A Scalable Collect

Jean-Marie BONNIN
bonnin@dpt-info.u-strasbg.fr
http://dpt-info.u-strasbg.fr/~bonnin

Jean-Jacques PANSIOT
pansiot@dpt-info.u-strasbg.fr
http://dpt-info.u-strasbg.fr/~pansiot

UNIVERSITE LOUIS PASTEUR - LSIIT URA-CNRS 1871
Département d'Informatique - 7, rue Descartes - 67084 Strasbourg Cedex - France
Tél: (33) 03 88 41 63 00 - Fax: (33) 03 88 60 26 54

Abstract

In a multicasting application it is often desirable for a source to get information on the status of receivers. We call this operation a **collect**. In large scale groups it is not feasible to have individual messages flowing from each receivers to the source. We present a collect using a hierarchical control structure which progressively aggregates data from subsets of receivers to the source. The hierarchical control structure is based on a control structure that is also used for implementing reliable multicast. In this paper we propose such a collect mechanism, called CAGE. We describe several types of collects (standard, scheduled, asynchronous) depending mainly on when aggregation of data starts. We then show how the collect propagates intermediate results into the control structure.

Keywords: Collect, reliability, multicast, large scale, group

1. Introduction

Peer to peer reliable transmission protocols (such as TCP) allow applications to adapt their behavior to network state and performance. Some researchers [DHT95] believe that applications using multicast « should be adaptive » to reach a maximum efficiency. In a point-to-point context even if an unreliable transport protocol such as UDP does not provide any information to the source, an application can obtain such knowledge by its own observation of application level acknowledgments. This allows the source to adapt its rate to the real functioning of the network. A point-to-point reliable protocol such as TCP needs only the error rate to control traffic. Unfortunately, it is more difficult in the multicast context to inform the source in order to ensure the accuracy of its actions. When a group grows even the error rate for each participant becomes hard to evaluate. Moreover, complex multimedia applications need more information about receivers to

adapt their transmission. There are two major directions to solve this problem. First, each receiver can react to its own reception conditions by modifying the resource reservation [ZDESZ93], by the means of filters [YMGH96] or by processing only a part of a hierarchical data flow [CJ96]. This supposes network capabilities. This approach is particularly interesting in the case of heterogeneous receivers and network links, since each receiver can adapt to its own conditions.
In another approach the source can adapt data flowing to receivers and the problem is to inform the source of receiver states. Note that adaptation by the source and by receivers are complementary. However for very large groups, the problem is to transmit and process a huge amount of information. A first solution is proposed by RTP/RTCP [SCFJ96]. This protocol allows each member of a multicast group to inform periodically all the others about its state by multicasting data to the entire group. But this type of mechanism has a scalability problem, as we will see. This leads us to develop a protocol that aggregates information before it reaches the source. We think it is necessary to provide the source with an amount of information that is bounded when the group grows, but with a controlled summarization.

Our goal is to provide a scalable many-to-one (or a *concentration* [Dio94]) communication to allow a multicast source to run adaptive applications. Some problems must be solved:

- the bandwidth used on any link should be bounded;
- the processing power needed in any node should also be bounded.

We propose a scalable gathering we called a « collect », to provide such a service.

We made the choice to use a hierarchical control structure already developed to ensure reliability in multicast communication. This control structure allows our collect to aggregate step by step information from receivers to the source of a collect request. Therefore, this aggregation process is naturally distributed over the group.

In the next section we first outline related works. Afterward we describe the aggregation operations allowed by our collect. And finally we sketch out the process of aggregation inside the hierarchical control structure and the functioning of a control entity.

2. Related works

In this section we analyze some problems of many-to-one information mechanisms when the group grows. We study solutions proposed in other works, not always concerned by large scale groups.

2.1. Informing the source

We have already emphasized that the application at the source should be informed to properly react to the network behavior. Useful information mentioned in some papers includes delay, jitter,

throughput, loss rate, ... This allows the source to adapt its data flow, compression rate, data encoding, frame rate, for a best use of network capabilities. A lot of recent research concerns adaptive applications, among them IVS [Tur94] and its successor « rendez-vous ». For example, to prevent a NACK implosion problem, IVS sends NACKs to request retransmission only if there are less than 10 receivers. Otherwise it uses a forward error correction scheme by increasing the intraframe rate. This rate depends on the loss rate observed by receivers and transmitted to the source [TH96].

