Implementation and Performance Analysis of the φ-Failure Detector

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

October 6, 2003
IS-RR-2003-013

JAIST Research Report

Implementation and Performance Analysis of the φ-Failure Detector

Naohiro Hayashibara[†] Xavier Défago[†,‡] Takuya Katayama[†]

[†]School of Information Science
Japan Advanced Institute of Science and Technology (JAIST)
1-1 Asahidai, Tatsunokuchi, Ishikawa 923-1292, Japan

[‡]"Information and Systems," PRESTO, Japan Science and Technology Agency (JST)
E-mail: {nao-haya,defago,katayama}@jaist.ac.jp

Abstract

Failure detection is a fundamental building block for ensuring fault tolerance in distributed systems. However, providing accurate and flexible failure detection in off-the-shelf distributed systems is difficult. Practical solutions to failure detection rely on some adaptive mechanism to cope with the unpredictability of networking conditions. However, while they provide reasonably good accuracy, they also lack the necessary flexibility to provide failure detection as a system-wide service. In particular, traditional solutions take a "one size fits all" approach, which prevents them from simultaneously supporting several distributed applications with very diverse QoS requirements.

In this paper, we present a novel approach to adaptive failure detection, called φ-failure detector, which addresses the flexibility issue mentioned above. We describe an implementation, and analyze its behavior over intercontinental communication links during several weeks. Our experimental results show that our failure detector compares well with other known adaptive failure detection mechanisms, with the considerable advantage that it provides virtually limitless flexibility.

Keywords: adaptive failure detection, φ-failure detectors, flexibility for application requirements, implementation issue, performance analysis and comparison

1 Introduction

A fundamental aspect of distributed systems is that they are subject to partial failures. This means that a portion of the system might fail while the remainder remains operational. In this situation, it is highly desirable that the system remains operational as a whole, in spite of the failure of some of its components.

Failure detectors. Nearly every distributed application, mechanism or protocol developed to tolerate the crash of some of its components relies on the ability to detect failures. A failure detection mechanism can be used explicitly, or implicitly by relying on the higher-level abstractions, such as a group membership service or other group communication primitives (e.g., consensus, total order broadcast). This makes failure detection a fundamental issue for ensuring fault-tolerance in distributed systems. This importance has led several authors to advocate that failure detection should be implemented as a generic service (e.g., [10, 13, 18, 17]), similar to the naming, authentication, or directory services. Unfortunately, this is still far from being a reality, and most distributed applications instead rely on some naive and ad hoc failure detection mechanism.

From a formal standpoint, the most notorious evidence of the central role of failure detection stems from the impossibility of solving the consensus problem in asynchronous systems[1] if even a single process might crash [12]. This impossibility is a consequence of the fact that, in asynchronous systems, a crashed process cannot be distinguished from a very slow one, with absolute confidence. It was shown later that the consensus problem can in fact be solved, if the system is augmented with an unreliable[2] failure detector oracle [4]. Formally, even the weakest failure detector needed to solve consensus, called $\Diamond\mathcal{W}$ [3], cannot be implemented by relying purely on message-passing, or else this would contradict the impossibility result mentioned above.

Adaptive failure detectors. Practical solutions can nevertheless be developed for systems in which message delays follow some probability distribution (e.g., [5]). In particular, adaptive failure detection mechanisms [2, 5, 11] consider some system where the parameters of this distribution are unknown, and can change over time, but eventually stabilize for periods that are "long enough" for the whole system to make some progress.[3] The idea of adaptive failure detection is that a monitored process p periodically sends a heartbeat message ("I'm alive!"). A process q begins to suspect p if it fails to receive a heartbeat from p after some timeout. Adaptive failure detection protocols change the value of the timeout dynamically, according to the network conditions measured in the recent past. Doing so, adaptive protocols are able to cope adequately with changing networking conditions, and hence they are particularly appropriate for common networking environment, or the Internet. In particular, they are able to maintain a good compromise between how fast they detect actual failures, and how well they avoid wrong suspicions.

The main drawback of the adaptive failure detection protocols that we are aware of [2, 5, 11] is their inability to address the QoS requirements of several distributed applications *simultaneously*; in other words, their lack of flexibility [13]. Let us illustrate this with a simple example. Consider for instance a situation where two applications are running simultaneously, and one is an interactive application while the other is a heavy-weight database service. The former application must always be highly responsive; it needs fast yet possibly inaccurate failure detection. Meanwhile, the latter application has a high reconfiguration overhead, and needs highly accurate failure detection, even though it might be slow. Addressing the requirements of both applications is not possible with the usual "one size fits all" approach adopted by the known adaptive protocols.

