Timeout Adaptive Consensus: Improving Performance through Adaptation

Mônica Dixit, Henrique Moniz, António Casimiro

DI–FCUL TR–2010–06

November 11, 2010

Departamento de Informática
Faculdade de Ciências da Universidade de Lisboa
Campo Grande, 1749–016 Lisboa
Portugal

Technical reports are available at `http://www.di.fc.ul.pt/tech-reports`. The files are stored in PDF, with the report number as filename. Alternatively, reports are available by post from the above address.

Timeout Adaptive Consensus: Improving Performance through Adaptation

Mônica Dixit, Henrique Moniz, António Casimiro
Faculty of Sciences of the University of Lisboa
Lisboa, Portugal
{mdixit,hmoniz,casim}@di.fc.ul.pt

November 11, 2010

Abstract

Algorithms for solving distributed system problems, such as consensus, often use timeouts as a mean to achieve progress, even if encapsulated in failure detection services. They are designed in a way that safety is always preserved despite timeouts being too small or too large. A "reasonable" timeout value is usually selected, such that the run-time performance is acceptable in the normal case.

In this paper we transform a fixed timeout consensus protocol into a timeout adaptive protocol, showing how this can be done in a structured way and demonstrating the performance improvements that we achieve. Our results are particularly significant in networking environments subject to uncertain or varying end-to-end delays, such as wireless environments with several nodes contending for medium access. With the timeout adaptive solution the number of transmitted broadcasts per consensus execution is always kept small, despite the number of involved processes. In addition, the overall protocol latency is also improved when comparing to the static version.

1 Introduction

The consensus problem is a fundamental building block in the design of distributed systems, as it contributes to the coordination of actions in order to achieve consistent decisions. Traditionally, consensus protocols assume a component failure model, relying on end-to-end communication and permanently faulty components, that can be processes or links [13, 14]. Various solutions for the consensus problem were proposed under this model, based on different approaches to enrich the synchrony of the system, such as assuming partial synchrony [8], assuming the availability of failure detectors [5] or using wormholes [11].

A different model, assuming transient communication failures [17, 18], is considered in a recently introduced randomized k-consensus protocol for wireless networks [10]. This is a more realistic model for dynamic settings and allows the

algorithm to take advantage of the natural broadcasting medium, reducing the cost of sending messages to multiple processes.

In this work we present an adaptive version of this consensus protocol. As usual in consensus algorithms, it relies on a timeout to decide whether and when messages should be retransmitted. The timeout value is fundamental to the performance of the algorithm: a too large timeout may delay necessary retransmissions, leading to a longer execution time, while a too small timeout may cause excessive and unnecessary retransmissions, overloading the network, and ultimately leading as well to a longer execution time. Using a fixed timeout which may be proper for some specific environment condition is not the best choice if the consensus will be executed under a dynamic environment. The timeout value should be adjusted to different enviroment conditions. In our solution, the timeout is dynamically adapted according to changes in the communication latency. The adaptation process is driven by *Adaptare*[7], a framework to support dependable adaptation of applications operating in stochastic environments.

Our contributions are twofold. First, we introduce an approach to architect adaptive solutions that may be generically applicable to other timeout-dependent distributed systems algorithms. Second, we improve the overall performance of the considered consensus algorithm, and thus we show that the proposed approach is effective and valuable.

The experimental results provided in the paper were obtained by running the static and adaptive versions of the algorithm in the Emulab testbed [20]. They show that by using *Adaptare* to support adaptation we can improve the performance of consensus under different communication conditions, both in more reliable settings (with loss rate around 1%) and less reliable ones (with loss rate of 10%), and with a varying number of processes. In particular, the overall latency of a consensus execution can be reduced in more reliable settings and may be kept unaffected despite the increased number of faults occurring in unreliable settings. Additionally, with the adaptive solution the network load is always lower, especially when the number of processes increases.

The paper is organized as follows. In the next section we present some related work. Section 3 introduces our solution, including a brief description of the considered consensus algorithm, the *Adaptare* framework, and the integration of both. Section 4 presents the practical evaluation of the proposed adaptive consensus algorithm. Finally, in Section 5, we conclude the paper.

2 Related work

The main problem addressed in this paper is how to improve the performance of a timeout-dependent distributed protocol by adapting the predefined timeout value instead of using a fixed one. This is a relevant problem when considering dynamic environments, in which the communication latency varies over time. However, research in distributed algorithms usually disregards this problem, since it is mostly

concerned with run-time performance and orthogonal to correctness. In fact, timers and timeouts are seen as artifacts that are necessary to ensure that processes make progress and do not wait indefinitely for possibly crashed processes or lost messages. This way, selecting timeout values is problem that may be addressed independently.