But, we can imagine higher level interaction, for example about degree of satisfaction of a televiewer, or possibly global management information similar to SNMP. It is hard to imagine all usages of such an information service. It could be very useful to permit the transmission of data with an application semantic. So we think the application should decide what information it will need and we emphasis our work on how it will be transmitted. This leads us to study a many-to-one communication channel. Note that for some applications it could be useful for each receiver to get information on all other receivers. It is always possible for the source to multicast a summary of receiver states to the whole group.

In RTCP, the control protocol of RTP [SCFJ96], each member of a multicast group informs periodically all the others, hence the source, about its state by multicasting data to the entire group. This type of mechanism has a scaling problem: either the periodicity is fixed and a member may be overwhelm by messages when the group becomes large, or the period increases (it is the case in RTCP) and the information gets less accurate.

The semantic of information carried by RTCP depends on the protocol used over RTP, since RTP was design in the ALF [CT90] framework and is a useful basis to develop a multipoint protocol. The multicasted information is application-dependent. For example, in SRM [FJCLZ95] negative acknowledgments are transmitted via RTCP.

RTP does not provide a centralized collect control mechanism, since each member of the group decides to multicast data about its own knowledge. Therefore there is no explicit means for an application to control when informations are multicasted. Some other protocols, like RMNP [AC95] or the older RM [Raj92], allow the requester of a collect to control when a collect starts, and to manage the collect until its end.

2.2. Aggregating information

RTCP does not provide any solution to store and process the important amount of data received by each member of a multicast group. This could overflow the capacities of some members. Note that if information is multicasted, all members must execute the same computation. Moreover if the collect requires some degree of reliability, each member must store all individual responses. This is an obvious waste of resources.

Other protocols we have already mentioned perform an aggregation before delivering gathered data to its destination. For example RM provides « gather » operations, which allow to obtain the set (by the NULL condition), minimum, maximum of the response set. The application level defines the semantics of messages, and RM only gathers values. When an error occurs the *gather* is aborted and the source should restart the collect to obtain the requested information. This is an important drawback since the probability of an error increases with the size of the group.
To be complete we must evoke RSVP [ZDESZ93]. To reserve resources a receiver issues a message that travels backward the routing tree to the sources. At each RSVP router, information about requested resources is aggregate with others until it reaches a link with a sufficient reservation. This mechanism is analogous to a collect mechanism, but the application has no choice about data issued from receivers to sources: it is a network level collect.

2.3. A hierarchical control structure

A hierarchical structure is a natural structure to aggregate data in a « divide and conquer » approach. It can be assumed that each entity at some level of the structure has a bounded number of child entities at the next lower level. This yields a scalable process in terms of number of messages and computing needs. Such hierarchical structures are already constructed for IP multicasting [Dee89], as in CBT [BFC93] and PIM [DEFJLW94], or for reliable multicasting as in LBRM [HSC95] or LGC [HBC95, Hof96a, Hof96b].

RM and RMNP respond timely an locally to negative acknowledgment and aggregate positive acknowledgment from receivers to the source, in the multicast tree. This permits to avoid the implosion problem at the source and overloading in the network. Moreover it is particularly simple to aggregate acknowledgments. Similarly we can see RSVP aggregation of QoS requested by receivers as an aggregation mechanism with minimum, maximum and addition operations. It is a continuous aggregation with no control of beginning and end. The source always sees the aggregated value of entries (with delay) selected by receivers on their own control.

Proposing a collect mechanism at the network level is interesting only in conjunction with a reliable multicast. If not, this requires a lot of new mechanisms. We think that the reliability of multicast should be ensured at the transport level to provide a more flexible service. This allows applications to tune easily the degree of reliability.
Many works propose reliable multicast at the transport level using the unreliable multicast network service [Dee89]. This leads them to develop their own control structure. LBRM and LGC, for example, work with their own control structure to manage acknowledgments and retransmissions. In contrast SRM works over RTP/RTCP without any control structure. So it does not perform any aggregation of acknowledgments and the source gets no more information than other members. Indeed the SRM mechanism has the advantage to distribute totally the management of losses and to

provide it as close as possible to the receiver. It is an important feature for reliability in large scale groups when distances increase.