Adaptive φ-failure detector. We propose a novel approach to adaptive failure detectors, called the φ-failure detector, which addresses the problem of flexibility mentioned above.[4] The basic idea is as follows. Other adaptive failure detectors provide information of a Shakespearean nature (i.e., *suspect* or *not suspect*) and change the threshold between these two possible values according to network conditions. In contrast, the φ-failure detector associates a value φ_p to some monitored process p. This value is expressed on a continuous scale that roughly represents the current level of confidence that process p has crashed. The scale itself is adapted dynamically to match the current network conditions and to ensure an adaptive behavior. Simultaneously running applications receive exactly the same information, and can set a threshold for φ_p according to their respective requirements; a small threshold value yields fast and inaccurate failure detection, whereas a large value results in accurate yet slow failure detection.

Contribution. In this paper, we propose a pragmatic implementation of the φ-failure detector, and analyze its behavior between Japan and Europe over the period of three weeks.

[1] An asynchronous distributed system in one in which there are bounds neither on communication delays nor on the speed of processes.

[2] An unreliable failure detector [4] is a failure detector that is allowed to make mistakes. For instance, it can suspect a process that has not crashed.

[3] Exact assumptions can vary slightly between authors.

[4] We sketched the basic idea in an earlier paper [14] without actually implementing or evaluating the approach.

Briefly speaking, the proposed implementation 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 [2, 5]. 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 the value φ_p with a scale that changes dynamically to match recent network conditions.

We have evaluated our failure detection scheme under normal transcontinental conditions (between Japan and Switzerland). Heartbeat messages were sent at a rate of one every 30 s using the user datagram protocol (UDP), and the experiment ran uninterruptedly for a period of three weeks, gathering a total of more than $60,000$ samples. Using these samples, we analyzed the behavior of our failure detector, and compared with traditional adaptive failure detectors [2, 5]. By providing exactly the same input to every failure detector, we could ensure the fairness of our comparisons. The results show that our failure detector implementation performs well when compared with traditional implementations, with the additional advantage that it provides virtually limitless flexibility by design.

The rest of the paper is organized as follows. Section 2 describes the system model and the basic assumptions. Section 3 discusses important facts about failure detectors and existing implementations. Section 4 presents the concept of the φ-failure detector in details. Section 5 describes our implementation of the φ-failure detector. In Sect. 6, we present our experiment, analyze the behavior of our failure detector under normal Internet conditions, and compare it with other failure detectors. Finally, Section 7 concludes the paper.

2 System Model and Definitions

2.1 System Model

We represent a distributed system as a set of processes $\{p_1, p_2, \ldots, p_n\}$ which communicate only by sending and receiving messages. We assume that every pair of processes is connected by two unidirectional quasi-reliable communication channels [1]. A quasi-reliable channel is defined as a communication channel which guarantees (1) no message loss, (2) no message corruption, and (3) no creation of spurious messages. We consider that processes may only fail by crashing, and that crashed processes never recover. We assume the system to be asynchronous in the sense that there exist bounds neither on communication delays nor on process speed.

2.2 Probabilistic Network Model

We assume that communication channels behave independently in regard to their respective timing behavior. For each communication channel, we assume message delays to be determined by some random variable whose characteristics are unknown, independent of other communication channels, and can change over time. We assume bursty traffic, which means that two consecutive messages are very likely to have similar probabilistic characteristics. Periods during which all consecutive messages follow the same probabilistic behavior are considered *stable*. During stable periods, we assume that inter-arrival times of periodically sent messages follow a normal distribution whose parameters (mean and variance) are not known a priori.

More intuitively, we can regard the system as moving form one stable period to another with different characteristics, and possibly with some unstable behavior during the transition. For instance, this can model the fact that traffic in a corporate network is significantly different between working hours or during the night, when fewer people are using it.

