UNIVERSITA' DEGLI STUDI DI MILANO
Dipartimento di Scienze dell'Informazione

RAPPORTO INTERNO N° 129-95

Comparison of Group Communication Protocols in a Faulty Environment

E. Pagani, G. P. Rossi

Comparison of Group Communication Protocols in a Faulty Environment

Elena Pagani
Gian Paolo Rossi

Technical Report
129/95

Dipartimento di Scienze dell'Informazione
Università degli Studi di Milano
Via Comelico, 39/41
I-20135 Milano (Italy)

ABSTRACT

Advances in high speed networks boosted the request for high level services suitable to support communications among grouped application processes. Although a number of solutions are available in literature, the identification of the protocol that better fits the given application requirements is still hard because of the lack of a unique problem statement and of a clear definition of the achievable quality of service. This paper provides a common framework of analysis, identifies the critical design issues and uses simulations to evaluate the role they play in coping with different failure conditions.

1 Introduction

Emerging applications, such as desktop conferencing, collaborative work, distributed real time control, are forcing the deployment of Common Application Services that are suitable to support group communications in a distributed environment [1]. Agreement [2], synchronization [3], reliable broadcast [4] and ordering of events [5] are just a few of the services that have to be provided among grouped application processes.

The key problem in the design of a protocol for group communication is to ensure the service in spite of the failures occurring at both the nodes (crash or permanent failures) and the network links where packets can be dropped (omissions or transient failures) [6]. Although the problem has been extensively studied from both the theoretical and practical point of view and a number of solutions are available in literature [4, 7, 8, 9, 10, 11], the identification of the protocol that better fits the given application requirements is still hard because of the lack of a unique problem statement and of a clear definition of the achievable quality of service. This paper aims to sketch a common framework of analysis, identifies critical design issues and uses simulations to evaluate different solutions. We specialize these general aspects to the Atomic Broadcast (AB) problem and we present simulation results for three AB protocols described in literature [4, 7, 8].

Throughout the paper we do not focus on typical performance indexes, such as throughput and delay, that result to be of limited interest; in fact, the paper shows that most of the existing protocols have equivalent performances, at least when normal working conditions are considered, while some design choice differentiates the protocols in terms of quality of service and fairness. Simulations are used to characterize the protocol behaviours and to analyse in some detail the protocol capability to react or to adapt to changing network conditions as a consequence of variable traffic and load. This represents a frequent condition to face with, for instance, in the Internet or when multimedia applications are in use among group of processes.

2 Background

In this paper we consider a group of n processes $p_1, \ldots, p_n$ that use a communication network for exchanging messages according to a group protocol. To allow the count of the amount of messages the protocol requires, n messages are counted for each broadcast. For sake of simplicity, we initially assume that processors have synchronized clocks and that the network guarantees a bounded and known transmission delay. As a consequence, the process communications proceed in *rounds*, each consisting of message transmission, reception and local processing, and roughly lasting half round trip delay.

The synchronous model does not introduce any limitation in describing practical systems; it properly applies to LANs of homogeneous workstations, where the transmission delay ranges within a small time interval and processors can run a clock synchronization protocol (e.g. `timed` for 4.3BSD systems [12]). Moreover, it has been proved that group protocols suitable for synchronous systems can be adapted to operate in a partially synchronous environment in which distances, load conditions and different technologies make variable both the message transmission and the processing performance [10]. Simulations are used to evaluate protocols under variable system conditions.

In such distributed system, failures affect both the communication network (transient failures) and the processors, where the group processes are running (crash or permanent failures). The key problem is to ensure the group service in spite of failures.

By placing the problem in the frame of a layered architecture, the transport layer is in charge of coping with network failures, while the upper layer provides the group service among processes. The two layers approach has been followed by K.Birman in designing the ISIS multicast primitives [7] and in the AMp protocol [11], and derives from traditional point to point communications. It implements an abstract layered model in which a specialized module, called the weakest failure detector (WFD), is responsible for detecting and recovering all of the transient network failures, thus reporting to the upper layer a perfectly reliable subnetwork [13]. Whenever an error is reported through the interface, it can only be generated by the crash failure of the process (processor) involved in the communication. In this case, the upper group entities are supposed to continue operation by running a group membership protocol to remove the crashed process from the group. Under these hypotheses, the group protocol results to be very simplified by facing only with crash failures. As a counterpart, some architectural costs derive from the amount of the active communication entities and from the communication between two adjacent layers.