LGC divides group in subgroups and selects in each a *controller*. This *controller* manages messages, losses and retransmissions and performs aggregation of acknowledgments to the source or to its parent controller if the structure is hierarchical as it is suggested. In RM and RMNP, it is the nearest gateway which manages losses, and forwards acknowledgments to its parents.

2.4. Conclusion

We have shown that a multicast source should be informed by receivers about their quality of reception. Moreover, it could be interesting to allow receivers to communicate data with an application semantic to the source. We have seen that information must be aggregated in a source controlled way before it reaches the source. Firstly this decreases the number of messages in the network. Secondly it decreases the amount of information processed at the source. Gathering operations running at the network level are already proposed but to support aggregation a better way is to use a hierarchical structure such as those developed for reliable transport multicast. This allows to distribute over the network the load due to aggregation processing, and to manage more locally collect messages.

3. Our data collect : a « scalable gathering »

Our collect takes place in the more general context of our work. We are developing a multicast service in three parts :

1. A control structure using a hierarchy of fault tolerant control entities ;
2. A reliable multicast protocol FAGE (stands for *Fiabilisation À Grande Échelle*, Large Scale Reliability) where control entities are responsible for retransmission and acknowledgment propagation [Bon96]. In fact FAGE makes reliable a pre-existent unreliable multicast at network level.
3. CAGE (stands for *Collecte À Grande Échelle*, Large Scale Collect), the collect mechanism using the control structure to aggregate receivers information.

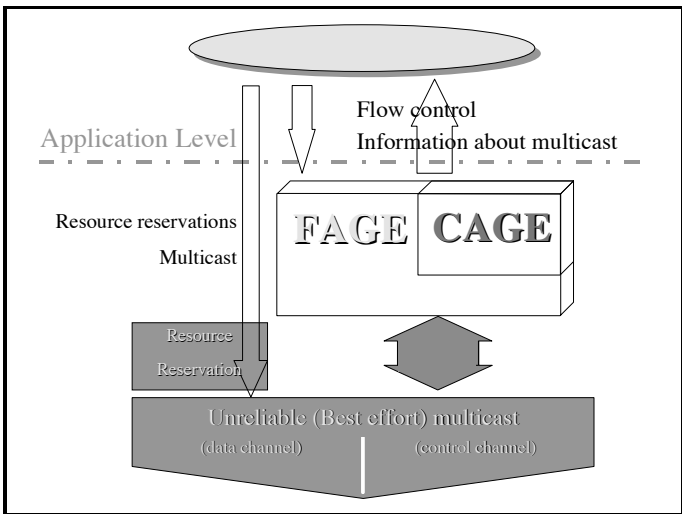

Figure 1 : CAGE in FAGE architecture.

FAGE and CAGE assume group-oriented communications : receivers must be able to dynamically join or leave groups, and senders are not involved in the membership management by the network

level. For simplicity, our multicast service allows a single data source, but it is always possible to define one group per source if an application uses multiple sources.

Although developed in the context of the FAGE service, we think the collect service and protocol are largely independent from the FAGE service and control structure. So CAGE requires only a hierarchical control structure. Each control entity could be responsible for a few hundred clients. This structure is spread over the multicast group, and could be implemented either on receivers or on specialized hosts (routers, gateways, ...).

In the remainder of this section we view the control structure as a single entity acting as a unique machine. We describe the collect as it will be observed by the source and receivers.
In order to satisfy the requirements of various applications, we define several types of collects. All have the same principles, and therefore, follow a similar scenario:
An application running at the source needs some information about members of the multicast group. Therefore it sends a request to the multicast group containing all receivers and the control structure through the one-to-many unreliable data channel. Each interrogated receiver computes its response if any and sends it to the control structure. Then the control structure which also receives the collect request, aggregates responses to provide the source with a usable (e. g. with a reasonable size) result.
In its request, the source can indicate which subset of the multicast group must respond, and the type of the collect. These parameters influence the time when the aggregation starts.

3.1. Different types of collect

We have defined three types of collect: *standard collect*, *asynchronous collect* and *control collect*. All of them gather the data specified in the collect request (see 3.2.). They may also indicate which class of members[1] are concerned by a specific collect. For example: all members, only reliable and collected members, … . Only reliable (i.e. *reliable* and *collected*) classes are explicitly known by the control structure which manages their membership. Therefore each collect must indicate when the gathering should end. This can be done in two ways:

- reliable collect: wait for all reliable members to respond;
- scheduled collect: wait until a specified time.