Interestingly, the problem has deserved some attention in the context of failure detectors, with the objective of improving the Quality of Service of failure detection [6]. Several timeout-based adaptive solutions have been proposed [3, 9, 12], which typically use relatively simple methods to determine and adapt timeout values, or solutions based on estimators and safety margins. However, in these works the problem is treated as a whole, without a clear separation between the monitoring and adaptation support framework, and the specific detection algorithm, being often necessary to adjust operational parameters in order to favor the detection time, or the detection accuracy. In our case we use the *Adaptare* framework, which provides a generic interface that might be used either for adaptive failure detectors as well as for adaptive consensus or other services that rely on temporal settings.

The need for dynamic adaptation of timeouts can be found, however, in several particular contexts, like in multimedia applications [2, 16], and in communication systems to improve congestion management [15, 21]. These approaches rely on the ability to monitor several operational parameters, among which the communication Round-Trip Time (RTT). Typically, timeouts are adapted in an ad hoc way, depending on the specific case, by inferring the new values based on the observed RTTs and using, for instance, smoothing and weighting methods. We use probabilistic techniques, implemented within *Adaptare*, which are tailored to explore the stochastic behaviors that are usually associated to communication delays, both in wired and wireless networks.

A distinguishing aspect of our work is that we use the *Adaptare* framework which, to our knowledge, is unique in the way it deals with application requirements. The application just has to specify a dependability-related value, the expected probability that the framework output (the timeout) will hold throughout the execution, as further detailed in Section 3.2.

3 Adaptive Consensus

In this paper we propose and evaluate an adaptive version of the consensus protocol introduced in [10]. The goal is to improve its performance by dynamically adjusting the timeout in response to network delay variations, following the indications provided by *Adaptare* [7], a framework to support adaptive applications.

This section briefly describes the consensus protocol and the *Adaptare* framework, for self-containment, but interested readers can find detailed information in the given references. The section also explains the integration between the consensus algorithm and the *Adaptare* framework.

3.1 The consensus protocol

The consensus protocol introduced in [10] solves the k-consensus problem in wireless ad hoc networks. In the k-consensus problem, each process p_i proposes an initial binary value $v_i \in 0, 1$, and at least $k > \frac{n}{2}$ of them have to agree on a common proposed value, where n is the number of participant processes. The remaining processes are not required to decide, but if they decide, it must be on the same value decided by the k processes.

The work assumes a communication failure model [17, 18], which is more appropriate to represent wireless ad hoc networks. An impossibility result presented in [17] states that under this model there is no finite time deterministic algorithm that allows n processes to reach k-agreement, if more than $n - 2$ transmission failures occur in a communication step.

The protocol introduced in [10] circumvents this impossibility result by employing randomization. In its randomized version, the k-consensus problem is formally defined by the following properties:

- **Validity.** If all processes propose the same value v, then any process that decides, decides v.
- **Agreement.** No two processes decide differently.
- **Termination.** At least k processes eventually decide with probability 1.

The algorithm tolerates omission faults. Safety, defined by the validity and agreement properties, is ensured regardless of the number of omission faults in each round, while liveness, defined by the termination property, is guaranteed if the number of omission faults per round is $f \leq \lceil \frac{n}{2} \rceil (n - k) + k - 2$.

In a round, each process p_i broadcasts a message containing its identifier, proposal, and internal state, and receives messages broadcast by the other processes. After analyzing the received messages, process p_i updates its state and possibly decides on some value, or initiates a new round. Because the algorithm is based on the communication failure model, some messages that a process is supposed to receive may be lost. If a process p_i does not receive sufficient messages to make progress within a pre-defined amount of time, the original message is retransmitted. Therefore, a timeout mechanism is used to trigger retransmissions.

The timeout value is fundamental to the performance of the algorithm and must be set in accordance with the network conditions, namely with the observed end-to-end delays. A too large timeout may delay necessary retransmissions, leading to a longer execution time, while a too small timeout may cause excessive and unnecessary retransmissions, overloading the network, and ultimately leading as well to a longer execution time.

We propose an adaptive version of this algorithm, based on a dynamic timeout value, which is adjusted in runtime according to the current environment conditions. The adaptation process is driven by the *Adaptare* framework, described in the next section.

The complete consensus algorithm and correctness proofs are presented in [10].

3.2 Adaptare framework

Adaptare [4, 7] is a framework developed to support adaptive systems and applications in stochastic environments, driving the adaptation process according to dependability requirements specified by the client applications.

More specifically, the *Adaptare* framework is fed with recent measurements of a temporal variable and performs a probabilistic analysis of these recent samples in order to determine an upper bound on the observed variable, which must hold with a given probability. This probability, referred as the bound *coverage*, is the dependability requirement specified by the client application.

