Broadcasting Messages in Fault-Tolerant Distributed Systems: the benefit of handling input-triggered and output-triggered suspicions differently[*]

Bernadette Charron-Bost[†] charron@lix.polytechnique.fr

Xavier Défago[‡] defago@jaist.ac.jp

André Schiper[*] andre.schiper@epfl.ch

[*] *LIX, École Polytechnique, 91128 Palaiseau Cedex, France*

[†] *Japan Advanced Institute of Science and Technology (JAIST), 1-1 Asahidai, Tatsunokuchi, Ishikawa 923-1292, Japan*

[‡] *École Polytechnique Fédérale de Lausanne (EPFL), 1015 Lausanne, Switzerland*

Abstract

This paper investigates the two main and seemingly antagonistic approaches to broadcasting reliably messages in fault-tolerant distributed systems: the approach based on Reliable Broadcast, and the one based on View Synchronous Communication (or VSC for short). While VSC does more than Reliable Broadcast, this has a cost. We show that this cost can be reduced by exploiting the difference between input-triggered *and* output-triggered *suspicions, and by replacing the standard VSC broadcast primitive by two broadcast primitives, one sensitive to input-triggered suspicions, and the other sensitive to output-triggered suspicions.*

1. Introduction

Reliable Broadcast [6] and *View Synchronous Communication* (*VSC*) [1, 8, 7, 5] are two communication abstractions that have been extensively considered in the context of asynchronous fault-tolerant distributed systems. Both abstractions allow the broadcasting of messages while ensuring some sort of agreement: for any message m, either all correct processes deliver m, or none of them do. However, when taking a closer look at the specification of each primitive, one has on one side a simple and clear definition (Reliable Broadcast), and on the other side a complex definition (VSC), which moreover varies from one author to another.

Now, why would one consider VSC at all, rather than the well-defined Reliable Broadcast primitive only? A careful analysis of the literature shows that theoretical papers tend to consider Reliable Broadcast, whereas more practical papers favor VSC. The goal of the paper is to show that, although the specification of Reliable Broadcast is simple, it leads to implementation problems that are addressed by VSC.

The implementation of Reliable Broadcast is usually described assuming reliable channels while the VSC approach considers the implementation of the VSC communication primitive over lossy channels. Obviously, assuming reliable channels is not realistic in practice. The implementation of Reliable Broadcast over lossy channels requires message retransmission; the same holds for VSC. In order for some process p to be able to retransmit message m, p needs to buffer m. This raises the question of how long p must buffer m? In this paper we argue that, unless the asynchronous system is augmented with a perfect failure detector (one that never makes mistakes) the implementation of Reliable Broadcast over lossy channels requires m to be buffered for an unbounded duration. In contrast, implementations of VSC are able to get around this problem: they are based on a group membership service which excludes slow processes from the membership and forces them to crash.[1]

However, the VSC approach has its own practical drawbacks. Processes that are excluded from the group might not have crashed. Thus the overhead of an incorrect failure suspicion is high in the VSC approach if, in order to keep the same degree of replication, every excluded process is replaced by a new one. For this reason, systems based on VSC are usually configured with a high timeout value to suspect crashed processes. The problem is that choosing a high timeout value also has drawbacks, namely it leads to high fail-over time.[2]

So, while the VSC approach addresses the issue of message buffering it favours high timeout values for suspecting crashed processes. We show that the two issues of buffering and fail-over time can be decoupled, with significant advantages for the fail-over time of applications. This decoupling can be achieved by distinguishing two "reliable broadcast" primitives instead of just one (i.e., VSC). These two primi-

[*]Research partially supported by the OFES under contract number 01.0537 as part of the IST Project REMUNE (IST-2000-65002).

[1]In the paper we consider the *Primary Partition* Group Membership Service.

[2]The fail-over time is the time elapsed between the crash of a process and the time at which the algorithm has recovered from the crash. During this interval the algorithm is blocked.

tives lead us to distinguish *input-triggered* suspicions from *output-triggered* suspicions. While output-triggered suspicions lead to exclusions from the membership, this is not the case with input-triggered suspicions. Moreover, we show that fail-over time is influenced only by input-triggered suspicions, and not by output-triggered suspicions. This allows aggressive input-triggered suspicions to coexist with conservative output-triggered suspicions.