Standard collect

A standard collect follows five steps to provide a result to the source (see Figure 2 page 7). The first one is a collect request (1); it is issued from the source to all members of the multicast group. After checking if it is concerned, each interrogated receiver looks for its local matching variables (2_r)

[1] FAGE defines three classes of receivers: *reliable*, *collected*, and *others*. They correspond to the degree of reliability requested and provided.

before answering (3). During this time, the control structure initializes internal data structures necessary to manage the collect in particular for aggregation (2_s). Responses to the control structure reach the nearest part of the structure which records which receivers have responded. Later on, the control starts the aggregation of the collected information at a time that depends on whether the collect is scheduled and/or reliable (see . It uses the aggregation operator to progressively aggregate data along the hierarchical structure (4). At the end of this process the result is in a concise (bounded size) and accurate form. It is ready to be transmitted to the source (5) with parameters describing the state of the group. These parameters allow to measure the precision of the result (see 3.4).

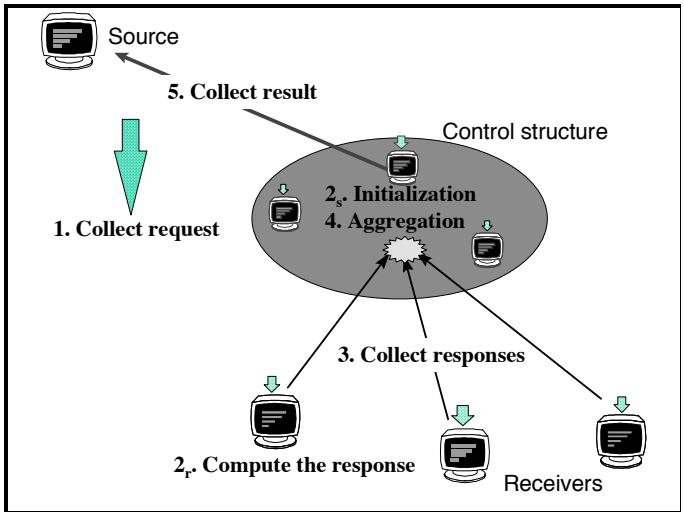

Figure 2 : Collect phases

Asynchronous collect

The main goals of the asynchronous collect design are to get results faster to the source and to allow only necessary data updates between receivers and the control structure. We use the control structure availability, reliability and liveliness properties to achieve these goals. Moreover, the structure can respond to a request faster than receivers. We define a mechanism to allow receivers to asynchronously send and update their data in the control structure. We have to split the standard scenario into 3 phases: preliminary collect, data updating and result request.

Preliminary collect:

The preliminary collect request is a message similar to the standard collect request. It initializes in each receiver the process that sends updates. In the same time the control structure initializes the data structure necessary to supervise future collects of this information and to store one value for each responding receiver.

It is necessary to have sufficient resources available on sites participating to the control structure. If not, they may cancel the collect by sending an error message to the source.

Data updating

After this setup phase, each receiver decides with its own application dependent criterion to update he stored data. It can do it periodically, or when the concerned variable changes, or each time the source sends a request of result.

Result request

When the source wants to retrieve data stored in the control structure, it sends a result request to the entire group. Only the structure has to respond to this demand ; it initiates the aggregation

mechanism. Receivers consider this demand as a reminder for those already participating in the collect, or as a preliminary collect request for newcomers.
When the request of result reaches sites in the control structure the aggregation starts immediately; this is an important speedup in comparison with the standard collect. Moreover, it is possible to have an arbitrary number of result requests for one preliminary collect request ; an asynchronous collect can last until the end of the multicast, polling periodically the control structure.
Ending an asynchronous collect
A time-out is associated to each pending asynchronous collect in order to free its allocated resources. This soft-state mechanism could avoid the use of an end collect message.

Control collect

As we will see in section 3.2, requested data are identified by logical names whose meaning is defined at the application level. It is the application entity of receivers that looks for matching data. The CAGE protocol is just a channel for this identifier from the source to receivers and for values collected from receivers to the control structure.
The control collect is a special case: the data collected concerns the functioning of the control structure itself. So the control collect request contains well-known identifiers specifying the protocol variables requested (à la SNMP). It will be necessary to define the list of well-known variables (similar to a MIB) in future works.

