Flexible Failure Detection with κ-FD

Naohiro Hayashibara[1], Xavier Défago[1,2], and Takuya Katayama[1]

[1]School of Information Science, Japan Advanced Institute of Science and Technology
[2]"Information & Systems," PRESTO, Japan Science and Technology Agency

February 25, 2004
IS-RR-2004-006

Flexible Failure Detection with κ-FD

Naohiro Hayashibara*, Xavier Défago*† and Takuya Katayama*
*School of Information Science
Japan Advanced Institute of Science and Technology (JAIST)
1-1 Asahidai, Tatsunokuchi, Ishikawa 923-1292, Japan
†PRESTO, Japan Science and Technology Agency (JST)
Email: {nao-haya,defago,katayama}@jaist.ac.jp

Abstract

Many people rightly consider that failure detection should be provided as a generic distributed system service to be used to support fault-tolerance within many applications, in spite of possibly very diverse requirements. Reality is however different, as classical techniques are normally unable to cope with many distributed applications running simultaneously. To overcome this problem, we advocate a different approach to failure detection, whereby the failure detection service associates a real number to each process being monitored, representing the level of confidence that this process has crashed.

In this paper, we present an adaptive failure detection algorithm based on the above principle. When compared experimentally with other recent failure detector implementations, we find that our algorithm performs significantly better with conservative failure detection (high accuracy, slow detection), and equally well with aggressive failure detection (quick detection, low accuracy). This comes in addition to a greater potential for flexibility offered by the model.

I. Introduction

Failure detection is a fundamental building block for ensuring fault tolerance in distributed systems. It is hence natural to consider that failure detection should be provided as a generic system service. This has been advocated by many people [1]–[5] without meeting much success so far in spite of several important breakthroughs. One of the major obstacles to building such a service is that applications with completely different requirements must be able to tune the service to meet their needs, even though they might run simultaneously. Moreover, many distributed applications can greatly benefit from setting different levels of failure detection to trigger different reactions (e.g., [6]–[8]). For instance, an application can take precautionary measures when the confidence in a suspicion reaches a given level, and then take more drastic actions once the confidence raises above a second (much higher) level.

In distributed systems, failure detectors are traditionally based on a simple interaction model wherein processes can only either trust or suspect the processes that they are monitoring. In contrast, we advocate an approach whereby a generic failure detection service outputs a value on a continuous normalized scale. Roughly speaking, this value captures the degree of confidence that the corresponding process has crashed. It is then left to each application process to set a suspicion threshold according to its own quality-of-service requirements. Beside, even within the scope of a single distributed application, it is often desirable to trigger different reactions to increasing degrees of suspicions. The main advantage of our approach is that its very design allows it to scale well with respect to the number of simultaneously running applications and/or triggered actions within each application.

In earlier work [9], [10], we outlined a failure detection scheme based on a similar approach, called the φ-failure detector. Experimental results in wide-area networks [10] have shown that current adaptive failure detection (including our failure detectors) is poorly adapted to very conservative failure detection. In particular, our experiments showed that, when tuned well, nearly all wrong suspicions come as the result of message losses. Unfortunately, current adaptive failure detectors either ignore the problem (e.g., [9], [11]) or are based on the assumption that the loss of consecutive messages are uncorrelated [12]. In contrast, our experiments have shown

us that message losses are strongly correlated and tend to occur in bursts of various length, which is consistent with observations made by Keidar et al. [13], as well as many people in the networking research community.

In this paper, we present the concept of κ-failure detector as a way to address the problems mentioned above. Rather than a failure detector per se, the concept should rather be seen as a way to extend an existing failure detection scheme in order to address the requirements of conservative failure detection. The κ-failure detector outputs a value which is calculated as a sum of contributions from expected heartbeats. Actions triggered by high thresholds will be less sensitive to long bursts of message losses and/or temporary network partitions. In this paper, we describe the κ-failure detector and prove some important properties. After this, we present an implementation based on the φ-failure detector mentioned above and evaluate its behavior over a transcontinental network connection. Our experiments show that the κ-failure detector can be tuned in the conservative range to avoid wrong suspicions.

The remainder of the paper is structured as follows. Section II describes the system and lists important definitions. Section III presents related work on the implementation of failure detectors. In Section IV, we describe the κ-failure detector and prove that it satisfies the property known as strong completeness. Section V describes a possible implementation of the κ-failure detector. This implementation was used to conduct experiments whose results are summarized in Section VI. In Section VII, we briefly review the relationship between our work and recent advances on group membership services. Finally, Section VIII concludes the paper.

II. System Model & Definitions

A. System Model

We consider a simple asynchronous model consisting of two processes p and q. The processes are subject to crash failures only and, crashes are permanent. The processes are connected by two unidirectional channels that cannot create, duplicate, or garble messages. The channels are fair-lossy which means that, if a process, say p, sends an infinite number of messages to process q and q is correct, then q eventually receives an infinite number of messages from p. In practice, a fair-lossy channel can be implemented by some best-effort lossy communication service, such as UDP.

In addition, processes have access to some local physical clock giving them the ability to measure time. We assume nothing regarding the synchronization of these clocks.

In the remainder of the paper, we consider the situation where process q monitors process p.

Our system model is similar to that used by Chen et al. [12].

B. Unreliable Failure Detectors

Being able to detect the crash of other processes is a fundamental issue in distributed systems. In particular, several distributed agreement problems (e.g., Consensus) cannot be solved deterministically in asynchronous systems if even a single process might crash [14]. The impossibility is based on the fact that, in such a system, a crashed process cannot be distinguished from a very slow one.

The impossibility result mentioned above no longer holds if the system is augmented with some unreliable failure detector oracle [15]. An unreliable failure detector is one that can make mistakes, to a certain degree. As an example, we present here the properties of a failure detector of class $\Diamond\mathcal{S}$, which is one of the weakest[1] failure detectors to solve Consensus:

Property 1 (Strong completeness): There is a time after which every process that crashes is permanently suspected by all correct processes.

Property 2 (Eventual weak accuracy): There is a time after which some correct process is never suspected by any correct process.

