Submission to DSN-2002 – Main Track: Dependable Computing and Communications

Implementation and performance evaluation of an adaptable failure detector

Marin BERTIER[2]
marin.bertier@lip6.fr

Olivier MARIN[1,2]
Olivier.Marin@univ-lehavre.fr

Pierre SENS[2]
pierre.sens@lip6.fr

[1]Laboratoire d'Informatique du Havre
University of Le Havre
25 rue Philippe Lebon
BP540 76058 Le Havre cedex

[2]Laboratoire d'Informatique de Paris 6
University Paris 6 - CNRS
4 place Jussieu
75252 Paris Cedex 05, France

Abstract

Chandra and Toueg introduced the concept of unreliable failure detectors. They showed how, by adding these detectors to an asynchronous system, it is possible to solve the Consensus problem. In this paper, we propose a new implementation of an Eventually Perfect failure detector ($\Diamond P$). This implementation is a variant of heartbeat failure detector which is adaptable and can support scalable applications. In this implementation we dissociate two aspects: a basic estimation of the expected arrival date to provide a short detection time, and an adaptation of the quality of service according to application needs. The latter is based on two principles: an adaptation layer and a heuristic to adapt the sending period of "I am alive" messages. This model allows us to use the failure detector as a shared service between several applications.

Keywords: failure detector, adaptation, heartbeat, performance, implementation
Submission category: insert category
Word count: 6000
Declaration: Material must be cleared through author affiliations.
Contact author:

Marin BERTIER
Laboratoire d'Informatique de Paris 6
4 place Jussieu
75252 Paris Cedex 05, France
marin.bertier@lip6.fr

CARTER Award: YES/NO (for Regular Papers only)

Nominated Student: Name, Firstname.
Advisor(s): Title, Name, Firstname.

1 Introduction

Failure detectors are well-known as a basic building block for fault-tolerant distributed systems. Chandra and Toueg introduce in [4] the concept of *unreliable failure detector*. They show how, by adding these detectors to an asynchronous system, it is possible to solve the Consensus problem. The Consensus is the "greatest common denominator" of agreement problems such as atomic broadcast or atomic commit. Fisher, Lynch, and Paterson [8] have shown that consensus cannot be solved deterministically in an asynchronous system that is subject to even a single crash failure. This impossibility results from the inherent difficulty of determining whether a process has actually crashed or is only "very slow".

Failure detectors can be seen as one oracle per process. An oracle provides a list of processes that it currently suspects to have crashed. Many fault-tolerant algorithms have been proposed [10, 7, 3] based on unreliable failure detectors, but there are few papers about implementing of these detectors [12]. In this paper we investigate how to implement and dynamically adapt failure detectors.

We propose a new implementation of an *Eventually Perfect* ($\Diamond P$) failure detector in models of partial synchrony [6]. This implementation is a variant of the heartbeat detector which is adaptable and can support scalable applications. Our algorithm is based on all-to-all communications where each process periodically sends an "*I am alive*" message to all processes using IP-Multicast capabilities. To provide a short detection delay, we automatically adapt the failure detection time as a function of previous receptions of "*I am alive*" messages.

Failure detectors are designed to be used over long periods where the need for *quality of detection* alters according to applications and systems evaluation. In practice, it is well known that systems are subjected to variations between long periods of instability and stability. We evaluate the maximal quality of service that the network can support in terms of detection time. Then we propose a heuristic to adapt the sending period of "*I am alive*" messages as a function of the network QoS and the application requirements.

The rest of the paper is organized as follows. In Section 2, we briefly describe the Chandra-Toueg failure detectors. In Section 3, we present the system model and we discuss different approaches in order to implement failure detectors. Section 4 describes the way we adapt the detection delays dynamically. In Section 5, we present the algorithm of our failure detectors, and we prove its correctness. Section 6 presents a performance evaluation of the failure detector and compares our results with some related studies. In Section 8, we describe a real application we develop on top of our failure detector. Finally, Section 9 concludes the paper.

2 Unreliable failure detectors

In this section, we present a short description of failure detectors and some metrics to compare their performances. Each process has access to a local failure detector which maintains a list of processes that it currently suspects to have crashed. Since a failure detector is unreliable, it can make mistakes by adding in its list a process which is still running. But if the detector later believes that suspecting this process is a mistake, it then removes the process from its list. Thus, a detector can repeatedly add and remove a same process from its list of suspect processes. Failure detectors are characterized by two properties: completeness and accuracy. Completeness characterizes the failure detector capability of suspecting every incorrect process permanently. Accuracy characterizes the failure detector capability of not suspecting correct processes. Two kinds of completeness and four kinds of accuracy are defined in [4], which once combined yield eight classes of failure detectors.

In this paper, we focus on the $\Diamond P$ detector, named *Eventually Perfect*. This detector requires these characteristics:

- **Strong completeness**: there is a time after which every process that crashes is permanently suspected by every correct process.

- **Eventual strong accuracy**: there is a time after which correct processes are not suspected by any correct process.

The above properties must be satisfied by our detector. In parallel, [5] proposes a set of metrics that can be used to specify the Quality of Service (QoS) of a failure detector. The QoS quantifies how fast a detector suspects a failure and how well it avoids false detection.

- *Detection time* (T_D): T_D is the time that elapses from p's crash to the time when q starts suspecting p permanently.

  The next metrics are used to specify the accuracy of a failure detector.

- *Mistake recurrence time* (T_{MR}): this measures the time between two consecutive mistakes.

- *Mistake duration* (T_M): this measures the time it takes the failure detector to correct a mistake.

3 Failure detection strategies

In this section, we present our target system model. We describe two classical implementations of failure detector, and then show why we have chosen the heartbeat failure detector model.