In addition to its types (*standard*, *asynchronous* and *control*) a collect has a finish mode that can be *reliable* and/or *scheduled.*

Reliable mode

In order to trust the collected information, some applications may request all known receivers to respond. So we propose that a collect request may indicate a reliable mode. Unfortunately this significantly slows down the collect process, since the control structure has to wait a response from all known members. It retransmits the request on time-out to each member which has not yet responded. There is a fixed maximum number of retransmission.

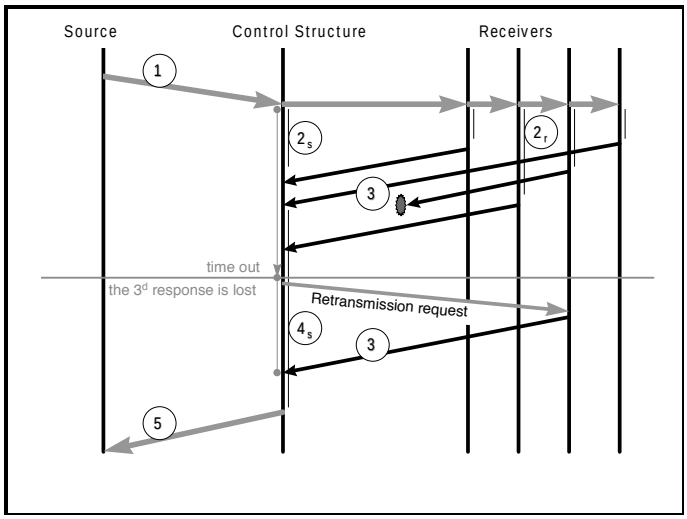

Figure 3 : Reliable standard collect with a loss

In the example of Figure 3 (page 8) we can see that the response from the third receiver

is lost or corrupted by the network. Therefore when the time-out occurs, a retransmission request is issued from the control structure to this receiver. Afterward the collect will continue normally.

Note that the reliable mode is possible only for a class of receivers whose members are explicitly known to the control structure (e. g. reliable class, collected class).

Scheduled mode

As the source can interrogate the entire multicast group, the control structure is unable to know individually all potentially answering sites. This leads us to specify a mode to limit the period of time during which the control structure and hence the source are waiting for responses (potentially there will be no more responses). In this mode, the collect request contains the time when the control structure must start the aggregation. Even if all members are known to the control structure this mode is useful to speedup the response to the source with a potentially non exhaustive result. Moreover scheduling the date to start aggregation allows the source to take a snap shot of the group at a more precise time excepting time for variations between sites.

In Figure 4 (page 9) we can see a scheduled standard collect. All phases are the same as for the previous collect, except that the beginning of the aggregation takes place at the indicated time. In a reliable standard collect the aggregation will start after the control structure receives the response of the third site. In an unreliable scheduled collect the aggregation starts at the scheduled date without waiting for all responses. In our example the third site takes too much time to respond, so the structure drops its late response.

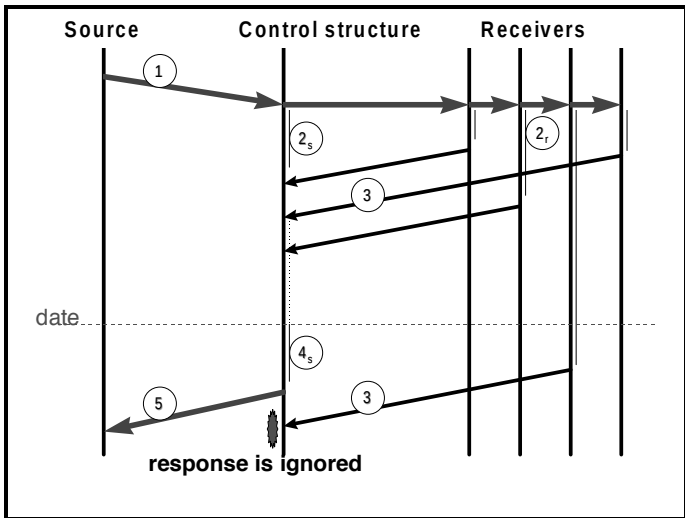

Figure 4 : Scheduled collect with a late response

Recall that a collect must be at least in one of the reliable and scheduled modes. A collect could also be both « reliable » and scheduled, but the reliability here is quite different from the reliability seen previously. Here aggregation starts either when all expected responses are received or when the scheduled time arrives. It also aggregates « non-response error ».

