# Comparing Performances and Quality of Service of Group Communication Protocols

Elena Pagani Gian Paolo Rossi
Dipartimento di Informatica, Università di Milano
e-mail: rossi@hermes.mc.dsi.unimi.it

## Abstract

*The deployment of a common platform of group services is considered the base for structuring fault tolerant distributed applications. 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 analisys, identifies the critical design issues that can affect the quality of service, and uses simulations to evaluate the role they play in coping with different failure conditions.*

## 1. Introduction

The deployment of a common platform of group services is considered the base for structuring fault tolerant distributed applications such as desktop conferencing, collaborative work, distributed real time control. Agreement [9], synchronization [12], reliable broadcast [13] and ordering of events [7] are just a few of the services that have to be provided among grouped application processes. The quality of service of such primitives can be evaluated in terms of *uniformity*, that is the guarantee that group decisions survive failures and recoveries leaving the system in a consistent state, *reliability*: a measure of continuous service delivery in spite of failures occurring at the system components [10] and *fairness* that is introduced here as a measure of the chances the correct processes have to terminate in spite of network failures.

The key problem in the design of a protocol for group communication is to ensure the quality of service in spite of the failures occurring at both the nodes (crash or permanent failures) and the network links (omissions or transient failures), where packets can be dropped [16]. Although the problem has been extensively studied from both the theoretical and the practical point of view and a number of solutions are available in literature [13, 10, 6, 11, 8, 1], 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 analisys, 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 [13, 6, 11].

## 2. Background

In this paper we consider a group of $n$ processes $p_1$, ..., $p_n$ that use a communication network for exchanging messages according to a group protocol. 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.

In such a distributed system, processors can stop working unexpectedly (crash or permanent failures) and communication channels can transiently drop some messages (transient failures). To ensure the quality of service in spite of failures different failure models can be adopted. Several authors (e.g. [13, 16]) have designed their algorithms by assuming that the only potential source of errors in a distributed system is an a-priori unknown set of processes, and that, once a processor fails during the execution of a protocol, it will be considered faulty "forever". This setting, which we will indicate as *static*, contrasts with a *dynamic* setting which allows that during the execution of a protocol,



1089-6503/96 $5.00 © 1996 IEEE
*Proceedings of EUROMICRO-22*

in the extreme case, a different set of faulty processors can be observed for each protocol step. Some authors [4, 15] have proved that dynamic and static settings are drastically different when considered for constructing distributed algorithms and that the former, under particular failure conditions, denies even the possibility of building the basic block of the group services: the agreement protocol. Roughly speaking, the capability of coping with dynamic failures has a larger practical extent but requires to weaken the initial problem requirements and resorts in a reduced quality of service.

Most of the existing group protocols, such those implemented within the systems ISIS [6], Transis [17], Delta-4 [10], can be viewed as having a two layer architecture. The lower layer is responsible for detecting and recovering all of the transient network failures. It acts as a failure detector [14] that notifies to the upper layer the 'suspected crash' events when excessive errors are observed. The upper entity provides the group service and assumes a crash failure model. The alternative approach derives from adopting the general omission failure model [13, 16] that directly ascribes both permanent and transient failures to processes.

## 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 [16] 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 so that the following conditions hold:*
**Validity**: *If a correct process sends a message m, 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; throughout this paper we will consider the atomic ordering. The definition of the Atomic Broadcast [16] 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 [16] in synchronous systems.

### 3.1. The uniformity requirement

Group applications often require that the property of *Uniformity* is ensured [16, 3]. 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.

### 3.2. The fairness requirement

We informally use the general term of *fairness* as an index of the fact that correct processes should have the same chances of properly terminating the protocol execution. This is actually a variable index because of transient failures that lead to erroneously suspect a correct process of being crashed (and thus removed from the group). The amount of 'still alive' processes that correctly terminate the protocol represents a good measure of fairness and should be observed under different failure conditions.

## 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 to either a distributed or a centralized control (e.g. [13, 10, 6, 11, 8]). The latter approach allows to generate an amount of messages that linearily grows with the group cardinality, but requires to execute an election algorithm when the *coordinator* fails. The *rotating coordinator paradigm* [5] is often used to overcome this problem.

### 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 [3] it has been proved that an agreement protocol can tolerate at most $\lceil n/2 \rceil - 1$ general omission failures, to ensure that uniform decisions are taken; this approach is adopted for example in [13, 10, 11, 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 the resilience of $n-1$ can coherently be used for the upper layer protocol, as the ABCAST protocol [6] does; however, this could not be sufficient in practice because processes being suspected crash by the underlying protocol 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 are unable of ensuring a uniform termination, as already pointed out in [2].