3.1 System model

We consider a distributed system consisting of a finite set of n processes $\Pi = \{p_1, p_2, \ldots, p_n\}$ that are spread throughout a local area network (LAN). These processes communicate only by sending and receiving messages. Every pair of processes is assumed to be connected by means of a reliable communication channel. We suppose that all the components in this system support IP-Multicast communication.

Processes can fail by crashing only, and this crash is permanent. Our algorithm does not need synchronized clocks, but local clocks must not drift with respect to real time. They must measure time intervals accurately.

We consider the model of partial synchrony proposed by Chandra and Toueg in [4]. This model stipulates that, for every execution, there are bounds on process speeds and on message transmission times. However, these bounds are not known and they hold only after some unknown time (called *GST* for *Global Stabilization Time*). We denote by Δ_{msg} the maximum time, after *GST*, between the sending of a message and the delivery and processing by its destination process, assuming that the destination process has not failed.

3.2 The Heartbeat strategy

This implementation is the most popular strategy for implementing failure detectors. Every process q periodically sends an "*I am alive*" message to the processes in charge of detecting its failure. If a process p does not receive such a message from q after the expiration of a timeout on q, it adds q to its list of suspected processes. If p later receives an "*I am alive*" message from q, p then removes q from its list of suspected processes.

This implementation is defined by two parameters (figure 1):

- **the heartbeat period** Δ_i: Δ_i is the time between two emissions of an "*I am alive*" message.
- **the timeout delay** Δ_{to}: Δ_{to} is the time between the last reception of an "*I am alive*" message from q and the time where p starts suspecting q, until an "*I am alive*" message from q is received.

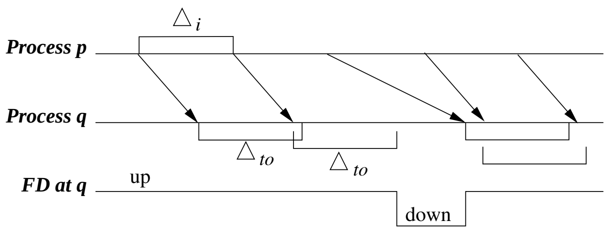

Figure 1: Failure detection _ the heartbeat strategy

An amelioration of this classic heartbeat implementation is proposed in [5]. In this new algorithm, in order to determine whether to suspect the process q, the process p uses a sequence $\tau_1, \tau_2, \ldots$ of fixed time points, called *freshness points*. The freshness point τ_i is an estimation of the arrival date of the i^{th} heartbeat message from q.

The advantage of this approach is that the detection time is independent from the last heartbeat. This modification increases the accuracy because it avoids premature timeout, and outperforms the failure detection time. If q sends a heartbeat message just before crashing and if the transmission delay is d, then the detection delay is: $d + \Delta_{to}$.

3.3 The Pinging strategy

A process p monitors a process q by sending "*Are you alive ?*" messages to q periodically. Upon reception of such messages, the monitored process replies with an "*I am alive*" message. If process p times out on process q, it adds q to its list of suspected processes. If p later receives an "*I am alive*" message from q, p then removes q from its list of suspected processes.

This implementation is also defined by two parameters (see figure 2):

- **the interrogation period** Δ_i: Δ_i is the time between two emissions of an "*Are you alive ?*" message.
- **the timeout delay** Δ_{to}: Δ_{to} is the time between the emission of an "*Are you alive ?*" message by p to q, and the time where p starts suspecting q, until p receives an "*I am alive*" message from q.

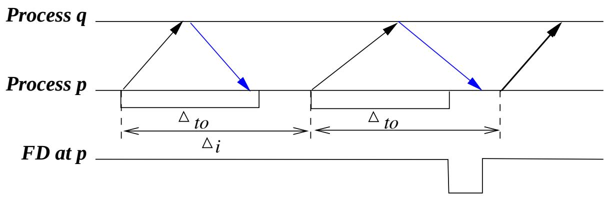

Figure 2: Failure detection _ the pinging strategy

3.4 Strategies comparison

Heartbeat failure detectors have many advantages over pinging failure detectors. The first advantage is that a heartbeat failure detector sends half as many messages as a pinging detector for the same detection quality.

The second advantage is the quality of the estimation for the timeout delay Δ_{to}. The heartbeat detector estimates the transmission delay of "*I am alive*" messages, whereas the pinging detector must estimate the transmission delay of "*Are you alive ?*" messages, the reaction delay, and the transmission delay of *"I am alive"* messages. Therefore it is easy to make a better estimation with a heartbeat message than by pinging another detector.

The pinging technique is often used in membership such as DCOM from Microsoft [11] where, in the absence of response to n successive "*Are you alive ?*" messages, the process is regarded as having failed. This approach is different from that of unreliable failure detectors, because here the detection of a process failure is definite, whereas unreliable failure detectors only suspect crashes: detected processes may be freed from suspicion later on.

4 Adaptation of the delays

4.1 Arrival date estimation

We believe that the heartbeat strategy is obviously better in order to implement failure detectors. In this section, we study how to estimate the arrival time of a heartbeat message. The QoS of the detection depends on the Δ_i and Δ_{to} parameters.

The timeout delay Δ_{to} is important because it determines the detection time. The estimation for Δ_{to} uses local information that each host possesses. This information is limited to the observation of heartbeat message arrival dates and the interrogation period Δ_i. On the other hand the arrival time of heartbeat messages can be altered by the network load and the host load.

In our solution the failure detector is structured into two layers. The first layer makes an accurate estimation to optimize the detection time. The second layer can modulate this detection time with respect to the needs in terms of QoS. In this part, we show how to estimate the arrival time as accurately as possible without increasing the number of false detections. For this purpose we compare three methods: the first one is proposed in [5]. The second one is inspired by Jacobson's algorithm [9], which is used to calculate the Round Trip Time (RTT) in the TCP protocol. The third method is the one we propose: it combines both the above mentioned techniques.