3.2. Data collected

The data requested by a collect is identified by a logical name at the application level. Upon receipt of a collect request each receiver tries to match data corresponding to the variable name (phase 2_r in

Selection Operator	Variable	Fixed operand
Equal	variable	value
Greater than	variable	value of threshold
Smaller than	variable	value of threshold

Table 1: List of selection operators

Figure 2 page 7). If it is unable to do it, it sends an error message to the control structure which counts the number of errors and aggregates it with the result.

The type of collected data can be integer or floating point. The source can join a threshold to the collect request to indicate which sites must respond (see Table 1). In this case each receiver checks the threshold and sends data in its response only if it satisfies the condition. This enables a collect to receivers participating to a response using application level criteria.

For example if the request is « *number of sites having an error rate greater than 10%* » the selection vector will be

```
(0, Number)
(0, Greater Than, ErrorRate, 0.1)
```

Example 1 : A "Number" Operation and its selector

where *Greater Than* is the operator, *Error Rate*, is the variable and 0.1 is the value of the threshold. The first vector indicates the aggregation operator as we will see in 3.3. The first parameter of each vector is the collect identifier, 0 in this case. Each requested member with an error rate greater than 10% should respond.

3.3. Aggregation operation

[CT96] proposes a capsule mechanism for active networks. Capsules could be a solution to provide aggregation in the network, or even to implement the collect mechanism. For simplicity we made the choice to think about a separate module, working at the transport level to manage the control structure, and the collect. But this module could be loaded into network components by different means, possibly allowing to dynamically load new aggregation operators.

Aggregation follows two main steps. First, immediate aggregation starts as soon as the control structure receives responses. The structure processes a response at the control entity which manages the receiver that issued it (i.e. as locally as possible). The structure does not store individual responses while it progressively aggregates data, except in the case of asynchronous collect.
Secondly, when all expected responses arrive at a control entity or when the delay expires, then the control entity decides to start global aggregation. Each entity waits for intermediate results coming from each of its child control entities; this is possible because each child entity is explicitly known. Then, the control entity aggregates these results with its own intermediate results to construct a new intermediate result message transmitted to its parent.

The control structure locally aggregates each response message as specified in the previously received collect request. Aggregation operations are those used by the control structure to compute one value with a large number of responses. We propose some aggregation operations: *Addition*, *Minimum*, *Maximum*, *Mean* and an operation that just counts responses, *Number*.

For example, if the source wants to know *the mean error rate for all receivers connected via a link with a 64 Kb bandwidth*, it issues the following request :

```
(1, mean, Error Rate)
(1, equal, Bandwidth, 64Kb)
```

Example 2 : Request of the mean of error rates

Here the first vector specifies that the aggregation operation *mean*, will be applied to the variable *Error Rate*. And the selection operator specifies that only receivers with a bandwidth equal to 64Kb, will participate to the mean calculation.

3.4. Result

Finally the root of the control structure sends the collect result message to the source. This message contains the result of the aggregation of all responses. Moreover, it indicates how many sites were expected by the control structure to respond, how many sites did respond and among those, how many satisfied the selection operation (Table 1). This is necessary to appreciate the result validity. The result message contains also aggregated information about error that occurred. For each type of error, it provides the number of such errors.
With all this information the source application can decide if the result is valid according to its own requirements.

4. The aggregation mechanism

In this section we will describe more precisely the aggregation process inside the control structure. There are three types of messages for a collect: the request from the source to the entire group (network level multicast group) including control structure entities; the response from each receiver to its neighbor control entity; and the result between control entities and to the source. Each of these messages contains vectors describing respectively a collect, a response and a result. Some other vectors may be added in response and result messages to describe errors.

4.1. Initialization

Upon receipt of a request a receiver begins to compute a response. It may store this response before sending it or compute it again if retransmission is requested. It must only store the identifier of the collect and the fact that it responded.
In case of an asynchronous collect, it must also initialize a data structure which contains information about collected variables and the selected updating strategy.

When a control entity receives a collect request, it initializes two main data structures. The first one will contain a description of the collect and intermediate results which will be updated when a response arrives. There is one such data structure for each running collect. The second data structure is a list of entities (groups members or child control entities) that are expected to send a response or a result. Group members are added to this list upon reception of either an explicit adhesion to the collect service, or a response to a collect request.
In case of asynchronous collect a third data structure will be initialized. It is the list of individual responses. Each response may be updated by the corresponding sender.