The framework is based on the assumption that the system behaves stochastically, alternating periods during which the conditions of the environment remain fixed (*stable phases*), with periods during which the environment conditions change (*transient phases*). During stable phases, the statistical process that generates the data flow is under control and hence we can compute the corresponding distribution using an appropriate number of samples. On the other hand, if the environment conditions are changing, then the associated statistical process is actually varying and no fixed distribution can describe its real behavior.

Figure 1 shows the *Adaptare* architecture. Three parameters must be provided at the framework interface: (i) the history size h, which defines the number of recently collected samples that will be analyzed; (ii) the most recently measured samples, used for the network characterization; and (iii) the required coverage for the computed bound. The framework uses this input data to characterize the network conditions and, based on this characterization and on the required coverage, it computes a new bound which is returned to the client application.

The network characterization process is composed by two phase detection mechanisms, which were implemented as goodness-of-fit (GoF) tests: the Kolmogorov-Smirnov (KS) test and the Anderson-Darling (AD) test. GoF tests are formal statistical procedures used to assess the underlying distribution of a data set. A stable period with distribution $\hat{D}$ is detected when some GoF test establishes the goodness of fit between the postulated distribution $\hat{D}$ and the evidence contained in the experimental observations [19].

Both AD and KS are distance tests based on the comparison of the cumulative distribution function (CDF) of the assumed distribution $\hat{D}$ and the empirical distribution function (EDF), which is a CDF built from the input samples. The current implementation of *Adaptare* tests five distributions: exponential, shifted exponential, Pareto, Weibull, and uniform. If the phase detection mechanisms do not identify a stable phase with any of these distributions, they assume that the environment is changing, i.e., a transient phase is detected.

To execute their statistical tests, the phase detection mechanisms need to estimate the parameters of each postulated distribution $\hat{D}$. Our parameters estimation equations are based on the maximum likelihood estimation (MLE) [1, 19] for the

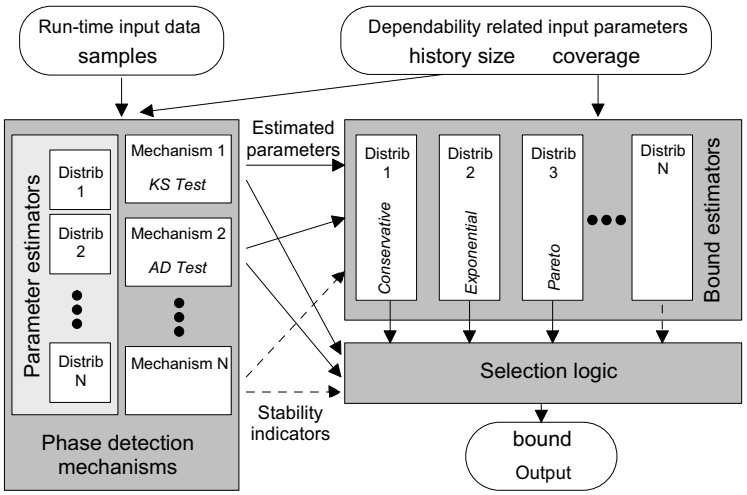


Figure 1: *Adaptare* framework architecture

exponential, shifted exponential, Pareto and uniform parameters, and on linear regression [19] for the Weibull distribution. The reason for using a different technique for the Weibull distribution is that the MLE method does not have a closed form solution for Weibull: the equations must be simultaneously solved using iterative algorithms, which are very time-consuming.

The output bound is computed depending on the phase detection mechanisms results. If some specific distribution is identified after the analysis of the provided measured samples, the output bound is derived from this distribution CDF and the required coverage C. Otherwise, a conservative bound which holds for all distributions is computed based on the one-sided inequality of probability theory.

The output bound and parameters estimators are presented in Table 1, where $\{s_1, s_2, ..., s_h\}$ are the *ordered* samples, $x_i = \ln(s_i)$, $y_i = \ln(-\ln(1 - F(s_i)))$, and $F(x_i) = (i - 0.3)/(h + 0.4)$ (approximation of the median rankings for the linear regression).

A complete description of *Adaptare*, including its architecture, implementation details, complexity analysis and a comprehensive performance evaluation is presented in [7].

3.3 Integration details

The *Adaptare* framework runs as an independent service, in each machine participating on the consensus execution. A well defined interface to use the service is provided, and may be used as a typical RPC service. In our implementation consensus processes and *Adaptare* communicate through UDP messages, but other approaches could be used as well. There are four different types of messages that are defined in this interface. Two indication messages that may be

Table 1: Adaptare bound and parameters estimators.