Unfortunately, all the implementations of the WFD are unable of totally removing transient failures; in fact, reliable transport protocols use retransmissions, i.e. temporal redundancies, to cope with transient failures and a t-error is reported when excessive errors affect the communications. As a consequence, a process can be erroneously considered crash and removed from the group just because the repeated network failures are eventually ascribed to it. This allows to introduce an alternative approach that, more pragmatically, directly ascribes both permanent and transient failures to processes. In literature (e.g. [4, 6]), it is referred to as the general omission failure model and can be defined as follows:

General Omission: a process fails crash, i.e. it prematurely stops executing the protocol, or it omits to send or to receive some message.

According to the original definition of the general omission model, as soon as a process p omits a single message it is considered as being a faulty process. With

the aim of modeling practical systems, some algorithms assume that a process that omits a message can be able of properly following the protocol in the next round. Processes affected by such transient failures are often called *erratic* processes [14].

In both the presented approaches, the excessive transient failures can be misinterpreted and lead to erroneously remove a process from the group; however, the design of a protocol for general omission, *i*) leads to a single layer implementation that embodies transport functions, thus reducing the architectural costs, *ii*) allows to provide reliable group services directly on top of a datagram subnetwork and, *iii*), the derived protocol presents different capabilities of tolerating failure conditions, as described in the next sections.

3 The Problem Description

Group communication is a general term to refer to a set of group services that are strictly related one another. Among them, the Reliable Broadcast [6] provides reliable message exchange among the members of a group. It can be defined as follows:

Definition 3.1: *The processes p_1, ..., p_n process the messages in such a way that the following conditions hold:*

Validity: *If a correct process sends a message m to the group, then all the correct processes eventually process m.*

Agreement: *If a correct process processes a message m then eventually all the correct processes process m.*

Integrity: *For each message m, each correct process processes m exactly once, and only if some process sent m to the group.*

Termination: *If a correct process receives a message m eventually all the correct processes process m.*

The Reliable Broadcast is the simplest version of the agreement problem. By the definition, two or more processes may exist that process the same set of messages but in a different order. According to the application requirements different ordering semantics can be added on top of the Reliable Broadcast [6]; throughout this paper we will consider the atomic ordering. The definition of the Atomic Broadcast problem is obtained by adding the following property to the above definition of Reliable Broadcast:

Atomic Ordering : *for every two correct processes p and q, p processes a message m' after a message m if and only if q processes m' after m.*

The Atomic Broadcast service ensures that messages are delivered in the same order to all the group members or to none of them. It has a wide spread of applications such as the ordering of events in a distributed control system or the achievement of consistency in updating multiple copies of data. Moreover, the problem has been proved to be equivalent to the Consensus problem [6, 15] in

synchronous systems, with its variants Multivalue Consensus [15], and Set Consensus [16].

3.1 The Uniformity Requirement

Group applications often require that the property of *Uniformity* is ensured [6, 17]. When applied to ordering problems the *uniformity* guarantees that if a process, either correct or faulty, decides on some order for a message, then all the correct processes in the group decide on the same value, while the faulty ones either decide with the group or do not decide at all. A uniform decision avoids to leave group partitions in inconsistent states and ensures that the changes the processes produce on the external environment survive failures and recoveries. This is typically needed in real time systems, where inconsistent actions on the controlled world may have critical effects, but it has also a general meaning; e.g. it can ensure that, when reliable updates on replicated data are performed, any client can receive the same reply from any server it enquiries.

4 Protocol Design

Despite the large amount of protocols for group communication, it is very difficult to fairly compare their behaviours to satisfy some given application requirements. In the sequel we highlight the main design issues that, although specialized for the atomic broadcast problem to allow comparisons throughout the paper, have a wider applicability.

4.1 Centralized Control

Group protocols may be designed according either a distributed or a centralized control. Most of the existing algorithms (e.g. [11, 9, 7, 4, 8]) adopt the latter approach because it allows to generate an amount of messages that linearly grows with the group cardinality. However, when the process having the control fails crash, the slave processes execute another group protocol to elect the new *coordinator*. The *rotating coordinator paradigm* is often used to overcome this problem; cyclically the group members are forced to act as coordinator according to a well known order and for a fixed amount of rounds.

4.2 Resilience of the algorithms and group membership

The resilience of an algorithm indicates the maximum amount of failures it can tolerate still behaving properly. In [17] it has been proved that an agreement protocol can tolerate at most $n/2 - 1$ general omission failures, to ensure that uniform decisions are taken; this approach is adopted for example in [11, 9, 4, 8]. If

more failures occur, the authors prove that the algorithm is unable of dealing with network partitions and, in the worst case, each process may terminate with different decisions.

