Using Light-Weight Groups to Handle Timing Failures in *Quasi-Synchronous* Systems

Carlos Almeida
Instituto Superior Técnico - Universidade Técnica de Lisboa
DEEC, Av. Rovisco Pais - 1096 Lisboa Codex - Portugal
cra@digitais.ist.utl.pt

Paulo Veríssimo
Faculdade de Ciências da Universidade de Lisboa
Bloco C5, Campo Grande, 1700 Lisboa - Portugal
pjv@di.fc.ul.pt

Abstract

In a quasi-synchronous *environment worst-case times associated with a given activity are usually much higher than the average time needed for that activity. Using always those worst-case times can make a system useless. However, not using them may lead to timing failures. On the other hand, fully synchronous behavior is usually restricted to small parts of the global system. In a previously defined architecture we use this small synchronous part to control and validate the other parts of the system. In this paper we present a light-weight group protocol that together with the previously defined architecture makes it possible to efficiently handle timing failures in a* quasi-synchronous *system. This is specially interesting when active replication is used. It provides application support for a fail-safe behavior, or controlled (timely and safe) switching between different qualities of service.*

1. Introduction

In the last few years there has been a proliferation of computers and communication networks. This creates the potential for the development of new classes of applications. Applications that until now were only centralized are now intended to run in a distributed environment. This new infrastructure also creates the potential for the existence of a set of new services with new requirements. Issues such as fault-tolerance and real-time characteristics are examples of aspects that must be addressed in this new context.

Although these problems are reasonable well understood in the context of synchronous systems, they are very difficult to solve (if even possible) in systems that are not fully synchronous, which is the case of these new infrastructures. Some new communication network technologies (e.g. ATM [11]) have improved the synchronism properties of these distributed environments, however they are not always fully synchronous. They are at most what we call *quasi-synchronous* [26]. Only a small part of the system can be considered as synchronous. The rest has a more dynamic behavior exhibiting, for a given activity, worst-case delays that are much higher than the normal delays[1] (see Figure 1).

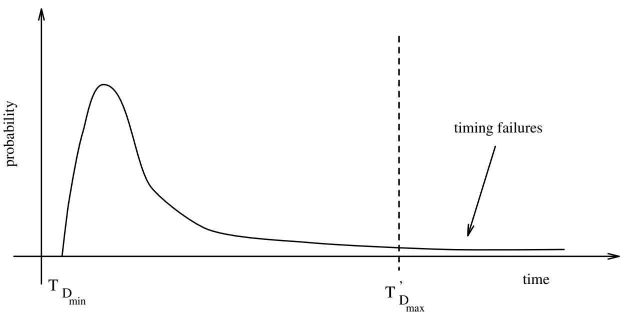

Figure 1. Distribution function for message delivery time (T_D) in a *quasi-synchronous* system. The worst-case is much higher than the normal case. Assuming as maximum a value closer to the normal case (due to practical reasons) increases the probability of having timing failures.

In this type of environment, the development of distribut-

[1] These delays can be related to execution if it is a CPU related activity, or transmission in the case of a network related activity.

ed fault-tolerant real-time applications is a difficult task. However, there is a demand for such applications, and therefore providing support for their development is of utmost importance.

We have been addressing these problems and in previous documents we have presented: the *quasi-synchronous* model [26]; group communication protocols that take into account the characteristics of these systems and provide message early-delivery [2]; and a timing failure detection service that has a key role in the proposed architecture [3]. In this paper we describe the protocols associated with the management of a hierarchy of groups, and explain how by using light-weight groups we can handle timing failures in a *quasi-synchronous* system.

The paper is organized as follows: in the next section we briefly describe our *quasi-synchronous* approach. In Section 3 we give a short reference to some related work. In Section 4 we present the group management protocols. In Section 5 we present and discuss some results obtained using simulation. The paper ends with the conclusions and some considerations about future work.

2. The quasi-synchronous approach

A synchronous system is one that exhibits known bounds on process execution times, message transmission delay and local clock rate drift. An asynchronous system is one where there are no such bounds. However, practical scenarios are not usually so well defined. Most systems, although not being fully synchronous, are not completely asynchronous either. They exhibit some form of synchronism. In this category are what we call *quasi-synchronous* systems [26].

A *quasi-synchronous* system can be modeled as if it was a synchronous system, in the sense that there are bounds on process execution times, message transmission delay and local clock rate drift, but some or all of those bounds are not precisely known, or have values that are too far from the normal case, that in practice one must use other values (closer to the normal case). In both cases it means that there is a non-null probability that the values we pick are not correct. This is a realistic scenario when there are situations of overload. Better synchronism properties are restricted to a small part of the system: a few high priority activities, and a small bandwidth channel for high priority messages.

Our approach is to use this small synchronous part of the system to build components able to control and validate the other parts of the system, thus making it possible to achieve safety in a timely fashion. This approach does not solve all timeliness problems *per se*, but can be used by applications to reach a safe state before stopping, or switch in a controlled manner between several different qualities of service. Together with group communication protocols and a hierarchy of group management it can also provide an efficient support for the use of active replication. As we show in this paper this approach can be very useful in the handling of timing failures. Using active replication to tolerate timing failures is possible if we can assume an independent failure mode. Although the assumption of an independent failure mode is not always realistic, there are many scenarios where it can be considered. This is what happens in some of our applications of concern. The distributed telecommunications information network database components suffer sporadic timing failures due to overload, caused by request arrival and local processing unpredictability. Making replicated requests may be a solution to mask these sporadic timing failures.