[1]To be exact, the weakest failure detector for solving Consensus is $\Diamond\mathcal{W}$ [16]. However, any $\Diamond\mathcal{W}$ failure detector can be transformed into a $\Diamond\mathcal{S}$ failure detector [15].

C. Quality of Service of Failure Detectors

Chen et al. [12] propose a set of metrics to evaluate the quality of service (QoS) of failure detectors. Given our assumption that there are two processes p and q where q monitors p, the metrics that we use in this paper are defined below. Notice that the first definition relates to the completeness, whereas the other metrics relate to the accuracy of the failure detector.

Definition 1 (Detection time T_D): The detection time is the time that elapses since the crash of p and until q begins to suspect p permanently.

Definition 2 (Mistake recurrence time T_{MR}): The mistake recurrence time measures the time between two consecutive wrong suspicions. T_{MR} is a random variable representing the time that elapses from an beginning of a wrong suspicion to the next one. T^U_{MR} and T^L_{MR} also denote an upper bound and a lower bound on the mistake recurrence time, respectively.

Definition 3 (Mistake duration T_M): The mistake duration measures the time that elapses from the beginning of a wrong suspicion until its end (i.e., until the mistake is corrected). This is represented by the random variable T_M of which we consider an upper and a lower bound, denoted by T^U_M and T^L_M respectively.

Definition 4 (Good period duration T_G): This is a random variable T_G representing the time that q trusts p and p is up. It can be expressed as $T_G = T_{MR} - T_M$.

Definition 5 (Average mistake rate λ_M): This measures the rate at which a failure detector generates wrong suspicions. This can be expressed by $\lambda_M = \frac{1}{E(T_{MR})}$.

III. Failure Detectors Implementations

A. Traditional heartbeat implementations

Using heartbeat messages is a common approach to implementing failure detectors. It works as follows: process p—i.e., the monitored process—periodically sends a heartbeat message to process q, informing q that p is still alive. The period is called the heartbeat interval Δ_i. Process q suspects process p if it fails to receive any message from p for a period of time determined by a timeout Δ_{to}.

The heartbeat approach leads to the following tradeoff. If the timeout (Δ_{to}) is short, crashes are detected quickly but the likeliness of wrong suspicions is high. Conversely, if the timeout is long, wrong suspicions become less frequent, but this comes at the expense of the detection time.

B. Adaptive implementations

The goal of adaptive failure detectors is to adapt to changing network conditions. In general, adaptive failure detectors are based on a heartbeat strategy (although nothing seems to preclude ping-style failure detection). The principal difference with the heartbeat strategy mentioned above is that the timeout is modified dynamically according to network conditions.

Chen et al. [12] propose an approach based on a probabilistic analysis of network traffic. The protocol uses arrival times sampled in the recent past to compute an estimation of the arrival time of the next heartbeat. The timeout is set according to this estimation and a safety margin, and recomputed for each interval. The safety margin is determined by QoS requirements (e.g. detection time) and network conditions (e.g. network load).

Bertier et al. [11] propose a different estimation function, which combines Chen's estimation with Jacobson's estimation of the round-trip time [17]. Bertier's estimation provides a shorter detection time, but generates more wrong suspicions than Chen's estimation. The resulting failure detector is proved to belong to class $\Diamond\mathcal{P}$ when executed within a specific partially synchronous system model.

There exist also several other proposals for adaptive failure detectors, although with a more conceptual (e.g., [18]) or more specific (e.g., [4]) flavor.

C. The φ-failure detector

Briefly speaking, the φ-failure detector [9] works as follows. The protocol samples the arrival time of heartbeats and maintains a sliding window of the most recent samples. This window is used to estimate the arrival time of the next heartbeat, similar to other adaptive failure detectors [11], [12]. In addition, the distribution of past samples is used as an approximation for the probabilistic distribution of future heartbeat messages. With this information, it is possible to compute a value φ with a scale that changes dynamically to match recent network conditions. Distributed applications use this value and set various thresholds to trigger appropriate reactions.

D. Comparison

We analyzed the behavior of the φ-failure detector over transcontinental links [10], and compared its behavior with that of other adaptive failure detectors [11], [12]. The results showed that the performance of the φ-failure detector was comparable with that of traditional implementations. We however found that none of the failure detectors considered could provide adequate support for conservative failure detection because of their vulnerability to message losses. In particular, neither φ nor Bertier's failure detector consider message losses. In contrast, Chen's takes account of message loses, but unfortunately relies on the unrealistic assumption that losses are uncorrelated. Since, in practice, message losses tend to be *strongly* correlated (i.e., losses tend to occur in bursts), their failure detector remains vulnerable to message losses.

IV. κ-Failure Detectors

In this section, we describe the κ-failure detector as a generic concept rather than a specific implementation (a possible implementation is described in Sect. V). The basic idea is that each missed heartbeat contributes to raise the level of suspicion of the failure detector. First, we define more precisely what the contribution of a heartbeat is. Then, we explain how the value κ output by the failure detector is determined. Finally, we prove the completeness[2] of the resulting failure detector.

A. Heartbeat Contributions (definition)

The κ-failure detector requires the existence of a function of time to represent the evolution of the confidence that a given heartbeat will not be received in the future, either because it was lost or because the sending process has crashed. The function returns a value between 0 and 1, where the latter means total confidence and the former means no confidence at all. Initially, the value is zero and remains so until some time when the heartbeat begins to be expected. Then, the value increases and ultimately converges to one. We consider this function as a black box here. We propose a possible implementation in Section V.

More precisely, the contribution function is defined as follows:

Definition 6 (Contribution function): The contribution function is a function of time which satisfies the properties below.

$$c : \mathbb{R} \longrightarrow [0; 1]$$

- c is monotonic.
- $c(0) = 0$
- $\lim_{t \to +\infty} c(t) = 1$

The function is used for each heartbeat to determine the evolution of the confidence with respect to that heartbeat. Notice that the function can be based on parameters that change dynamically, when new heartbeat are received. We consider that there is a time, called the starting time, before which the heartbeat is not expected.