4.1.1 Chen's estimation In [5], the authors propose several implementations relying on clock synchronization and a probabilistic behaviour of the system. The estimation presented here is carried out for unsynchronized clocks.

This technique estimates the arrival time for heartbeat messages (EA) and adds a constant safety margin. The expected arrival date is calculated as shown below, and the safety margin is determined by a preliminary calculation with respect to QoS requirements.

Each process q considers the n most recent heartbeat messages, denoted $m_1, m_2, \ldots, m_n$. Let $A_1, A_2, \ldots, A_n$ be their receipt times according to q's local clock. When at least n messages have been received, $EA_{(k+1)}$ can be estimated by:

$$EA_{(k+1)} \approx \frac{1}{n}\left(\sum_{i=k-n}^{k} A_i - \Delta_i.i\right) + (k+1).\Delta_i$$

The next timeout delay Δ_{to} (which expires at the next freshness point $\tau_{(k+1)}$) is composed of EA and α the constant safety margin. EA represents the theoretical arrival date. The safety margin is added to avoid false detections caused by transmission delay or processor overload.

$$\tau_{(k+1)} = \alpha_{(k+1)} + EA_{(k+1)}$$

This technique provides a good estimation for the next arrival time. However, it uses a constant safety margin because the authors estimate that the model presents a probabilistic behaviour.

4.1.2 Jacobson's estimation

On the contrary to the first estimation, in which we suppose that we can determine a constant safety margin $\alpha_{(k+1)}$ beforehand, Jacobson's estimation assumes less knowledge about the system model. The original algorithm is used in TCP to estimate the delay after which the transceiver retransmits its last message.

This estimation supposes that the behaviour of the system is not constant. It adapts the safety margin each time it receives a message. The adaptation of the margin α uses the $error$ in the last estimation. Parameter γ represents the importance of the new measure with respect to the previous ones. $delay$ represents the theoretical margin, and var estimates the magnitude between errors. The original algorithm is:

$$\begin{aligned}
error_{(k)} &= A_k - EA_{(k)} - delay_{(k)} \\
delay_{(k+1)} &= delay_{(k)} + \gamma.error_{(k)} \\
var_{(k+1)} &= var_{(k)} + \gamma.(|error_{(k)}| - var_{(k)}) \\
\alpha_{(k+1)} &= \beta.delay_{(k+1)} + \phi.var_{(k+1)} \\
\text{And}\quad \tau_{(k+1)} &= \tau_{(k)} + \Delta_i + \alpha_{(k+1)}
\end{aligned}$$

This method constantly adapts the margin with respect to the network state. However, it can take a long time to converge and the estimation time is less accurate.

4.1.3 Our estimation

Our method is a combination of the two previous ones. We estimate the arrival time with the first method and we evaluate the safety margin dynamically with Jacobson's algorithm.

Chen's estimation supposes that we stock all the information about the n last received messages, but you can transform it into a recursive equation.

Until process q receives at least n heartbeat messages from process p, q estimates the next arrival time by:

$$\begin{aligned}
U_{(k+1)} &= \tfrac{A_k}{k+1}.\tfrac{k.U_{(k)}}{k+1} \\
EA_{(k+1)} &= U_{(k+1)} + \tfrac{k+1}{2}.\Delta_i \\
\text{with}\quad U_{(1)} &= A_0
\end{aligned}$$

When process q has received more than n heartbeat messages, it uses:

$$EA_{(k+1)} = EA_{(k)} + \frac{1}{k}(A_k - A_{(k-n-1)})$$

The safety margin $\alpha_{(k+1)}$ is calculated similarly to Jacobson's estimation. The next timeout $\Delta_{to_{(k+1)}}$, activated by q when it receives m_k, expires at the next freshness point:

$$\tau_{(k+1)} = EA_{(k+1)} + \alpha_{(k+1)}$$

The three methods are compared with respect to the performances in section 6

4.2 Dynamic adaptation of the interrogation delay

Failure detectors are designed to be used over long periods of time. The needs in terms of QoS are not constant, they vary according to each application. To adapt the QoS, we can change the interrogation delay Δ_i. The other reason for adapting the QoS of the detector is to adapt the bandwidth required by detectors with respect to the network load.

The idea is to allow the adaptation of Δ_i during the execution. In order to achieve this, all the detectors must reach a consensus over the new Δ_i, because if one detector changes its delay, all the others must change theirs accordingly.

The reasons to make this change are: the deliberate increase or decrease of the quality of detection, situations where the network capacity cannot allow to maintain the current quality of detection anymore, or where the network capacity increases and allows to obtain a higher quality of detection.

When a detector reaches one of the above situations, it starts a consensus. The expression for the QoS used here is the heartbeat emission frequency.

Each detector has a list of its client applications with an estimation of their respective qualities of detection. According to this list each detector can evaluate a *quality margin* that it may propose for the consensus. The quality margin is composed of two values: namely the ability quality, and the required quality.

The ability quality is obtained by means of a heuristic, which evaluates the maximum quality of service that the network can support. The required quality is obtained with respect to the maximum quality of detection required by the applications.

At the end of the consensus a margin is decided. The new heartbeat sending interval Δ_i is the minimum of the two quality of service values. We use the consensus algorithm introduced in [4] for $\diamondsuit S$ detectors.

5 Failure Detector Algorithm

5.1 Algorithm

We now present our algorithm for a failure detector of class $\diamondsuit P$. Our implementation is composed of two layers. In the first layer, we implement a basic failure detection service, which provides an estimation for the arrival date of the next heartbeat message optimized for detection time. This estimation is obtained from the expected arrival date and a dynamic margin (see section 4.2). The aim of this layer is not to avoid all false detections but to provide a compromise between the number of false detections and the accuracy of the detection time. The second layer adapts the detection service provided by the first layer with respect to the application needs.