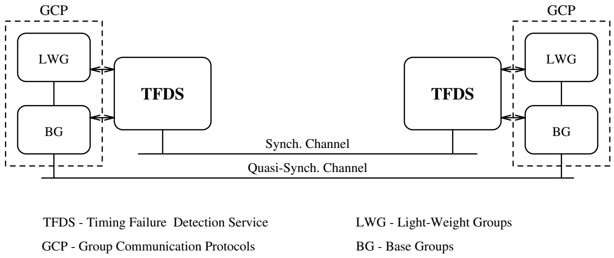

Figure 2. Use of the Timing Failure Detection Service

The proposed architecture is represented in Figure 2. It uses a timing failure detection service (TFDS) to disseminate with timeliness guarantees control information that is used by the communication protocols to validate their properties [3]. The implementation of TFDS may require the use of a specific network or a dedicated channel in a more generic network. We are exploiting several approaches. Depending on what is available and on the existing resources for a given application one may choose the approach that best fits in a given scenario. If an ATM network is available, for example, TFDS can be implemented using a channel with better guarantees than the channels used for normal communication.

Group communication protocols and group management are a powerful tool to support the development of applications that require the use of replication. Replication is often used as a way to increase availability and dependability. However, its existence implies replica management and communication support. This is specially important when there is a need for consistent dynamic updates in the presence of failures, including timing failures. This is a situation where our approach can be very helpful. By using TFDS as an oracle it is possible to detect timing failures (e.g. delays in updates) in a timely fashion. This in conjunction with a hierarchical structure for group management allows an efficient handling of timing failures.

In our architecture (see Figure 2) we use two layers for group management. At low level there is a basic group layer (BG) corresponding to traditional group management (a participant leaves the group when it wants, or when it crashes). On top of the BG layer there is a layer with light-weight groups (LWG). A participant may leave the LWG group even when the BG group is still operational. This makes it possible to efficiently enforce safety properties when timing failures are detected. With the help of TFDS we can *force* a *fail stop mode* related to timing failures. A given participant is only considered active (able to generate messages) if it is in a consistent state (up-to-date). Keeping the BG layer operational reduces recovery time because late messages can still be collected.

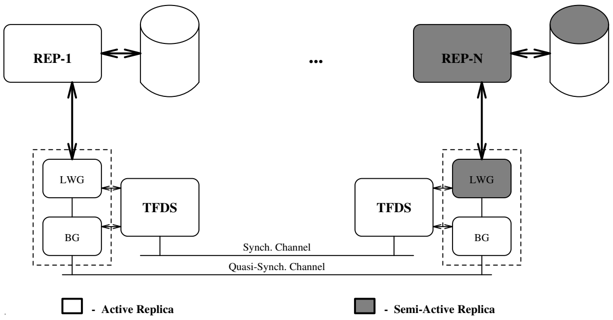

Figure 3. Use of TFDS and LWG in a distributed and replicated real-time database

The use of TFDS and LWG in a distributed and replicated real-time database is represented in Figure 3. The database application runs on top of TFDS and group communication protocols. When a given replica fails to receive the update in the specified time (detected by TFDS), it is temporarily disconnected from the group of active replicas (this is represented in gray in the figure). As there are two different group layers, only the upper layer is disconnected from the group when a timing failure is detected. The disconnection avoids consistency and contamination problems [21]. As the replica is not in a consistent state, it is prevented to respond to queries. However, keeping the low layer active, the replica may continue to receive updates thus minimizing the recovery time. This replica can be seen as a *semi-active replica*. As soon as the replica reaches a consistent state again, it may rejoin the group and become an active replica. Recovery can be done by means of state transfer, but, provided that there are some execution idle times between requests, it is faster to collect and execute late messages to catch-up. Message order can be enforced using the control information available from TFDS.

2.1. Feasibility of the assumptions on synchrony

The assumption of a channel with *better* characteristics is a reasonable assumption in scenarios usually available in real-time environments. Even in a real-time networks such as Token-bus, only high priority messages have hard real-time guarantees. In order to have guarantees for all messages, the configuration and network load must be completely controlled. In more general network architectures, it is normally possible to give guarantees to the highest priority messages. An example of such a network technology where different guarantees are given to different channels, is ATM (Asynchronous Transfer Mode) [11]. In this case it is possible to reserve channels with a given bandwidth and with a given level of guarantee associated with them. More recently a new type of network was standardized, called Iso-Ethernet [7], where traditional Ethernet and ISDN channels (Integrated Services Digital Network) coexist, with the latter having temporal guarantees. In some scenarios, it is also possible to have a dedicated network, as it is sometimes used for clock synchronization, for example. In the context of new telecommunications infrastructures, there are also examples of services that make our approach more viable, such as the short message service of GSM (SMS-GSM) and the foreseen message services of low earth orbit satellite constellations (LEO-Iridium)[9].

In a generic way, the fact that we only need a small bandwidth for the synchronous channel (the amount of information to transmit on the synchronous channel is much smaller than the amount of information to transmit on a normal channel), makes it easier to obtain. Note that there is usually an almost constant relation between latency and throughput in all networks. A low latency is usually obtained at the cost of having a decrease in throughput. When a high throughput is not needed, it makes it easier to obtain a channel with better temporal guarantees. In the limit, "a zero bandwidth" channel would provide very good temporal guarantees with probability 1!