A higher resilience of $n-1$ is allowed when only crash failures occur. This means that when adopting the two layer approach, as introduced in section 2, the resilience of $n-1$ can coherently be used, as the ABCAST protocol [7] does; however, this could not be sufficient in practice because processes being supposed crash are actually still alive and capable of getting wrong decisions. As a consequence, the protocols that follow this approach are capable of tolerating a very high amount of failures but unable of ensuring uniform termination, as already pointed out in [18].

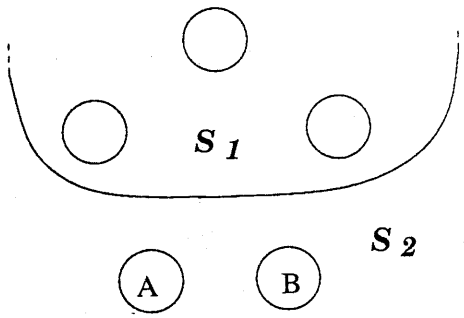

Figure4.1: Example of network partition.

These considerations can be motivated with the aid of the following example. Let us consider the 5 processes of Figure 4.1 and assume that processes in the subsets $\mathcal{S}_1$ and $\mathcal{S}_2$ are temporarily unable to communicate because of failures. If the resilience of $n/2-1$ is adopted, the processes in $\mathcal{S}_1$ represent the majority and can consistently terminate and proceed to a new decision, while processes A and B cannot. On the contrary, if the resilience is $n-1$, processes belonging to the different subsets can reciprocally consider the remaining as being crashed and autonomously decide on a different value. In both the approaches, protocols use the group membership mechanisms to resize the group cardinality to fit the set of processes with the same decision. When $t \leq n/2-1$ only one group survives failures while the remaining may either commit suicide or remain undecided. Undecided processes could be eventually recovered by maintaining the history of the taken decisions [8, 9]. When $t \leq n-1$ two groups are created and the original group might progressively degenerate into several fragments. In this case, process rejoin is affordable only through a voting mechanism amongst the group fragments; this can involve roll-back of a part of the processes. Furthermore, a coordinator process that repeatedly fails in receiving packets can force the group to remove a correct process.

Since the protocols follow different mechanisms to eliminate processes supposed faulty, it is interesting the evaluation of the total amount of processes that terminate with a consistent decision. We have evaluated this point by means of simulations; a detailed discussion is given in section 6.

4.3 Quality of service

The Atomic Broadcast problem definition, given in section 3, is very general and applies to most of the existing algorithms. However, the algorithms provide different quality of service as the consequence of their different interpretations of process correctness. A class of algorithms (e.g. those presented in [4, 10]), mainly derived from a theoretical approach, ensures uniform termination only for the set of correct processes that never failed, neither crash nor transiently (omissions), during the whole algorithm execution. By adopting the resilience $n/2-1$, at least $n/2+1$ correct processes exist and terminate with a common decision. Since a number of non crashed and still active processes exist that are suffering transient network failures, the algorithms in general adopt proper mechanisms to enlarge the amount of processes that uniformly terminate. The remaining processes are either crashed or undecided.

Another class of algorithms accepts that all of the group members can be dynamically affected by transient failures [7, 8, 19]. This makes harder to satisfy the uniformity property, thus reducing the provided quality of service. For instance, in [8] up to $n/2-1$ failures are tolerated per round and, at the end of one protocol execution, all of the group processes may have incurred some failure. The algorithm ensures that a round exists in which at least the majority of the group members agrees on the same value and that this set includes the correct processes that might exist. As previously described, the group membership forces to remove faulty processes and to agree on the set of processes that have the same decision. Among them the uniformity is ensured; however, the algorithm is unable of avoiding that one of the excluded processes processed a wrong decision before being forced to die. This limits the application of these algorithms in real time systems where a decision might act upon the controlled environment. The same considerations apply to [7]; however, in this case, as described in the previous section, the adopted resilience prevents from providing any form of uniformity since partitions holding different decisions might be created.

4.4 Recovery from Failures

The design of a group protocol combines the decision activity that provides the group service, e.g. define the processing order of a given message, the group membership and the mechanism to recover erratic processes. The group membership and the recovery can either be: *i*) blocking, i.e. a new decision is not initiated until all the members of the group have terminated the current one, either by deciding or by leaving the group [7, 11], or *ii*) non blocking, i.e. these activities are concurrently executed [8, 9]. The non blocking approach is affordable, for example, by forcing the processes to maintain the *history* of the past decisions; a process can recover from the most updated one. When a history is used, the higher throughput is paid with the problem of purging stabilized decisions from the histories and with the

occupancy of memory space. Whether adopt the blocking approach or not depends on, for instance, the expected message arrival rate; when it is slow, blocking or not are equivalent.

