TIMELY AND DEPENDABLE QOS ADAPTATION IN QUASI-SYNCHRONOUS SYSTEMS*

Carlos Almeida
Department of Electrical and Computer Engineering
Instituto Superior Técnico - Technical University of Lisbon
Av. Rovisco Pais
1049-001 Lisboa - Portugal
email: cra@comp.ist.utl.pt

Abstract

Most common distributed communication infrastructures are not fully synchronous. They are at most quasi-synchronous. Only a small part of the system can be considered as synchronous. The rest presents a more dynamic behavior where the duration of a given activity (processing or communication) is usually relatively small, but there is a non-null probability of taking a much higher value occasionally. In these type of environments, it is very difficult (if even possible) to always offer the desired timeliness properties. However, some applications need more than just a best-effort policy. They need at least the ability to adapt the offered quality-of-service (QoS) in a timely and consistent way.

In the quasi-synchronous approach, the small synchronous part of the system is used 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*,

*This work was partially supported by FCT, through Project POSC/EIA/56041/2004 (DARIO).

but can be used by applications to reach a safe state before stopping, or switch in a controlled manner between several different QoS. In this paper we explain how this architecture can be used to provide the infrastructure to build communication protocols that are able to dynamically adapt their offered QoS in a timely and dependable way.

Key Words

Real-Time Group Communications, QoS Adaptability, Quasi-Synchronous Systems, Fault-Tolerance

1. Introduction and Motivation

"It is not the strongest of the species that survive, nor the most intelligent, but the ones most responsive to change." – Charles Darwin

The last few years have watched a proliferation of computers and communication networks that create a distributed infrastructure where there is the potential for the development of several different classes of applications. Some of these applications have dependability and real-time requirements that are not easy to fulfill in these new infrastructures. Although the aspects related to real-time and fault-tolerance have been addressed, and are reasonable well understood, in the context of synchronous systems, they are much more difficult to solve (if even possible) in systems that are not fully synchronous, which is the case of these new infrastructures.

Although specific communication network technologies (e.g. ATM [1]) do provide synchronism properties, in a more generic case these distributed environments are not always fully synchronous. They are at most *quasi-synchronous* [2]. Only a small part of the system can be considered as synchronous. The rest has a more dynamic behavior exhibiting, for the duration of a given activity (processing or communication), a probability distribution function similar to the one that is represented in Fig. 1. Most of the time these durations are relatively small, but there is a non-null probability that they take a much higher value occasionally.

Real-time applications developed for this type of environment face the following dilemma: if they use always the real worst-case time, they may become useless because that time is far away from the normal time; however, not using it may lead to timing failures, which, if not correctly handled, may lead to system failure.

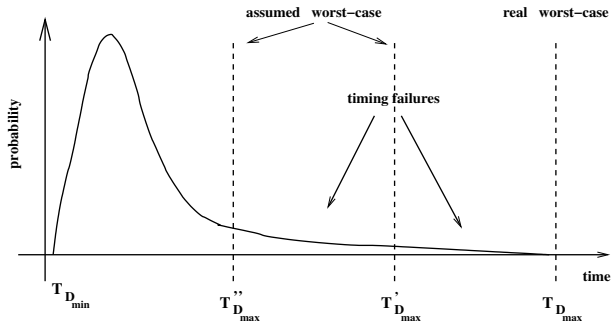

Figure 1. Distribution function for message delivery time (T_D) in a *quasi-synchronous* system. The worst-case is much higher than the normal case. Assuming as maximum a value closer to the normal case (due to practical reasons) increases the probability of having timing failures.

Although it may not be possible to provide full timeliness in this type of environment, it is desirable and important to provide some form of control to detect timing failures in a timely fashion and provide the mechanisms to ensure safety. Based on those mechanisms, applications can (if they want to, and if that makes sense for that particular application) adapt to the global system evolution, by switching between several different qualities-of-service (QoS) and thus providing a graceful degradation instead of a possible abrupt failure.

Being able to provide a timely and dependable adaptation is of utmost importance, even if full timeliness is not possible. This is more than a simple best-effort policy. It provides some guarantees. Although one might need to offer the possible service instead of the desired service, the adaptation is done in a timely and consistent way and avoiding inconsistency and contamination problems. For critical applications it may imply to stop the service, but it will be a stop in a safe state.

