] A. Abouaissa and A. Benslimane, “Dynamical Grouping Model for Distributed Real Time Causal Ordering”, Internal report, LaRIS, Technical University of Belfort, December 1999.

[3] J. F. Adam, et al., “A network architecture for distributed multimedia systems”, Proceeding of ICMCS’94, Boston, May 1994.

[4] I. F. Akyildiz, W. Yen, “Multimedia group synchronization Protocols for Integrated Services Networks”, IEEE Journal Selected Areas Comm. Vol. 14(1), 1995.

[5] D. P. Anderson, G. Homsy, “A Continuous Media I/O Server and its Synchronization Mechanism”, IEEE Computer, October 1991.

[6] J. Burgin, “Dynamic capacity management in the BISDN”, Int. Journal Digital Analog comm. Sytesm 3, 1990.

[7] A. Campell et al., “Integrated Quality of Service for Multimedia Communication,” Proc. Of the INFOCOM `93 Conference, pp. 732-740, Apr. 1993.

[8] J. Escobar, et al., “Flow Synchronization Protocol,” IEEE Global Communications Conference, pp. 1381-1387, Dec. 1992.

[9] M. Gerla et al., "Topology design and Bandwidth allocation in ATM nets", IEEE Journal Selected Areas Comm., October 1983.
[10] S. J. Golestani, "A Framing Strategy for Congestion Management," IEEE Journal on Selected Area in Comm., Vol. 9, No. 7, pp. 1067-1077, Sep. 1991.
[11] S. Kadur, F. Golshani, and B. Millard, "Dealy-jitter in multimedia applications", Multimedia Systems vol. 4. N° 1 : 30-39 1996.
[12] R. Kawamura et al., "Fast VP-Bandwidth management with distributed control in ATM Networks", IEICE Trans. Comm., E77-B (1) 1994.
[13] L. Lamont and D. Georganas, "Synchronization Architecture and Protocols for a Multimedia New Service Application," Proc.of ICMCS `94, Boston, May 1994.
[14] A. A. Lazar et al., "Control of resources in broadband networks with quality of service guarantees", IEEE comm. Mag., October 1991.
[15] T. D. C. Little and A. Ghafoor, "Synchronization and Storage Models for Multimedia Objects," IEEE Journal on Selected Areas in Comm., vol. 8, No. 3, pp. 427-430 Apr.
[16] S. Ramanathan and P.V. Rangan, "Adaptive Feedback Techniques for Synchronized Multimedia Retrieval over Integrated Networks," IEEE/ACM Transactions on Networking, vol. 1, No2, pp. 246-260, Apr. 1993.
[17] P. V. Rangan, H. M. Vin and S. Ramanthan, "Communication Architectures and Algorithms for Mixing in Multimedia Conferences," IEEE/ACM Transactions on Networking, vol. 1, No. 1, pp. 20-30, Feb. 1993.
[18] J. Won Choi et al.," A Synchronization Model for User Interactive Control Using the Script Language," E-mail : jwchoi@ipl.chungnam.ac.kr
[19] W. Yen, I. F. Akyildiz, "The Hierarchical architecture for buffer management in integrated srvices networks", Multimedia Systems Vol. 4(3) : 131-139, 1996
[20] R. Gusella, S. Zatti,'The Accuracy of the Clock Synchronization Achieved by TEMPO in Berkley UNIX 4-3 BSD', IEEE Trans. on Software Engineering, 15(7) :847-853, july 1989.
[21] D. Mills,"Internet Time Synchronization : The Network Time Protocol", IEEE Trans. on Comm. 39(10) :1482-1493, October 1991.
[22] K. Jeffay D.L. Stone and F.D. Smith, "Transport and display mechanisms for multimedia conferencing across packet switched networks", Computer Networks, vol.26, pp : 1281-1304, 1994.
[23] K. Ravindran an V. Bansal, "Delay Compensation Protocols for Synchronization of Multimedia Data Streams", IEEE Trans. On Knowledge and Data Engineering, vol.5, No . 4 pp :574-589, 1993.
[24] A. Benslimane and A. Abouaissa, "A Synchronization Protocol for Group Communication Systems", Proc. of the IEEE Seventh International Symposium on Modeling, Analysis and Simulation MASCOT'99, College Park , Maryland USA, IEEE Computer Society Press, October 24-28, 1999, ISBN 0-7695-0381-0, pp. 322-329.

Authors' biographies :

Abderrahim BENSLIMANE is Associate Professor of computer science and Engineering at the Technical University of Belfort, France. He received his PhD in Computer Science from the Franche-Comte University of Besançon in 1993.
His research and teaching interests are in Specification and Verification of Communication Protocols, Computer and High speed Networks as ATM, Distributed Multimedia and languages Programming.

Abdelhafid ABOUAISSA received his B.S. degree from Technical university of Wroclaw, Poland, in 1995, and M.S. degree from Franche-Comte University of Besançon, France, in 1996. He is currently a Ph.D Student at Technical University of Belfort, France, where his interests include Multimedia Synchronization, group communication systems, ATM, QoS management.