Estimator	**Output bound t (Coverage=C)**	**Parameter estimators**
Exponential	$t = \frac{1}{\lambda} \ln \frac{1}{1-C}$	$\hat{\lambda} = \frac{1}{\bar{s}}$
Shifted Exponential	$t = \gamma + \lambda \ln \frac{1}{1-C}$	$\hat{\lambda} = h\frac{(\bar{s}-s_{min})}{h-1}$ $\hat{\gamma} = s_{min} - \frac{\hat{\gamma}}{h}$
Pareto	$t = \frac{k}{\sqrt[\hat{\alpha}]{1-C}}$	$\hat{k} = s_{min}$ $\hat{\alpha} = \frac{h}{\sum_{i=1}^{h} \ln \frac{s_i}{\hat{k}}}$
Weibull	$t = \lambda \sqrt[\gamma]{\ln \frac{1}{1-C}}$	$\hat{\alpha} = e^{-\frac{\bar{y}-\hat{\gamma}\bar{s}}{\hat{\gamma}}}$ $\hat{\gamma} = \frac{\sum_{i=1}^{h} x_i y_i - \frac{\sum_{i=1}^{h} x_i \sum_{i=1}^{h} y_i}{h}}{\sum_{i=1}^{h} x_i^2 - \frac{(\sum_{i=1}^{h} x_i)^2}{h}}$
Uniform	$t = C(b-a) + a$	$\hat{a} = s_{min}$ $\hat{b} = s_{max}$
Conservative	$t = E(D) + \sqrt{\frac{V(D)}{1-C} - V(D)}$	$\hat{E(D)} = \bar{s}$ $\hat{V(D)} = \frac{\sum_{i=1}^{h}(s_i-\bar{s})^2}{h-1}$

sent to *Adaptare*, `MSG_PROGRESS` and `MSG_TIMEOUT`, one request message, `MSG_TIMEOUT_REQUEST`, and one reply, `MSG_TIMEOUT_REPLY`.

As described in Section 3.1, after sending a broadcast message, a process waits for sufficient responses from other processes during a given amount of time, referred here as the algorithm's `timeout`. If it receives sufficient responses to make progress within this time, a `MSG_PROGRESS` message is sent to *Adaptare*. Otherwise, a `MSG_TIMEOUT` message is sent. Both messages include the amount of time elapsed since the last broadcast, which is an end-to-end measure of the time it took to either complete a successful round or skip to a new round of retransmissions.

Upon the reception of either a `MSG_PROGRESS` or a `MSG_TIMEOUT` message, *Adaptare* updates the local history of measurements (adding the new one and removing the older one to keep a constant history size h) and computes a new value for the `timeout`. However, if the received message type is `MSG_TIMEOUT`, the new computed value for `timeout` will be a conservative value, which holds for all probability distributions (see Section 3.2). In fact, when a timeout expires this can be an indication that the network became unstable or exhibits increased delays, and thus assuming a higher and more conservative timeout will be positive. Note that being conservative does not necessarily mean using very high timeouts: this depends on the required coverage, which is specified for the overall execution. To keep the approach dependable, the timeout (conservative or not) just has to be as high as necessary to ensure the required coverage.

Before sending a broadcast message, each consensus process requests a new

`timeout` value by sending a `MSG_TIMEOUT_REQUEST` message. *Adaptare* replies each request with a `MSG_TIMEOUT_REPLY` message, which contains the most recently computed `timeout`. Note that this value is updated whenever a `MSG_PROGRESS` or a `MSG_TIMEOUT` message is received. In this way, *Adaptare* does not need to compute a new value when a `MSG_TIMEOUT_REQUEST` message is received, decreasing the process waiting time.

4 Performance Evaluation

We executed a set of experiments in order to compare the performances of the static timeout consensus protocol presented in [10] and of the timeout adaptive version that we considered in this paper. In this section we present the achieved results and compare the two solutions in terms of (i) algorithm's latency, (ii) number of broadcast messages sent by each process per consensus execution and (iii) average timeout.

4.1 Environment setup

The experiments were carried out on the Emulab testbed [20]. A total of 16 nodes were used, each one with the following hardware characteristics: Pentium III processor, 600 MHz of clock speed, 256 MB of RAM, and 802.11 a/b/g D-Link DWL-AG530 WLAN interface card. The operating system was the Fedora Core 4 Linux with kernel version 2.6.18.6. The nodes were located on the same physical cluster and were, at most, a few meters distant from each other.

4.2 Methodology

We performed a set of experiments by combining different parameters for the consensus protocol and the *Adaptare* framework. Our evaluation is based on the analysis of three performance metrics: the algorithm latency, which is the time that a particular process takes to decide, the number of broadcast messages sent by a process until reaching a decision state, and the average timeout in a consensus execution.