Another aspect that we assume in this paper and we do not address here is clock synchronization. There is however, published work about this subject that show how that can be done in similar environments [27, 16]. Anyway, the precision and granularity of the clocks will only impose limitations on application time granularity not in the feasibility of the model, and the availability of a synchronous channel can make it easier to obtain.

3. Related work

The way we address some problems in our work has similarities with other works, even if in different contexts. For example, the idea of using early-delivery protocols, that we address in [2], appears in some other papers, and we used the work of Gopal *et al.* [20] as a starting point. In our paper we introduced extensions in order to be able to provide early-delivery to all participants of the communication group.

The use of a failure detection service together with the communication protocols, that we address in [3], can also be related to work done by the Isis Group [24, 8] to *solve* the problem of consensus in an asynchronous system [17]. The major difference between the two scenarios is that we want to reach agreement in a known bounded time and also detect timing failures. This puts some special requirements on the failure detection service – it must be implemented using a synchronous channel. Our timing failure detection service is also extended to provide the dissemination of control information.

Another work that deals with systems that are not completely synchronous is the one described by Dwork *et al.* in [12] about *partial synchrony*. However, they are mainly concerned with safety properties not related with time, assuming a liveness perspective, whereas we are deeply concerned with time under the perspective of timeliness. This essentially means having safety in a bounded time in a *quasi-synchronous* system. That is why we have to pick *a priori* a given value as the bound for message delay. In [12] the bound is not known or the point in time when it starts to hold is not known.

More recently there has been another work that has some similarities to our work from the point of view of goals [15]. That work is based on a model called *Timed Asynchronous* [10]. That system model assumes that processes have access to local clocks that although not synchronized have a bounded drift rate. The fault model assumes that processes may crash or experience performance failures and messages can have omissions or performance failures. On top of this model a datagram service is built that has knowledge about failures (*fail-aware*) [13]. It calculates an upper bound for message transmission delay using round trip delays and local clocks (not synchronized). Based on that calculus a message is classified either as *fast* or *slow*. This information can be used by upper layers to build, for example, a membership service able to define logical partitions [14]. This way it is possible to have applications with a fail-safe behavior [15].

Although with some similar goals (support for the development of real-time applications on environments that are not fully synchronous), the work just described is based on a system model (*Timed Asynchronous*) that is "more asynchronous" than our *Quasi-Synchronous* model. This way the qualities of service that can be offered in each environment are not exactly the same. Both models make it possible to build applications with a fail-safe shutdown. However, the Quasi-Synchronous model provides more flexibility for system reconfiguration. Because of the timely control information provided by TFDS, it is possible to reach an agreement before making a decision about the way that reconfiguration should be done.

The use of light-weight groups (that is the focus of this paper) is also something that has been used by other researchers [23, 19, 25]. However, they were used in a different perspective not related to real-time and the handling of timing failures.

Having group members that exclude themselves from the group upon failure detection is also used in [1]. However, they do not use a hierarchical structure for groups and the excluding members do so by crashing. Another difference is that in our case it is possible to reach an agreement before making the decision to leave or not the group.

4. Group management and timing failures

As we explained above, in order to handle the uncertainty inherent to a quasi-synchronous system we propose the use of a hierarchy of groups. At a low level, communication is addressed to base groups (BG) whose elements remain in the group as long as they do not crash. At a higher level there are light-weight groups (LWG). A LWG group is mapped to a BG group, however its elements are only considered as active elements as long as they do not experience timing failures. When a timing failure is detected (a message that is received late) the faulty element leaves the LWG group. At the BG group level that element is still operational and can collect late messages in order to have a fast recovery.

4.1. Basic model

Our protocols for the management of groups are supported on the Timing Failure Detection Service (TFDS). TFDS works as an oracle that periodically disseminates control information among all nodes containing group participants. Each node has a TFDS instance. The period of TFDS is Π_F and the latency is Δ_F. TFDS is implemented on top of a small bandwidth synchronous channel. We also assume synchronized clocks. Generic communication is done using channels with more relaxed synchronism properties where it is possible to have timing failures. Messages are supposed to be disseminated within a time interval $T_{Dis_{max}}$. If a message with timestamp T_m is received after $T_m + T_{Dis_{max}}$ there is a timing failure.

When agreement is required, a message is not immediately delivered upon reception. Instead, there is an additional latency increment in order to be able to make the decision about delivering or not. This increment is λ and is related to TFDS parameters: $\lambda \geq \Pi_F + \Delta_F$.

Messages are ordered accordingly to their timestamps T_m that are generated by the sender at send time using the current value of its clock (which is synchronized). Messages are kept in a delivery queue but are only delivered when there are guarantees about the order and agreement (depending on the quality of service desired). Those guarantees are obtained through the control information disseminated by TFDS. Early-delivery is provided if those guarantees are obtained before the deadline for delivery ($T_m + T_{Dis_{max}} + \lambda$). For a more detailed description of the early-delivery group communication protocols and TFDS, the interested reader is referred to [2] and [3].

The protocols for group management, both BG and LWG, also use the delivery queue and TFDS in order to achieve a consistent behavior. Membership changes are done in a consistent way by all participants and are ordered with respect to message delivery. Messages are delivered in the same view by all correct participants. This is related to virtual synchrony that was first introduced in the Isis system [6] and has been used since then by other researchers, sometimes with some modifications [18].

