# Self-Tuned Failure Detector

Raul Ceretta Nunes *

Universidade Federal do Rio Grande do Sul - Instituto de Informática
Av. Bento Gonçalves, 9500 - Agronomia - 91501-970 - Porto Alegre / RS - Brazil
Phone: +55(51)316-6804 - Fax: +55(51)319-1576
ceretta@inf.ufsm.br

**Abstract** The main problem on asynchronous systems is the impossibility of a process/object to distinguish a slow process/object from crashed or disconnected ones. This result comes from the indeterminism in the communication delay measurements. Despite of this, failure detector abstraction is an important building block to solve many problems, like membership and consensus protocols. Unfortunately, the indeterminism turns hard the task of tuning a timeout-based failure detector on these systems.
This work proposes a new self-tuned failure detector. We are working on an adaptive timeout mechanism that uses the inter-arrival times history concept to dynamically determine the most adequate tuning of the timeout period used by the failure detector algorithm.
**Key words:** failure detector, adaptive timeout, group communication.

## 1 Introduction

In the last few years, the interest in dependable (reliable and available) distributed applications is increasing, mainly on the Internet. Unfortunately, the design and the implementation of such applications over an unreliable asynchronous distributed system is not a trivial task. The reason is the impossibility of distinguishing a slow process/object from crashed or disconnected ones [6]. As a consequence, ensuring the correct state of a distributed application is a challenge.

An important building block that helps the resolution of the impossibility problem is the unreliable failure detector abstraction [2], specified by completeness and accuracy properties[1]. This abstraction is useful for solving fundamental problems like membership and consensus. For example, group membership services can use a failure detector module to discover which members are suspected of failure. Thus, suspecting that

* Computer Science PhD. Student at Federal University of Rio Grande do Sul.

[1] Roughly speaking, completeness requires that a failure detector eventually suspects every process that actually crashes, while accuracy restricts the mistakes that a failure detector can make.

a member is on failure results in the execution of the group membership agreement protocol. By this way, either the suspected member will be removed from the group or it will remain in the group with all suspicions removed. On the other hand, although many algorithms, as membership and consensus, are designed to tolerate some failure detector mistakes, the wrong suspicions of some correct members decrease the performance of these algorithms and must be avoided.

To ensure the algorithm termination, despite of unbounded communication delays, most of failure detector implementations use some kind of time limit (*timeout*), although it can make mistakes. An adequate timeout can reduce the side effect of the timeout based failure detector approach, i.e., the occurrence of wrong suspicions.

Two basic approaches to tune the timeout period of the failure detectors appear at literature: one carry out a tuning based on a careful analysis of the system environment (off-line setting) [12][13] and, the other, adapting the timeout according to the system behavior (at runtime) [10][9][8][3]. Our work aims at to explore the second one.

The following strategies may appear in the second approach: (A) set the timeout using the first approach and, at runtime, to increase it by a constant $k$ whenever a wrong suspicion is detected [10]; (B) set the timeout taking into account the communication delay supplied by a communication time indicator (CTI) module [9][8]; and (C) set the timeout taking into account the expected arrival time of the next heartbeat message, given by an estimator from the heartbeat probabilistic behavior (EHPB) module [3].

On strategy A, the timeout never decreases. Consequently, after a long time, the timeout generally turns continuously big and the time to detect a failure becomes greater. The B and C strategies are better then A, since it adapts (increasing or decreasing) the timeout according to the communication behavior. On strategy B, the CTI module works by analyzing, all the time, the current operating system and the network load, and on strategy C the EHPB module works by computing the next expected arrival time from last $n$ arrival times of the heartbeats. Considering the system behavior as an ergodic process[2], strategy B works well in relative stable systems.

[2] By definition ([11] section 9.2.2), a stochastic process $x(t)$ is said to be an ergodic process if the time average of a single record is approximately equal to the ensemble average.