This paper is organized as follows: in the next section we explain what are the main problems to solve and briefly describe our *quasi-synchronous* approach. In Section 3 we give a short reference to some previous and related work. In Section 4 the handling of QoS adaptability is presented. The paper ends with the conclusions and some considerations about future work.

2. Main Problems to Solve and Approach

As explained above and represented in Fig. 1, in the type of environment under consideration the synchronism is not perfect. The best that is possible to achieve is to have a small part of the system that can be considered as synchronous, but the other parts exhibit a more dynamic behavior where it is possible to have a high variability in response time.

This high variability in response time makes it difficult to build real-time applications that are both efficient (from the point of view of timeliness) and safe. There is a trade-off between tight timeliness and timing failures. If, due to practical reasons, a "worst-case" is assumed that is smaller than the real worst-case, then the probability of having timing failures increases, as shown in Fig. 1.

The high variability in response time that is present in this type of environments is due to the fact that it is not cost-effective to use a resource adequacy policy (due to the high dynamic behavior of applications and/or environments) and so load is not completely controlled, being it possible to have overload scenarios.

Although this situation makes this type of environment not suitable for safety-critical hard real-time applications, there are other real-time applications that can still be built in such scenarios and offer an acceptable service, provided that there are validation mechanisms.

The basic idea is to try to provide optimizations, using a best-effort approach, but at same time introduce control mechanisms able to ensure validation in a timely fashion. These control mechanisms will provide the basis to achieve safety in a bounded time.

Having this type of mechanisms, applications can adapt in a controlled way to environment changes. It is, thus, possible to maximize the offered value by choosing different working envelopes. Applications can choose the best parameter values (e.g. thresholds / deadlines) that fit in a given working scenario. By using the control mechanisms referred above, these mode changes can be done in a bounded time and preserving safety. This allows a graceful degradation of QoS when it is not possible to ensure the full service. Being able to provide a dynamic adaptability makes it possible to avoid situations where the application would fail. For example, in an active replicated application, it may be possible to try to avoid a scenario where all replicas are excluded, which would mean service unavailability.

To deal with this problem we propose the use of the quasi-synchronous approach as a support to build the protocols to handle dynamic adaptation.

2.1. 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 usually not 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 can be called *quasi-synchronous* systems [2].

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. Tight 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.

In the quasi-synchronous approach, this small synchronous part of the system is used 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 QoS. Together with group communication protocols and a hierarchy of group management it can also provide an efficient support for the use of active replication. This subject was addressed in a previous document [3].

The proposed architecture is represented in Fig. 2. A timing failure detection service (TFDS), supported on a small bandwidth synchronous channel, is used to disseminate, with timeliness guarantees, control information. That control information is used by the communication protocols to validate their properties in a safe and timely manner [4].

The implementation of TFDS may require the use of a specific network or a dedicated channel

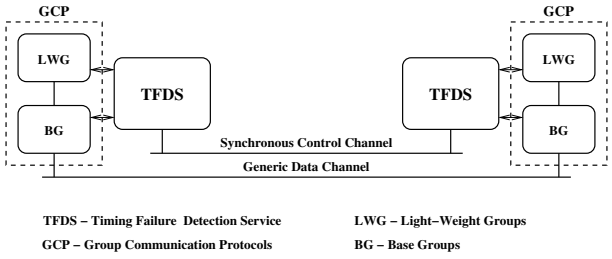

Figure 2. The quasi-synchronous approach architecture

in a more generic network. We can exploit 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. In other settings, one may need to reserve a small part of the global resources in order to be able to build a correct TFDS.

In this paper we explain how this architecture can be used to provide the infrastructure to build group communication protocols that are able to dynamically adapt their offered QoS in a timely and dependable way.

3. Previous and Related Work

We have been addressing these problems of having real-time in systems that are not fully synchronous. Besides presenting the quasi-synchronous model [2, 5], we have proposed group communication protocols that provide message early-delivery [6], and we explained the use of the timing failure detection service and its architecture [4]. In [3] we describe the protocols associated with the management of a hierarchy of groups, and explain how we can handle timing failures in a quasi-synchronous system by using light-weight groups. As we show in that paper, the quasi-synchronous approach can be very useful in the handling of timing failures in an active replication scenario, provided that an independent failure mode can be assumed.