In our experiments, the number of processes participating in the consensus execution was set to 4, 7, 10, 13, and 16. Processes with odd identifiers initially propose the value 1, while processes with even identifiers propose 0, guaranteeing a divergent initial proposal set. The static timeout version of the protocol was configured to use a timeout of 10ms. This value was obtained by running a set of empirical tests with different static values, being the value that provided the better results on average. The initial timeout for the timeout adaptive version is also 10ms, but it is dynamically adjusted according to *Adaptare* outputs.

The experiments were carried out under two different environment conditions. The first one corresponds to the baseline environment provided by the Emulab

network, in which the omission rate is very small, always below 1%, in typical no-contention situations, thus constituting what we call a failure-free (with no losses) environment. The second environment represents a more unreliable network, in which the baseline message loss is 10%. This behavior was obtained by injecting omission faults in the broadcast operation such that every broadcast is sent with a probability of 90%.

Regarding the *Adaptare* configuration, we configured the history size to $h = 30$, since the evaluation presented in [7] indicated that this value provides the best results. For the required coverage, we considered two values: 99% and 80%. A higher coverage is typically used when it is fundamental to ensure that assumed bounds are never violated, but in this case this is not a crucial aspect, and therefore we also considered a smaller coverage value, which also yields smaller bounds (timeouts) that may be more adequate for performance objectives.

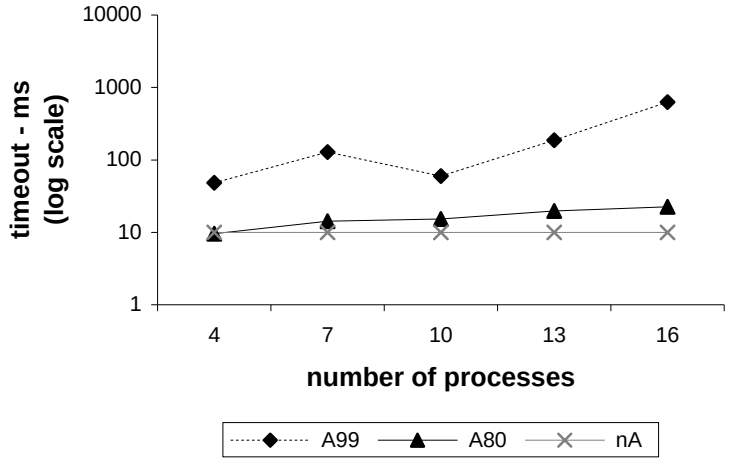
An experiment comprises 20 consecutive executions of the algorithm, for each considered configuration. The experiments were executed during five different days. The values presented in this evaluation correspond to the average of the several results obtained during the five days.

4.3 Results

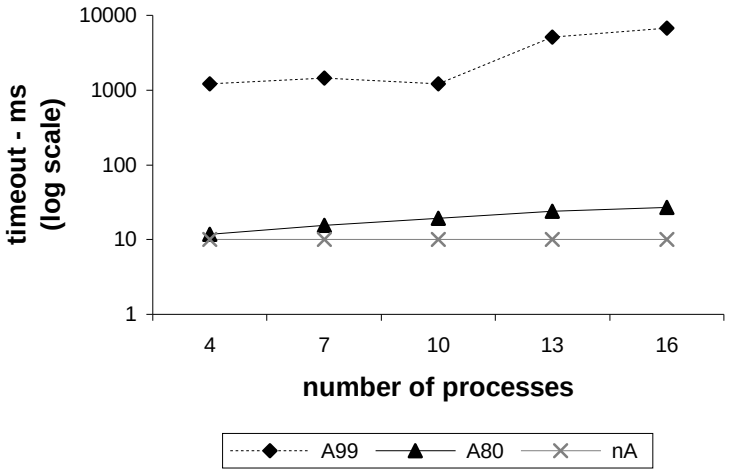
Figures 2, 3 and 4 present the average results for the timeout, number of broadcasts and latency of the consensus algorithms. In these graphs, the `A99` and `A80` series correspond to the adaptive version of the algorithm with required coverage of 99% and 80%, respectively, while the `nA` series corresponds to the non adaptive algorithm. The average timeout values are plotted in logarithmic scale in order to facilitate visualization.