Figure 3 shows the whole algorithm of our failure detector. The second layer presented here (*Task 3*) provides an eventually perfect failure detector $\diamondsuit P$.

The failure detector works as follows. In *Task 1* each process p periodically sends an "*I am alive*" message

Every process $p \in \Pi$ *executes :*

Initialization:

$suspect_p \leftarrow \emptyset$
for all $q \in \Pi - \{p\}$
  $\Delta_p(q) = 0$ { $\Delta_p(q)$ belongs to the second layer and allows to moderate the detection}
  $\tau_0(q) = 0$ { Initially, all process q will be suspected by process p }
  $EA_{(0)}(q) = U_0(q) = 0$
  $delay_0(q) =$ initial value
  $\alpha_0(q) = var_0(q) = error_0(q) = 0$
  $A_{(0)}(q) = 0$
  $k(q) = -1$ { $k(q)$ keeps the largest sequence number in all the messages p received from q so far }

Task 1:

at time $i.\eta$, sends heartbeat m_i to $\Pi - \{p\}$ {η is the detection interval}

Task 2:

upon receive message m_j at time t from q :
  if $j > k(q)$ **then** { received a message with a higher sequence number }
    $k(q) \leftarrow j$

    $error_k(q) \leftarrow t - EA_{(k)}(q) - \alpha_k(q)$
    $delay_{(k+1)}(q) \leftarrow delay_k(q) + \gamma.error_k(q)$
    $var_{(k+1)}(q) \leftarrow var_k(q) + \gamma.(|error_k| - var_k(q))$
    $\alpha_{(k+1)}(q) \leftarrow \beta.delay_{(k+1)}(q) + \phi.var_{(k+1)}(q)$

    if $j < n$ **then**
      $U_{(k+1)} = \frac{t}{k+1} * \frac{k}{k+1} U_k$
      $EA_{(k+1)} = U_{(k+1)} + \frac{k+1}{2}.\eta$
    else
      $EA_{(k+1)} = EA_{(k)} + \frac{1}{k}.(t - A_{(k-n-1)}(q))$
    endif

    $A_{(k)}(q) \leftarrow t$ { A(q) is an array which contains the n last message arrival dates from q}
    $\tau_{(k+1)}(q) \leftarrow \tau_k(q) + EA_{(k+1)}(q) + \alpha_{(k+1)}(q)$ { set the next freshness point $\tau_{\ell+1}(q)$ }
    if $q \in suspect_p$ **then**
      $suspect_p \leftarrow suspect_p - \{q\}$ { trust q since $m_k(q)$ is still fresh at time t }
      $\Delta_p(q) \leftarrow \Delta_p(q) + 1$ { increase the timeout period }
    endif

Task 3:

upon $\tau_{k+1}(q) =$ the current time :
{ if the current time reaches τ_{k+1}, then none of the messages received is still fresh }
  wait during $\Delta_p(q)$ and if no message receive from q { detection moderation }
    $suspect_p \leftarrow suspect_p \cup \{q\}$
{ suspect q since no received message is still fresh at this time }

Figure 3: Implementation of $FD \in \Diamond P$

to all processes.

Task 2 performed every time a process p receives an "*I am alive*" message from a process q. It is the first layer of our detector: process p estimates the expected arrival date $EA(q)$ and the safety margin α for the next "*I am alive*" message from q. From these results, process p determines the next freshness point $\tau(q)$ for process q. If process p currently suspects q, then it stops suspecting it and increases its moderator timeout $\Delta_p(q)$ because p knows that its previous timeout on q was premature.

Task 3 starts when process p does not receive an "*I am alive*" message from q before the next freshness point $\tau(q)$. Process p waits again for a message from q during the moderator timeout $\Delta_p(q)$. If after this delay, it still does not receive a message from q, it starts suspecting it.

5.2 Proof

We show here that the algorithm of figure 3 implements a failure detector of class $\Diamond P$, on condition that the system is in accordance with the system model defined in section 3.1.

Consider a partially synchronous system S. For every run of S there is a Global Stabilisation Time (GST) after which some bounds on relative process speeds and message transmission times hold. The values of GST and these bounds are not known.

A failure detector of class $\Diamond P$, must verify the two properties represented by the two Theorems 1 and 2.

Theorem 1 *Strong completeness. Eventually every process q that crashes is permanently suspected by every correct process.*

$$\exists\, t_0 : \forall t \geq t_0, \forall p \in correct(t),\ \forall q \in crashed,\ q \in suspect_p(t)$$

Theorem 2 *Eventual strong accuracy. There is a time after which the correct processes are not suspected by any correct process.*

$$\exists\, t_{bound}, \forall t \geq t_{bound}, \forall p, q \in correct(t), q \notin suspect_p(t)$$

Strong Completeness Theorem 1 is verified if Lemma 1 and 2 are verified. That is if there is a time t_{mute} after which no correct process p receives heartbeat messages from the crashed process q, and if there is a time $t_{timeout_k}$ after which all correct processes p permanently suspect q.

Lemma 1 *If process q crashes at t_{crash}, then there is a time t_{mute} after which process p stops receiving messages from q.*

$$t_{mute} \leq t_{crash} + \Delta_{msg}$$

Proof: All the time instants considered in the rest of this section are assumed to be after GST. We also assume that, at these instants, all the messages sent before GST have already been delivered and processed.