[2]We do not prove the accuracy of the failure detector essentially for the model assumed in the paper does not allow to ensure accuracy in the formal sense (deterministically), although it does in a more pragmatic way (i.e., stochastically). Nevertheless, we evaluate QoS parameters describing the accuracy of our failure detector experimentally in Section VI-C.1.

Definition 7 (Starting time): Let H^i denote the i-th heartbeat (with $i = 1, 2, \cdots$). Its starting time T^i_{st} has the following property.

- $\forall j \left(i < j \Leftrightarrow T^i_{st} < T^j_{st} \right)$

It follows that the contribution of some heartbeat H^i can be computed simply by (1).

$$c^i(t) = c(t - T^i_{st}) \tag{1}$$

In practice, the nature of the contribution function is important for aggressive failure detectors but not so much for conservative ones. This is because the contribution function defines the meaning of fractional part of the value output by κ.

In reality, one can think of various possible contribution functions. In this paper (Sect. V), we propose an implementation based on the φ-failure detector as described in a recent technical report [10]. Alternatively, the contribution of a heartbeat could be defined as a step function, thus matching single-heartbeat failure detectors based on a conventional *"trust-or-suspect"* scheme, such as Bertier's failure detector [11].

B. Computing the κ Function

The value output by the failure detector is given by a function of time $\kappa(t)$, obtained by summing the contributions of all expected heartbeats with a rank higher than the most recent heartbeat received so far. This is expressed by the function $\kappa(t)$ defined below.

Definition 8 (κ): Let k be the rank of the most recent heartbeat received so far.

$$\begin{aligned} \kappa \quad : \quad & \mathbb{R} \longrightarrow \mathbb{R}^+ \\ & \kappa(t) = \sum_{i=k+1}^{\infty} c(t - T^i_{st}) \end{aligned} \tag{2}$$

Notice that we also assume that, if process p is correct, then p sends infinitely many heartbeat messages.

C. Important Properties

All properties mentioned below are based on the assumption that, when process q monitors process p, q suspects p based on a positive constant[3] threshold K.

Lemma 1: Let p and q be two processes, where q is correct and monitors p. For any finite threshold K, if p crashes, then eventually $\kappa(t) > K$ and this is permanent.

Proof: If p crashes, there is a time after which q never receives any heartbeat from p. Let H^k be the most recent heartbeat received from process q.

Let $c(t)$ denote the contribution function after that time. Since no more heartbeat messages are received, the function does not change.

By its definition, the contribution function is monotonic and converges to one. This means that there a time after which the contribution is always greater than say $\frac{1}{2}$. Let us call this time $T_{\frac{1}{2}}$.

[3]The assumption that K that is constant is done in order to keep the proofs simple. This need not be the case in practice.

Given a finite threshold K, we prove the lemma by showing that there exists a time $\overline{T}$ for which $\kappa(\overline{T}) > K$. Let us choose $\overline{T}$ as follows: $\overline{T} = T_{st}^{k+2\lceil K\rceil+1} + T_{\frac{1}{2}}$. We can now start from $\kappa(\overline{T})$ and develop.

$$\begin{array}{rcl} \kappa(\overline{T}) & = & \kappa(T_{st}^{k+2\lceil K\rceil+1} + T_{\frac{1}{2}}) \\ & & \textit{Eq. (2):} \\ & = & c(\overline{T} - T_{st}^{k+1}) + \cdots + c(\overline{T} - T_{st}^{k+2\lceil K\rceil+1})\{...\} \\ & & \textit{Def. 6:} \\ & \geq & c(T_{\frac{1}{2}}) + \cdots + c(T_{\frac{1}{2}}) \\ & \geq & (2\lceil K\rceil + 1) \cdot \frac{1}{2} \\ & > & 2\lceil K\rceil \cdot \frac{1}{2} = \lceil K\rceil \geq K \end{array} \quad (3)$$

This proves the first part of the lemma. It is now easy to show the second part. Indeed, being the sum of monotonically increasing functions, $\kappa(t)$ is itself monotonically increasing. It follows that, for any time $t' > \overline{T}$, $\kappa(t') \geq \kappa(\overline{T}) > K$. ■

Theorem 1 (Strong completeness): A crashed process is eventually suspected by all correct processes.

Proof: This assumes that all processes monitor each other. Let p be some crashed process, and q be some correct process that monitors p. By Lemma 1, q eventually suspects p (i.e., $\kappa(t) > K$). Since both p and q have been chosen arbitrarily, this completes the proof. ■

V. Implementation

This section describes a possible implementation of the κ-failure detector, based on a failure detection strategy developed in earlier work. We have used this implementation to run the experiments presented in Section VI.

A. Description

In this implementation, the contribution function of heartbeats is computed from the arrival intervals between two consecutive heartbeats. Namely, we use the estimation made for the φ-failure detector [10], and consider that the arrival interval between two consecutive heartbeats is a random variable that we approximate with a normal distribution. The failure detector module is divided into two main tasks, namely, the sampling of heartbeat arrivals, and the computation of the current value for $\kappa(t)$. We now describe the two tasks of the failure detector.

Task 1: Sampling: The sampling task is executed whenever a new heartbeat is received, and gathers information about recent heartbeat arrivals. In particular, the task maintains a sliding window of past arrivals with parameter WS as the window size. Upon receiving a new heartbeat, the task reads the process clock and stores the heartbeat rank and arrival time into the sliding window (thus discarding the oldest heartbeat if necessary).

The task keeps track of four values that are of particular importance for the estimation of κ: the mean μ and the variance σ^2 of inter-arrival times, as well as the rank k and the arrival time A_k, where k is the highest rank among all received heartbeats. For the first two values, this is done by simply keeping track of the sum and the sum of squares of inter-arrival times.

Task 2: Computing κ: This task is invoked when some application process queries the failure detector. The task reads the process clock and computes the value for the function $\kappa(t)$. This is done by approximating the contribution function of expected heartbeats and summing each of them.