5 The considered AB protocols

Three of the AB protocols available in literature have been evaluated and compared by means of simulations. They have in common a centralized control, while they differ for the interpretation of process correctness and for other design choices as sketched in Figure 2. The protocol described in [4] solves the Reliable Broadcast problem. We have adapted it to solve the Atomic Broadcast problem by simply forcing the processes to execute only one instance of the protocol at a time, thus obtaining a total ordering among the processed messages.

Protocol	Resilience	Transport	Recovery	Uniformity	Failures
ABCAST	$n-1$	YES	Blocking	no	crash
Toueg's	$n/2-1$	NO	Blocking	all	crash/gen.om.
urgc	$n/2-1$	NO	Non Blocking	active	general om.

Figure5.1: Summary of the algorithms.

The ABCAST Protocol The algorithm, [7], solves the Atomic Broadcast problem. It uses the *flush* protocol to update the group membership when a crash is suspected by the underlying reliable transport protocol. The *flush* protocol blocks the decision activity. The algorithm has a resilience of $n-1$.

ABCAST processes communicate asynchronously and a decision is initiated only when at least one process has a message. To facilitate comparisons, we supposed that the group processes have backlogged messages to order, thus allowing to consider communications as being roundized.

While executing a *flush* protocol, a new *flush* instance might be nested as a consequence of continuous network failures, thus leading to up to $n-1$ *flush* concurrent executions; the protocol is capable of installing the group view obtained by the intersection of the groups generated by all the *flush* instances. In our simulations we obtained the same result by executing a single *flush* protocol and by properly constructing the group. Furthermore, we assumed that the transport service calls are blocking. This can affect the observed throughput of the protocol; anyhow, we do not focus on this performance index, thus making uninfluent this limitation.

Chandra-Toueg's Protocol In [4] different protocols for Reliable Broadcast have been presented in different failure models; we considered the one in the general omission failure model and we adapted it to perform the Atomic Broadcast. The algorithm has a resilience of $n/2-1$ lasting the whole execution, i.e. no more than $n/2-1$ processes can fail; if a process p fails at round r, from that round on it is considered faulty. The algorithm uses the rotating coordinator paradigm; each phase lasts 7 rounds and guarantees the uniformity.

We implemented the optimized version of the algorithm that introduces a pipelining mechanism to concurrently execute 4 phases.

The description given in [4] does not face with the problem of the exclusion of crashed processes. To simulate practical conditions in which a number of decisions are sequentially taken, we added an agreement algorithm that allows processes to agree on the group composition. It is run in between two consecutive decisions to avoid to forward messages towards crashed processes. However, the cost of this algorithm has not been considered in our measurements.

The *urgc* Protocol The protocol in [8] solves the atomic broadcast problem in presence of general omission failures. The algorithm uses the rotating coordinator paradigm and makes use of a history of past decisions to recover erratic processes. It has a resilience of $n/2-1$ per round, and the termination is guaranteed if a correct coordinator eventually exists. The group resize does not block the decision activity, unless failures affect a coordinator. In this case, the ordering is suspended until the group has been reconstructed and the processes have agreed on the most recently ordered message. The purge of the *stable* messages exploits the same mechanisms adopted to provide the group service and are concurrently executed.

6 Comparison

We used simulations to compare the behaviours of the 3 algorithms in the frame of the system and failure model described in section 2. We focus on the amount of processes that correctly terminate the algorithm, the exchanged messages and rounds to terminate. We also discuss the capability of the protocols of coping with variable network conditions.

6.1 Simulation conditions

Simulations assume that processes operate in a peer-to-peer group architecture, i.e. message senders belong to the group. We also assumed that the round, i.e. the time unit in which communication proceeds in *urgc* and in Toueg, and the time-out, that is used by the ABCAST to initiate a retransmission, last the same time. The parameter k indicates either the number of rounds spent by the *urgc* algorithm before deciding a crash failure or the amount of retransmissions used by

the ABCAST. The protocol [4] does not use such a mechanism to decide on the crash of a process. Different values of k have been used throughout simulations to evaluate the protocol sensitiveness to the parameter for a given failure pattern.

The overheads due to local memory management or to protocol architecture have not been considered. Furthermore, the protocols are forced to order only one message a time; this allows to fairly compare the protocol given in [4], while both ABCAST and *urgc* would support multiple decisions.