4.2. Light-weight groups

As we said above, timing failures are handled at LWG level and crash failures are handled at BG level. A timing failure will force a LWG participant to leave the group. After recovering by collecting late messages it will request to join again the group. Recovering can also be done by state transfer. However, collecting late messages is usually faster. So, in this paper we consider the state transfer only in the context of base groups. The procedure would be similar for LWG.

Leave

A timing failure occurs when a given message m with timestamp T_m takes more than the assumed maximum time $T_{Dis_{max}}$ to be disseminated to all participants of the group, i.e., there is at least one participant for which we have

$$T(reception) > T_m + T_{Dis_{max}}.$$

When this happens, in order to provide a fail-stop behavior related to timing failures, the participant that did not receive the message in time must leave the LWG group (switch from active to not active). Furthermore, the other participants must also remove the failed member from their list of active members (view). This operations must be done in a consistent way by all participants, and the "leave" operation must be ordered with respect to message deliveries: all participants see the same order.

This is achieved by doing that operation at message delivery time ($T_m + T_{Dis_{max}} + \lambda$). Note that messages are ordered by their timestamps and there is a total order. As $\lambda \geq \Pi_F + \Delta_F$, between $T_m + T_{Dis_{max}}$ and $T_m + T_{Dis_{max}} + \lambda$, TFDS will disseminate to all participants the information about who has received or not the message by $T_m + T_{Dis_{max}}$. This way, a consistent decision can be made by all participants, and faulty participants are removed from the view (membership).

Although the failure is due to the fact that a message m was not received until $T_m + T_{Dis_{max}}$, that failure is not effective until $T_m + T_{Dis_{max}} + \lambda$, the delivery deadline. Only at the delivery deadline an inconsistent state is reached when/if the other participants deliver the message. Between $T_m + T_{Dis_{max}}$ and $T_m + T_{Dis_{max}} + \lambda$ the participant can still be considered as correct because the missing/late message does not affect yet the "visible" state of that participant. Moreover, only at delivery time it is possible to have a consistent view of the situation.

At delivery time a consistent decision is made by all participants. A fail-stop behavior is enforced by the "failed" element itself by switching from active to not active. Note that the TFDS instance associated with the failed participant is still operational. As we are only assuming timing failures in the generic communication channel, TFDS is supposed to be full operational and so protocol correction can be ensured (crash failures are handled at BG level).

Messages in transit

It is possible to have messages that were already sent when a given participant fails, but were not accepted yet by all the other participants of the group. Those messages are supposed to be delivered and they must be taken into account to define order of delivery. TFDS provides the required information to achieve this. In the case of a crash, that situation is detected by not receiving the information periodically disseminated by TFDS. Such case implies not having early-delivery. It is necessary to wait the maximum time in order to make sure that a "flush" is done. As we do not have the information usually provided by TFDS, we do not know if there are messages in transit, and so we must wait to possibly receive those messages.

Join

When a given participant that was out of the group reaches a correct state again (from the point of view of order and agreement) it can request to rejoin the group. For that it can "send" a "join message" using TFDS. This "message" is timestamped as a regular message and inserted in the message queue accordingly to the timestamp. This way there is a "flush" of previous messages corresponding to the view where this participant did not belong.

When a given participant is in recovering mode, it can reach an up-to-date state by collecting late messages or by asking for a state transfer. For LWG we consider here only the first case. A state transfer would be handled in a manner similar to the one described for the base groups.

Messages with a timestamp greater than the timestamp of the request to join ($T_m > T_j$) are delivered after the join and need to have the confirmation from the new participant. Order is ensured by TFDS. Messages with a timestamp $T_m < T_j$ are delivered before the join and so do not have the confirmation from the new participant that is not an active member at that point.

All correct participants deliver a given message in the same view. They all know who accepts/delivers. However, a given message can be sent in a view different from the view in which it is delivered. A participant can leave a view before delivering a message sent in that view.

A given participant can deliver a message sent in a view to which it did not belong, provided that it had already request the join at that point. When the message was sent it was not an active member but it had already requested to join. The "join" will be "delivered" before the message.

In this mode of operation a sender does not know with accuracy at sending time which is the view in which the message will be delivered. All possible participants are potential elements of the view. Consistency is ensured by TFDS that remains operational even when a given participant (from a LWG) experiences a timing failure.

Locally, a participant only becomes an active member when/if it is in a consistent state. This behavior makes it possible to optimize the join by requesting it sooner.

4.3. Base groups

The protocol used for the management of base groups (BG) is also supported on TFDS. The control information is disseminated by TFDS with a period Π_F and a latency Δ_F. The protocol is intended to handle crash failures. The absence of a periodic message from the TFDS instance of a given participant p means that participant p has crashed. As TFDS uses a synchronous channel, the absence of information is assumed as a crash. Timing failures only occur in generic communication channels.

Leave

As explained above, the detection of a crash failure is done by not receiving the periodic information disseminated by TFDS. If by time $\Pi_F^i + \Delta_F$ the information related to period i is not received from a given participant p, it means that p has crashed. All correct participants detect this situation. However, they do not update their membership list immediately. This is necessary in order to ensure the correctness of the early-delivery protocols.