However, when there is the possibility of having all replicas fail – and that is not acceptable for the given application – then some form of adaptability is required. One possibility is to adapt a specific parameter, for example the value used as threshold to decide if there is a timing fault or

not. We addressed that subject in [7]. In the present document we revise and extend that work.

Another possibility is to have a set of different protocols with different QoS to choose from. Besides providing a set of protocols with different QoS, those protocols can be built using a component based approach. Some preliminary work was presented in [8].

There are other works that have some similarities to our work from the point of view of goals. One such work is based on a model called *Timed Asynchronous* [9]. 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*). 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. This way it is possible to have applications with a fail-safe behavior [10].

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 we present in [3]) is also something that has been used by other researchers [11, 12, 13]. 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 [14]. 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.

More recently, there has been some work trying to unify both the Timed Asynchronous model and the Quasi-Synchronous model [15]. Presented as an extension to the quasi-synchronous model, they use a similar approach: a synchronous part of the system is used to control and to provide services to the other part.

Another line of research related to QoS control in adaptive real-time systems, is the one that tries to apply control theory to this problem. An example of such work is the one presented in [16]. They use feedback control to achieve the desired dynamic responses.

The subject of having QoS adaptability has received some attention in the last few years [17]. However, most of the existing work is more concerned with multimedia applications, dealing with delay and throughput adaptations, but without much concern with respect to the timeliness and safety of the adaptation procedures. In our case, we are supporting real-time group communication protocols where it is important that all group members have a consistent view of the system. This implies that the QoS adaptation must be done in a timely and safe way. We also use QoS in a more generic sense than it is usually used in the context of multimedia applications. For us, a given QoS can be defined based on the properties of the group communication protocols such as: order, agreement and timeliness, for example.

4. Handling QoS Adaptability

In [3] we used active replication and a hierarchy of groups (light-weight groups (LWG)) to handle timing failures and achieve a reduction in response time. This can be done when an independent failure mode can be assumed for the replicas. The timeliness improvement is achieved by assuming a "worst-case" dissemination time closer to the normal value (recall Fig. 1). Having independent replicas (failures do not occur at the same time), and with a careful management of the group (LWG are used to enforce a "fail silent" mode related to timing failures, thus avoiding inconsistency and contamination problems), it is possible to provide a reliable service.

In a highly dynamic environment, there is, however, the possibility of having all group members excluded from the group, if the chosen value for "worst-case" is too small, and there is an increase in the timing failure rate. In this situation, it is not possible to continue to offer the same QoS, and, if nothing is done, the system will fail, possibly in an uncontrolled way. To make sure that

safety is preserved, avoiding contamination problems, a consistent decision must be reached by all group members in a timely fashion, and a new safe state obtained.

There are several options to deal with the situation (see Fig. 3):

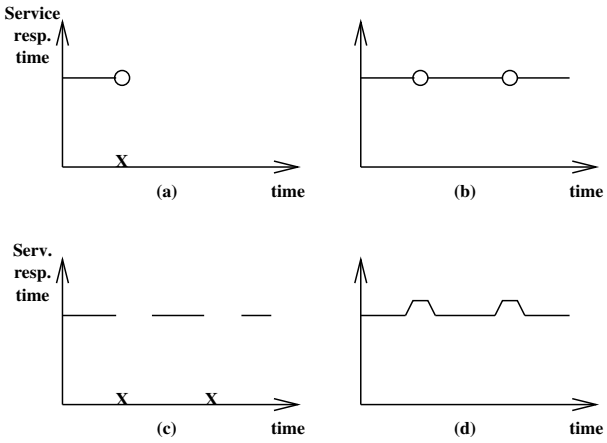

Figure 3. Several options for adaptability when timing failures do occur: (a) - stop in a safe state; (b) - discard message; (c) - stop, wait, join; (d) - adapt threshold

stop in a safe state – This corresponds to a hard real-time approach. Timeliness is very important, and thus a timing fault implies a system failure. The best that can be done is to ensure that there are no catastrophic consequences, that is, the system has a fail-safe behavior – when there is a failure it stops in a safe state.

discard late messages – This scenario corresponds to a situation where it is more important to keep service availability than to wait for late messages, provided that all group members have the same view of the system (consistency). In this case, if waiting for a late message would imply having all group members leave the group, a consistent decision is made for all members to discard that message and continue active.

stop-wait-join – In this case, all messages are important (or at least taken into account). One needs to wait for late messages, even if that means to have the service unavailable for a given period of time. The system stops (becomes temporarily unavailable), waits for recovery, and then restarts in a new consistent state. The recovery can be done waiting for late messages, doing state transfer, or through some form of compensation (forward error recovery).

adapt threshold – Increase threshold to avoid failure. In this case it is more important to keep the server operational. Instead of failing, we are willing to decrease temporarily the quality of timeliness thus avoiding the stop.