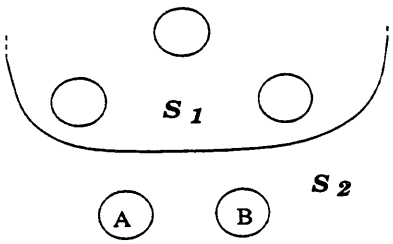


**Figure 1. Example of network partition.**

These considerations can be motivated with the aid of the following example. Let us consider the 5 processes of Figure 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 $\lceil n/2 \rceil - 1$ 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 $t \leq \lceil n/2 \rceil - 1$ 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 [11, 8]. 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, thus reducing the fairness degree of the algorithm.

### 4.3. Quality of service

To understand how the dynamic and static settings may affect the protocol termination we use another simple example. In Figure 2, we consider 5 processes that have to decide on a given value $v$ provided by

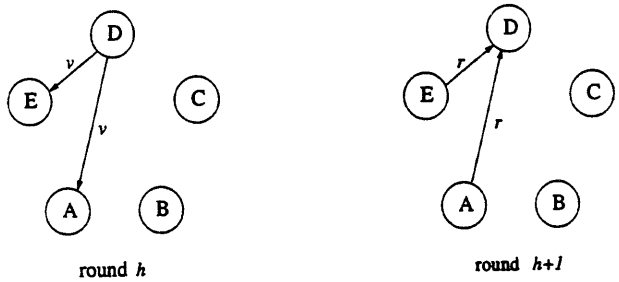


**Figure 2. Example of the decision activity in the static and the dynamic settings.**

the coordinator (process D). The activity requires two rounds: in the first D broadcasts its proposal $v$; in the second it collects the replies $r$ from the partners. D can be either correct or faulty. When static failures are considered, if D is correct, at round $h$ it reaches the majority of correct processes and at round $h+1$ it receives back their replies. D can decide $v$ because it knows that there cannot exist another majority deciding for a different value. If D is faulty, it won't decide because it won't receive enough replies, but, by rotating the coordinator, a correct coordinator eventually

exists that is capable of deciding. When dynamic failures are considered, D could not be able to decide even if correct. In fact, at round $h$ it reaches the current majority of correct processes (say A, E and D). At round $h+1$ E fails and D is unable to decide because it does not receive enough positive replies. Since these failures can repeatedly affect communications, decision cannot be guaranteed.

As a consequence, the protocols that adopt a dynamic setting can ensure termination only by weakening the requirements in terms of quality of service. This can resort in either renouncing to uniformity, as in [6], where, in the worst case, all processes terminate with different decisions, or in weakening the uniformity, as in [11] where 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 all correct processes that might exist; however, the algorithm is unable of avoiding that one of the excluded processes processed a wrong decision before being forced to die. Always, dynamic failures lead to design protocols that use the group membership to remove the processes affected by excessive transient failures from the group.

On the contrary, the set of algorithms (e.g. those presented in [13]) that has been designed for a static setting ensures uniform termination by adopting the resilience $\lceil n/2 \rceil - 1$ [3].

### 4.4. Recovery from failures

When failures occur, the protocol does its best to recover them; if they last too long, the group membership protocol is activated to avoid slowing down decisions of the more correct processes. The group membership and the recovery activities 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), [10, 6], or *ii*) non blocking, i.e. decisions and recoveries can be concurrently executed [11]. The non blocking approach is affordable, for example, by forcing the processes to maintain the *history* of the past decisions; a faulty process can recover from the most updated one. When the 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 Table 1. The protocol described in [13] 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.

**The ABCAST protocol.** The algorithm, [6], 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 [13] 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 $\lceil n/2 \rceil - 1$ lasting the whole execution, i.e. no more than $\lceil n/2 \rceil - 1$ 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 and guarantees the uniformity. It is devided in phases and each phase lasts 7 rounds.

We implemented the optimized version of the algorithm that introduces a pipelining mechanism to con-

| Protocol | Resilience | Two Layers | Recovery | Uniformity | Failures |
|---|---|---|---|---|---|
| ABCAST | $n-1$ | YES | Blocking | no | crash |
| Toueg's | $\lceil n/2 \rceil - 1$ | NO | Blocking | all | crash/gen.om. |
| *urgc* | $\lceil n/2 \rceil - 1$ | NO | Non Blocking | active | general om. |

**Table 1. Summary of the algorithms.**

currently execute 4 phases.