Given the values obtained by the first task (i.e., μ, σ^2, k, A_k), the contribution function $c(t)$ is approximated from the cumulative normal distribution function.

$$c(t) = \begin{cases} \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{t} e^{-\frac{(x-\mu)^2}{2\sigma^2}} dx & \text{if } t > 0 \\ 0 & \text{otherwise} \end{cases} \quad (4)$$

It follows that, for some heartbeat H^i, where $i > k$, the contribution is computed by the function $c^i(t)$ shown below. Also, in Eq. (4), the contribution of heartbeat H^i starts one heartbeat interval before its estimated arrival.

Hence, the starting time T^i_{st} of heartbeat H^i is given by the following equation.

$$\begin{aligned} T^i_{st} &= A_k + (i - k - 1)\mu \\ c^i(t) &= c(t - T^i_{st}) \end{aligned} \tag{5}$$

Computing the current value of the function $\kappa(t)$ is done by summing the contribution of all heartbeats H^i for which the starting time is past (i.e., where $t > T^i_{st}$).

VI. Experiments

We have analyzed the behavior of our implementation of the κ-failure detector over a transcontinental network connection for a total duration of three weeks. We have set several experimental scenarios of which we present the most relevant ones. We begin the section by describing our experimental setup, then we describe our measurements, present the experimental results, and finish the section with a discussion.

The main goal of our experiments was to observe the ability of the κ-failure detector to tolerate message losses in a tunable way. Besides, we wanted to compare κ-failure detector with other adaptive failure detectors, and observe the effect of certain parameters on a real-world environment.

A. Experimental Setup

Our experiments involved two computers, with one in Japan and the other in Switzerland, and connected through a normal intercontinental Internet connection. One machine was sending heartbeats (thus acting like process p) while the other one was recording the arrival times of each heartbeat (thus acting like process q).

The sending host was located in Switzerland, at the Swiss Federal Institute of Technology in Lausanne (EPFL). The machine was equipped with a Pentium III processor at 766 MHz and 128 MB of memory. The operating system was Red Hat Linux 7.2 (with Linux kernel 2.4.9).

The receiving host was located in Japan, at the Japan Advanced Institute of Science and Technology (JAIST). The machine was equipped with a Pentium II processor at 450 MHz and 512 MB of memory. The running operating system was Red Hat Linux 9.0 (with Linux kernel 2.4.20).

All messages were transmitted using the UDP/IP protocol. Interestingly, using the `traceroute` command has shown us that most of the traffic was actually routed through the United States, rather than directly between Asia and Europe.

B. Experiments Overview

The experiment was done in two phases. First, we have recorded heartbeat arrivals using the experimental setup described above. Then, we have used simulation to replay the recorded traces with different failure detector implementations. As a result, the failure detectors are compared based on *exactly* the same scenarios, thus ensuring the proper fairness of the comparisons.

Phase 1: Recording heartbeat arrivals: For the first phase, we have run a program on the EPFL machine to generate heartbeat messages. Another program ran on the JAIST machine to record the arrival time of each heartbeat and log the information into a file. Neither machine failed during the experiment. The experiment lasted for three weeks, during which heartbeat messages were generated at a constant rate of one every 30 seconds. A total of $60,489$ heartbeat messages were generated. Among those messages, 219 messages were lost; this corresponds to a loss rate of about 0.36%. We observed that message losses tended to occur in bursts, the longest of which was 13 heartbeats long (i.e., it lasted for about 7 minutes). We observed 27 different bursts of consecutively 99 lost messages (see Fig.2). The mean arrival interval of received heartbeats (filtering out lost messages) was 30.00467 seconds with a standard deviation of about 24.5 ms, meaning that arrival times were quite stable (see Fig. 1). As a final note, we have monitored the CPU load average on the two machines during the whole period of the experimentation. We observed that the load was nearly constant throughout, and that the load was well below the capacity of the machines.