These assumptions allow us to consider in the rest of the section, that the unknown bounds on process speeds and on message transmission times hold.

$$\exists\, t_{GST} : \forall m_k \mid t_{s_k} \geq t_{GST} : (t_{r_k} - t_{s_k}) < \Delta_{msg}$$
$$t_{s_k} \text{ is the time when } q \text{ sends } m_k \text{ and } t_{r_k} \text{ is the time when } p \text{ receives } m_k \tag{1}$$

Suppose a process q crashed at t_{crash}. Then q stops sending "*I am alive*" messages.

$$\nexists\, m_k \mid t_{s_k} \geq t_{crash} \tag{2}$$

The process p cannot receive message k from process q after $t_{r_k} + \Delta_{msg}$. Hence process p cannot receive any message from process q after $t_{crash} + \Delta_{msg}$. □

Lemma 2 *For any sequence of k messages received by process p from q, there is a time τ_k after which process p starts suspecting process q if it does not receive any message from q.*

Proof: From *Task 2*, when the process p receive a message $m_{(k-1)}$ from process q, it calculates a new τ_k after which it starts suspecting process q. We must prove that the τ_k is always bounded. The τ_k is calculated as follows (Task 2 and 3):

$$\tau_k = (EA_k + \alpha) + \Delta_p(q)$$

In Task 2, if $k > n$, EA_k is calculated as follows:

$$\begin{aligned} EA_k &= \frac{1}{n}\left(\sum_{i=k-n}^{k} t_{r_i} - \eta i\right) + k.\eta \\ \text{from our model} \quad EA_k &< \frac{1}{n}\left(\sum_{i=k-n}^{k} (t_{s_i} + \Delta_{msg}) - \eta i\right) + k.\eta \\ \text{as } t_{s_i} = t_{s_{(i-1)}} + \eta \text{ then} \quad EA_k &< t_{s0} + \Delta_{msg} + k.\eta \end{aligned}$$

Partial result 1 *The expected arrival date of the m_k message is bounded by:*

$$t_{s_k} < EA_k \leq t_{s_k} + \Delta_{msg}$$

In Task 2, the safety margin α_k is obtained:

$$\begin{cases} error_k = t_{r_{(k-1)}} - EA_{(k-1)} - delay_{(k-1)} \\ delay_k = delay_{(k-1)} + \gamma.error_{(k-1)} \\ var_k = var_{(k-1)} + \gamma(|error_{(k-1)}| - var_{(k-1)}) \\ \alpha_k = \beta.delay_k + \nu.var_k \end{cases} \tag{3}$$

From partial result 1 and partially synchronous system, we can bound $error$ by:

$$-\Delta_{msg} - delay_{(k-1)} \leq error_k \leq \Delta_{msg} - delay_{(k-1)}$$

Therefore, we can deduce:

$$\begin{aligned} delay_k &\leq (1-\gamma).delay_{(k-1)} + \gamma.\Delta_{msg} \\ delay_k &\leq (1-\gamma)^k delay_0 + \gamma.\Delta_{msg} \sum_{i=0}^{k-1} (1-\gamma)^i \end{aligned}$$

$$As \begin{cases} \sum_{i=0}^{(k-1)} (1-\gamma)^i = \frac{1-[1-\gamma]^k}{1-[1-\gamma]} = \frac{1-[1-\gamma]^k}{\gamma} \\ \text{and} \quad \gamma \in]0,1[\implies 1-\gamma \leq 1 \\ \text{then} \quad \forall k \in \mathbf{N} \mid (1-\gamma)^k < 1 \end{cases}$$

Then in the worst case, $delay$ is bounded by: $0 \leq delay_k \leq delay_0 + \Delta_{msg}$

We can apply the same reasoning to var:

$$\text{If } \Delta_{msg} > delay_0 : \quad var_k \leq (1-\gamma)^k + \gamma(2\Delta_{msg} - delay_0) \textstyle\sum_{i=0}^{k-1} (1-\gamma)^i$$

We can conclude that var, in the worst case (note that this case is not compatible with the worst of case for $delay$), is bounded by:

$$0 \leq var_k \leq 1 + 2\Delta_{msg} - delay_0 \tag{4}$$

Partial result 2 *From intermediate results, if $delay$ and var are bounded, we can conclude that α is also bounded by:*

$$0 < \alpha_k \leq \beta(delay_0 + \Delta_{msg}) + \nu(1 + 2\Delta_{msg} - delay_0)$$

Partial result 3 *From task 3, every time where p times out and q is correct then $\Delta_p(q)$ is increased. There is a time t_{bound} where $\Delta_p(q)$ is large enough to avoid false detection and stops increasing. When $\Delta_p(q)$ becomes upper than Δ_{msg} then no false detection can happened.*

$$\exists\, t_{bound}, \forall t \geq t_{bound}, \quad \Delta_p(q)(t) \geq \Delta_{msg} \quad \text{and} \quad \Delta_p(q)(t) = \Delta_p(q)(t+1)$$

In Partial result 1, we show that the expected arrival date for any message m_k is bounded, in result 2, α is bounded and in result 3, $\Delta_p(q)$ is bounded. All components of τ_k are bounded then we can deduce that τ_k is bounded. If for each message $m_{(k-1)}$ received from process q, process p activate a bound timeout, then there is a time after which p suspects q, if it receives no new message from q. The strong completness is proved. □

Eventual Strong Accuracy The Theorem 2 is verified if the $\tau_k(q)$ of process p is large enough to avoid that process p wrongly suspects process q. From our model, if the lemma 3 is verified then the Theorem 2 is a direct deduction.

Lemma 3 *There is a time after which τ_k is greater than $(t_{s_k} + \Delta_{msg})$.*

$$\exists t_{bound}; \forall t \geq t_{bound}, \quad \tau_k \geq t_{s_k} + \Delta_{msg}$$

Proof: From partial results 1, 2 and 3 we can say that:

$$\begin{array}{rrcl} & \forall m_k \quad \left\{ \begin{array}{rcl} t_{s_k} & < & EA_k \\ 0 & \leq & \alpha_k \end{array} \right. & & \\ \exists\ t_{bound}, \forall t > t_{bound} & \Delta_{msg} & \leq & \Delta_p(q)(t) \end{array} \tag{5}$$

We can conclude that:

$$\exists t_{bound}; \forall t > t_{bound}, \quad \tau_k > t_{s_k} + \Delta_{msg} \tag{6}$$

The Theorem 2 is verified because if τ_k is larger than $(t_{s_k} + \Delta_{msg})$ then process q cannot be considered as having failed by process p. □

6 Performances

This section is dedicated to studying the behaviour of our failure detector and of its underlying estimation method. It is important to note that these measures are focused on the first layer of our detector, the aim of which is to smooth the detector observation.