The description given in [13] 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 [11] 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 omission failures. It has a resilience of $\lceil n/2 \rceil - 1$ 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

A few simple analytical considerations help to quantify the performances of the algorithms in absence of failures. Table 2 reports the indexes: amount of rounds to terminate, amount of messages and message size. Under normal system conditions, the algorithms ter-

| algorithm | rounds | messages | coord. msg size |
|---|---|---|---|
| TOUEG | 8 execution<br>3 decision | $16(n-1)$ | 5 bytes $+data$ |
| *urgc* | 2 | $2(n-1)$ | $(28 + \frac{17}{4}n)$ bytes |
| ABCAST | 1 | $2(n-1)$ | 17 bytes |

**Table 2. Performance indexes of the AB algorithms.**

minate in a very short time; by considering 7.81 *msec.* per round (this is the averaged measure we obtained over our Departmental Ethernet network) ABCAST and *urgc* terminate in 7.81 *msec.* and 15.62 *msec.* respectively. The amount of exchanged messages linearily grows with the group cardinality when considering fully connected point to point subnetworks. For instance, the *urgc* algorithm exchanges 38 messages, about 113 bytes each when a group of 20 processes is considered. The required aggregated bandwidth over a fully connected network can be computed in 17.2 Kbit per round. The ABCAST is even cheaper.

When failures occur the amount of rounds is likely to be increased together with the exchanged messages as the consequence of failures occurring at the coordinator's nodes. To evaluate the behaviour of the algorithms under different failure conditions, we simulated their executions on top of a model of distributed system where both static and dynamic failure conditions were reproduced.

### 6.1. Simulation conditions

Simulations assume 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 in [13] 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 [13], 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.

The omission failures have been uniformily distributed between send and receive omissions. To model the dynamics of changing failure conditions two $\beta$ dis-

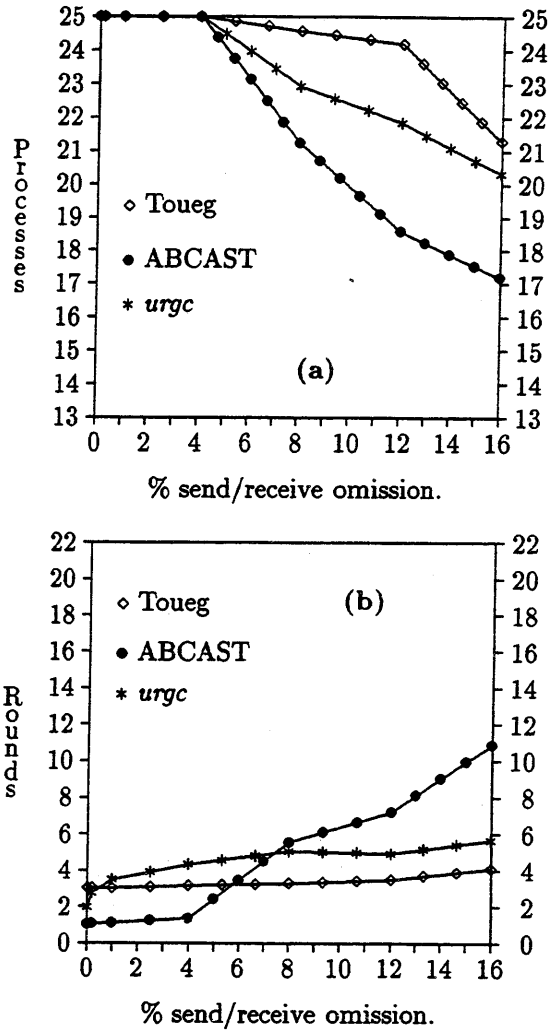


**Figure 3. (a) Mean number $x$ and (b) mean number $y$ vs. omission failure rate. $k = 5$, $\beta_F = 2$ and constant correct majority.**

