*Published in the proceedings of the work in progress session of theReal-Time Systems Symposium. London, 3-6 Dec 2001.*1

# A Timely Distributed Consensus Solution in a Crash/Omission-Fault Environment

George M. A. Lima* Alan Burns

Real-Time Systems Research Group
Department of Computer Science
University of York, England
{gmlima,burns}@cs.york.ac.uk

## Abstract

*A timely protocol to solve the distributed consensus problem that tolerates process crashes and message omissions is described. The protocol is optimal in terms of the number of communication steps needed to achieve consensus. The model on which the protocol is based relies on a priority-based communication network, a kind of network commonly used in practice to support real-time systems.*

**Keywords**: Distributed Consensus, Fault Tolerance, Hard Real-Time System

## 1. Introduction and Background

Distributed consensus, a fundamental problem in distributed systems, can be formulated as follows: all correct processes from a set of processes propose a value and then they have to decide on a common value. A distributed consensus server can provide a basic building block to implement other fault-tolerant services [8, 3, 13, 9].

More formally, the consensus problem can be specified in terms of the following properties [12]:

*Termination*: Every correct process eventually decides some value.

*Validity*: If a process decides $v$, then $v$ was proposed by some process.

*Agreement*: No two processes decide differently.

In this work we are interested in a solution for the consensus problem which can provide support for fault-tolerant hard real-time systems, whose correctness depends on *timeliness*. Thus, we consider the timed termination property for specifying the *timed* consensus problem:

*Timed Termination*: Any process that decides, decides within a maximum known period of time.

The specifications of both timed and non-timed consensus problems are very simple. Their simplicity, however, may hide the real difficulties in solving them. In fact, it has been shown that in asynchronous distributed systems, where no assumption on time is made, there is no deterministic solution for the consensus problem when faults are considered [7]. This has motivated several theoretical and practical researchers to look for other alternatives to solve the consensus problem [6, 2, 1, 5, 4]. They have proposed some *semi-synchronous* models of computation, which are based on the assumption that *eventually* the required level of synchrony for solving consensus holds. Hence, they do not solve the timed consensus problem since they do not deal with the timed termination property.

Usually, the timed consensus problem is addressed in *synchronous* distributed systems [11]. These are systems that have known bounds on message transmission delays and processing speeds. In this work we show that it is possible to solve the timed consensus problem in a less strict model of computation (section 2), where: (a) the message transmission delay bound may be violated by all but one process provided that it does not crash and; (b) there is a priority-based network, which is able to schedule messages for transmission according to their priorities. By assuming (a) and (b) and assigning priorities to messages appropriately, we divide the system into parts and assure that at least one of them will be timely. In other words, our assumed model of computation can be considered synchronous but only with respect to the process that transmits the highest priority message.

Other models of computation [14, 15] have assumed different levels of synchrony in different parts of the system. These models consider the system made of *two* distinct parts, one synchronous and the other semi-synchronous.

*Supported by CAPES/Brazil under grant: BEX1438/98-0.

The synchronous part is used to control and adjust the semi-synchronous one, where complete synchronism may not hold from time to time. Unlike them, our model does not require a special role for different parts. For example, if the timely process (synchronous part) crashes, another process automatically assumes the role of the crashed process. Interestingly, this is done without any extra computational effort by the processes due to the scheduling property of the network.

Based on the assumed model we propose a protocol that solves the timed consensus problem (section 3). The protocol tolerates both message timing/omission faults and process crashes. The protocol is optimal in terms of the number of communication steps needed to achieve consensus and the number of tolerated process crashes. These characteristics make the protocol very attractive and provide a step towards the design of flexible and fault-tolerant real-time systems.