The rest of the paper is organized as follows. Section 2 discusses Reliable Broadcast and introduces the *time-bounded buffering* problem. Section 3 shows how View Synchronous Communication solves the *time-bounded buffering* problem. Section 4 introduces the distinction between *input-triggered* and *output-triggered* suspicions, and shows the drawback of the VSC approach. Section 5 shows that the drawback of the VSC approach can be overcome by having two broadcast primitives, rather than just one. In Section 6, the use of the two broadcast primitives is illustrated by an example. Related work is discussed in Section 7. Finally, Section 8 concludes the paper.

2. Reliable Broadcast and its Limitations

In this section, we discuss the implementation of reliable communication over fair-lossy channels, which highlights the major drawback of the Reliable Broadcast approach.

2.1. Processes, channels and Reliable Broadcast

The Reliable Broadcast approach assumes an asynchronous system model where the set of processes is fixed. Processes are only subject to crash failures (no Byzantine failures) without recovery. A *correct* process is a process that never crashes. Processes are completely connected by a fair-lossy channels.

Reliable Broadcast is specified in terms of two primitives R-BROADCAST and R-DELIVER, which satisfy the following properties [6]:

- **Validity:** If a correct process R-BROADCASTS m, then it eventually R-DELIVERS m.
- **Agreement:** If a correct process R-DELIVERS m, then all the correct processes eventually R-DELIVER m.
- **Integrity:** For any message m, every correct process R-DELIVERS m at most once, and only if m was previously R-BROADCAST.

2.2 Reliable Broadcast over reliable channels $vs.$ fair-lossy channels

Reliable Broadcast can be easily implemented in an asynchronous system with reliable channels: when a process p wishes to R-BROADCAST a message m, p sends m to all processes. When some process q receives m for the first time, then (1) q sends m to all processes and (2) q R-DELIVERS m. Clearly, this implementation does not work with fair-lossy channels.

Reliable (or rather quasi-reliable) channels, can be implemented over fair-lossy channels. Consider message m sent by p to q. Upon sending m, p copies m into an output buffer, and transmits repeatedly m to q until it receives an acknowledgment $ack(m)$ from q. Each time q gets m, it transmits $ack(m)$ to q. When p gets $ack(m)$. it deletes m from its output buffer.

However, if q crashes, process p may never get $ack(m)$, and keeps m in its output buffer forever. This naturally leads to the following question: is there an implementation in which p can safely delete m from its output buffer after a finite amount of time? To formalize this issue, we introduce the *time-bounded buffering problem*: a time-bounded buffering implementation of reliable communication is an implementation wherein every message is eventually discarded from the output buffers of all processes. Time-bounded buffering is related to the notion of *message stability*, a terminology used in the context of group communication. Solving Reliable Broadcast with time-bounded buffering is equivalent to ensuring that, for every process p and all messages m in p's output buffer, eventually m is stable at p. We argue in [4] that no implementation of Reliable Broadcast over fair-lossy channels can solve the time-bounded buffering problem, solely based on failure detectors of either class $\mathcal{S}$ or class $\Diamond\mathcal{P}$ [3]. However the problem can be solved with a perfect failure detector $\mathcal{P}$ (i.e., one that does not make any mistake) as follows: a process p that has some message m in its output buffer discards m once it knows that for every process q, either q has acknowledged m or q is suspected.

2.3 Reliable Broadcast over lossy channels: time-bounded buffering and program-controlled crash

The impossibility result for a time-bounded buffering implementation of Reliable Broadcast with a $\Diamond\mathcal{P}$ failure detector is quite a limiting constraint in practice. Systems based on View Synchronous Communication overcome this impossibility by relying on *program-controlled crash* [2]. Program-controlled crash gives the processes the ability to kill other processes or to commit suicide. It can be used to implement Reliable Broadcast over fair-lossy channels with time-bounded buffering. Consider process p with message m in the output buffer to q. If after some duration p has not received $ack(m)$ from q (directly or indirectly), p decides (1) to kill q, and (2) to discard m from its output buffer. Indeed, as q eventually crashes, there is no obligation for q to R-DELIVER m, i.e., p can safely discard m.