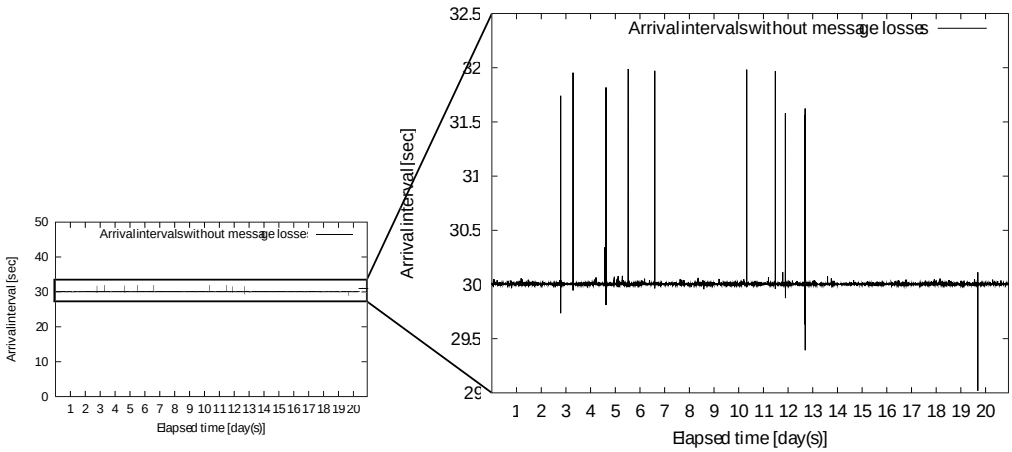

Fig. 1. Heartbeat arrival intervals without message losses

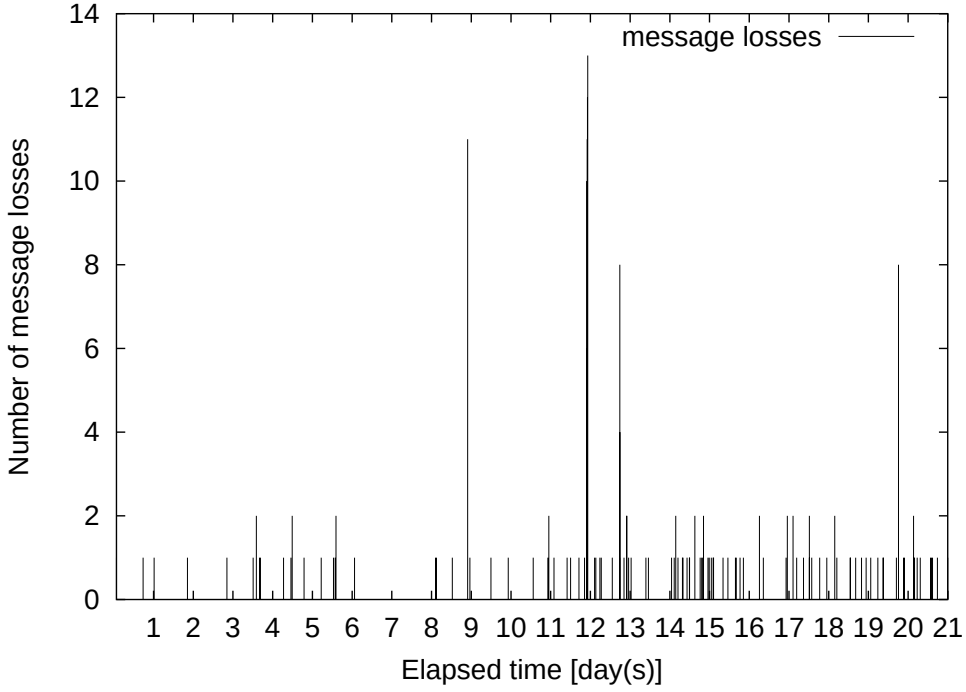

Fig. 2. Occurrence of message losses and network partitions

Phase 2: Simulating failure detectors: Using the trace file obtained during the first phase of the experiments, we have run several simulations involving the κ-failure detector with various parameters. To provide a reference for comparison, we have also run simulations with the failure detector of Chen et al. [12]. In particular, we have done simulations according to the three scenarios described below (the corresponding results are discussed in Section VI-C).

Scenario 1 (Mistake rate): The first scenario measures the mistake rate λ_M, obtained with the κ-failure detector. In particular, we observe the evolution of the mistake rate when the threshold K that triggers suspicions increases. The scenario is important to determine how the κ-failure detector behaves with respect to conservative failure detection. We do the same simulations with three different window sizes for the history, namely $1,000$, $5,000$, and $10,000$ samples.

Scenario 2 (Estimated timeout): The second scenario measures the impact of the threshold K on the estimated timeout (and thus, somewhat indirectly, the detection time). Similar to the previous scenario, we do the simulations

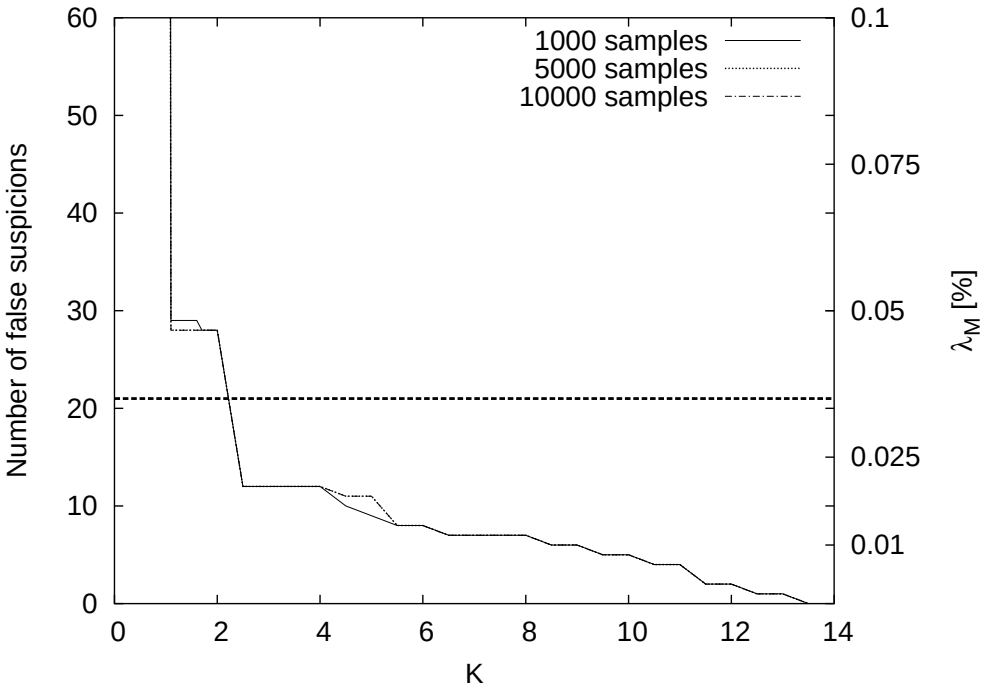

Fig. 3. Number of wrong suspicions with certain K_q

with several window size.

Scenario 3 (Comparison with Chen's FD): In the third scenario, we compare the κ-failure detector with the failure detector of Chen et al. [12]. The goal of the comparison is meant as a way to provide a reference.

For a fair comparison, we have tuned the parameters of both the κ-failure detector (threshold K) and Chen's failure detector (safety margin α) so that they both have the same average mistake rate. We have determined that mistake rate by choosing a threshold of $K = 1$ for the κ-failure detector, and measuring the resulting mistake rate, thus obtaining $\lambda_M = 0.73\%$. Note that we chose a small threshold to ensure a fair comparison of the two failure detectors. Indeed, κ is designed to tolerate message losses whereas Chen's failure detector is not.

In addition, both failure detectors were set to use the same window size of $1,000$ samples for computing their estimation. In order to compare the failure detectors in their stable state, all results obtained during the warmup period—i.e., the period before the window is full—were simply ignored. With the above settings, we have most notably measured the longest mistake duration T_M^U and the shortest mistake recurrence interval T_{MR}^L.

C. Experimental results & discussions

This section presents the results obtained after running the experiments described in Sect. VI-B. The first two scenarios measure the behavior of the κ-failure detector, whereas the last scenario compares κ and Chen's failure detector [12].