First, we describe the experimentations environment, then we compare different estimation techniques presented in Section 3 We show how parameters influence the behavior of our strategy.

6.1 Environment

The experiment described in this Section was performed on a cluster of six PCs. We consider a heterogeneous network composed of four Pentium III 600 MHz and two Pentium IV 1400 MHz linked by a 100 $Mbit/s$ Ethernet. This network is compatible with the IP-Multicast communication protocol. The algorithms were implemented in Java (Sun's JDK 1.3) on top of a 2.4 Linux kernel.

We consider crash failures only. All disruptions in these experiments are due to processor load and transmission delay. For this experiment, the interrogation interval adaptation is disabled so as not to interfere with the observation.

All experimentations are parameterized as follows: $\Delta_i = 5000ms$, $\gamma = 0.1$, $\beta = 1$ and $\nu = 2$, $n = 1000$.

The following graphics representations show how $host_1$ perceives $host_2$. For this purpose, each graphic compares the real interval between two successive heartbeat message receptions $(t_{r_{(k+1)}} - t_{r_k})$ with the results from the different estimation techniques: that is the interval between the arrival date of the last heartbeat message and the estimation for the arrival date of the next heartbeat message $(\tau_{(k+1)} - t_{r_k})$. False detections are brought to the fore when the plot for the real interval is above the plot for the estimations.

6.2 Initialization

This scenario illustrates the evolution of delay when the service starts. The figure 4 presents a representative example from the detector on $host_1$.

Here the initial Δ_{to} is equal to $5700\ ms$, meaning $delay = 700ms$ and $var = 0$. The detector must then adapt the margin to optimize the detection time.

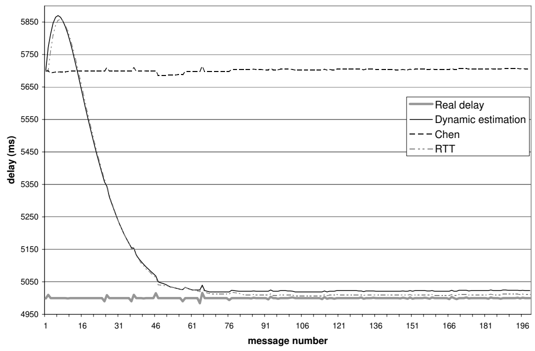

Figure 4: Δ_{to} evolution at detector initialization

Figure 4 illustrates the initialization of a failure detector which monitors another host amongst the five in the system. We can observe the evolution of Δ_{to} with respect to the received message number for each estimation method.

The real delay between two receptions of heartbeat messages is nearly constant. In spite of this Chen's estimation is not a dynamic method so the Δ_{to} is almost constant; the two other methods converge to the Δ_i interval ($5000\ ms$). These latter methods retain a small safety margin which, as observed in our estimation (Dynamic evaluation), is always higher than the RTT estimation. This comes from the fact that our expected time evaluation is obtained with the real arrival dates average, and the real deviations are most often positive.

6.3 Punctual overload

The aim of this experimentation is to show how reacts failure detector when there is a punctual deviation of the sending period. This deviation is obtained by the creation and destruction of 100 threads in the J.V.M. of the sending host which initializes many variables and activates the garbage collector. The garbage collector is activated after the emission of the 38^{th} and the 195^{th} heartbeat messages. Figure 5 illustrates the evolution of Δ_{to} in this situation. In this scenario, the initial value of Δ_{to} is equal to $5100\ ms$: $delay = 100\ ms$ and $var = 0\ ms$.

The shape of this deviation is due to the load on the transmitting host: every negative deviation is instantly followed by an equivalent positive deviation. Indeed this load slows the execution of the heartbeat process; when the process resumes, it compensates this interruption by sending consecutive heartbeat messages.

We observe that Chen's estimation increases lightly Δ_{to}, because the last deviation has the same weight as all the previous delays in the Δ_{to} calculation. Whereas in the RTT estimation, as the last delay is more important than the others, the estimation is more reactive, in proportion to parameter γ. For our estimation,

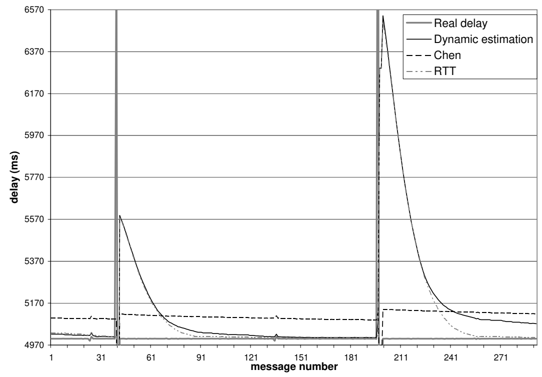

Figure 5: Δ_{to} evolution with punctual deviation

this deviation has consequences on the calculation of the expected date and the safety margin.

It has the same reaction as the RTT estimation in the short term and the same reaction as Chen's estimation in the medium term.

This scenario is not very adequate for a dynamic estimation because the deviations due to overload are too spaced out. However, it illustrates the reaction of failure detectors; the adaptation layer algorithm ought to be more suited to deal with such deviations.

6.4 Constant overload

A constant overload implies that the emission periods of heartbeat messages are very irregular. This scenario shows how failure detectors can avoid false detection while upholding a correct detection time.

This load is generated by an external program on the sending host which periodically creates 100 processes. The initial configuration is: $\Delta_{to} = 5700\ ms$, $delay = 700\ ms$ and $var = 0\ ms$.