However, program-controlled crash has a non negligible cost. To see that, consider some process q that is forced to crash. In order to keep the same degree of replication, another process q' will have to be created in order to replace q. This requires a dynamic system model. The management of the processes that are part of the system is handled by a *group membership service*. So, the suicide of q triggers a costly sequence of operations: (1) membership change to exclude q, (2) membership change to include q', which incorporates (3) the costly state transfer operation to bring q'

to an up-to-date state. In other words, each exclusion of a correct process leads to an important overhead. From a practical point of view, this means that incorrect failure suspicions should be avoided as much as possible. This can be achieved by choosing a conservative timeout value in the implementation of the failure suspicion mechanism. Unfortunately the price is a high fail-over time. We come back to this issue later in the paper (Section 4).

3. VSC ensures Reliable Broadcast in the context of a view

In this section we compare Reliable Broadcast with View Synchronous Communication and show that View Synchronous Communication ensures the properties of Reliable Broadcast in the context of a view.

3.1. Group membership and View Synchronous Communication

View Synchronous Communication (or VSC for short) [5] assumes an asynchronous system model where processes may fail by crashing and may recover. It is based on a *group membership service*, which manages the the successive memberships of a group, called *views*. One distinguishes two types of group membership services: *primary-partition* and *partitionable*. Primary-partition group membership services attempt to maintain a *single* agreed view of the current membership of the group. In this paper, we only consider the primary-partition membership service. The primary-partition membership service is defined by an agreement property on view history: if p installs v as the ith view and if q installs v' as the ith view, then we have $v = v'$.

VSC allows processes to broadcast messages to the members of their current view with certain guarantees. Let V-BROADCAST^v denote the primitive by which a message is broadcast by a process in view v, and by V-DELIVER^v the primitive that delivers a message to a process in view v. VSC is defined by the following core properties [5]:

- **Validity:** If a correct process executes V-BROADCAST$^v(m)$, then it eventually V-DELIVERS m (in view v or in a subsequent view).
- **Termination:** If a process executes V-BROADCAST$^v(m)$, then eventually (1) every process in the view v V-DELIVERS$^v(m)$ or (2) every correct process in v installs a new view.
- **View Synchrony:** If process p belongs to two consecutive views v and v', and V-DELIVERS$^v(m)$, then every process q in $v \cap v'$ that installs v', also V-DELIVERS$^v(m)$, i.e., delivers m before installing v'.
- **Sending View Delivery:**[3] A message broadcast in view v, if delivered, has to be delivered in view v. In other words, if V-DELIVER$^{v'}(m)$ and V-BROADCAST$^v(m)$ occur, then $v' = v$.

[3]Some specification consider a property called “Same View Delivery” instead of “Sending View Delivery” [5].

- **Integrity:** For any message m, every correct process V-DELIVERS m at most once, and only if m was previously V-BROADCAST.

3.2 VSC implementation ensuring time-bounded buffering

VSC over fair-lossy channels can be implemented with time-bounded buffering by relying on program-controlled crash. The idea is very simple. Consider V-BROADCAST$^v(m)$, and message m in the output buffer of p. If p receives $ack(m)$ from all the processes in view v, then p can discard m from its output buffer. If p does not receive $ack(m)$ from some process r, then p eventually triggers a view change in order to exclude r. Upon installation of a new view v' from which r is excluded, p can discard m from its output buffer. If r discovers that it has been (incorrectly) excluded from the new view v', then it commits suicide.

3.3 VSC ensures Reliable Broadcast in the context of a view

We now show that VSC ensures the three properties of Reliable Broadcast in the context of a view. The Validity and Integrity properties of VSC clearly enforce the Validity and Integrity properties of Reliable Broadcast.