1) Mistake rate λ_M (Scenario 1): We measure the mistake rate λ_M of the κ-failure detector, when the threshold K varies. Figure 3 shows the mistake rate expressed as a number of wrong suspicions on the left vertical axis, and as the equivalent relative value in percent on the right. The figure shows three curves obtained by using different window size (i.e., $1,000$, $5,000$, and $10,000$ samples). The horizontal dashed line is used as a reference and represents the mistake rate equivalent to generating one wrong suspicion per day. However, the mistake rate obtained when considering only the wrong suspicions that are due to message losses, too high, could not be represented on the figure.

Figure 3 illustrates the fact that, as the threshold increases, fewer wrong suspicions are generated, until no suspicions are made during the three weeks period of the experiment.

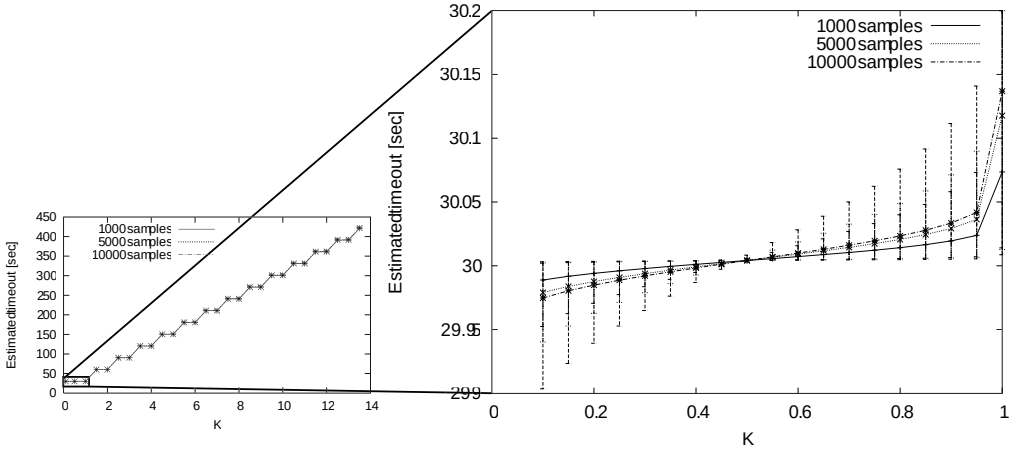

Fig. 4. (Scenario 2) Relation between the threshold K and the estimated timeout. The three curves correspond to different window size ($1,000$, $5,000$, $10,000$ samples).

In particular, when the threshold was set to about $K = 2$, the mistake rate was determined by the message losses. With higher thresholds, some lost messages no longer caused wrong suspicions, until $\overline{K} = 13.5$, beyond which not a single wrong suspicion was generated during the whole duration of our experiment. Evidently, a longer experimentation period or a different environment would almost certainly yield different values for the threshold $\overline{K}$, and hence the value is not particularly important. What is important is simply that such a value exists, and that we were able to observe it under real-world conditions.

The figure allows us to make some other observations. First, we can see that changing the window size had only very little effect on the mistake rate of the failure detector. In light of this, the smallest window size of $1,000$ was sufficient to ensure the stability of the estimation. Second, the mistake rate decreases gradually for threshold values $K \geq 4$. This shows that fine-tuning the κ-failure detector is possible for conservative failure detection. On the other hand, the curve has a staircase-like shape for smaller values of the threshold. This part, important for aggressive failure detection, is essentially determined by the choice of contribution function.

Another interesting value is that, with a threshold of $K = 1$, we obtained 438 wrong suspicions, thus an average mistake rate of $\lambda_M = 0.73\%$. We have used this value to tune the parameters of Scenario 3.

2) Estimated timeout (Scenario 2): We measure the estimated timeout of the κ-failure detector in relation with the threshold K (Fig. 4). Then, in Figure 5, we observe the variation over time of the estimated timeout, with a threshold set to $K = 1$. We repeat the simulations for several window size, namely $1,000$, $5,000$, and $10,000$ samples.

On Figure 4, we see that the estimated timeout grows in a staircase curve as the threshold K increases. This is because the interval between heartbeats was set to 30 seconds, and the standard deviation of inter-arrival times is small. The figure on the right shows a closeup for $K \leq 1$. It shows that the window size has an influence on the granularity of the estimated timeout. In particular, a small window size results in a more staircase-like behavior.

Figure 5 shows the evolution of the estimated timeout over time, when the threshold is set to $K = 1$ and the window size to $1,000$, $5,000$, and $10,000$ samples. The plot on the right is a closeup around 30 seconds. The plot confirms the intuition that a small window is more sensitive to transient behaviors on the network, while a larger window size results in a larger expected timeout.

3) Comparison with Chen's FD (Scenario 3): We compare the behavior of the κ-failure detector with that of Chen et al. [12]. To do so, we set the threshold of κ to $K = 1$, and tuned the parameter α (i.e., the safety margin) of Chen's failure detector, so that both failure detectors have the same mistake rate ($K = 1$, $\lambda_M = 0.73\%$, $\alpha = 3.95$ ms). We then compared the failure detectors based on other QoS metrics. In particular, we have measured the average estimated timeout,[4] the range of the mistake duration $[T_M^L, T_M^U]$, as well as the range of the mistake

[4]The estimated timeout is indirectly related to the detection time T_D. However, since we did not have synchronized clocks between the two machines, we could not accurately measure the detection time.