While executing simulations for the ABCAST, we prevented failures from affecting the coordinator process, thus avoiding the implementation of the election algorithm.

Crash failures have never been simulated, since all the protocols can effectively detect them.

All the simulations have been made with a group of 25 processes; the shown results are averaged on 50 decisions. The omission failures have been reproduced by means of a uniform distribution and when reporting 0.2% of omission failures we mean that each process has probability 0.1 of failing in send omission and probability 0.1 of failing in receive omission. The duration of transient failures is computed according to a β distribution.

6.2 Simulation results

Under the above conditions we observed the amount of generated messages and the number of correctly terminating processes at different failure rates. Results for the three considered algorithms are reported in Figure 3 under the hypothesis that the majority of correct processes is static for the whole protocol execution. Results for the ABCAST and *urgc* are reported in Figure 4 under the condition of having a majority of correct processes that dynamically changes, at the rate generated according to a β distribution, as the consequence of transient failures. Figures 3 and 4 report averaged results from several simulations an show that the protocols present different behaviours only under extreme failure conditions.

When few failures occur the protocols decide in a very short time; 3 or 4 rounds are required by *urgc* and Toueg's algorithms, while ABCAST is even faster with 1 or 2 rounds in average. The Toueg's algorithm pays this performance with a high amount of exchanged messages (due to the pipelining mechanism). No partitions are observed with the ABCAST. All the processes correctly terminate the protocol in spite of the growing rate of failures up to the 5%. When failures increase, ABCAST is penalized for starting the *flush* protocol and progressively reduces its performances.

When moving to a partially synchronous environment in which different workstation technologies and traffic conditions may introduce large variances in the observed transmission delays, communications may randomly slow down, thus making critical the definition of the round width and of the amount of retries (k). If they are wrong sized, quite a number of transient failures might be observed; if the slowing condition

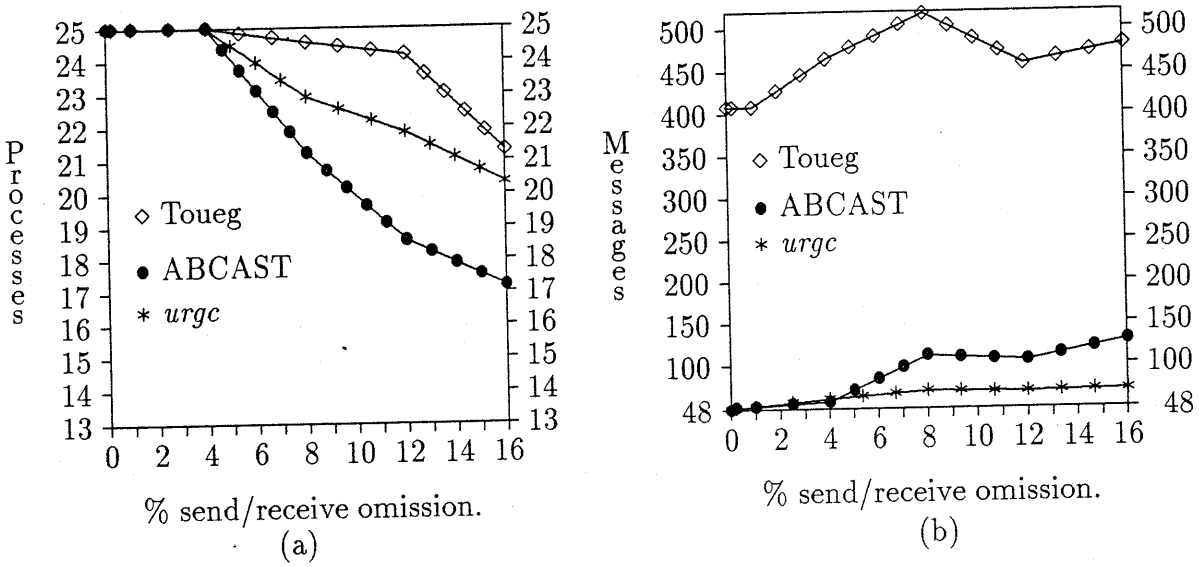

Figure6.1: (a) Mean number of decided processes and (b) mean number of generated messages per agreement vs. percentage of omission failures with $k = 5$ and a constant correct majority.