This paper presents a first draft of a self-tuned failure detector design based on an adaptive timeout mechanism. This approach allows us to increase/decrease the timeout period as needed. The strategy used in this work sets the next timeout to the expected inter-arrival time by taking into account an ergodic behavior. It differs from strategy C by working with the inter-arrival time between two consecutive heartbeat messages instead of using arrival times. Besides, to predict the next timeout, the estimator of the next inter-arrival time is computed by the adaptable timeout mechanism that uses the second-order moving average prediction method ([7] section 3.3).

The paper is organized as follows: the adaptive timeout mechanism is described on section 2; the basic self-tuned failure detector algorithm is presented on section 3; and finally some partial conclusions are presented on section 4.

## 2 The Adaptable Timeout Mechanism

Consider two hosts in the asynchronous network and a process/object associated to each of them. If the two processes exchange heartbeat messages, whenever a heartbeat message ($hbMsg$) is received by one of them, the following steps are done by the adaptable timeout mechanism (see the algorithm on figure 1): (a) the difference between the arrival reception time of the last two messages is stored in the last position of the sample vector (line 2) and the last message arrival time is updated (line 3); (b) the sample mean (simple moving average) is computed and stored in the last position of the averages vector (line 4); (c) the double moving average is computed (line 5); and (d) the next inter-arrival time estimator ($\widehat{ET}$) is set to $2.Mean[oldest] - \widehat{\omega}$ (line 6).

Note that the new timeout (based on the next inter-arrival time estimator) can increase or decrease according to moving mean variation. Of course, new adjustments may be required to accommodate extra time differences and minimize the number of the wrong suspicions of the failure detector. These adjustments can be included by additional parameters in $\widehat{ET}$ expression (line 6). At the present moment, the probabilistic analysis and the search for additional parameters are objects of study.

1 **Upon** $receive(hbMsg)$ **do**:
2 $Sample[oldest] \leftarrow hbMsgRecTime - hbMsgLastRecTime$;
3 $hbMsgLastRecTime \leftarrow hbMsgRecTime$;
4 $Average[oldest] \leftarrow \frac{1}{n} \sum_{i=1}^{n} Sample[i]$;
5 $\widehat{\omega} \leftarrow \frac{1}{m} \sum_{i=1}^{m} Average[i]$;
6 $\widehat{ET} \leftarrow 2.Average[oldest] - \widehat{\omega}$;

**Figure 1:** Adaptable timeout mechanism.

The statistic theory shows ([1] section 7.1) that if we consider a large random sample (greater than 30), any probability distribution associated to inter-arrival times (random variable computed by algorithm) can be approximated to a normal distribution. Thus, the simple mean used here, based on parameter $n$ (line 4 in figure 1), is an unbiased estimator to the expected mean. A similar approach can be used to set the $m$ parameter (line 5 in figure 1), where $1 \leq m \leq n$. To achieve the best timeout estimator, the $m$ parameter must be set to a value that minimizes the moving mean variation.

## 3 Self-tuned Failure Detector

According to Felber and Guerraoui [5], the failure detectors can be represented by first-class objects. There are two basic strategies for implementing them: push (based on *heartbeat* messages) and pull (based on *are you alive?* messages). Moreover, these two strategies can also be combined to obtain new failure detector strategies. However, independent from the strategy, if we are considering the Internet environment, to set some timeout according to the network topology and the communication pattern is a difficult task because these parameters are continuously changing. In our work, we have chosen the push strategy to show how an adaptable timeout mechanism can be used, but other failure detector strategies could have been used.

The push failure detector strategy [5] is based on two temporal references: the heartbeat interval ($t_h$) and the timeout to receive a message ($timeout$). The basic algorithm is: (a) at every $t_h$ time units, each object $q$ broadcasts a $hbMsg$ message to every other monitored object; (b) if other object $p$ has received an $hbMsg$ message from $q$ by a $timeout$, then $p$ re-starts the $timeout$ associated to $q$; (c) if other object $p$ has not

received an $hbMsg$ message from $q$ for some $timeout$, then $p$ adds $q$ to its suspect list ($SuspectList_p$). Note that the basic algorithm neither specifies if an object can recover after being added in the SuspectList nor specifies if the timeout is variable.