For the Agreement property, consider a correct process p that executes V-DELIVER$^v(m)$ (in view v). By the Integrity property of VSC, some process must have executed V-BROADCAST(m). The Sending View Delivery property guarantees that m was V-BROADCAST in view v. If all the processes in view v V-DELIVER m, then the Agreement property of Reliable Broadcast holds in v. Otherwise, the Termination property of VSC implies that every correct process eventually installs a new view v'. By the View Synchrony property, every process in v that installs v' has V-DELIVERED m in view v.

So, VSC ensures the properties of Reliable Broadcast in the context of a view with time-bounded buffering, but with a price: the high overhead of program-controlled crash (Sect. 2.3).

4. Limitations of the VSC approach

When looking closer at the role of GMS in the context of VSC one can make the following observation:

1. GMS ensures the time-bounded buffering property.
2. As a failure detection mechanism, GMS prevents blocking in (application) algorithms: if $q \in v$ waits for a message broadcast by p in view v, then a view change that excludes p allows q to stop waiting for m.

The VSC approach handles these two different aspects uniformly. This may have some bad effects since timing constraints are quite different in (1) and (2). One the one hand, detecting failures quickly is crucial with respect to (2) for reducing blocking periods of algorithms, and hence to ensure reasonable fail-over time. On the other hand, one tolerates longer delays for forcing time-bounded buffering. Indeed, these longer delays have no direct impact on the timing behaviors of algorithms as long as buffer resources are available. In the VSC approach, we cannot take advantage of the timing flexibility allowed for enforcing time-bounded buffering since the VSC approach artificially binds this problem to the one of preventing blocking. There is a clear dilemma between short timeout values and high timeout values.

One can escape from the dilemma by noticing that the two roles of GMS are in fact related to two different failure detection mechanisms. Process p can suspect q with respect to the fact that either (1) messages in its output buffer to q are never received, called *output-triggered* suspicions, or (2) because its input buffer from q is empty, called *input-triggered* suspicions. While the two suspicion mechanisms have been used in implementations, their specificities have not been been exploited effectively. These differences can be exploited by introducing two broadcast primitives instead of just one.

5 Two broadcast primitives instead of just one

In order to exploit the difference between output-triggered and input-triggered suspicions, we split the features of V-BROADCAST into two broadcast primitives that we call V-R-BROADCAST and V-FD-BROADCAST, (R stands for *Reliable* and FD for *Failure Detection*). Both alike satisfy the specification of VSC given in Section 2.3, but with a different GMS: the views used by V-R-BROADCAST are different from the ones used by V-FD-BROADCAST. The issue pointed out in Section 4 is non-functional, so it is not surprising that our two new broadcast primitives have the same specification. Typically, V-R-BROADCAST would be used to reliably broadcast a message in a view while ensuring time-bounded buffering; on the other hand, V-FD-BROADCAST should be used whenever view changes are needed to prevent blocking.

5.1 Membership views $vs.$ ranking views

Two GMS define two types of views. We call them *membership views* (or simply *views*), and *ranking views* (or *rk-views*). Ordinary views are identical to the views of View Synchronous Communication, and they are denoted similarly by $v_0, v_1, \ldots, v_i, \ldots$. Each ordinary view v_i is a *set* of processes. Ranking views are installed between membership views. Processes agree on the sequence of both membership views and rk-views. The rk-views between v_i and v_{i+1} are noted

$$v_i^0,\ v_i^1, \ldots, v_i^j, \ldots, v_i^{last_i}$$

Each ranking view v_i^j is a *sequence* of processes. The membership of v_i^0 is equal to the membership of v_i, and the membership of the last ranking view $v_i^{last_i}$ is equal to the membership of v_{i+1}. Moreover, the membership of all ranking views $v_i^0, \ldots, v_i^{last_i - 1}$ is the same as the membership of v_i. Only the order of the processes differ (the reason will be explained below), e.g., $v_i = v_i^0 = [p, q, r]$, $v_i^1 = [q, r, p]$, $v_i^2 = [r, p, q]$, $v_i^3 = [p, q, r]$, etc. During the existence of the rk-views $v_i^0, \ldots v_i^{last_i - 1}$ the membership view remains v_i.