The exact moment of crash is not known with accuracy. It can be in the interval $[\Pi_F^{i-1}, \Pi_F^i]$. This means that it is possible that there are messages sent by the failed participant that are still in transit. The last one can have a timestamp $T_m = \Pi_F^i$. So, it is necessary to wait until $\Pi_F^i + T_{Dis_{max}} + \lambda$. This can be handled by inserting a fictitious message with timestamp Π_F^i when the failure is detected. This "message" will be "delivered" at normal delivery time ($T_m + T_{Dis_{max}} + \lambda$, being $T_m = \Pi_F^i$). This "delivery" corresponds to the indication of participant failure and respective membership update. During this period of time it may not be possible to provide early-delivery.

Join

TFDS has a reserved slot for each possible participant. This is used to request the join. When a participant restarts at time Π^0, it sends the request to join. At time $\Pi^0 + \Delta_F$ all correct participants know about the request. At this point the new participant is considered an "observer", not a full member. It starts to accept (without delivering) messages with timestamps $T_m > \Pi^0 + \Delta_F$. Other messages are discarded (they will be taken into account in the state transfer).

At time $T_i = \Pi^0 + \Delta_F + T_{Dis_{max}} + \lambda$ the group coordinator (for example the one with the lowest identifier) sends a state transfer to the joining participant. This time interval is necessary in order for the state transfer to take into account possible messages that were in transit ($T_m < \Pi^0 + \Delta_F$).

At time $T_i + T_{Dis_{max}} + \lambda$ the new participant updates its state based on the state transfer and on the messages it collected in the mean time ($T_m > \Pi^0 + \Delta_F$) and becomes an active member of the group. All other participants also update their membership list.

5. Discussion of results

In order to have a better idea about the behavior of the proposed architecture and protocols, we made some simulations corresponding to possible situations of target applications. In this section we present the results associated with the scenario where we use TFDS and light-weight groups to handle timing failures. A more complete study of several scenarios can be found in [4].

5.1. Simulation characteristics

The simulations that we present here were obtained using NETSIM [22], a simulator developed by the "MIT LCS Advanced Network Architecture" group. This simulator allows the specification of a network of components that exchange messages. It offers a graphical interface for visualization and parameterization of network topology and components. Essentially it works as an event manager. Events are generated by components accordingly to protocol implementations.

For generic communication data channels we used an Ethernet network at 10Mbits/s with support for multicast. We do not consider collisions but we consider transmission delay and medium occupation. Messages wait in queues until transmission is possible. Transmission duration takes into account message size. As we are mainly concerned with timing failures, we do not consider here omissions. That would imply the use of retransmissions in order to have reliability. We assume it included at this level. A specific treatment of this issue is not our main goal, that is why we did not address it here. From a practical point of view it would mean essentially an increase of message dissemination time. This means that the time values represented in the Figures [4-7] would be higher in a real setting. However, for our study relative values are more important than the absolute values. Absolute values will depend always on system and application time granularity.

Messages are generated in a random fashion following a Poisson distribution. For message size we used 1000 bytes. There is time associated with message processing. This time takes into account not only the time required to process the message but also possible CPU load related to other activities due for example to the existence of other applications. This situation corresponds to extra load that is independent of the message being processed and may differ between different processors. It is also modeled as a Poisson distribution. This aspect of load independence between different processors is important when considering active replication.

For TFDS we assume the existence of a synchronous channel with timeliness guarantees. The simulation of that channel if done in an independent fashion generating reception events directly on the destination accordingly to the period and latency considered for that service. For the simulations presented in this paper, we used $\Pi_F = 500\mu s$ and $\Delta_F = 100\mu s$.

5.2. Simulation scenarios

When application requirements implies agreement between all correct replicas about who has received or not a given message, system response time is conditioned by the time required for message dissemination and its worst case ($T_{Dis_{max}}$).

It is possible to still offer an early-delivery service, but if unanimity is wanted, it is always necessary to wait for reception confirmation from the latest replica. The value assumed for $T_{Dis_{max}}$ is the waiting threshold. If by that time (more precisely, $T_m + T_{Dis_{max}}$) there is a replica that has not received the message, then there are two options: the message is discarded; or the late replica is excluded from the group. In any case the decision must be made in a consistent way by all correct replicas (in the second option, late replicas are considered not correct and leave the group).

The maximum message delivery time ($T_{D_{max}}$) is, however, $T_{D_{max}} = T_{Dis_{max}} + \lambda$. The latency increase ($\lambda = \Pi_F + \Delta_F$) is necessary to have the information needed to make a consistent decision. This way, if from

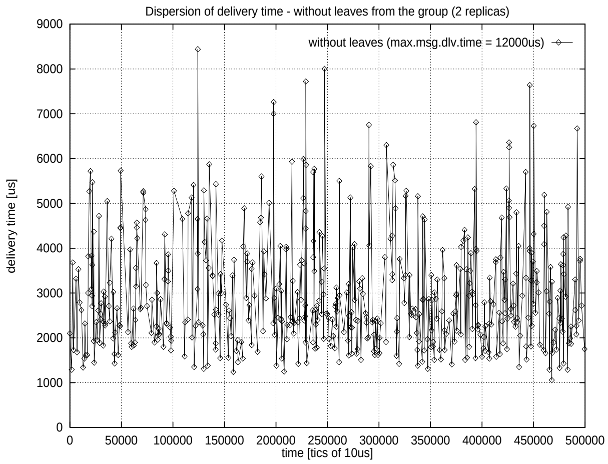

Figure 4. Scenario (a) - Active replication with updates (dispersion of delivery time). Use of TFDS and LWG. No leaves (maximum message delivery time = $12000\mu s$, 2 replicas).