Figure 2 shows the average timeouts. For both networking conditions, the timeouts follow the same trend. The lower timeout is the fixed value of the non adaptive algorithm, 10ms. The `A80` adaptive algorithm presents average timeouts of 9ms to 20ms, therefore close to the non adaptive timeout. On the other hand, the `A99` adaptive timeouts are at least one order of magnitude higher than in the other two cases. Clearly, when requesting a higher coverage, *Adaptare* will compute higher bounds (timeout values), such that only in 1% of the cases the timeout will expire and retransmissions will occur. Given that safety is always ensured, even if timeouts expire, requiring a high coverage is not really necessary in this case, and is in fact an overkill in some cases, as revealed by the performance results that we discuss ahead. We note that in other situations it may be necessary to require a high coverage. For instance, in the implementation of failure detectors it is fundamental to ensure that the detector does not make mistakes, and therefore coverage must typically be very high, even if other metrics, like detection time, are negatively affected.

An interesting observation is that in the lossy environment (Figure 2(b)) the `A99` adaptive timeouts are even higher, while the `A80` timeouts are not significantly affected by losses. There is an simple explanation for this fact. Recall that every



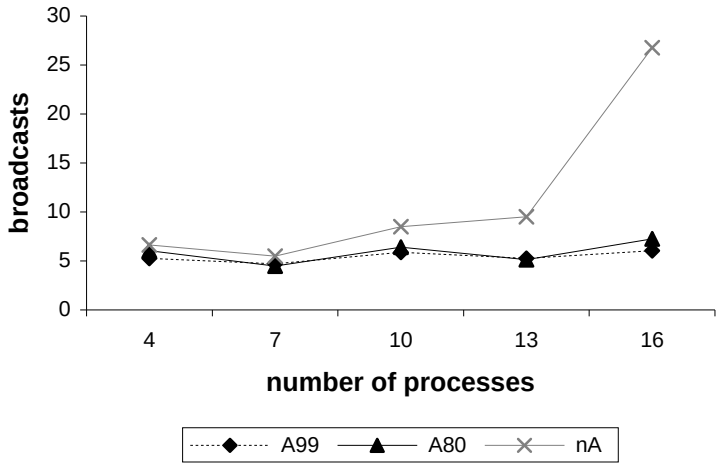
(a) No losses.



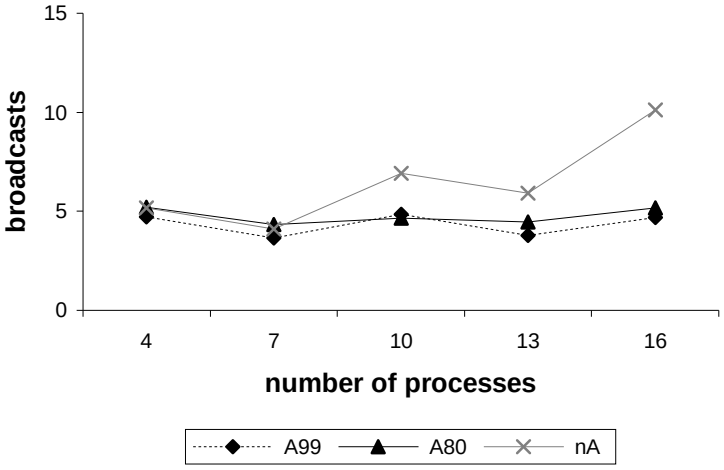
(b) Loss rate = 10%.

Figure 2: Average timeout.

loss counts to *Adaptare* as a sample with the value of the assumed timeout. This means that every loss in the A80 case adds a small timeout value, while in the A99 case it adds a large value. The consequence is that the statistical mechanisms will compute an even higher timeout in the A99 case, thus leading to the observed results.



(a) No losses.

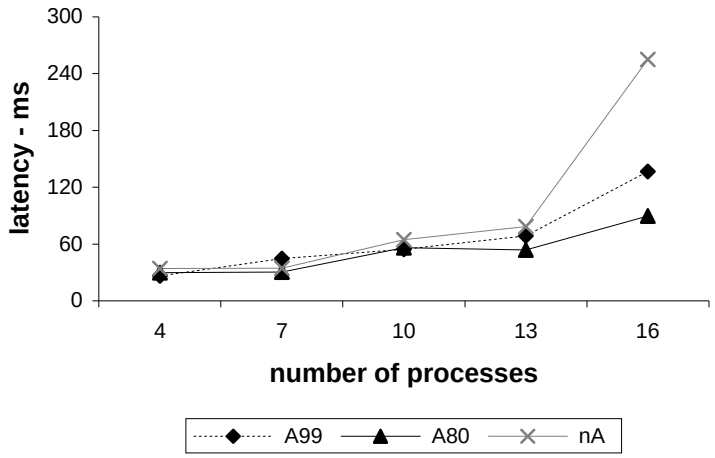


(b) Loss rate = 10%.