In all the above cases there is always a consistent decision. All group members make the same decision. And this decision is achieved in a timely fashion. This way, even when the desired timeliness is not possible, a timely and dependable QoS adaptation is ensured.

For example, the dynamic adaptation of the threshold must be done in such a way that the properties of the communication protocols are not jeopardized. It may be necessary to preserve order and agreement properties among group members while updating the threshold. Old messages, new messages and the changing control information may need to be causally related. The duration of this update operation must also be bounded.

All these integration aspects must be handled with care. This is where our quasi-synchronous approach plays an important role. TFDS is used to disseminate the control information, needed for the dynamic adaptation, in a timely fashion, thus making it possible to perform the update operation in a safe way.

4.1. Basic System Model

The basic system model is similar to what was considered in [3]. Our group communication and adaptation protocols 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 it 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 [6] and [4].

We also use the hierarchy of groups proposed in [3]. 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.

In this paper we extend that work to provide dynamic adaptation of the threshold ($T_{Dis_{max}}$) that is used to decide if a given group member is excluded or not from the group.

Timely info from TFDS

There is no specific restriction on the type of information to be disseminated by TFDS. The only limitation is on the available bandwidth. Otherwise, the communication protocol can specify a given amount of information that it wants all group members to receive. Besides group membership information, communication protocols may want to disseminate some control information related to message activity, as for example:

- messages received in $[T - T_{Dis_{max}},\ T]$
- messages sent in $[T - T_{Dis_{max}},\ T]$
- messages delivered in $[T - T_{Dis_{max}},\ T]$
- messages received after the deadline

With this information they will be able to make a decision accordingly to the QoS they want to provide.

4.2. QoS Adaptability

Let's assume, for now, that there are no crashes. If there are no crashes, TFDS is fully operational. Timing failures only occur in the generic channel, not in the TFDS channel, which is synchronous.

QoS adaptation (in this case modification of the time assumed as maximum message dissemination time, $T_{Dis_{max}}$) can be triggered in two different ways: automatically in response to membership changes; or triggered by the application. In the first case, it corresponds to have pre-defined values associated with the number of active replicas, for example, and the modifications are done at the same time as membership changes. It corresponds to a small modification to the membership protocol. The benefit of this type of service is to try to keep the number of active replicas more or less constant and thus avoid a situation where there are no active replicas. It is also faster to get the modification done, because we act immediately upon the membership change, instead of making a request after that event. On the other hand, by doing these modifications automatically, the timeliness properties of the communication protocols change without an explicit action from the application, which may be undesirable for some applications.

In order to have a better control from the application, a QoS adaptation triggered directly by the application is advised. This way it is also possible to do the modifications at anytime, even if there are no membership changes.

Independently of the way the parameter change is done, it is necessary to carefully address the situations where the modification of the threshold (both increase and decrease) occur while there are messages being disseminated. One must ensure that there is a consistent view by all group members.

As we said above, the modification of parameters can be triggered by a group membership change ("join" or "leave"), or by an explicit request for a new QoS. In the first case, the new parameters are related to the number of active members, and are pre-defined. In the other case, there is information associated with the request for new QoS. This request is "sent" as a "message" using TFDS, and disseminated and delivered to all group members. Its delivery is handled together

with data messages so as to preserve order and consistency among group members.

In any case, although the new QoS parameters become *available* at "delivery" time, one may need a *transition phase*. The "delivery" time of new QoS ($T_{Dlv_{nQ}}$) is the delivery time of the request for new QoS, or the delivery time of the membership change caused by the join or the leave events. The way the transition phase is handled depends on the type of parameter modification: increase or decrease.