## 2. Computational Model

We consider a set of processes $\Pi = \{p_1, p_2, \ldots, p_n\}$ allocated to a set of distributed nodes, which are fully connected by a communication network. Processes are uniquely identified and their identities are common knowledge in the system. Any transmitted message in the system has a priority and it is scheduled for transmission according to its priority. Notice that networks capable of scheduling messages according to their priorities are desirable to support real-time applications and, in fact, they are largely used in practice (e.g. CAN [10]).

We assume a message transmission delay threshold, $\delta$. Any message transmitted within $\delta$ time units is considered timely. A process is timely when its messages are timely. We assume that a real-time scheduling mechanism guarantees that any local computation carried out by processes is timely. In particular, we assume that any process updates its state upon the receiving of a message within $\alpha$ time units. By relying on the timeliness of local computation, we can say that the occurrence of errors and possible overload in the communication network are the only source of system unpredictability.

Regarding the fault model, we assume that messages can be dropped or arbitrarily delayed by the communication network and processes may crash. A process crashes at time $t$ if it stops both receiving and transmitting messages indefinitely from time $t$. Crashed processes do not recover. Note that we exclude from the fault model the possibility of having corrupted/spurious messages in the network. However, in practice, this does not restrict the applicability of the protocol. Since most corrupted/spurious messages can be detected by checksum based mechanisms, one can assume that these kinds of fault can be easily transformed into message omission faults.

Despite error occurrences we assume that in the worst-case scenario the communication network has enough available bandwidth to guarantee some timeliness in the system. More specifically, we assume that if two or more messages are being transmitted concurrently, at least one of them is timely, provided that its sender process does not crash. As messages are scheduled for transmission according to their priorities, the process that transmits the highest priority messages is guaranteed to be timely, provided that it does not crash during the transmission. Note, on the other hand, that this assumption allows a reasonable level of flexibility. For example, if a process that transmits the highest priority messages crashes when transmitting a message, its message may not be delivered.

Processes have access to hardware clocks and there is a known bound on their clock drift rates, denoted as $\rho$. Note that this assumption fits well the actual hardware found in practice.

## 3. Solving the Timed Consensus Problem

Whenever a process in $\Pi$ wants to agree on a common value with the other processes, it calls the primitive $\mathsf{consensus}(v)$, where $v$ is its proposed value. This primitive is described in figure 1. The protocol tolerates $f < |\Pi|$ process crashes and message timing/omission faults. During the execution of the protocol processes exchange their messages throughout $f + 1$ communication steps (rounds) and in each of them $|\Pi|^2$ messages are exchanged. At the end of the last round, all non-crashed processes agree on a common value. Our protocol is optimal with regard to the number of rounds needed to achieve consensus since it is well known that to tolerate $f$ process crashes at least $f + 1$ communication steps [11] are needed.

At each round $r$, any process $p_i$ sends a message at priority $r.i$ with its current estimated decision value to all processes in $\Pi$ (including itself), where $r.i$ is a bit concatenation operation between $r$ and $i$. The higher the value of $r.i$ is, the higher the priority is. The initial estimated value of any process is its proposed value. For the other rounds ($r > 1$), the estimated value of $p_i$ is set to the estimated value carried by the highest priority received message. Thus, if the process which sends the highest priority message in round $r$, say $p_k$, does not crash, then all non-crashed processes will receive the message sent by $p_k$ and update their estimated values to $p_k$'s estimate. However, $p_k$ may fail during the broadcast of its estimated value by crashing. This may lead to situations in which only some processes in $\Pi$ receive $p_k$'s message.

