On the Inherent Cost of Generic Broadcast

Fernando Pedone⋆ André Schiper†

⋆Computer Networking Laboratory
Swiss Federal Institute of Technology (EPFL), CH-1015 Lausanne
Phone: +41 21 693 4797 Fax: +41 21 693 6600
E-mail: fernando.pedone@epfl.ch

†Distributed Systems Laboratory
Swiss Federal Institute of Technology (EPFL), CH-1015 Lausanne
Phone: +41 21 693 4248 Fax: +41 21 693 6770
E-mail: andre.schiper@epfl.ch

EPFL Technical Report IC/2004/46
20 May 2004

Abstract

This short paper establishes lower bounds on the time complexity of algorithms solving the *generic broadcast* problem. The paper shows that (a) to deliver messages in one round, no failures can be tolerated, (b) to deliver messages in two rounds, at most $f < n/3$ failures can be tolerated, where n is the number of processes in the system, and (c) to deliver messages in three rounds, at most $f < n/2$ failures can be tolerated. The lower bounds are tight: a simple algorithm capable of delivering messages in one round if $f = 0$ is presented, and algorithms solving generic broadcast in two rounds when $f < n/3$ and in three rounds when $f < n/2$ are known in the literature. The paper also shows that even in runs in which messages do not conflict, generic broadcast cannot achieve the same performance of reliable broadcast algorithms.

Keywords: group communication, algorithm analysis, fault tolerance

1 Introduction

This short paper establishes lower bounds on the time complexity of algorithms solving the *generic broadcast* problem. Generic broadcast assumes a symmetric, non-reflexive conflict relation on the set of messages, and requires ordered delivery only for conflicting messages [6]. If messages m and m' conflict, processes are required to deliver them in the same order; if they do not conflict, some process may deliver m and then m', and some other process may deliver m' and then m. The conflict relation is defined by the application. For example, in a system in which *read* and *write* messages are broadcast to replicated processes, read messages do not conflict with each other, and so, do not have to be delivered in the same order.

Formally, generic broadcast is defined by the primitives *broadcast*(m) and *deliver*(m), the conflict relation $\sim$, and the following conditions: (a) if a correct process p broadcasts a message m, then p eventually delivers m (*validity*); (b) if a correct process p delivers a message m, then every correct process q eventually delivers m (*agreement*); (c) for any message m, every process delivers m at most once, and only if m was previously broadcast by some process (*integrity*); and (d) if correct processes p and q both deliver conflicting messages m and m' (i.e., $m \sim m'$), then p and q deliver m and m' in the same order (*total order*).

Generic broadcast implementations can take advantage of the fact that some messages do not conflict and only order messages when really necessary. Ordering messages may be expensive or, if processes cannot use *oracles,* impossible. Even if processes have access to oracles, they should use them sparingly since oracles can make mistakes (e.g., failure detectors), which may degrade the performance of the system. Thus, if messages do not conflict, efficient generic broadcast algorithms will not require processes to always query their oracles.

The lower bounds established in this paper relate the resilience of generic broadcast algorithms (i.e., the total number of failures f the algorithms can tolerate) to their time complexity in runs in which oracles are not used. The paper shows that (a) to deliver messages in one round (formally defined in Section 2), no failures can be tolerated, (b) to deliver messages in two rounds, at most $f < n/3$ failures can be tolerated, where n is the number of processes in the system, and (c) to deliver messages in three rounds, at most $f < n/2$ failures can be tolerated. These lower bounds are tight: we give a simple algorithm that is capable of delivering messages in one round if $f = 0$. Algorithms solving generic broadcast in two rounds when $f < n/3$, and algorithms solving generic broadcast in three rounds when $f < n/2$ are known in the literature [1, 6, 7].

One can compare the cost of generic broadcast in runs in which no conflicting messages are broadcast (i.e., processes never query the oracle) with the cost of reliable broadcast. Reliable broadcast (both uniform and non-uniform) can be solved in asynchronous systems with reliable channels [4]. Non-uniform reliable broadcast algorithms can tolerate any number of failures and deliver messages in the end of the first round [3]. Uniform reliable broadcast algorithms, in which processes do not use oracles, require $f < n/2$ [5] and can deliver messages in the second round—actually, it turns out that this is the best that can be achieved without oracles.

Ideally, one would like to have a generic broadcast algorithm that performs like a reliable broadcast algorithm in runs in which no conflicting messages are broadcast. This paper shows that such generic broadcast algorithms do not exist. The difference between the cost of delivering non-conflicting messages with an optimal generic broadcast algorithm and the cost of delivering messages with an efficient reliable broadcast algorithm can be understood as "the inherent cost of generality."