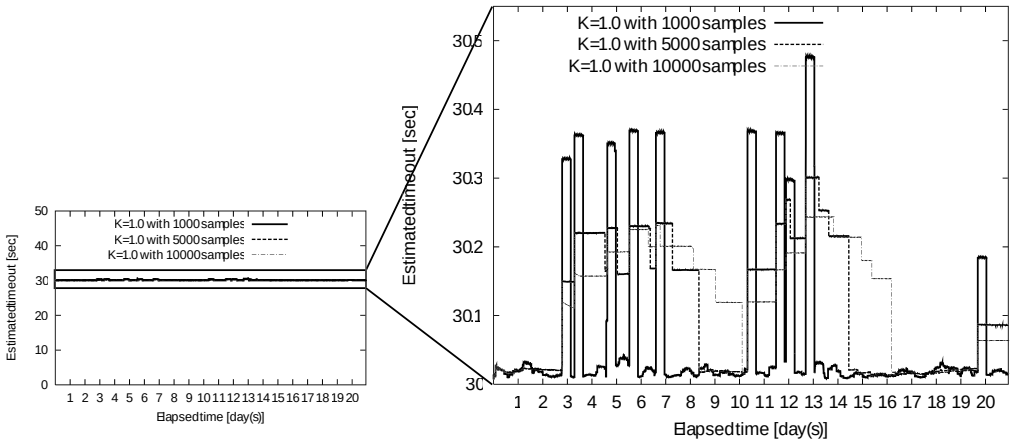

Fig. 5. Estimated timeout over time, with $K = 1$. The three curves correspond to different window size ($1,000$, $5,000$, $10,000$ samples).

TABLE I
κ-FD VS. CHEN'S ESTIMATION.

	κ-FD	Chen's estimation
Parameters	$K = 1$	α = 3.95e-03
Window size	1,000	1,000
Average estimated timeout [s]	30.073	30.117
Wrong suspicions	438	438
λ_M [%]	0.73	0.73
T_M^L [s]	1.68e-05	7.16e-06
T_M^U [s]	387.952	386.828
T_{MR}^L [s]	30.010	30.008
T_{MR}^U [s]	48,786.912	35,435.059

recurrence time $[T_{MR}^L, T_{MR}^U]$. The results are summarized in Table I.

The results in Table I do not show any significant difference in performance between the two failure detectors, except for the upper bound of the mistake recurrence time which is about 37% longer for κ. This tends to suggest than κ benefits from longer good periods (measured by T_G) than Chen's failure detector. However, many more experiments would be needed to verify this, and we are satisfied to conclude that both failure detectors had similar performance when κ was set as an aggressive failure detector.

Another point of interest is about the warmup period. The results in Table I have been computed after discarding suspicions that occurred during the warmup period. We have looked at the behavior of the failure detectors during the warmup period. We have found that, during this period, Chen's failure detector generated 80 wrong suspicions, whereas κ generated only 6.

Figure 6 depicts the evolution of the estimated timeout for both κ and Chen's failure detectors, during the three weeks that the experiment lasted. The parameters are set to the same values as those described in Table I.

Fig. 6 shows the transition of the estimated timeout in both failure detectors over three weeks according to the setting described in Table I. The upper bound on the estimated timeout of the κ-failure detector is less than one in Chen's failure detector. The plots show that, although the failure detectors have comparable performance, their behavior is different. For instance, it is interesting to note that the κ-failure detector was more sensitive during the first half of the experiment, while Chen's was subject to two peaks during the 12-th and the 13-th days.

VII. RELATION WITH GROUP MEMBERSHIP

Group membership is a popular approach to ensuring fault-tolerance in distributed applications. In short, a group membership keeps track of what process belongs to the distributed computation and what process does not. In

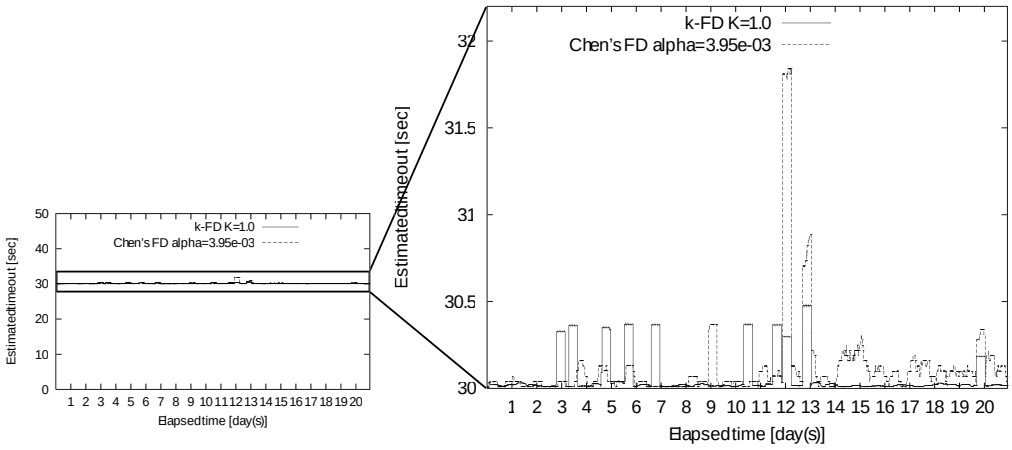

Fig. 6. Evolution of the estimated timeout for κ and Chen's failure detectors. (parameters are described in Table I.)

particular, a group membership usually needs to exclude processes that have crashed or partitioned away. For more information on the subject, we refer to the excellent survey of Chockler et al. [19]. A group membership can also be seen as a high-level failure detection mechanism that provides consistent information about suspicions and failures [8].

In a recent position paper, Friedman [20] proposed to investigate the notion of a fuzzy group membership as an interesting research direction. The idea is that each member of the group is associated with a fuzziness level instead of binary information (i.e., member or not member). Although Friedman does not actually describe an implementation, we believe that a fuzzy group membership could be built based on the κ-failure detector.

Similarly, the κ-failure detector could also be useful as a low-level building block for implementing a partitionable group membership, such as Moshe [13]. Such a group membership must indeed distinguish between message losses, network partitions, and actual process crashes. For instance, Keidar et al. [13] decide that a network partition has occurred after more than three consecutive messages have been lost. Typically, this could be done by using the κ-failure detector and setting an appropriate threshold.

VIII. Conclusion

In this paper, we have presented a novel approach to implementing tunable conservative failure detection in distributed systems. The κ-failure detector presented in this paper addresses the problem of conservative failure detection by taking account of message losses and short-lived network partitions. In addition, the failure detector outputs information on a continuous scale rather than using the traditional "trust-or-suspect" model. This improves its flexibility as applications can trigger suspicions based on their own requirements, without interfering with each other.