Each round of the protocol has a maximum duration of time, $\Delta$. The function $\mathsf{SetTimer}(\Delta)$ sets this timeout. When the timeout expires the function $\mathsf{ExpTimer}()$ returns

% Each process $p_i$ in $\Pi$
% executes the following algorithm
% to achieve consensus

**procedure** consensus($v$)
$est_i \leftarrow v$
**for** $r = 1$ **to** $f + 1$
    SetTimer($\Delta$)
    **send** $(est_i, r)$ at priority $r.i$ to all $p_j$
    **wait until** $[\forall p_j \in \Pi :$
        received $(est_j, r) \lor$ ExpTimer()$]$
    Let $(est_k, r')$ be the
        highest priority received message
    $est_i \leftarrow est_k$
**endfor**
return($est_i$)

**Figure 1. The consensus algorithm to reach agreement on a common value**

true. This avoids a process awaiting either late/lost messages or messages that are from processes that are already crashed. A safe choice of $\Delta$ is $(|\Pi|\delta + \alpha)(1 + \rho)$ since: (a) $|\Pi|\delta$ is the time necessary to transmit $|\Pi|$ concurrent messages scheduled by their priorities; (b) $\alpha$ is the worst-case response time of a local computation (i.e. the receiving of messages); and (c) local clocks may drift $\pm\rho$ apart from each other.

As an illustration of the consensus protocol behaviour, consider figure 2, where two scenarios are presented. The horizontal arrows represent the real-time line and the diagonal ones represent sent messages. If a message is not successfully received at its destination, a vertical segment indicates the occurrence of an error. Suppose that $\Pi = \{p_1, p_2, p_3\}$, assume $f = 2$ and let a, b and c be the values proposed by $p_1$, $p_2$ and $p_3$, respectively. The rounds of processes are considered synchronised in order to make the illustration clear. In scenario 2 (a), where $p_3$ is timely and does not crash in the first round, $p_2$ and $p_1$ update their estimated value to c. Thus, from the next round onwards, c is the only possible value on which all processes can agree. Note that this holds even if $p_3$ (or any other process) fails in future rounds, as illustrated in the figure. In scenario 2 (b), on the other hand, $p_3$'s message does not arrive at $p_2$ since $p_3$ crashes during the first broadcast. Because of this, the highest priority message $p_2$ receives in the first round is its own message while $p_1$ receives $p_3$'s message. Hence, in the second round the estimated values of processes $p_1$ and $p_2$ are c and b, respectively. At the end of the second round, though, they agree on a common estimated value and in the third round they can decide on it.

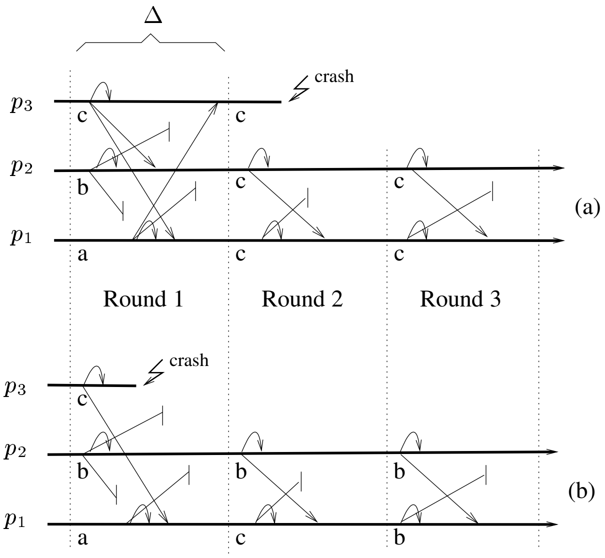


**Figure 2.** Illustration of an execution of the protocol described figure 1

It is not difficult to show that the described protocol is a timely solution for the consensus problem. What follows is some argumentation since the formal proof cannot be described here due to the lack of space. The timed termination property follows from the fact that the maximum duration for any round is $\Delta$. Thus, any process that decides, finishes executing the protocol within $(f + 1)\Delta$ from the time it started. If this does not violate the application deadline, we say that timeliness is guaranteed. Also, it is clear that the validity property holds since processes update their estimated values based on their received messages, which by assumption cannot be arbitrarily created or corrupted. The agreement property follows from the following arguments. Firstly, notice that there is a round $r$ during any execution of the protocol in which no process crashes. This is because at most $f$ processes may crash and all correct processes execute $f + 1$ rounds. Also, it can be shown that all processes that do not crash in $r$ will agree on a common value by the end of $r$. Finally, it is clear that if there is a round in which all correct processes have the same estimated value, they will keep this value until the end of the protocol since there is no corrupted/spurious message in the system.