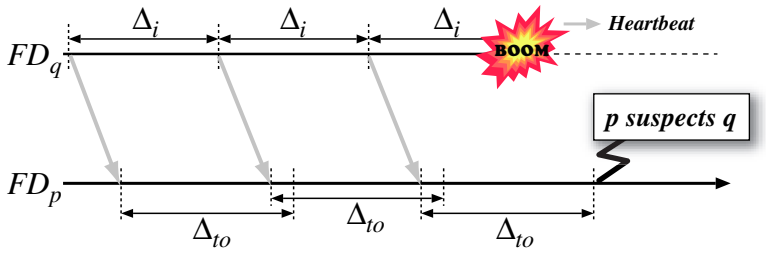

Figure 1. Heartbeat messages

3 Failure Detectors

3.1 Unreliable Failure Detectors

Chandra and Toueg [4] define failure detectors as a distributed oracle with well-defined properties. A failure detector is a distributed entity which consists of a set of failure detector modules, one attached to each process. A failure detector module FD_p, attached to a process p, maintains a set of suspected processes, that p can query at any time. Whenever some process q appears in the set maintained by FD_p, we say that p *suspects* q (to have crashed). The failure detector is however unreliable in the sense that its modules are allowed to make mistakes (1) by erroneously suspecting some correct process (wrong suspicion), or (2) by failing to suspect a process that has actually crashed. A module can also change its mind, for instance, by stopping to suspect at some time t' some process that it suspected from time $t < t'$.

Several classes of failure detectors are defined according to two properties which restrict the mistakes that the failure detector can make. For instance, a failure detector of class $\Diamond\mathcal{P}$ must meet the following properties of completeness and accuracy.

(STRONG COMPLETENESS) Eventually 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.

3.2 Heartbeat Strategy

Using heartbeat messages is quite a common approach to implement failure detectors, and works as follows. Every failure detector module periodically sends a heartbeat message to the other modules, informing them that the process is still alive (see Fig. 1). The period is determined by the heartbeat interval Δ_i. A process p suspects a process q if FD_p, the module attached to process p, fails to receive any message from FD_q for a period of time determined by a timeout Δ_{to}.

There is 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, the chance of wrong suspicions is low, but this comes at the expense of the detection time. Beside, the fact that the timeout is fixed means that the failure detection mechanism is unable to adapt to changing conditions. This is because a long timeout in some system setting can turn out to be very short in a different environment. Beside, in practical systems, network conditions can greatly vary over time. (e.g., depending on the load).

3.3 Adaptive Failure Detectors

Adaptive failure detectors address the problem of adapting to changing network conditions, mentioned previously. There exist several proposals for adaptive failure detection [2, 5, 11, 16]. The proposed solutions are based on a heartbeat strategy, although nothing seem to preclude the use of other strategies such as ping-style failure detection. The principal difference with the heartbeat strategy mentioned above is that the timeout is modified dynamically according to network conditions.

The protocol proposed by Fetzer et al. [11] has a simple adaptation mechanism. It adjusts the timeout by using the maximum arrival interval of heartbeat messages. The protocol supposes a partially synchronous system model [9], wherein an unknown bound on message delays eventually exists. The authors show that their algorithm belongs to the class $\Diamond\mathcal{P}$ in this model.

Chen et al. [5] propose a different 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. [2] propose a different estimation function, which combines Chen's estimation with another estimation due to Jacobson [15] and developed in a different context. Bertier's proposal provides a shorter detection time, but generates more wrong suspicions than with Chen's estimation. The resulting failure detector is proved to belong to class $\Diamond\mathcal{P}$ in a partially synchronous system model.

Sotoma et al. [16] propose an implementation of an adaptive failure detector with CORBA. Their algorithm compute the timeout based on the average time for arrival intervals of heartbeat messages, and some ratio between arrival intervals.

None of the adaptive failure detectors mentioned in this section address the problem of adapting to application requirements. In fact, the failure detection protocol provides a "hardwired" degree of accuracy which must be shared by all applications.

3.4 Tailored Failure Detectors

As far as we know, there exist only few failure detector implementations which allow non-trivial tailoring on the part of the applications, let alone the requirements of several applications running simultaneously.

Cosquer et al. [6] propose configurable failure "suspectors" whose parameters can be fine tuned by a distributed application. The suspectors can be tuned directly, but they are used only through a group membership service and view synchronous communication. There is a wide range of parameters that can be set, but the proposed solution remains unable to simultaneously support several applications with very different requirements.

The failure detector implementation proposed by Chen et al. [5] can also be tuned to application requirements. However, the parameters must be dimensioned *statically*, and can only match the requirements of a *single* application. It can be said that they provide a "hardwired" degree of accuracy which must be shared by all applications.

The two timeout approach [7, 8] can also be seen as a first step toward adapting to application requirements, but the solution lacks generality. The two timeout approach was proposed and discussed in relation with group membership and consensus. In short, it was proposed to implement failure detection based on two different timeout values; an aggressive and a conservative one. The approach is well suited for building consensus-based group communication systems. However, the protocol was not rendered adaptive to changing network conditions (although this would be feasible) and, more importantly, still lacks the flexibility required by a generic service. Indeed, this could support only two classes of applications.