By considering that links may delay or loose messages, we have adapted the basic push specification for our environment. From basic push algorithm, our failure detector adds the following rule: if an object $p$ receives a $hbMsg$ message from another object $q$ that it already suspects, $p$ removes $q$ from the suspect list and set its timeout period to the next inter-arrival time estimated by the adaptable timeout mechanism. The new self-tuned failure detector algorithm is shown in figure 2.

Note that whenever an object $p$ receives a $hbMsg$ message from $q$, first of all it verifies if $q$ is either in its suspect list or not. Note also that $q$ broadcasts $hbMsg$ messages until it really crashes. This property ensures its future recovery by the failure detector.

```
   For every member p:
   (Task 1)
1     do forever
2        for each t_h:
3           broadcast(hbMsg, G);
   (Task 2)
1     upon receive(hbMsg, q) do
2        if q ∈ SuspectList_p than
3           SuspectList_p ← SuspectList_p − q;
4        timeout_q ← ÊT;
5     upon timeout do
6        SuspectList_p ← SuspectList_p + q;
```

**Figure 2:** Self-tuned failure detector.

## 4 Conclusion

The solution proposed in this paper works by dynamically tuning the timeout period of the failure detector. Thus, it is better applied when the communication time is not previously bounded and also presents variations along the time, as occurs on Internet environment.

Although the solution does not avoid neither wrong suspicions nor the possibility of unstable behavior (successive suspicions), its probabilistic nature allow us to reduce the probability of the wrong suspicions in most of the time, increasing the accuracy of the failure detector.

## References


1. Allen, Arnold O. Probability, Statistics, and Queueing Theory with Computer Science Applications. [S.l.]: Academic Press, 1990.
2. Chandra, T.D.; Toueg, Sam. Unreliable Failure Detectors for Reliable Distributed Systems. Journal of the ACM, New York, v.43, n.2, p.225-267, Mar. 1996.
3. Chen, Wei. On the Quality of Service of Failure Detectors. Ithaca, EUA: Department of Computer Science, Cornell University, 2000. Ph.D. Thesis. 140p.
4. Dolev, Denny et al. Failure Detectors in Omission Environments. In: ACM SYMPOSIUM ON PRINCIPLES OF DISTRIBUTED COMPUTING (PODC), 16., 1997. Santa Barbara, EUA: ACM Press, 1997.
5. Felber, Pascal; Fayad, Mahamed E.; Guerraoui, Rachid. Putting OO Distributed Programming to Work. Communications of the ACM, New York, v.42, n.11, p.97-101, Nov. 1999.
6. Fischer, M.; Lynch, N.; Paterson, M. Impossibility of Distributed Consensus with One Faulty Process. Journal of the ACM, New York, v.32, p.374-382, Apr. 1985.
7. Hamilton, James D. Time Series Analysis. New Jersey: Princeton University Press, 1994.
8. Macêdo, Raimundo. Failure Detection in Asynchronous Distributed Systems. In: WORKSHOP ON FAULT TOLERANCE, 2., 2000. Proceedings ... Curitiba, Brazil: PUC/PR, 2000. p.76-81
9. Macêdo, Raimundo. Implementing Failure Detection through the use of a Self-Tuned Time Connectivity Indicator. Salvador: Laboratório de Sistemas Distribuídos - LaSiD, Brazil, 1998. (Technical Report RT008/98).
10. Montresor, Alberto. System Support for Programming Object-Oriented Dependable Applications in Partitionable Systems. Bologna, Italy: Department of Computer Science, University of Bologna, 2000. Ph.D. Thesis. 220p.
11. OCHI, Michel K. Applied Probability and Stochastic Processes in Engineering and Physical Sciences. New York: John Wiley & Sons, 1990.
12. Raynal, Michel and Tronel, Frédéric. Group Membership Failure Detecton: a Simple Protocol and its Probabilistic Analysis. Distributed Systems Engineering Journal, v.6, n.3, p.95-102, 1999.
13. Renesse, Robbert van; Minsky, Yaron; Hayden, Mark. A Gossip-Style Failure Detection Service. In: MIDDLEWARE, 1998. Proceedings ... Sep., 1998.