Referring to the discussion of Section 4, membership views are generated by suspicions resulting from conservative timeout values, while rk-views are generated by suspicions resulting from aggressive timeout values. As all the ranking views v_i^j, except $v_i^{last_i}$, are composed of the same set of processes as v_i, they do not force the crash of processes. So, the role of rk-views is to contribute to a short fail-over time, while membership views ensure time-bounded buffering of messages.

As mentioned above, the specification of the two broadcast primitives is identical to the specification of VSC given in Section 3.1, but with different views. This affects only the Sending View Delivery property, which becomes:

- **Sending View Delivery:**
  If V-R-DELIVER$^v(m)$ and V-R-BROADCAST$^{v'}(m)$ occur, then $v = v'$ is the same membership view (rk-view changes could have occurred between V-R-BROADCAST(m) and V-R-DELIVER(m)).
  If V-FD-DELIVER$^v(m)$ and V-FD-BROADCAST$^v(m)$ occur, then $v = v'$ is the same rk-view, i.e., no view change and no rk-view change occurred between V-FD-BROADCAST(m) and V-FD-DELIVER(m).

5.2 The two broadcast primitives and output-triggered $vs.$ input-triggered suspicions

The two broadcast primitives, V-R-BROADCAST and V-FD-BROADCAST allow us to take advantage of the two types of suspicions: input-triggered vs. output-triggered. As explained in Section 4, exclusions (resulting from suspicions) ensure message stability (i.e., time-bounded buffering), whereas ranking views (resulting from suspicions, too) prevent algorithms from blocking. Message stability is an issue related to output buffers, while blocking is an issue related to input buffers. Thus it is natural to base message stability (i.e., process exclusion) on output-triggered suspicions. On the other hand, prevention of blocking (i.e., delivery of ranking views) ought to be based on input-triggered suspicions. Consequently, the GMS related to V-R-BROADCAST should define membership views based on output-triggered suspicions (the suspicion of some process p leads to the exclusion of p, and the definition of a new membership view), whereas the GMS related to V-FD-BROADCAST should define rk-views based on input-triggered suspicions.

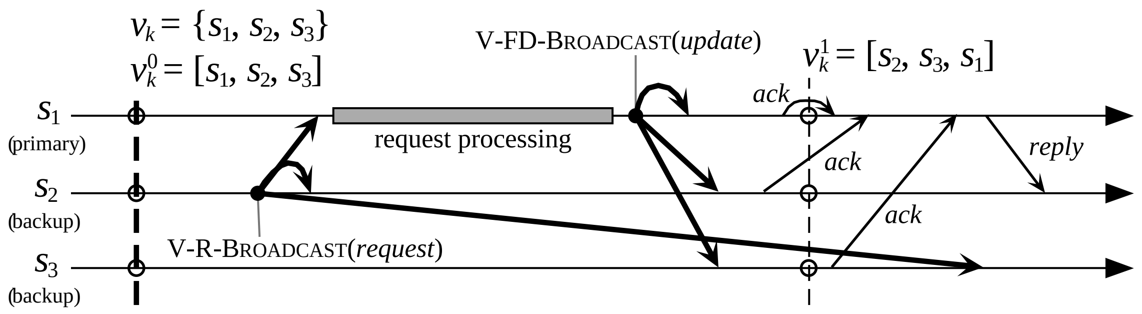

Figure 1. Primary-backup replication with V-R-BROADCAST **and** V-FD-BROADCAST

5.3 Defining ranking views

In this section we discuss the definition of rk-views. Various options are possible, the simplest one being the *rotating coordinator rk-views*.[4] In the rotating coordinator rk-views, the first process in some rk-view v,[5] and only this process, is monitored by all other processes in the rk-view. If this process is suspected (input-triggered suspicions), the GMS is invoked to install a new rk-view v', where v' is obtained from v by a permutation that moves the head of the sequence v to the tail of the sequence v', e.g., if $v = [p, q, r]$, then $v' = [q, r, p]$. This corresponds to a coordinator change, from coordinator p in the rk-view v, to the coordinator q in the rk-view v'. The monitoring of the first process of some rk-view v can be implemented using heartbeat messages: the first process of the rk-view v periodically sends *I am alive* messages to the other processes of v.