2 System Model

We consider an *asynchronous* system composed of a set $\Pi = \{p_1, ..., p_n\}$ of processes, augmented by an oracle (e.g., a *failure detector* [3]) in order to make the generic broadcast problem solvable. Processes may fail by crashing, but do not behave maliciously (i.e., no Byzantine behavior). Processes that do not crash are *correct;* otherwise they are *faulty.* We assume a fully connected and reliable FIFO network (i.e., no loss, no duplication, and no creation of messages). Each process p_i has a buffer, $buffer_i$, that represents the set of messages that have been sent to p_i but not yet received; p_i receives the message when it removes it from its buffer. We assume a notion of round similar to the one in [2]: In any *run* of an algorithm, until it crashes, each process p_i repeatedly performs the following two *steps*, which define one *round*:

1. In the first step, p_i generates the (possibly *null*) messages to be sent to each process based on its current state, and puts these messages in the appropriate process buffers. If p_i crashes in round r, only a subset of the messages created in r by p_i are put in the buffers.
2. In the second step, p_i may query its oracle or not. If p_i decides not to query its oracle, it waits until there is one or more messages in its buffer, removes these messages, and determines its new state based on its current state and on the messages received. We do not define the behavior of p_i if it decides to query its oracle; our results will be stated in runs in which processes do not query their oracles.

Given the asynchrony of the system, it can be that one process terminates round r, while another has not started round r', $r' \leq r$. Thus, it is possible for a process in r to receive messages sent in r'. Moreover, without querying an oracle, no process can wait for messages from more than $n - f$ different processes in a round without risking being blocked forever [2].

3 The Lower Bounds

We establish conditions under which a message m broadcast in round 1 can be delivered at the end of round $r = 1$, at the end of round $r = 2$, and at the end of round $r \geq 3$. The bounds hold for algorithms in which processes query their oracles iff they have received two conflicting messages.

Proposition 1 *Let $\mathcal{C}$ be a non-empty conflict relation and $\mathcal{A}$ a generic broadcast algorithm using $\mathcal{C}$. In runs in which the oracle is not used, if messages broadcast in round 1 are delivered in round 1, then $\mathcal{A}$ does not tolerate any failures.*

PROOF: Assume for a contradiction that $\mathcal{A}$ tolerates one failure. Consider runs R_1 and R_2, and let m_1 and m_2 be two conflicting messages. In run R_x, $x \in \{1, 2\}$, process p_{3-x} does not execute any step (i.e., it fails in the beginning of the run). Process p_x broadcasts message m_x, and no other process broadcasts a message in R_x. By assumption, p_x delivers m_x in round 1.

Let R_3 be a failure-free run in which p_1 broadcasts m_1 and p_2 broadcasts m_2, such that any messages sent by p_1 only reach p_2 after round 1, and any messages sent by p_2 only reach p_1 after round 1. R_3 is such that for p_1, round 1 in R_3 is indistinguishable from round 1 in R_1, and since p_1 delivers m_1 in round 1 in R_1, it also delivers m_1 in round 1 in R_3. Similarly, for p_2, in round 1, runs R_2 and R_3 are indistinguishable, and since p_2 delivers m_2 in round 1 in R_2, it also delivers m_2 in round 1 in R_3. From agreement of generic broadcast, all processes deliver m_1 and m_2 in R_3, but p_1 delivers m_1 and then m_2, and p_2 delivers m_2 and then m_1, violating total order, and contradicting the fact that $\mathcal{A}$ solves generic broadcast. □

The lower bound of Proposition 1 is tight. Consider the following generic broadcast algorithm, which does not tolerate any failures. In the first round, if p_i wants to broadcast m, it sends m to all processes; otherwise it sends a *void* message to all processes. Each process waits for all messages, applies some deterministic function to decide on the delivery order and delivers every message received different from *void* in this order.

Proposition 2 *Let $\mathcal{C}$ be a non-empty conflict relation and $\mathcal{A}$ a generic broadcast algorithm using $\mathcal{C}$. In runs in which the oracle is not used, if messages broadcast in round 1 are delivered in round 2, then $\mathcal{A}$ does not tolerate $n/3$ failures.*

Proof: For a contradiction, assume that $\mathcal{A}$ tolerates $f \geq n/3$ failures. We divide set Π in three disjoint subsets, P_1, P_2, and P_3, of size f or less. By assumption, $\mathcal{A}$ can tolerate the failure of all processes in one single set.

Assume that m_1 and m_2 are conflicting messages. Let R_i, $i \in \{1, 2\}$, be a run in which processes in P_{3-i} do not execute any steps (i.e., they fail in the beginning of the run). Process p_i in P_i broadcasts message m_i, and no other process broadcasts any messages in R_i. Every process in $P_i \cup P_3$ executes rounds 1 and 2, and at the end of round 2 delivers m_i. By assumption, no process queries its oracle in R_i. Notice that processes in P_3 send in the first round of R_1 the same messages (if any) they send in the first round of R_2.