4.2. in-fly aggregation

When a control entity receives a response, it records in the list of linked member that this receiver has responded to this collect. If this member has already responded to this collect the message will be discarded. If not, the aggregation operator associated to this collect is applied to perform preliminary aggregation, and is update the intermediate result.

In case of asynchronous collect, aggregation is delayed until a result request. Because a receiver may update its data several times after the preliminary collect, the control structure must store individual data without any aggregation until a result request is received. Another solution would be to « rollback » the aggregation of the first data coming from this receiver before restart with the new data. In both cases the entity must store individual responses, so we choose the first solution to avoid the costly « rollback » operation. Therefore the control entity store all valid responses in the list of individual responses after checking for multiple responses.

4.3. aggregation

When a control entity receives a result message, it aggregates it with its own intermediate result, as it does with responses during the preliminary collect. In addition it must update the number of expected responses, the number of effective responses, and the number of responses used in the result message.
For example if the source requested a mean, a control entity receives a result message containing :

```
(1, 0.15, 7, 7, 4)
```

where the first number identifies the collect, the second is the intermediate result, the third, the number of expected responses, the next, the number of effective responses, and the last the number of responses used to compute the intermediate result.
The control entity aggregates it as follows

$$IntermediateResult = \frac{IntermediateResult \times UsedNumber + 0.15 \times 4}{UsedNumber + 4}$$

$NumberExpected+ = 7$

$NumberReceived+ = 7$

$NumberUsed+ = 4$

The time when a control entity decides to transmits upward its aggregated result depend on two factors: the type and mode of collect, and the number of results received.
Moreover if a control entity has child control entities, it must wait for their responses. Only then, it can issue a result to its parent. When a collect is reliable, the control entity must wait all responses from its directly attached receivers.

When a result is directly received by the source the collect is finished.

5. Conclusion and perspective

We outlined a mechanism called collect that allows the multicast source to be informed of the status of the receivers in an internet context. Basically we define a many-to-one response channel. We saw why a hierarchical structure is really suited to support this service. This allows a scalable service. If we consider a control entity having at most N children (receivers or child control entities), we observe that for one collect :

- The number of messages received by this entity is linear in N;
- The size of message is basically independent of group size;
- The amount of memory necessary is linear in N;
- The processing time is also linear in N.

Therefore resources necessary for our collect in any single component (link, control entity, receivers, source) is bounded by the out degree of any control structure entity.
Obviously a difficulty with our approach will be to automatically construct the control structure while limiting the out degree of each entity, independently of the actual group membership distribution over the network.

We plan to simulate FAGE and CAGE algorithms. We are currently developing a runable simulation model over a discrete event simulator called SAMSON [tou91]. This model will permit comparisons between FAGE and other reliable multicast protocols. We intend also to compare CAGE and RTP/RTCP ([SCFJ96]) information mechanisms. A difficult problem is to build the control structure and to decide where different control entities must take place in the network. The simulation will show the influence of the tree shape on FAGE and CAGE performances. We want also to evaluate the costs of each type of collect in terms of number of messages, amount of control structure resources necessary, and delays.

6. Bibliography

[AC95] **Azcorra A., Calderon M.**, *A Network-Based Approach to Reliable Multicast*, PROMS'95 (Salzburg - Austria), October 1995, pp. 245-256

[AFM92] **Armstrong S., Freier A., Marzullo K.**, *Multicast Transport Protocol*, RFC 1301, February 1992, 38 p.

[BFC93] **Ballardie T., Francis P., Crowcroft J.**, *Core Based Trees (CBT): An Architecture for Scalable Inter-Domain Multicast Routing*, in Proceedings of SIGCOMM'93, September 1993, pp. 85-95

[Bon96] **Bonnin J-M**, *Vers un service de diffusion fiable*, JDIR'96 (Paris - France), September 1996, 6 p.

[CJ96] **McCanne S., Jacobson V.**, *Receiver-driven Layered Multicast*, Proceedings of ACM SIGCOMM'96 (CCR:v29n4), October 1996, pp 117-130

[CT90] **Clark D., Tennenhouse D.**, *Architectural Considerations for a new Generation of Protocols*, SIGCOMM'90, September 1990, pp. 200-208

[Dee89] **Deering S.**, *Host Extensions for IP Multicasting*, RFC 1112, August 1989, 17 p.