5.4 Performance issues

Most of the time, the performance of group communication is measured in "nice" runs, i.e., in runs with no crashes and no incorrect failure suspicions. The reason is that the performance of group communication in the case of a crash is dominated by the timeout value used for failure detection, which leads to embarrassingly large figures.

Assume that V-FD-BROADCAST is implemented with a small input-triggered timeout value (e.g., 1s), and V-R-BROADCAST is implemented with a large output-triggered timeout value (e.g., 100s). This means that the cost of V-FD-BROADCAST in the case of a crash will be on average around 1 second (i.e., better than VSC with a timeout of 10s), and the cost of V-R-BROADCAST will be on average around 100 seconds (i.e., worst than VSC). To understand that we gain in both cases compared to VSC with a timeout value of 10 seconds, the reader must understand that the crash of a process — in the context of reliable broadcast — impedes the group only whenever the rest of the group is blocked waiting for a message from the crashed process:

- If the group blocks in the case of a crash, then V-FD-BROADCAST should be used (in order to have a small blocking period).
- If the crash of a process does not block the group, then V-R-BROADCAST should be used: V-R-BROADCAST instead of VSC (which has a smaller timeout value) reduces the probability of incorrectly excluding processes. This also has a positive impact on the overall performance.

6 Example: VSC and primary-backup replication

Here, we illustrate the use of the two broadcast primitives in the context of the primary-backup replication technique (for more details, see [4]). In order to simplify the example, we assume that the role of the clients are played by the servers, i.e., servers issue requests.

The use of the two broadcast primitives is shown in Figure 1: V-R-BROADCAST is used for broadcasting *request* messages, while V-FD-BROADCAST is used by the primary for broadcasting *update* messages. On Figure 1, the leftmost membership view v_k and rk-view v_k^0 is $[s_1, s_2, s_3]$: this rk-view defines s_1 as the primary (i.e., the first process in the sequence). A new rk-view $v_k^1 = [s_2, s_3, s_1]$ is later installed, which defines s_2 as the new primary: the membership view v_k remains unchanged, i.e., though s_1 has been suspected, s_1 remains in the membership view v_k. The Sending View property of V-FD-BROADCAST ensures that V-FD-BROADCAST$(update)$ and V-FD-DELIVER$(update)$ occur in the same rk-view $v_k^0 = [s_1, s_2, s_3]$. The Sending View property of V-R-BROADCAST also ensures that V-R-BROADCAST$(request)$ and V-R-DELIVER$(request)$ occur in the same membership view $v_k = \{s_1, s_2, s_3\}$ (but not necessarily in the same rk-view). Point-to-point messages (ack, $reply$) are transparent to view changes.

[4]The rotating coordinator paradigm is well known in the context of fault-tolerant computing, e.g., [3]. In a given group communication system, various rk-view paradigms could be predefined. The choice of the paradigm would have to be specified as a parameter upon creation of a group.

[5]A rk-view is a "sequence" of processes. The "coordinator" is the first process in the rk-view.

In the light of this example, we can see the benefit of having two broadcast primitives instead of just one (as in the classical VSC context). The crash of s_2 (which broadcasts a request) and the crash of s_1 (which processes requests and broadcasts updates) do not have the same impact on the system: the crash of s_1 should be quickly detected (it blocks the group), whereas fast detection of the crash of s_2 is not essential (the crash of s_2 does not block the group, since the primary waits for a majority of ack messages). With only one broadcast primitive, it is impossible to handle the broadcast of s_1 differently from the broadcast of s_2.

7 Membership and ranking views compared to partitionable group membership

Wrong suspicions related to V-FD-BROADCAST do not lead to the exclusion of processes. This can be seen as similar to a partitionable membership service, wherein processes in a minority partition are not forced to crash [5]. Apart from this similarity, our proposal differs from VSC in a partitionable group membership (called *extended* VSC).