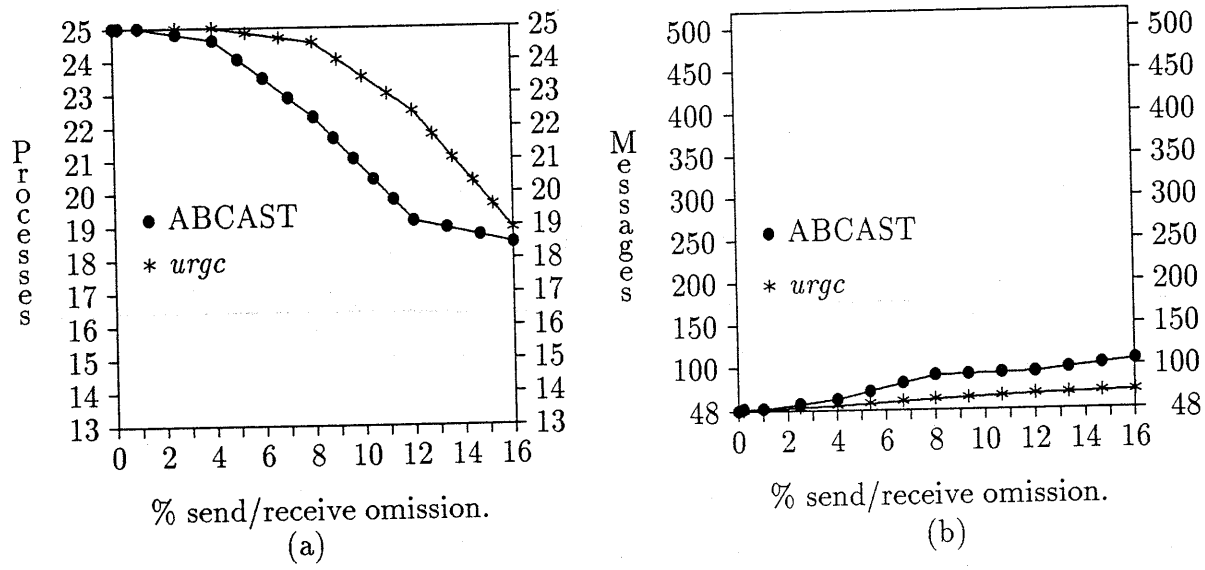

Figure6.2: (a) Mean number of decided processes and (b) mean number of generated messages per agreement vs. percentage of omission failures with $k = 5$ and a correct majority variable according to $\beta(7, 0.02)$.

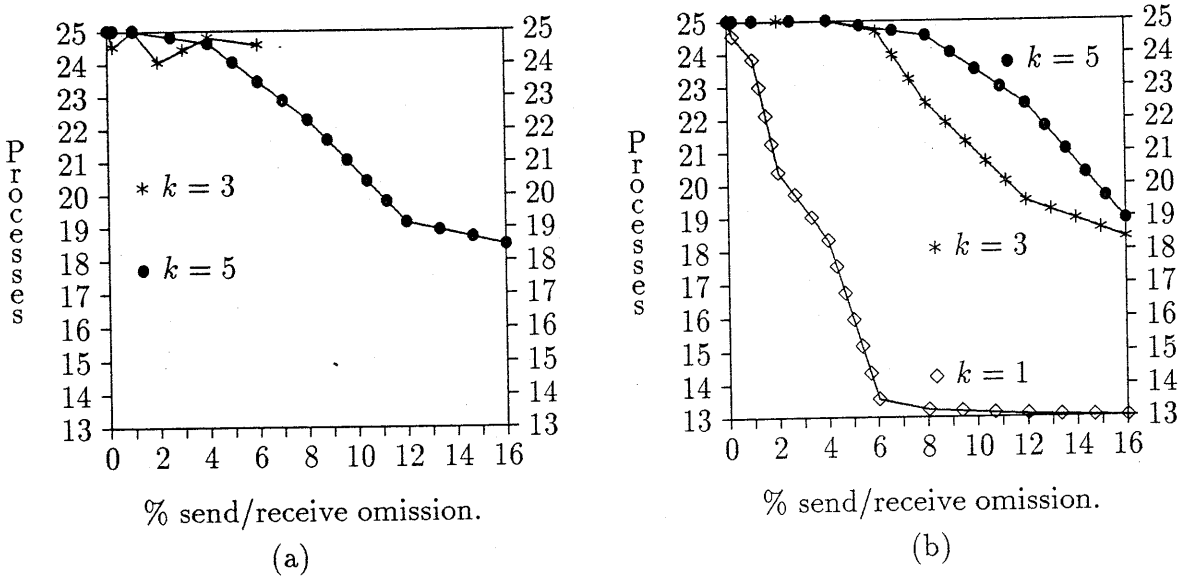

Figure6.3: Mean number of decided processes per agreement vs. percentage of omission failures for different values of k, for: (a) the ABCAST protocol and (b) the *urgc* protocol.