the application point of view we want to select a given maximum message delivery time ($T_{D_{max}}$), then we have to parameterize the communication protocols so as to choose the right waiting threshold $T_{Dis_{max}}$. This value is obtained by subtracting $\Pi_F + \Delta_F$ from the application chosen value. For example, if the application wants a maximum message delivery time of, say, $T_{D_{max}} = 6000\mu s$, then (assuming $\Pi_F = 500\mu s$ and $\Delta_F = 100\mu s$) we will have:

$$T_{Dis_{max}} = 6000 - \Pi_F - \Delta_F = 6000 - 500 - 100 = 5400\mu s.$$

Thus, the waiting threshold to decide about the timing failure will be $5400\mu s$, in order to make sure that the maximum message delivery time is $6000\mu s$.

When timeliness is a key factor, the possibility of "ignoring" (temporarily disconnecting from the group) late participants makes it possible to reduce system response time. It is however important to have an efficient group management so as to avoid excessive overhead. This is achieved by the use of Light-Weight Groups (LWG) as explained before. When there is a timing failure (late reception of a message by a given replica), that replica leaves the group so as to avoid system inconsistency. However, it continues to be a member of the Base Group (BG) and keeps collecting late messages in order to have a faster recovery. This group management is done in a controlled way with the help of TFDS, as previous explained.

The simulation scenario that we present here is intended to show the effect of using LWG and TFDS as a way to improve system response time in a situation where there are dynamic updates in a set of replicas. This scenario is subdivided into four scenarios (a, b, c, d) in order to better

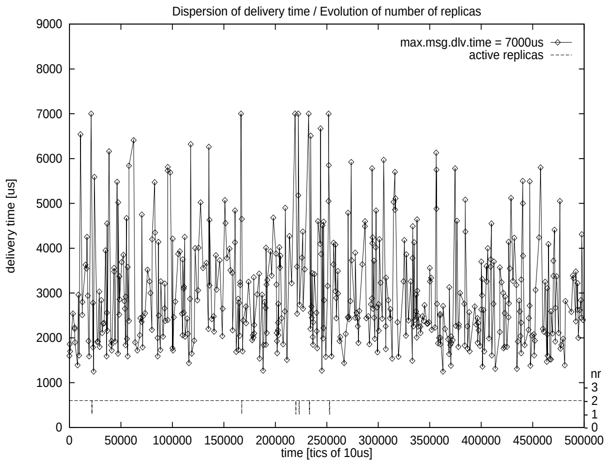

Figure 5. Scenario (b) - Active replication with updates (dispersion of delivery time). Use of TFDS and LWG. Leaves and joins (maximum message delivery time = $7000\mu s$, 2 replicas).

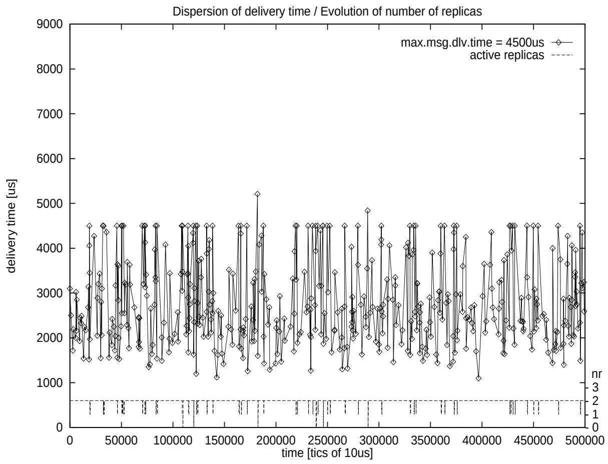

Figure 6. Scenario (c) - Active replication with updates (dispersion of delivery time). Use of TFDS and LWG. Leaves and joins (maximum message delivery time = $4500\mu s$, 2 replicas).

clarify the results obtained. This way we start by considering the situation in which group elements are stable, i.e. there are no leaves from the group. This happens because the value chosen for maximum message delivery time ($T_{D_{max}}$) is high enough to avoid being reached. Figure 4 corresponds to that situation. The value used for $T_{D_{max}}$ was $12000\mu s$ (implying $T_{Dis_{max}} = 11400\mu s$, used as threshold to define if a given replica experiences or not a timing failure). The figure represents the dispersion of message delivery time for a given time interval. It is a situation where there is active replication using 2 replicas, and it is necessary to keep consistency between them. It implies that the decision about delivering or not a given message can only be made after message reception by both replicas. As can be seen in the figure, although for most cases delivery time is between $1000\mu s$ and $5000\mu s$, there are other cases where it can reach, or even exceed, $8000\mu s$.

In order to reduce these delays we can assume a lower maximum message delivery delay, thus lowering also the threshold $T_{Dis_{max}}$, which forces the decision to be made early. The reduction of delivery time is obtained at the expense of excluding late replicas. This operation must be done in a consistent way (which is achieved with the help of TFDS) and furthermore there is a tradeoff between timeliness and number of active replicas. If the threshold is too low there is the possibility of excluding most replicas (in the limit, all replicas). Anyway, a careful use of this mechanism allows to obtain significant improvements in the system response time.