In Figure 1, consider the membership view $v_k = \{s_1, s_2, s_3\}$, and the rk-view $v_k^0 = v_k$. Let processes s_1, s_2, s_3 be correct, but consider a temporary link failure inducing the formation of two temporary partitions $\pi_1 = \{s_1\}$ and $\pi_2 = \{s_2, s_3\}$. Assume that this partition leads to the definition of a new rk-view $v_k^1 = [s_2, s_3, s_1]$, installed on s_2 and s_3 (and on s_1 after the partition is repaired). With extended VSC, the messages broadcast by processes in partition π_1 are sent to the processes in π_1, whereas the messages broadcast by processes in π_2 are sent to the processes in π_2. This is not the case with our broadcast primitives. If no (membership) view changes occurs, then all messages V-R-BROADCAST or V-FD-BROADCAST (1) before the partition, (2) during the partition, or (3) after the repair of the partition, are eventually delivered to $\{s_1, s_2, s_3\}$. The layer implementing VSC has thus the responsibility to buffer messages during the existence of the partition, and to transmit these messages outside the partition, once the partition is repaired. In other words, *the occurrence of the partition is totally transparent*. This is not guaranteed by extended VSC: if a partition occurs, the application has the responsibility to forward messages broadcast within one partition to the processes outside of the partition, during the merge of the partitions. *The occurrence of the partition is not transparent to the application.*

8 Conclusion

The paper has introduced the *time-bounded buffering* problem in the context of the implementation of reliable communication over fair-lossy channels, and has shown how VSC addresses the issue thanks to the *program-controlled crash* feature. The paper has also shown that, while VSC provides more than Reliable Broadcast with time-bounded buffering, it has failed to do it adequately. This is related to the fact that VSC has overlooked the fundamental difference between output-triggered and input-triggered failure suspicions. The paper has shown the benefit that results from distinguishing between these two types of failure suspicions. It has also shown how this difference can be exploited by replacing the single VSC broadcast primitive by two broadcast primitives, called V-R-BROADCAST and V-FD-BROADCAST. In addition, instead of considering only the usual time-based suspicions, space constraints rather than time can be considered for output-triggered suspicions: as long as there is enough space to hold outgoing messages for retransmission, there is no reason to exclude any process based on timeouts.

We believe that the novel approach to building fault-tolerant distributed algorithms introduced in the paper is an important step toward improving the fail-over time of applications built on top of a VSC infrastructure.

References

[1] K. Birman and T. Joseph. Reliable communication in the presence of failures. *ACM Trans. on Computer Systems*, 5(1):47–76, February 1987.

[2] T. D. Chandra, V. Hadzilacos, S. Toueg, and B. Charron-Bost. On the impossibility of group membership. In *Proc. of the 15th ACM Symposium on Principles of Distributed Computing (PODC-15)*, pages 322–330, Philadelphia, Pennsylvania, USA, May 1996.

[3] T. D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of ACM*, 43(2):225–267, 1996.

[4] B. Charron-Bost, X. Défago, and A. Schiper. Broadcasting Messages in Fault-Tolerant Distributed Systems: the benefit of handling input-triggered and output-triggered suspicions differently. Technical Report IC-2002/020, EPFL, May 2002.

[5] G.V. Chockler, I. Keidar, and R. Vitenberg. Group Communication Specifications: A Comprehensive Study. *Computing Surveys*, 4(33):1–43, December 2001.

[6] V. Hadzilacos and S. Toueg. Fault-Tolerant Broadcasts and Related Problems. Technical Report 94-1425, Department of Computer Science, Cornell University, May 1994.

[7] R. De Prisco, A. Fekete, N. Lynch, and A. Shvartsman. A Dynamic View-Oriented Group Communication Service. In *Proc. of the 17th ACM Symposium on Principles of Distributed Computing*, Puerto Vallarta, Mexico, June 1998.

[8] A. Schiper and A. Sandoz. Uniform reliable multicast in a virtually synchronous environment. In *Proc. of the 13th IEEE Int'l Conf. on Distributed Computing Systems (ICDCS-13)*, pages 561–568, May 1993.