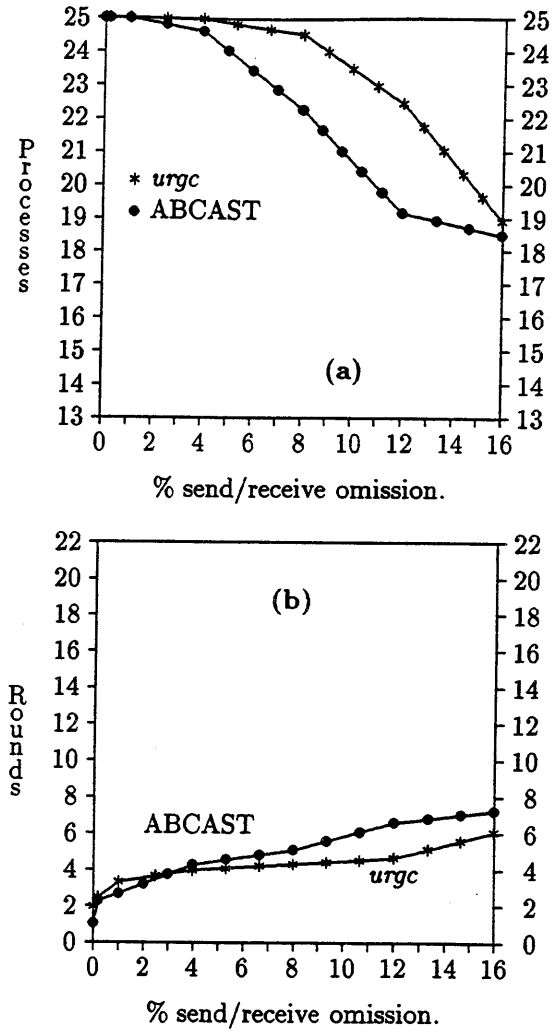


**Figure 4. (a) Mean number $x$ and (b) mean number $y$ vs. omission failure rate. $k = 5$, $\beta_C = 7$ and $\beta_F = 2$**

tributions have been used: $\beta_F$ allows to model how long (in rounds) transient failures last, while $\beta_C$ describes the amount of rounds a given majority of correct processes remains stable. The amount of terminating processes and the amount of rounds required to terminate have been observed for a system of 25 processes and reported against growing failure rates.

### 6.2. Simulation results

Figures 3 and 4 report the number $x$ of correctly terminating processes (a) and the amount $y$ of rounds to terminate (b) for the considered algorithms and observed under growing failure rates. Figure 3 refers to a static setting, while Figure 4 refers to the dynamic one. Both ABCAST and *urgc* resort to remove processes from the group as soon as the failure rate starts to grow up. This reduces the termination chances of some processes that, under the same conditions, the Toueg's algorithm leads to termination (Figure 3(a)). *urgc* shows a good fairness degree because it requires that a majority agrees before eliminating a process, while in the ABCAST a single process can force the elimination of another one. Both Figures 3(b) and 4(b) show that the amount of rounds required to terminate is not highly influenced by the considered growing failure rates. For the *urgc* this derives from the protocol structure that concurrently executes the recovery with the decision process. The ABCAST performances are negatively influenced by the *flush* protocol. When the

omission failures are concentrated over a minority of processes (as in the case of Figure 3) the size of $k$ is not sufficient to recover them, and the *flush* execution progressively increases the amount of required rounds. This also reduces the amount of terminating processes.

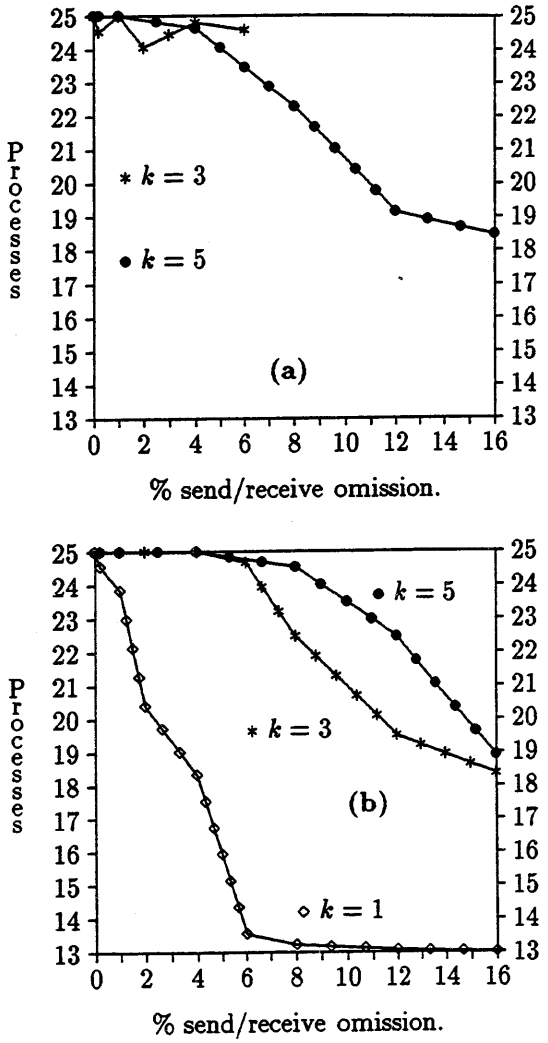


**Figure 5. Mean number of decided processes per agreement vs. omission failure rates for different values of $k$, $\beta_C = 7$ and $\beta_F = 2$. (a) ABCAST protocol and (b) *urgc* protocol.**