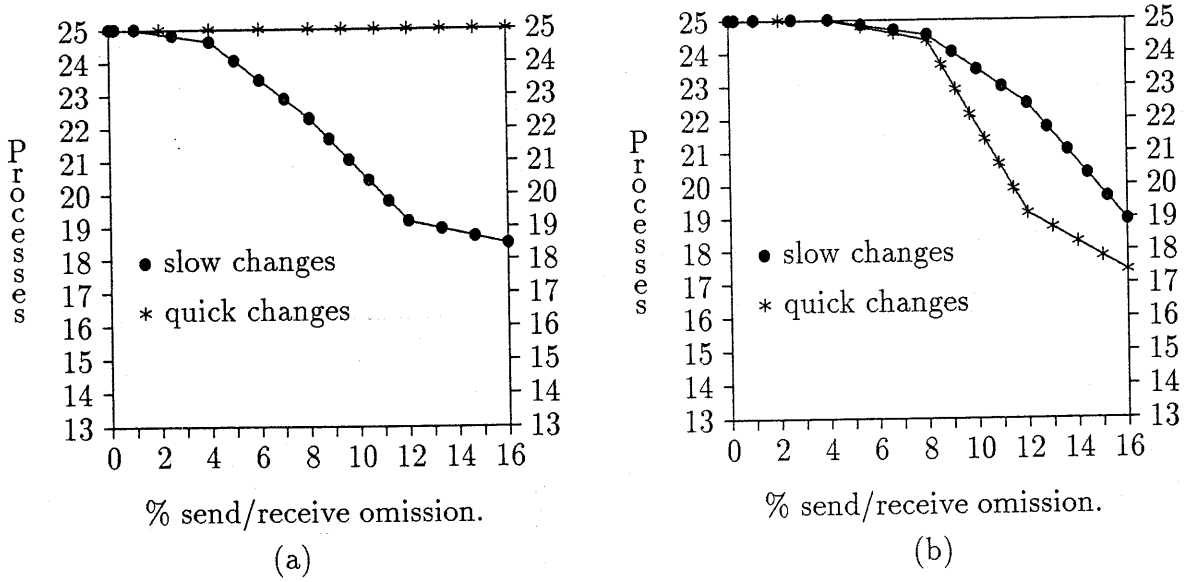

Figure6.4: Mean number of decided processes per agreement vs. percentage of omission failures in different dynamic failures conditions, with $k = 5$ and for: (a) the ABCAST protocol and (b) the *urgc* protocol.

is persistent, it could lead to detect wrong crashes. In the Internet or in high speed networks, where workstations may represents the bottleneck, the transmission delay may vary of some orders of magnitude from a normal condition to a congested one. The choice of a large amount of retries would be highly inefficient, while a protocol less sensitive to k should be preferred because it can survive a wrong setting, thus making less critical the sizing of time-outs and retries. We have observed the ABCAST and *urgc* behaviour under these conditions with decreasing values of k. We simulated the length of transient failures by means of a β distribution with mean 3 and variance 0.02, we used the values (1,3,5) for k and assumed failures affecting at most $n/2-1$ processes. Results are reported in Figure 5 and show that ABCAST has more problems in facing with transient failures with a wrong k. As failures grow up, it starts flushing, the group partitions into several subgroups, the amount of messages increases to collapse. *urgc* shows a graceful decrease of the number of terminating processes when changing k from 5 to 3. In this case ABCAST is penalized by the centralized control; the coordinator persists in solving the observed failures, it starts to flush which likely suffers the same problem. On the contrary, the rotating coordinator approach helps in turning around the problem. When $k = 1$, *urgc* fastly removes processes from the group and, thanks to the resilience, ensures that at least the majority of the processes terminate, thus succeeding against stressed critical conditions.

Slowing conditions due to congestion and traffic overload are likely to last for a relatively long and unpredictable time, thus emphasizing some protocol characteristics that allow to cope with them. The protocols observe them as being slowly changing failure patterns. When failure patterns are quickly changing, the k and the amount of retries are less important. Figure 6 reports the number of terminated processes under quickly changing failure patterns for growing failure rates and compared with the results obtained with slowly changing conditions. The β distribution with mean 1 and k set to 5 have been used. The situation is turned over; at a high failure rate, the rotation of the coordinators increases the probability of being pursued by the changing failures conditions and penalizes *urgc* that removes more processes and requires more rounds and messages. ABCAST is nearly transparent to this condition.