We build now an auxiliary run R_3. In R_3, p_1 and p_2 broadcast m_1 and m_2, respectively, in round 1. Further, (a) for processes in P_1, the first round of R_3 is indistinguishable from the first round of R_1, (b) for processes in P_2 the first round of R_3 is indistinguishable from the first round of R_2, and (c) processes in P_3 crash in round 1 immediately after having sent their messages. From generic broadcast, every process in $P_2 \cup P_3$ delivers m_1 and m_2 in the same order in R_3. Let r' be the smallest round in which all correct processes have delivered both messages, and assume m_1 is delivered before m_2. Using R_3, we construct the failure-free run R_4 as follows:

1. For processes in P_3, (a) rounds 1 and 2 are indistinguishable from rounds 1 and 2 in R_2, and (b) the messages sent by processes in P_3 in rounds $2, ..., r'$ are only received by processes in $P_1 \cup P_2$ after round r'. Item (a) is satisfied as follows: processes in P_2 send in round 1 of R_4 the same messages they send in round 1 of R_2. Messages sent from processes in P_1 do not reach processes in P_3 until after round 2.

2. For processes in $P_1 \cup P_2$, rounds $1, ..., r'$ are indistinguishable from rounds $1, ..., r'$ in R_3.

By (2), processes in $P_1 \cup P_2$ deliver messages m_1 and m_2 as in R_3, i.e., m_1 before m_2. By (1), processes in P_3 deliver m_2 in round 2 before delivering m_1. Since no process crashes in R_4, eventually, processes in P_3 also deliver m_1. Thus, the order property is violated, contradicting the fact that $\mathcal{A}$ solves generic broadcast. □

Proposition 3 *Let $\mathcal{C}$ be a non-empty conflict relation and $\mathcal{A}$ a generic broadcast algorithm using $\mathcal{C}$. In runs in which the oracle is not used, if messages broadcast in round 1 are delivered in round $r \geq 3$, then $\mathcal{A}$ does not tolerate $n/2$ failures.*

Proof: First we prove a basic fact about generic broadcast algorithms.

Lemma 1 *Let $\mathcal{C}$ be a non-empty conflict relation. There is no generic broadcast algorithm $\mathcal{A}$ using $\mathcal{C}$ that tolerates $n/2$ failures.*[1]

Proof: The proof is by contradiction using a partition argument. Assume that $\mathcal{A}$ tolerates $f \geq n/2$ failures. We divide set Π in two disjoint subsets, A and B, of size f or less. By assumption, $\mathcal{A}$ can tolerate the failure of all processes in one single set.

Let R_1 be a run in which processes in B do not execute any step. Some p_i in A broadcasts m_1, and no other process broadcasts any message in R_1. From the properties of generic broadcast, there is a round r_1 such that all processes in A have delivered m_1 by the end of round r_1.

Let R_2 be a run in which processes in A do not execute any step. Some p_j in B broadcasts m_2, which conflicts with m_1, and no other process broadcasts any message in R_2. From the properties of generic broadcast, there is a round r_2 such that all processes in B have delivered m_2 by the end of round r_2.

Consider now the failure-free run R_3 such that messages from processes in A (resp. B) to processes in B (resp. A) are very slow and do not reach their destinations before round $r = \max(r_1, r_2)$. For processes in A, until round r, runs R_1 and R_3 are indistinguishable, so processes in A deliver m_1 by the end of round r_1. For processes in B, until round r, runs R_2 and R_3 are indistinguishable, so processes in B deliver m_2 by the end of round r_2.

Since R_3 is a failure-free run, from the agreement property of generic broadcast, all processes deliver m_1 and m_2. Thus, processes in A deliver m_1 before m_2 and processes in B deliver m_2 before m_1, violating order and contradicting that $\mathcal{A}$ solves generic broadcast. □ Lemma 1

The proof of Proposition 3 follows immediately from Lemma 1 and Proposition 2. □

References

[1] M. K. Aguilera, C. Delporte-Gallet, H. Fauconnier, and S. Toueg. Thrifty generic broadcast. In *Proc. of the 14th International Symposium on Distributed Computing (DISC'2000)*, October 2000.

[2] M. Ben-Or and R. El-Yaniv. Resilient-optimal interactive consistency in constant time. *Distributed Computing*, 16:249–262, 2003.

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

[4] V. Hadzilacos and S. Toueg. Fault-Tolerant Broadcasts and Related Problems. In Sape Mullender, editor, *Distributed Systems*, pages 97–145. ACM Press, 1993.

[5] D. Malki, K. Birman, A. Ricciardi, and A. Schiper. Uniform Actions in Asynchronous Distributed Systems. In *Proc. of the 13th ACM Symposium on Principles of Distributed Computing*, August 1994.

[6] F. Pedone and A. Schiper. Generic Broadcast. In *13th. Intl. Symposium on Distributed Computing (DISC'99)*, pages 94–108. Springer Verlag, LNCS 1693, September 1999.

[7] F. Pedone and A. Schiper. Handling message semantics with generic broadcast protocols. *Distributed Computing*, 15(2):97–107, 2002.

[1] This result has also been stated, but not proved, in [1].