In environments with different workstation technologies and highly variable traffic conditions [1] large variances may be observed in the transmission delays, thus making critical the definition of the round width and of the amount of retries ($k$). If they are wrong sized, the slowing conditions can be misinterpreted as transient failures and, if this condition is persistent, quite a number of processes might be suspected crash and removed from the group. The choice of a large amount of retries would be highly unefficient, while a protocol less sensitive to $k$ should be preferred because it can survive a wrong setting. 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 $\beta$ distribution with mean 2 and variance 0.02, we used the values (1,3,5) for $k$ and assumed failures affecting at most $\lceil n/2 \rceil - 1$ processes. Results

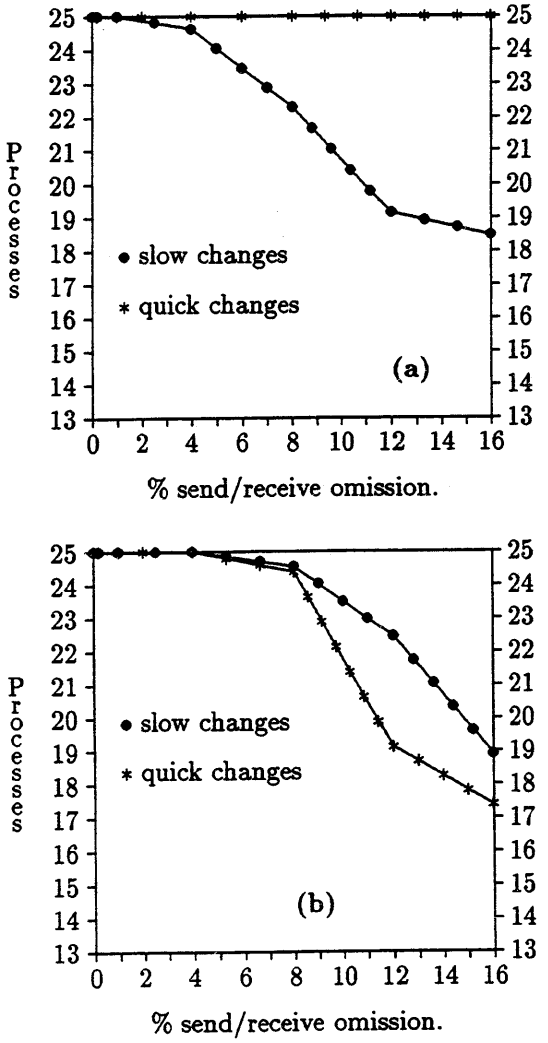


**Figure 6. Mean number of decided processes per agreement vs. omission failure rates under dynamic failures conditions. $k = 5$, slow changes $\beta_C = 7$, quick changes $\beta_C = 1$, $\beta_F = 2$ . (a) ABCAST protocol and (b) *urgc* protocol.**

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. Even in the extreme case of $k = 1$, *urgc* fastly removes processes from the group and, thanks to the resilience, ensures that at least the majority of the processes terminate.

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 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 failures.

## 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 operates in a crash model, 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 lines of thinking; the first derived from the experiences in designing practical networks, and the second from a theoretical background. Simulation results show that both the approaches are almost equivalent when normal conditions are considered, although ABCAST is slightly ahead; when critical conditions are experienced, ABCAST progressively degenerates, while the others can ensure the service.

Simulations have also shown that the resilience of $\lceil n/2 \rceil - 1$ 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] 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.

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

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

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

[5] J. Chang, N. F. Maxemchuk. *"Reliable Broadcast Protocols". ACM Trans. on Computer Systems, Vol. 2, No. 3,* pages 251–273, Aug. 1984.

[6] 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.

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

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

[9] M. Pease, R. Shostak, L. Lamport. *"Reaching Agreement in the Presence of Faults". Journal of ACM, Vol. 27, No. 2,* pages 228–234, Apr. 1980.

[10] P. Verissimo, L. Rodrigues, M. Baptista. *"AMp: A Highly Parallel Atomic Multicast Protocol". Proceedings of the ACM SIGCOMM'89 Conference. Computer Communication Review, Vol. 19, No. 4,* pages 83–93, Austin, TX, Sep. 1989.

[11] 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.

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

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

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

[15] V. Hadzilacos. *"Connectivity Requirements for Byzantine Agreement under Restricted Types of Failure". Distributed Computing, Vol. 2, No. 2,* 1987.

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

[17] Y. Amir, D. Dolev, S. Kramer, D. Malki. *"Transis: A Communication Sub-System for High Availability". Proc. 22nd Annual Int. Symp. on Fault-Tolerant Computing,* pages 76–84, Jul. 1992.