7 Concluding Remarks

A common framework to interpret and to analyse the behaviours of group communication protocols has been presented. Simulations have been used to evaluate some protocol characteristics. We compared the ISIS ABCAST primitive, that follows a two layer approach, and the *urgc* and Toueg's algorithms that are designed to operate in a general omission failure model. The two approaches are examples of two different line of thinking; the first derived from the experiences in designing

practical networks, and the second from a theoretical background. Simulation results show that the algorithms are almost equivalent when normal conditions are considered, although ABCAST performs slightly better; when critical conditions are experienced, ABCAST progressively degenerates, while the others continue to ensure the service.

Simulations have also shown that the resilience of $n/2 - 1$ and the choice of the rotating coordinator mechanism succeed in overwhelming slowly changing failure conditions such as those generated by network congestions. They could not be helpful when the failure patterns are rapidly changing.

References

[1] J. Crowcroft, K. Paliwoda. *"A Multicast Transport Protocol". ACM Computer Communication Review, Vol. 18, No. 4*, pages 247–260, 1988.

[2] M. Pease, R. Shostak, L. Lamport. *"Reaching Agreement in the Presence of Faults". Journal of ACM, Vol. 27*, 1980.

[3] S. Toueg, T. Srikanth. *"Optimal Clock Synchronization". Journal of ACM, Vol. 34*, 1987.

[4] T. D. Chandra, S. Toueg. *"Time and Message Efficient Reliable Broadcast". Proc. 4th International Workshop of Distributed Algorithms*, Vol. 486 of Lecture Notes in Computer Science:289–303, Sep. 1990.

[5] L. Lamport. *"Time, Clocks, and the Ordering of Events in a Distributed System". Communications of the ACM, Vol. 21, No. 7*, page 558, 1978.

[6] V. Hadzilacos, S. Toueg. *"Fault-Tolerant Broadcasts and Related Problems"*. Distributed Systems:97–145, S. Mullender, Addison-Wesley, 1993.

[7] K. Birman, A. Schiper, P. Stephenson. *"Lightweight Causal and Atomic Group Multicast". ACM Transactions on Computer Systems, Vol. 9, No. 3*, pages 272–314, Aug. 1991.

[8] R. Aiello, E. Pagani, G. P. Rossi. *"An Efficient Algorithm for Group Communication". Proceedings of the 5th IEEE Symposium on Parallel and Distributed Processing*, pages 226–232, Dallas, TX, Dec. 1-4 1993.

[9] M. F. Kaashoek, A. Tanenbaum. *"Fault Tolerance using Group Communication". ACM Operating Systems Review, Vol. 25, No. 2*, pages 71–74, Apr. 1991.

[10] C. Dwork, N. Lynch, L. Stockmeyer. *"Consensus in the Presence of Partial Synchrony". Journal of the ACM, Vol. 35, No. 2*, pages 288–323, Apr. 1988.

[11] P. Verissimo, L. Rodrigues, M. Baptista. *"AMp: A Highly Parallel Atomic Multicast Protocol". ACM Computer Communication Review, Vol. 19, No. 4*, pages 83–93, Sep. 1989.

[12] W. R. Stevens. Unix Network Programming, Prentice Hall, 1990.

[13] T. D. Chandra, V. Hadzilacos, S. Toueg. *"The Weakest Failure Detector for Solving Consensus". Cornell Technical Report No. 92-1293*, 1992.

[14] B. Chor, M. Merrit, D. B. Shmoys. *"Simple Constant-Time Consensus Protocols in Realistic Failure Models". Journal of the ACM, Vol. 36, No. 3*, pages 591–614, Jul. 1989.

[15] N. Lynch. *"Distributed Algorithms"*. Lecture Notes of a MIT Ph.D. course, Fall Semester 1992.

[16] S. Chauduri. *"More Choices Allow More Faults: Set Consensus Problem in Totally Asynchronous Systems". Information and Computation, Vol. 105*, pages 132–158, 1993.

[17] G. Neiger, S. Toueg. *"Automatically Increasing the Fault Tolerance of Distributed Algorithms". Journal of Algorithms, Vol. 11*, pages 374–419, 1990.

[18] D. R. Cheriton, D. Skeen. *"Understanding the Limitations of Causally and Totally Ordered Communication". Proceedings ACM SIGOPS'93 Conference*, pages 44–57, Dec. 1993.

[19] J. A. Garay. *"Reaching (and Maintaining) Agreement in the Presence of Mobile Faults". Proceedings WDAG '94*, pages 253–264, Oct. 1994.