# 4. Final Comments

By describing a simple protocol, we have shown that it is possible to solve the timed consensus problem in an environment where messages may be lost or arrive too late and processes may crash. This represents a significant step towards the design of flexible and fault-tolerant real-time systems. The protocol is currently being extended to cope with more general cases, where more than a group of processes concurrently execute the consensus protocol. Also, future works must include other agreement problems such as clock synchronisation.

## References


[1] M. K. Aguilera, W. Chen, and S. Toueg. Quiescent Reliable Communication and Quiescent Consensus in Partitionable Network. *Theoretical Computer Science (special issue on distributed algorithms)*, 200(1):3–30, June 1999.

[2] T. Chandra and S. Toueg. "Unreliable Failure Detectors for Reliable Distributed Systems". *Journal of ACM*, 43(2):225–267, Mar. 1996.

[3] T. D. Chandra, V. Hadzilacos, S. Toueg, and B. Charron-Bost. "On the Impossibility of Group Membership". In *Proc. of the 15th Annual ACM Symp. on Principles of Distributed Computing (PODC 96)*, pages 322–330, New York, USA, 1996. ACM.

[4] F. Cristian and C. Fetzer. "The Timed Asynchronous Distributed Model". *IEEE Transaction on Parallel and Distributed Systems*, 1999.

[5] D. Dolev, R. Friedman, I. Keidar, and D. Malkhi. "Failure Detectors in Omission Failure Environments". In *ACM Symp. on Principles of Distributed Computing (PODC 97)*, page 286, Santa Barbara, California, USA, Aug. 1997.

[6] C. Dwork and N. Lynch. "Concensus in the Presence of Partial Synchrony". *Journal of ACM*, 35(2):288–323, Apr. 1988.

[7] M. J. Fischer, N. A. Lynch, and M. S. Peterson. Impossibility of Distributed Consensus with One Faulty Process. *Journal of ACM*, 32(2):374–382, 1985.

[8] V. Hadziacos and S. Toueg. "Fault-Tolerant Broadcast and Related Problems". In Sape Mullender, editor, *Distributed Systems*, chapter 5. Addison-Wesley, 2nd edition, 1993.

[9] V. Hadzilacos. "On the Relationship between the Atomic Commitment and Consensus Problems". In *Proc. of the Workshop on Fault-Tolerant Distributed Computing. Lecture Notes in Computer Science Vol. 448 (B. Simons and A. Spector, eds.), Springer-Verlag*, volume 448, pages 201–208, 1990.

[10] Int'l Standards Organisation. *ISO 11898. Road Vehicles – Interchange of digital information – Controller area network (CAN) for high speed communication*, 1993.

[11] N. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publisher, 1996.

[12] G. Neiger and S. Toueg. "Automatically Increasing the Fault-Tolerance of Distributed Systems. *Journal of Algorithms*, 11(2):374–419, Sept. 1990.

[13] L. S. Sabel and K. Marzullo. Election vs. consensus in asynchronous systems. Technical Report TR95-1488, Cornel University, Feb. 1995.

[14] P. Veríssimo and C. Almeida. "Quasi-Synchronism: A Step Away from the Traditional Fault-Tolerant Real-Time Systems Model". In *Bulletin of the Technical Committee on Operating Systems and Application Environments (TCOS 95)*, pages 35–99, 1995.

[15] P. Veríssimo, A. Casimiro, and C. Fetzer. "The Timely Computing Base: Timely Actions in the Presence of Uncertain Timeliness". In *Int'l Conf. on Dependable Systems and Networks (DSN 00)*, June 2000.