Figures 5 and 6 show the results obtained when maximum message delivery time $T_{D_{max}}$ is $7000\mu s$ and $4500\mu s$, respectively. The figures also show (bottom) the evolution of the number of active replicas (leaves and joins). As can be seen in Figure 5, by reducing $T_{D_{max}}$ to $7000\mu s$ makes it possible to have a delivery time always less or equal to that value. This is possible because when a replica is late it is excluded from the group, as can be seen in bottom part of the figure. This exclusion is usually very short because, as explained before, late messages are still received, allowing a fast recovery. The re-integration in the group (join) is done in a controlled way using TFDS.

As can be seen in Figure 5, for $T_{D_{max}} = 7000\mu s$, there are few situations where it is necessary to exclude a replica. This means that it might be possible to have an acceptable system behavior even with a lower threshold. In Figure 6 it is shown the results obtained when $T_{D_{max}}$ is $4500\mu s$. In this situation, system response time is reduced to values less or equal to $4500\mu s$ in almost all cases. However, there are a few cases where there are temporarily no active replicas. In those situations, it is not possible to guarantee the maximum message delivery time assumed. In Figure 6 there are two situations in which the message is delivered after the deadline. Delivering or not messages that are in these situations is optional. One can choose to discard the message and not deliver it to any replica. The final decision is application dependent. It might be acceptable to have a small percentage of messages being delivered with some delay (this is a typical scenario in soft real-time applications).

In cases where it is important to minimize the possibility of excluding all active replicas, one possible solution is to increase the number of active replicas. This way the probability of having all replicas late at the same time is

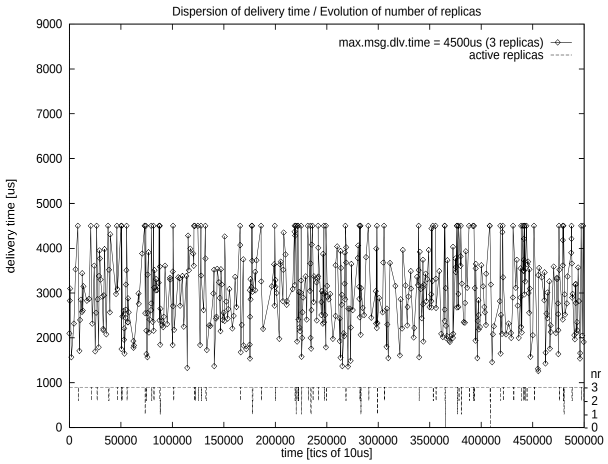

Figure 7. Scenario (d) - Active replication with updates (dispersion of delivery time). Use of TFDS and LWG. Leaves and joins (maximum message delivery time = $4500\mu s$, 3 replicas).

reduced. Figure 7 shows the results obtained when we use 3 replicas instead of 2. In this situation, for the same threshold ($T_{D_{max}} = 4500\mu s$), during the observation period, there were only two cases where all replicas received the same message late. (Even in that situation, the delivery was done before the assumed deadline.) These results show the potential of our approach.

5.3. Improvements and limitations

The results presented in this section show that by using the proposed architecture it is possible to improve in a significant way the behavior of a real-time system that runs in an environment that is not fully synchronous (quasi-synchronous). There are improvements both at system timeliness level and logical safety level.

By restricting synchronism properties to a few system modules, which makes its implementation viable in a larger setting, it is possible to build an infrastructure that supports the development of fault tolerant real-time applications in scenarios where is was not possible before.

It is not our goal to support the development of critical hard real-time applications on environments that are not fully controlled. It is our goal to use an approach that allows the improvement of real-time characteristics, and at same time improve the validation of situations where timeliness goals are not fully achieved.

In a generic way, we can consider two fundamental aspects in our approach:

1. Use of a best-effort policy at "normal" system components level trying however to introduce optimizations.
2. Add validation mechanisms that by having better synchronism properties make it possible to control the other system components, thus providing safety.

The first aspect *per se* improves the support for soft real-time applications. However, it is the second aspect that introduces a more significant improvement in the system. It makes it possible to support hard real-time applications provided that they have a fail-safe state. So, we use a best-effort policy to try to fulfill application timeliness requirements, but we also validate safety properties. If, by the assumed deadline, it is not possible to obtain the desired results, one can opt to stop in a fail-safe state (hard real-time), or one can relax timeliness requirements and wait for late results (soft real-time).

The characteristics of the validation mechanism that we refer above (TFDS) puts some restrictions on the type of environment where it is possible to obtain its implementation. The need of a synchronous channel in order to have timeliness guarantees limits its utilization. However, as we have already said, we only need a small bandwidth channel for control information, not generic data. New network technologies such as ATM (*Asynchronous Transfer Mode*) are also improving this scenario, thus making the use of our approach more viable.

6. Conclusions and future work

The work presented in this paper is part of a more generic architecture that aims at providing an infrastructure to support the development of fault-tolerant distributed real-time applications. Our target applications are mainly those with dynamic characteristics or those that run on environments that are not completely synchronous (*quasi-synchronous*).

In previous documents we have presented our *quasi-synchronous* model, group communication protocols that provide early-delivery, and a timing failure detection service that is a key component in the proposed architecture. In this paper we presented group management protocols supported on that previous defined architecture. They are structured in a hierarchical way providing a layer of light-weight groups. As shown in the paper, this is used to efficiently handle timing failures in a *quasi-synchronous* system. These results are very important to a new class of applications that is appearing due to the widespread use of computers and communication networks.

The results presented so far are mainly related to communications. In order to obtain a "full" system we need to also address similar problems related to processing. We are now addressing those problems in the DDRAFT system [5].