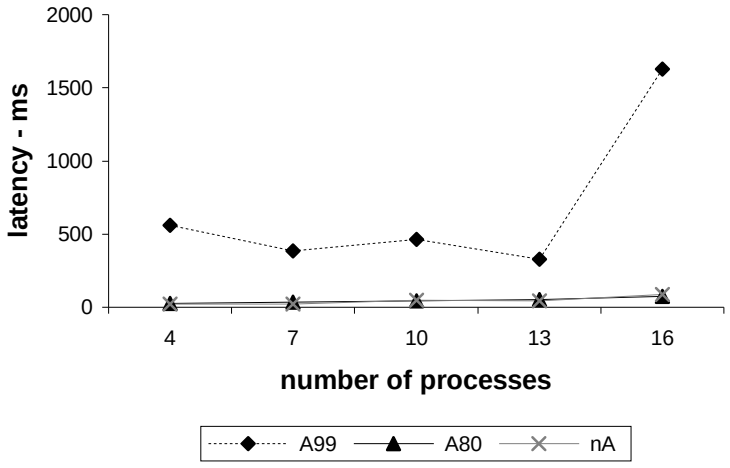
Figure 3: Average number of broadcasts.

Analyzing the number of broadcast messages, shown in Figure 3, the relation between timeouts and retransmissions is clear. Algorithms with higher timeouts have more chances of receiving sufficient messages before timeout expiration, hence less broadcast messages will be sent.

However, it is important to note that although the `A80` adaptive version pre-



(a) No losses.



(b) Loss rate = 10%.

Figure 4: Average latency.

sented average timeouts very close to the static timeout algorithm, it required the transmission of a lower number of broadcasts, at least half of those transmitted by the non adaptive algorithm, being the improvement more visible as the number of processes increases. In fact, in the `A80` version the number of broadcasts is almost as low as the number of broadcasts in the `A99` version. This is a clear indication

of the benefits that may be achieved by adapting the timeout. When the number of processes increases, creating more contention and higher transmission delays in the wireless network, the timeouts produced by *Adaptare* in the `A80` adaptive algorithm are also slightly increased. Remarkably, this small increase is sufficiently relevant to ensure the reception of the required messages within the defined timeout, avoiding the retransmissions that we observe in the non adaptive version of the consensus algorithm.

Finally, the average latency is presented in Figure 4. This is the most important metric, as the execution time of an algorithm is the ultimate performance indicator for end users.

In failure-free environments it is in principle useless to retransmit messages because all messages will be eventually received and therefore retransmissions will not speed up the protocol. In fact, retransmitting messages will have a negative impact in the sense that more network bandwidth will be used. This becomes even worse with an increasing number of processes. Therefore, higher timeouts will be better, because they will prevent retransmissions. This is perfectly visible in Figure 4(a).

In the same figure it is also visible that the `A80` adaptive algorithm performs better than the `A99` version, despite the similar number of broadcasts observed in the two algorithms. This is due to the influence of residual message losses, still observed in the baseline Emulab network, particularly with an increased number of processes. These losses may imply retransmissions, which will be done only after the defined timeout, which is much higher in the `A99` version. The will lead to an increased average latency.

When the number of losses is higher, the impact of large timeouts will be amplified, leading to a significant degradation of the average latency. The effect is clearly observable in Figure 4(b), with the `A99` algorithm presenting poor latency results. Once more, the results obtained with the `A80` adaptive algorithm are very good.

The timeout adaptive consensus with a required coverage of 80% is clearly the best one with respect to latency: it presented the best latency results in the failure-free environment, and latency values similar to the non adaptive algorithm in the network with losses.

This set of experiments emphasizes the importance of dynamically adjusting time-related variables of distributed algorithms according to observed changes in the operating conditions. The small timeout adjustments that took place in the `A80` adaptive algorithm lead to an improvement of the network load up to 80% (with 16 processes in the no loss environment), and of the latency up to 65% in the normal case of few communication failures, without compromising the execution time in the presence of losses.

5 Conclusions

In this paper, we addressed the problem of adapting timing variables and timeouts used in algorithms to the actual conditions of the environment. This problem is sometimes disregarded, but it is practically relevant to improve the overall performance of distributed algorithms, as we show in our evaluation results.

The problem is particularly relevant in dynamic environments, specially when communication latencies can be affected by varying the number of processes and network load. Therefore, the paper proposes an adaptive version of a consensus protocol that was designed for wireless networks, and provides several results that show the improvements that may be achieved in comparison with a non-adaptive version of the same protocol.

A fundamental building block of the proposed adaptive consensus protocol is the adaptation support framework, which continuously feeds the protocol with the timeout values that must be used at a given moment. We use *Adaptare*, which is a generic framework that is easily configured and may be used as an independent building block, in the context of different applications.