Table 1. The relation between φ_p and P_{acc}

φ_p	0	1	2	3	...	∞
P_{acc}	0	0.9	0.99	0.999	...	1

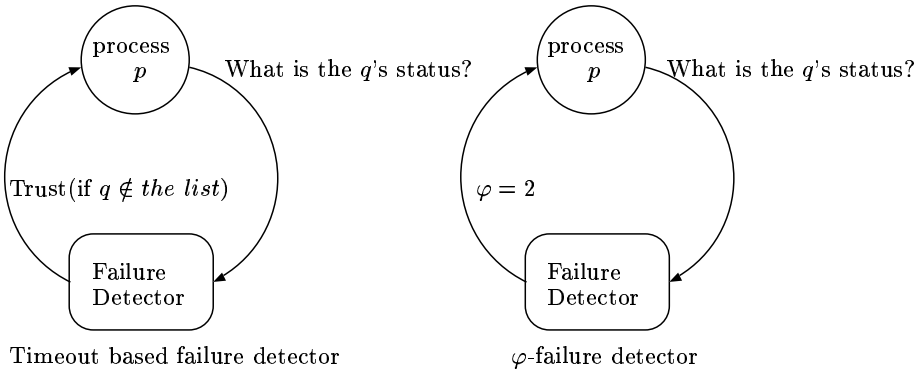

Figure 2. Timeout based failure detector vs. φ-failure detector

4 Adaptive φ-Failure Detectors

As mentioned above, adaptation can occur in several different ways. To be truly generic, a failure detection service must be equally adaptive to (1) changing network conditions, and (2) applications requirements. More concretely, a failure detection service must be able to meet the requirements of a wide range of distributed applications *running simultaneously*. Timeout-based failure detectors are intrinsically limited by the fact that one timeout value is necessary for each set of requirements or, in the worst case, one timeout for each concurrent application. In particular, this is also the case with adaptive failure detectors (see Sect. 3.3). Thus, the latter can adapt to changing network conditions, but they are unable to realistically meet the different requirements of several concurrent applications.

We have developed a novel approach to failure detection, called the φ-failure detectors, which can address both adapt to changing network conditions, and meet the requirements of multiple distributed applications. The principle of the φ-failure detector is as follows. Each failure detector module associates a value $\varphi_p \in \mathbb{R}^+$ to every known process p instead of managing a list of suspected processes. The value φ_p represents the degree of confidence that process p has crashed. This value is expressed according to a normalized scale, where $\varphi_p = 0$ means that there is currently no reason to doubt that p is operational, and conversely, $\varphi_p = \infty$ indicates an absolute confidence that p has crashed. Thus, failure detection modules maintain a list of pairs (p, φ_p) for every monitored process, and which can be queried at any time by any application. More precisely, φ_p is defined along the following scale. Let P_{acc} denote the probability that the statement "process p has crashed" will not be contradicted in the future (by the reception of a late heartbeat). Then, φ_p can be determined by Eq. (1), which leads to the scale illustrated on Table 1.

$$\varphi_p = -log_{10}(1 - P_{acc}) \tag{1}$$

The failure detector must guarantee that φ_p increases monotonically, between two periods where it is reset to 0.

The interactions between the applications and the failure detector are hence different than in the traditional case. Indeed, distributed applications use the value φ_p associated with a process p to decide on a course of action. For

instance, applications can set some finite threshold for φ_p and decide to suspect p if φ_p crosses that threshold. Different applications can then set different thresholds for the same process. For instance, some applications would set a low threshold to obtain prompt yet inaccurate failure detection (i.e., with many wrong suspicions), while applications with stronger requirements would set a higher threshold and obtain more accurate suspicions. Consequently, this approach can effectively adapt to application requirements because the threshold can be set on an per-application basis (and also on a per-communication channel basis within each application). Beside, the scale ensures that (1) the value set as a threshold retains some meaning for the application (it represents the degree of confidence), and (2) the failure detection adapts to changing network conditions even with a fixed threshold (because the scale adapts).

5 Implementation Based on a Sliding Window

In this section, we describe our implementation of the φ-failure detector, which uses a stochastic approach. In short, the approach is simple; a sliding window is maintained and used to compute estimated arrival times, as well as approximate the probabilistic distribution of future arrivals.