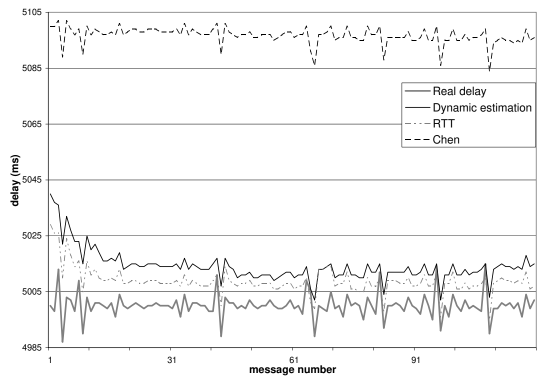

Figure 6: Δ_{to} evolution with constant load

This experimentation shows that our estimation allows to avoid more false detections than the RTT estimation, and at the same time upholds a better detection time than Chen's estimation. This experimentation is summarized in figure 7.

	Dynamic estimation	RTT estimation	Chen's estimation
number of false detections	0	4	0
Detection Time average (ms)	$5016, 6$	$5011, 9$	$5089, 9$

Figure 7: Summary of constant load experienement.

To complete this comparison, figure 8 summarizes an experimentation performed over two days. The hosts which take part in the detection are submitted to a normal use by the laboratory staff.

This experimentation is in accordance with the previous result: our estimation is a compromise between a good detection time and the need to avoid false detections.

	Dynamic estimation	RTT estimation	Chen's estimation
number of false detections	24	51	19
Mistake duration average (ms)	$76, 6$	$25, 23$	$51, 61$
Detection Time average (ms)	$5152, 6$	$5081, 49$	$5672, 53$

Figure 8: Summary of the "real" experienement

6.5 Impact of parameters

Our estimation uses many parameters; these parameters do not have the same influence on the failure detector behaviour. For example the initial $delay$ and var values are not important overall because the aim of this estimation is to adapt those values to the environment. The important parameters are those for the margin calculation: (γ) and for the expected arrival date estimation: (n).

γ allows to adjust the influence of the last delay in relation to the previous results and n is the number of messages which take part in the calculation. These parameters allow to adjust the memory of the estimation. Therefore, the higher their values, the less dynamic the expected arrival date becomes, as seen in the previous experimentations. However, if the values are too small, the expected arrival date is indeed dynamic but it is not that interesting because we already have a short-term dynamic safety margin. Figure 9, illustrates the influence of parameter n in our dynamic estimation technique.

7 Shared Service

The current functional architecture we propose includes an adaptation layer between the basic failure detection layer and the user application. Adaptors provide higher-level algorithms to enhance the characteristics of the failure detectors. There are two motives for this adaptation layer.

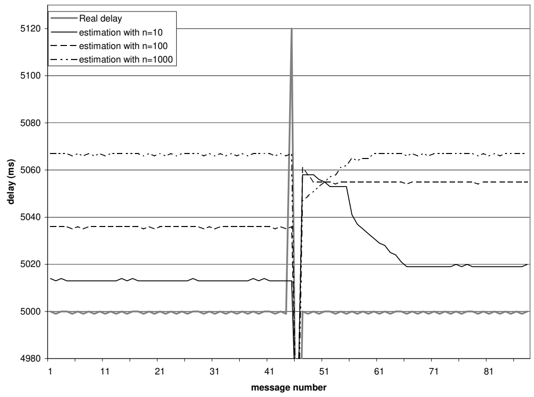

Figure 9: the n last messages influence on Dynamic estimation

Firstly, it is a simple means of improving the quality of the detection. For instance, with a view to avoiding malfunctions or for optimization as well as enhancement purposes, it may be adequate to apply a particular smoothing method to the peaks observed in the performances. In the context of this article, the developed adaptor guarantees that the supplied detection satisfies the $\diamondsuit P$ requirements. Moreover, the number of false detections is memorized in order to evaluate the degree of reliability associated to the detector. This degree is then included in the calculation to determine the most suitable value for the new detection delay.

Secondly, the adaptation layer guarantees the adequacy of the detection: it allows to adjust the provided quality of service to the needs of each application. By maintaining statistics such as the meantime between failures or the rate of mistaken detections, as well as by exchanging information with the adaptation layer from other nodes, an application-specific failure detection service can be achieved.

The reason why ensuring the adequacy of the failure detection isn't left entirely to the responsibility of the client software developer is that integrating the adaptation within the application seems fastidious and inefficient. Typically, application overheads might be cut by piggybacking some messages on the communications exchanged between detectors. The conception of the piggybacking mechanism is sure to be simpler as part of the relevant adaptor. An implementation of this example is detailed in the next section. In addition, the adaptation layer provides a simple way of defining various qualities of service for the same application. Each definition will then be embodied in a corresponding adaptor.

Furthermore, the adaptation layer enables several applications to rely on the same failure detector. As a matter of fact, nothing prevents the existence of more than one adaptor on the same node. Thus it is possible to ensure the coexistence of any amount of clients, even if their needs and constraints are very different in terms of such a service.

8 Application

The failure detector implementation presented in this article is part of the DARX (Dynamic Agent Replication eXtension) project [13]. The goal of this project is to provide a framework for designing large-scale agent systems. Such systems theoretically comprise thousands of agents; in general, they are developed in the field of distributed artificial intelligence. Many multi-agent platforms [2, 1, 14] propose solutions to deploy agent applications over networks. However, to our knowledge no platform provides the required characteristics for massive agent organisations running over asynchronous systems such as the world wide web.

As part of the means to supply adequate support for large-scale agent applications, the DARX platform includes a hierarchical, fault-tolerant naming service. In order to provide a synchronous abstraction of the underlying network, this distributed service is mapped upon the failure detection service through the adaptation layer presented in section 7 With a view to supporting large-scale integration, the naming service comprehends two levels: a local and a global one.