Increase of Threshold

When the modification that we want to make is the increase of the threshold, so as to reduce the timing failure rate, we have a situation where we gain time to make the decision. We only have a problem if the threshold for a given message was already reached, and that message delivery was not done yet. In such case, we can not immediately increase the threshold for those specific messages, otherwise we could be in a scenario where some members would use the old threshold and others (late ones) would use the new one. We would be in a inconsistent state.

To avoid this situation, we can not make the update of the threshold immediately. If the deadline (threshold), considering the old parameter, has already passed, it is possible to have inconsistencies among group members. A late group member could receive a given message after the old deadline but before the new one, and assume it was correct. That would imply an inconsistency in the membership, because other group members would exclude it from the membership but it would not exclude itself.

In order to avoid this situation, the new parameter ($T_{nDis_{max}}$) becomes only effective for messages for which the following applies:

$$T_m + T_{Dis_{max}} > T_{Dlv_{nQ}}$$

otherwise the old parameter ($T_{Dis_{max}}$) is used.

The update must be done like this because: if the deadline has not arrived yet, then all group members will make the update (increase of threshold) and so order and consistency are preserved; but, if the deadline has already passed, then the decision is already in progress, and so, late members must use the old parameter to preserve consistency among all group members. They

can not assume they are going to stay in the group if other members are going to decide their exclusion.

Decrease of Threshold

When the modification corresponds to a decrease of the threshold, there is a reduction in the time available to make a decision. Some restrictions must be observed if one wants to preserve consistency. We may need to reduce less than desired during a transition phase.

As in the case of a threshold increase, a threshold reduction can only be done if

$$T_m + T_{Dis_{max}} > T_{Dlv_{nQ}}$$

that is, the deadline has not arrived yet. Otherwise, the old value must be used because the decision is already in progress, and so, a consistent decision must be done by all group members.

The reduction can only be fully done if

$$T_m + T_{nDis_{max}} > T_{Dlv_{nQ}}$$

that is, there is still time for all group members to use the new value ($T_{nDis_{max}}$). Otherwise, an intermediate value ($T_{xDis_{max}}$) should be used:

$$T_{xDis_{max}} = T_{Dlv_{nQ}} - T_m \ .$$

This temporary threshold ($T_{xDis_{max}}$) corresponds to an optimization. Although it is not possible to use already the new value ($T_{nDis_{max}}$), it is still possible to make a threshold reduction. As all group members have a consistent view of all deliveries, including the one corresponding to the new QoS ($T_{Dlv_{nQ}}$), they will all use the same threshold value in a consistent way.

4.3. Handling Crashes

The scenario with crashes is mainly addressed at Base Groups (BG) level, and the major consequence is that we may not be able to provide early-delivery. The crash is detected by not receiving the periodic information from TFDS. By not having that control information, we need to wait the normal dissemination time to possible receive messages that were already in transit. This is similar to what was done in [3].

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.

Base Groups 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.

Base Groups 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.

4.4. Other QoS

The desired properties (QoS) for a protocol are not always the same. It depends on applications characteristics. For example, to maintain a consistent state among all group members, may not be the most important issue. In some situations, timeliness may be more important than order or agreement, for example. In that case, instead of waiting for late messages to preserve the order of delivery, it may be more important to deliver messages as soon as they are received, provided that they are timely. If they are late, they are discarded. Based on properties such as timeliness, agreement and order, it is possible to provide a set of different communication protocols with different QoS, that applications can choose.

Another issue that can be addressed is the way protocol parameters are handled by group members, and its relation with messages. The thresholds that we are considering in the proposed protocols are valid for all messages. That is, there is only one threshold value at a given time. All group members use the same threshold, that is valid for all messages until a new one is obtained by a consistent modification. We could consider a different situation where the threshold would be message specific (for example, deadlines associated with messages). That situation may imply the existence of many thresholds at the same time, and poses additional problems in the case where besides deadline we also want to preserve order. But, depending on the relative importance of the properties referred above (timeliness, agreement and order) it may be interesting for some applications. We plan to address these issues in future work.

5. Conclusions and Future Work

In the new emerging distributed communication infrastructures, there is the potential and the demand to build new classes of applications with ever increasing requirements in what concerns dependability and real-time. However, the dynamic characteristics and temporal uncertainty due to the (lack of) synchronism properties of these environments prevent a direct utilization of traditional techniques to address those subjects.