Acknowledgments

We would like to thank José Rufino for many helpful discussions and comments.

References

[1] T. Abdelzaher, A. Shaikh, F. Jahanian, and K. Shin. RT-CAST: Lightweight multicast for real-time process groups. In *Proceedings of Real-Time Technology and Applications Symposium*, Boston, MA, June 1996. IEEE.

[2] C. Almeida and P. Veríssimo. An adaptive real-time group communication protocol. In *Proceedings of the First IEEE Workshop on Factory Communication Systems*, Leysin, Switzerland, Oct. 1995.

[3] C. Almeida and P. Veríssimo. Timing failure detection and real-time group communication in *quasi-synchronous* systems. In *Proceedings of the 8th Euromicro Workshop on Real-Time Systems*, L' Aquila, Italy, June 1996. Also available as INESC technical report RT/20-95.

[4] C. Almeida and P. Veríssimo. Timing failure detection service: Architecture and simulation results. Technical Report CTI RT-97-05, Instituto Superior Técnico, Lisboa, Portugal, Dec. 1997.

[5] C. Almeida and P. Veríssimo. DDRAFT: Supporting dynamic distributed real-time applications with fault-tolerance. Technical Report CTI RT-98-02, Instituto Superior Técnico, Lisboa, Portugal, Feb. 1998.

[6] K. P. Birman. Virtual synchrony model. Technical report, Cornell University, July 1993.

[7] R. Brand. Iso-Ethernet: Bridging the gap from WAN to LAN. *Data Communications*, July 1995.

[8] T. Chandra and S. Toueg. Unreliable failure detectors for asynchronous systems (preliminary version). Technical report, Department of Computer Science, Cornell University, Ithaca, USA, July 1991.

[9] G. Comparetto and R. Ramirez. Trends in mobile satellite technology. *Computer*, 30(2):44–52, Feb. 1997.

[10] F. Cristian and C. Fetzer. The timed asynchronous system model. Technical Report CSE97-519, Dept. of Computer Science, UCSD, La Jolla, CA 92093-0114, January 1997.

[11] M. de Prycker. *Asynchronous Transfer Mode: Solution For Broadband ISDN (Third Edition)*. Number ISBN 0-13-342171-6. Prentice Hall, 1995.

[12] C. Dwork, N. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM*, 35(2):288–323, Apr. 1988.

[13] C. Fetzer and F. Cristian. A fail-aware datagram service. Technical Report CS96-498, Department of Computer Science and Engineering, University of California, San Diego, La Jolla, CA 92093-0114, November 1996.

[14] C. Fetzer and F. Cristian. A fail-aware membership service. Technical Report CS96-503, Department of Computer Science and Engineering, University of California, San Diego, La Jolla, CA 92093-0114, November 1996.

[15] C. Fetzer and F. Cristian. Fail-awareness: An approach to construct fail-safe applications. In *Proceedings of 27th Annual International Symposium on Fault-Tolerant Computing*, Seattle, Washington, USA, June 1997.

[16] C. Fetzer and F. Cristian. Integrating external and internal clock synchronization. *Journal of Real-Time Systems*, 12(2), 1997.

[17] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the Association for Computing Machinery*, 32(2):374–382, Apr. 1985.

[18] R. Friedman and R. van Renesse. Strong and weak virtual synchrony in horus. Technical Report TR-95-1537, Cornell University, Ithaca, USA, August 1995.

[19] B. Glade, K. Birman, R. Cooper, and R. Renesse. Light-weight process groups in the isis system. *Distributed System Engineering*, (1):29–36, 1993.

[20] A. Gopal, R. Strong, S. Toueg, and F. Cristian. Early-delivery atomic broadcast. In *Proc. of Ninth ACM Symposium on Principles of Distributed Computing*, Quebec City, Canada, August 1990.

[21] A. Gopal and S. Toueg. Inconsistency and contaminations. In *Proceedings of the Tenth ACM Annual Symposium on Principles of Distributed Computing*, pages 257–272, Montreal, Quebec, 1991.

[22] A. Heybey. The network simulator version 2.1. Technical report, M.I.T., USA, September 1990.

[23] D. Powell, editor. *Delta-4 - A Generic Architecture for Dependable Distributed Computing*. ESPRIT Research Reports. Springer Verlag, Nov. 1991.

[24] A. M. Ricciardi and K. P. Birman. Using process groups to implement failure detection in asynchronous environemnts. Technical Report TR 91-1188, Cornell University, Department of Computer Science, 1991.

[25] L. Rodrigues, K. Guo, A. Sargento, R. van Renesse, B. Glade, P. Veríssimo, and K. Birman. A dynamic light-weight group service. In *Proceedings of the 15th IEEE Symposium on Reliable Distributed Systems*, Niagara on the Lake, Canada, Oct. 1996. (also available as Cornell University Technical Report, TR96-1611).

[26] P. Veríssimo and C. Almeida. Quasi-synchronism: a step away from the traditional fault-tolerant real-time system models. *Bulletin of the Technical Committee on Operating Systems and Application Environments (TCOS), IEEE Computer Society*, 7(4):35–39, Winter 1995.

[27] P. Veríssimo, L. Rodrigues, and A. Casimiro. Cesiumspray: a precise and accurate global time service for large-scale systems. *Journal of Real-Time Systems*, 12(3):243–294, 1997.