The system is composed of local groups bound together by a global group. Every local group elects exactly one leader which will participate to the global group. At the global level, each name server maintains a list of the known agents within the application. This information is shared and kept up-to-date through a consensus algorithm implying all the global name servers. When a new agent is created, it is registered locally and the info is passed on to the group leader; likewise in the case of an unregistration.

This organization supposes that two different failure detector types are distinguished. This distinction is important, since a failure does not have the same interpretation in the local context as in the global one. A local failure corresponds to the crash of a host, whereas in the global context a failure represents the crash of an entire local group. In this situation, the ability to provide different qualities of service to the local and the global detectors is a major asset of our implementation.

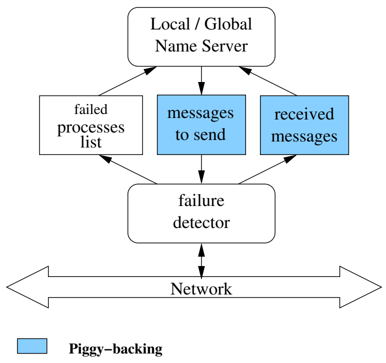

Figure 10: Usage of the failure detector by the name server

Figure 10 details the integration of each failure detector within the naming service. The information is exchanged between name servers via piggybacking on the failure detection messages, that is the "*I am alive*" notifications. The local lists of processes which are suspected to be faulty are directly reused to maintain the global view of the application. In the DARX context, this means that the list of agents present in the system is systematically updated. When a DARX server is considered as having crashed, all the agents it hosted are removed from the list and replaced by replicas located on other hosts. The election of a new leader within an agent replication group is performed through a consensus at the global level of the naming service.

9 Conclusion

In this paper, we have presented a new vision of failure detectors. This notion is based on considering failure detection as a shared service between several applications. We dissociate two layers: the first layer, called the *basic layer*, provides a basic estimation of the timeout delay Δ_{to} and the second layer, called the *adaptation layer*, adapts the information provided by the first layer to the application needs.

We have studied the impact of different estimation algorithms on the QoS of the detection provided by the basic layer, and we have seen that our algorithm provides a good compromise between the optimization of the detection time and the need to avoid false detections. Build upon this basic layer, we prove that it is possible to obtain an *Eventually Perfect* failure detector $\diamondsuit P$ given a specific adaptation layer, although the first aim of this layer is to adapt the QoS to the application needs. Therefore, this layer is specific to the application and it's possible to use several implementations which can provide different visions of the same environment.

In conclusion, we have presented a new implementation of the hearbeat failure detector whose main characterics are to be adaptive as well as dynamic, with a detection delay Δ_{to} composed of a short-term dynamic safety margin and a medium-term dynamic expected arrival date. This failure detector can also change its interrogation delay Δ_i to adapt its adequacy in terms of network load to the application needs and the network capacities. It is particularly adapted for large-scale applications using a detection group division and a hierachical organisation. The use of IP-Multicast and the capacity to use piggybacking on heartbeat messages confirm this ability.

It would now be interesting to study how an application can express its needs in terms of quality of detection. In fact, we have the capacity to implement many adaptation layer algorithms, and although we have the QoS metrics, we cannot express the compromise between these different metrics. It might also be interesting to identify different typical environment behaviours and implement the corresponding adaptive layers. The aim would be to trigger the adequate adaptive layer modification on the fly in order to adapt to the evolution of the environment behaviour.

References

[1] *IBM Aglets homepage*. http://www.trl.ibm.com/aglets/.

[2] *ObjectSpace Voyager 4.0 documentation*. http://www.objectspace.com.

[3] M.K. Aguilera, W. Chen, and S. Toueg. Using the heartbeat failure detector for quiescent reliable communication and consensus in partitionable networks. *TCS: Theoretical Computer Science*, 220, 1999.

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

[5] W. Chen, S. Toueg, and M. K. Aguilera. On the quality of service of failure detectors. In *Proceedings of the First International Conference on Dependable Systems and Networks*, Cornell University, 2000.

[6] D. Dolev, C. Dwork, and L. Stockmeyer. On the minimal synchronism needed for distributed consensus. *Journal of the ACM*, 34(1):77–97, 1987.

[7] D. Dolev, R. Friedman, I. Keidar, and D. Malkhi. Failure detectors in omission failure environments. In *Symposium on Principles of Distributed Computing*, page 286, 1997.

[8] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, apr 1985.

[9] Network Working Group. Rfc 2988 : Computing tcp's retransmission. *http://www.rfc-editor.org/rfc/rfc2988.txt*, 2000.

[10] R. Guerraoui, M. Larrea, and A. Schiper. Non blocking atomic commitment with an unreliable failure detector. Technical report, ESPRIT Basic Research Project BROADCAST, June 1995.

[11] M. Horstmann Mand M. Kirtland. The distributed component object model architecture. *http://msdn.microsoft.com/library/backgrnd/html/msdn_dcomarch.htm*, July 1997.

[12] M. Larrea, A. Fernández, and S. Arévalo. Optimal implementation of the weakest failure detector for solving consensus. In *Proceedings of the 19th Annual ACM Symposium on Principles of Distributed Computing (PODC-00)*, pages 334–334, NY, July 16–19 2000. ACM Press.

[13] O. Marin, J-P. Briot P. Sens, and Z. Guessoum. Towards adaptive fault-tolerance for distributed multi-agent systems. In *Proceedings of European Research Seminar on Advances in Distributed Systems*, pages 195–201, Bertinoro, Italy, May 2001.

[14] M. Strasser, J. Baumann, and M. Schwehm. An agent-based framework for the transparent distribution of computations. In *Arabnia (ed.), Proceedings of Parallel and Distributed Processing Techniques and Applications*, 1999.