To deal with this problem, being able to provide some form of adaptability is of utmost importance. That adaptability, to be done in a safe way, requires control information provided in a timely fashion. To obtain that, we use the quasi-synchronous approach, where a Timing Failure Detection Service, using a small bandwidth synchronous channel, is in charge of disseminating that control information.

This way, we create the conditions to have control over the dynamic adaptation of QoS. It is thus possible to have different application working envelopes that will be traversed accordingly to environment conditions. The transitions are done in a timely and safe way, allowing to take advantage of the best possible QoS available at a given time.

In this paper we explained how to handle the adaptation of a group communication protocol parameter: the assumed maximum message dissemination time, that is used as a threshold to decide when there is a timing failure implying the exclusion of a group member. This parameter modification (change of QoS) is done in a timely and consistent way by all group members, preserving safety.

This overall architecture creates the conditions to have classes of applications in these environments, that otherwise would not be possible.

As future work, we plan to continue to build a set of different QoS protocols. Although the work presented here is more communications oriented, we are also pursuing the adaptation of the quasi-synchronous approach to problems related to processing.

6. Acknowledgments

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

References

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

[2] Paulo Veríssimo and Carlos 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.

[3] Carlos Almeida and Paulo Veríssimo. Using light-weight groups to handle timing failures in *quasi-synchronous* systems. In *Proceedings of the 19th IEEE Real-Time Systems Symposium*, Madrid, Spain, December 1998.

[4] Carlos Almeida and Paulo 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.

[5] Carlos Almeida, Paulo Veríssimo, and António Casimiro. The quasi-synchronous approach to fault-tolerant and real-time communication and processing. Technical Report CSTC RT-98-04, Instituto Superior Técnico, Lisboa, Portugal, July 1998.

[6] Carlos Almeida and Paulo Veríssimo. An adaptive real-time group communication protocol. In *Proceedings of the First IEEE Workshop on Factory Communication Systems*, Leysin, Switzerland, October 1995.

[7] Carlos Almeida. Dynamic QoS adaptability in quasi-synchronous systems. In *Proceedings of the 14th IASTED International Conference on Parallel and Distributed Computing and Systems - PDCS2002*, pages 691–696, Cambridge, USA, November 2002. IASTED.

[8] Carlos M. R. Almeida. Component based QoS for real-time group communications. In *Proceedings of the IASTED International Conference on Parallel and Distributed Computing and Networks - PDCN2004*, Innsbruck, Austria, February 2004. IASTED.

[9] Flaviu Cristian and Christof Fetzer. The timed asynchronous system model. In *Proceedings of 28th Annual International Symposium on Fault-Tolerant Computing*, Munich, Germany, June 1998.

[10] Christof Fetzer and Flaviu 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.

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

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

[13] 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, October 1996.

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

[15] Paulo Veríssimo, António Casimiro, and Christof Fetzer. The Timely Computing Base: Timely actions in the presence of uncertain timeliness. In *Proceedings of the International Conference on Dependable Systems and Networks*, New York, USA, June 2000.

[16] Chenyang Lu, John A. Stankovic, Tarek F. Abdelzaher, Gang Tao, Sang H. Son, and Michael Marley. Performance specifications and metrics for adaptive real-time systems. In *Proceedings of the 21st IEEE Real-Time Systems Symposium*, Orlando, Florida, USA, December 2000.

[17] C. Diot. Adaptive applications and QoS guaranties. In *IEEE Multimedia Networking*, pages 27–29, Aizu (Japan), September 1995.

Biography

Carlos Almeida graduated in Electrical and Computer Engineering at Instituto Superior Técnico,

Technical University of Lisbon, in 1984. In 1989 and 1999, he completed, respectively, a Masters of Science and a PhD in Electrical and Computer Engineering also at the same University. Since 1985 he teaches in the Department of Electrical and Computer Engineering of Instituto Superior Técnico, where he is currently an Assistant Professor. He was a researcher at INESC from 1984 to 1996. In 1990-1991 he visited with the French company Chorus Systems (area of real-time operating systems), and from 1991 to 1993 he visited with Cornell University, USA, in the ISIS group (Computer Science Department). He has been participating in several national and international projects. His main research interests are: real-time and embedded systems, distributed systems, fault-tolerance and sensor networks.