We believe that our contribution is important both from a practical perspective, as we show how to improve the performance of a consensus protocol in particular, and of distributed timeout-based protocols in general, and from an engineering and architectural perspective, as we propose a solution that may be used as a pattern for structuring timeout adaptive applications.

6 Acknowledgments

This work was partially supported by the FCT, through the Multiannual and CMU-Portugal programmes.

References

[1] N. Balakrishnan and A.P. Basu. *The exponential distribution: Theory, methods and applications*. CRC Press, 1995.

[2] Ali C. Begen and Yucel Altunbasak. An adaptive media-aware retransmission timeout estimation method for low-delay packet video. *IEEE Transactions on Multimedia*, 9(2):332–347, 2007.

[3] Marin Bertier, Olivier Marin, and Pierre Sens. Implementation and performance evaluation of an adaptable failure detector. In *In 2002 International Conference on Dependable Systems and Networks*, pages 354–363, Washington, DC, USA, 2002.

[4] A. Casimiro, P. Lollini, M. Dixit, A. Bondavalli, and P. Verissimo. A framework for dependable qos adaptation in probabilistic environments. In *23rd*

ACM Symposium on Applied Computing, Dependable and Adaptive Distributed Systems Track, pages 2192–2196, Ceara, Brazil, 2008.

[5] Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *J. ACM*, 43(2):225–267, 1996.

[6] W. Chen, S. Toueg, and M. K. Aguilera. On the quality of service of failure detectors. *IEEE Trans. on Computers*, 51(1):13–32, 2002.

[7] M. Dixit, A. Casimiro, P. Lollini, A. Bondavalli, and P. Verissimo. A probabilistic framework for automatic and dependable adaptation in dynamic environments. Tech Report TR-09-19, Dep. of Informatics, Univ. of Lisboa, 2009.

[8] Danny Dolev, Cynthia Dwork, and Larry Stockmeyer. On the minimal synchronism needed for distributed consensus. *J. ACM*, 34(1):77–97, 1987.

[9] Lorenzo Falai and Andrea Bondavalli. Experimental evaluation of the qos of failure detectors on wide area network. In *In 2005 International Conference on Dependable Systems and Networks*, pages 624–633, Washington, DC, USA, 2005.

[10] Henrique Moniz, Nuno Neves, Miguel Correia, and Paulo Verissimo. Randomization can be a healer: Consensus with dynamic omission failures. *Distributed Computing (to appear)*, 2010.

[11] Nuno F. Neves, Miguel Correia, and Paulo Verissimo. Solving vector consensus with a wormhole. *IEEE Trans. Parallel Distrib. Syst.*, 16(12):1120–1131, 2005.

[12] Raul Ceretta Nunes and Ingrid Jansch-Porto. Qos of timeout-based self-tuned failure detectors: The effects of the communication delay predictor and the safety margin. In *In 2004 International Conference on Dependable Systems and Networks*, page 753, Washington, DC, USA, 2004.

[13] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2):228–234, 1980.

[14] K J Perry and S Toueg. Distributed agreement in the presence of processor and communication faults. *IEEE Trans. Softw. Eng.*, 12(3):477–482, 1986.

[15] Ioannis Psaras, Vassilis Tsaoussidis, and Lefteris Mamatas. CA-RTO: A Contention-Adaptive Retransmission Timeout. In *ICCCN 2005, San Diego, CA, USA*, October 2005.

[16] Rouzbeh Razavi, Martin Fleury, and Mohammed Ghanbari. Fuzzy control of adaptive timeout for video streaming over a bluetooth interconnect. In *Proceedings of the 12th IEEE Symposium on Computers and Communications (ISCC 2007)*, Aveiro, Portugal, July 2007.

[17] Nicola Santoro and Peter Widmayer. Time is not a healer. In *STACS '89: Proceedings of the 6th Annual Symposium on Theoretical Aspects of Computer Science*, pages 304–313, London, UK, 1989. Springer-Verlag.

[18] Nicola Santoro and Peter Widmayer. Agreement in synchronous networks with ubiquitous faults. *Theor. Comput. Sci.*, 384(2-3):232–249, 2007.

[19] K. S. Trivedi. *Probability and Statistics with Reliability, Queuing and Computer Science Applications*. John Wiley and Sons, 2002.

[20] Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb, and Abhijeet Joglekar. An integrated experimental environment for distributed systems and networks. In *5th Symposium on Operating Systems Design and Implementation*, pages 255–270, Boston, MA, USA, 2002.

[21] George Xylomenos and Christos Tsilopoulos. Adaptive timeout policies for wireless links. In *20th International Conference on Advanced Information Networking and Applications (AINA 2006)*, pages 497–502, Vienna, Austria, April 2006.