The κ-failure detector was described as a generic concept whereby a loss-intolerant detection strategy can be used as the basis for computing the contribution of a single heartbeat. Yet, the combination of contributions makes it possible to set a threshold so that consecutive message losses are tolerated.

The paper describes an implementation of the κ-failure detector, where the contribution of a heartbeat is based on the φ-failure detector [9], [10] described in our earlier work. The resulting implementation is compared with the failure detector of Chen et al. [12]. Our results show that the κ-failure detector behaves as expected in the conservative range, since it can be set so that messages losses do not trigger wrong suspicions. Also, when setting κ for aggressive failure detection, we found that its performance were comparable to that of the failure detector of Chen et al. [12]. However, we believe that there exists room for improvement, especially when κ is set as an aggressive. In particular, it might be interesting to evaluate other contribution functions, but we leave this for future work.

ACKNOWLEDGMENTS

We are grateful to Péter Urbán for giving us precious advice and insightful comments, as well as André Schiper for allowing us to remotely use a machine in his laboratory at EPFL, so that we could conduct our experiments.

REFERENCES

[1] R. van Renesse, Y. Minsky, and M. Hayden, "A gossip-style failure detection service," in *Middleware'98*, N. Davies, K. Raymond, and J. Seitz, Eds., The Lake District, UK, Sept. 1998, pp. 55–70.

[2] P. Stelling, I. Foster, C. Kesselman, C. Lee, and G. von Laszewski, "A fault detection service for wide area distributed computations," in *Proc. 7th IEEE Symp. on High Performance Distributed Computing*, July 1998, pp. 268–278.

[3] P. Felber, X. Défago, R. Guerraoui, and P. Oser, "Failure detectors as first class objects," in *Proc. 1st IEEE Intl. Symp. on Distributed Objects and Applications (DOA'99)*, Edinburgh, Scotland, Sept. 1999, pp. 132–141.

[4] I. Sotoma and E. R. M. Madeira, "Adaptation - algorithms to adaptive fault monitoring and their implementation on CORBA," in *Proc. 3rd Intl. Symp. on Distributed-Objects and Applications (DOA'01)*, Rome, Italy, Sept. 2001, pp. 219–228.

[5] N. Hayashibara, A. Cherif, and T. Katayama, "Failure detectors for large-scale distributed systems," in *Proc. 21st IEEE Symp. on Reliable Distributed Systems (SRDS-21), Intl. Workshop on Self-Repairing and Self-Configurable Distributed Systems (RCDS'2002)*, Osaka, Japan, Oct. 2002, pp. 404–409.

[6] B. Charron-Bost, X. Défago, and A. Schiper, "Broadcasting messages in fault-tolerant distributed systems: the benefit of handling input-triggered and output-triggered suspicions differently," in *Proc. 21st IEEE Intl. Symp. on Reliable Distributed Systems (SRDS-21)*, Osaka, Japan, Oct. 2002, pp. 244–249.

[7] X. Défago, A. Schiper, and N. Sergent, "Semi-passive replication," in *Proc. 17th IEEE Intl. Symp. on Reliable Distributed Systems (SRDS-17)*, West Lafayette, IN, USA, Oct. 1998, pp. 43–50.

[8] P. Urbán, I. Schnayderman, and A. Schiper, "Comparison of failure detectors and group membership: Performa nce study of two atomic broadcast algorithms," June 2003, pp. 645–654.

[9] N. Hayashibara, X. Défago, and T. Katayama, "Two-ways adaptive failure detection with the φ-failure detector," Printouts distributed to participants, Oct. 2003, presented at Workshop on Adaptive Distributed Systems, as part of the 17th Intl. Conf. on Distributed Computing (DISC'03).

[10] ——, "Implementation and performance analysis of the φ-failure detector," Japan Adv. Inst. of Sci. and Tech., Ishikawa, Japan, Research Report IS-RR-2003-013, Sept. 2003.

[11] M. Bertier, O. Marin, and P. Sens, "Implementation and performance evaluation of an adaptable failure detector," in *Proc. 15th Intl. Conf. on Dependable Systems and Networks (DSN'02)*, Washington, D.C., USA, June 2002, pp. 354–363.

[12] W. Chen, S. Toueg, and M. K. Aguilera, "On the quality of service of failure detectors," *IEEE Trans. Comput.*, vol. 51, no. 5, pp. 561–580, May 2002.

[13] I. Keidar, J. Sussman, K. Marzullo, and D. Dolev, "Moshe: A group membership service for WANs," *ACM Trans. Comput. Systems*, vol. 20, no. 3, pp. 1–48, Aug. 2002.

[14] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of distributed consensus with one faulty process," *J. ACM*, vol. 32, no. 2, pp. 374–382, 1985.

[15] T. D. Chandra and S. Toueg, "Unreliable failure detectors for reliable distributed systems," *J. ACM*, vol. 43, no. 2, pp. 225–267, 1996.

[16] T. D. Chandra, V. Hadzilacos, and S. Toueg, "The weakest failure detector for solving consensus," vol. 43, no. 4, pp. 685–722, July 1996.

[17] V. Jacobson, "Congestion avoidance and control," in *Proc. of ACM SIGCOMM'88*, Stanford, CA, USA, Aug 1988.

[18] C. Fetzer, M. Raynal, and F. Tronel, "An adaptive failure detection protocol," in *Proc. 8th IEEE Pacific Rim Symp. on Dependable Computing(PRDC-8)*, Seoul, Korea, Dec. 2001, pp. 146–153.

[19] G. Chockler, I. Keidar, and R. Vitenberg, "Group communication specifications: A comprehensive study," vol. 33, no. 4, pp. 427–469, May 2001.

[20] R. Friedman, "Fuzzy group membership," in *Future Directions in Distributed Computing: Research and Position Papers (FuDiCo 2002)*, ser. LNCS, A. Schiper, A. A. Shvartsman, H. Weatherspoon, and B. Y. Zhao, Eds., vol. 2584. Bertinoro, Italy: Springer-Verlag Heidelberg, June 2003, pp. 114–118.