[DEFJLW94] **Deering S., Estrin D., Farinacci D., Jacobson V., Liu C.-G., Wei L.**, *An Architecture For Wide-Area Multicast Routing*, in proceedings of SIGCOMM'94 (v24n4), August 1994, pp. 126-135

[DHT95] **Diot C., Huitema C., Turletti T.**, *Multimedia Applications should be Adaptative*, Proceedings of 3d IEEE Workshop on High Performance Communication Subsystems HPCS (Mystic - Connecticut - USA), August 1995, 9 p.

[Dio94] **Diot C.**, *Reliability in Multicast Services and Protocols ; A Survey*, International Conference on Local Metropolitan Communication Systems (Kyoto), December 1994, 19 p.

[FJCLZ95] **Floyd S., Jacobson V., Mc Canne S., Liu C., Zhang L.**, *A Reliable Multicast for Light-Weight Sessions and Application Level Framing* , SIGCOMM'95 (Cambridge - USA), August 1995, pp. 342-356

[HBC95] **Hofmann M., Braun T., Carle G.**, *Multicast Communication in Large-Scale Networks*, Proceedings of 3d IEEE Workshop on High Performance Communication Subsystems HPCS (Mystic-Connecticut-USA), August 1995, 5 p.

[Hof96a] **Hofmann M.**, *A Generic Concept For Large-Scale Multicast*, Proceedings of International Zurich Seminar on Digital Communication ISZ'96 (Zurich - Switzerland), February 1996, 12 p.

[Hof96b] **Hofmann M.**, *Adding Scalability to Transport Level Multicast*, 3^{d} International COST Workshop, November 1996, 15 p.

[HSC95] **Holbrook H., Singhal S., Cheriton D.**, *Log-Based Receiver-Reliable Multicast for Distributed Interactive Simulation*, SIGCOMM'95 (Cambridge - USA), August 1995, pp. 328-341

[Hui94] **Huitema C.**, *Le routage dans l'internet*, Editions Eyrolles, 1994, 420 p.

[Mon94] **Montgomery T.**, *Design, Implementation and Verification of Reliable Multicast Protocols*, Ph. D. Thesis of the University of Virginia (USA), 1994, 131 p.

[Par96] **Parnes P.**, *RTP extension for Scalable Reliable Multicast* , Work in progress: Internet Draft, March 1996, 7 p.

[Raj92] **Rajagopalan B.**, *Reliability and Scaling Issues in Multicast Communication*, SIGCOMM'92, August 1992, pp. 188-198

[SCFJ96] **Schulzrinne H., Casner S., Frederick R., Jacobson V.**, *RTP: A Transport Protocol for fiReal-Time Applications*, RFC 1889, January 1996, 75 p.

[TH96] **Turletti T., Huitema C.**, *Videoconferencing on the Internet*, in ACM Transactions on Networking (v4n3), June 1996, pp. 340-351

[Tou91] **Toutain L.**, *SAMSON : Un simulateur pour systèmes répartis et temps-réel*, Thèse de l'université du Havre, November 1991, 115 p.

[Tur94] **Turletti T.**, *The INRIA Videoconferencing System (IVS)*, in Connexion (v8n10), October 1994, 7 p.

[TW96] **Tennenhouse D., Wetherall D.**, *Towards an Active Network Architecture*, Computer Communication Review (v26n2), April 1996, pp 5-18

[WKM95] **Whetten B., Kaplan S., Montgomery T.**, *A High Performance Totally Ordered Multicast Protocol*, ?, 1995, 30 p.

[Yea93] **Yeadon N.**, *Supporting Quality of Service in Multimedia Communication via the use of Filters*, Technical Report of Lancaster University, December 1993, 18 p.

[YMGH96] **Yeadon N., Mauthe A., Garcia F., Hutchinson D.**, *QoS Filters: Addressing the Heterogeneity Gap*, Proceedings of IDMS'96 European workshop on Interactive Distributed Multimedia Systems and Services (Berlin - Germany), March 1996, 17 p.

[ZDESZ93] **Zhang L., Deering S., Estrin D., Shenker S., Zappala D.**, *RSVP: A New Resource ReSerVation Protocol*, IEEE Network Magazine, September 1993, pp. 8-18