Computing φ_p based on a sliding window. For each monitored process p, the failure detector modules maintain a bounded history H_p of arrival intervals of heartbeat messages sent by p. The history H_p, i.e., the sliding window, is implemented as a simple circular array. Let A_k^p be the arrival time for the k-th heartbeat message from process p. Then, the history H_p is a sequence $\{A_1^p, A_2^p, A_3^p, \ldots, A_{|H_p|}^p\}$, where A_1^p is the arrival time of the most recent heartbeat from process p, and $|H_p|$ is the length of the sliding window for that process. We assume that arrival time *intervals* follow a normal distribution. So, based on the history H_p, we compute the mean μ_p and the variance σ_p, and use these parameters to estimate the probabilistic distribution of arrival time.

In fact, computing the mean and variance require only little computation. To do this, we keep two additional variables; the sum and the sum of squares. Whenever we receive a new sample, we subtract $A_{|H_p|}^p\}$ (or its square) from the sums, add the new sample, and append it to the bounded history H_p. Consequently, the size of the sliding window has no effect on the amount of computation needed to obtain the parameters of the distribution, and compute φ_p.

Interaction with applications. The φ-failure detector provides a simple interface for distributed applications; the failure detector is queried through a function call which returns the time when it was called and the computed value for φ_p (see Fig. 2). When an application process queries its failure detector module on the current status of some process p, the module computes the value φ_p at that time, based on the estimated distribution and the time elapsed since the receipt of the last heartbeat (see Fig. 3). Then, the value computed for φ_p is returned to the application process. When polling is not acceptable, the application process sets can set a callback that is triggered when φ_p grows beyond a given value.

Message losses. In our system model, we say that we assume no message loss. In fact, the occurrence of message loss generates a wrong suspicion. This is consistent with the behavior of both Chen's [5] and Bertier's [2] failure detectors. Implicitly, we assume that, taken over long periods, the probability of message loss remains low. This assumption is confirmed by our experimental results between Japan and Switzerland, where we measure an average loss rate below 0.4% (see details in Sect. 6). In fact, we observed that messages are rarely lost individually, but rather that several consecutive messages are lost, probably as the result of some network partition. During our experiments, a total of 219 messages were lost, but only 117 suspicions resulted from these losses.

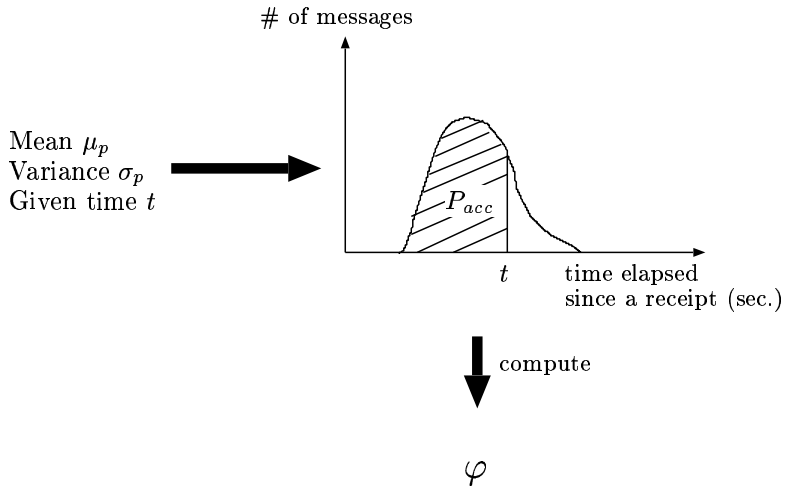

Figure 3. The mechanism for the φ-failure detector

6 Performance Analysis

In this section, we show experimental results about our implementation of the φ-failure detector, as well as a comparison with two other adaptive failure detector implementations, namely Chen et al. [5] and Bertier et al. [2]. Our experiments were done over transcontinental links, for a consecutive duration of three weeks.

6.1 Objective

From a practical point of view, two failure detector metrics [5] are especially relevant to evaluate the practical performance of failure detectors; namely, the *mistake rate*, and the *average detection time*. Roughly speaking, the *mistake rate* measures how many wrong suspicions the failure detector generates, and the *average detection time* measures how fast it detects actual failures.

6.2 Scenarios and parameters

We ran our experiments between two machines, with one machine located at JAIST in Japan, and the other located in Switzerland, at the Swiss Federal Institute of Technology in Lausanne (EPFL).[5] For three weeks, the machine at EPFL was sending heartbeat messages every 30 seconds to the machine at JAIST, using the user datagram protocol (UDP). Upon reception of each heartbeat, the receiving host at JAIST wrote the arrival time of the message into a log file. In total, the sending host at EPFL generated $60,489$ heartbeat messages, out of which the receiving host actually received $60,270$. Consequently, 219 heartbeat messages were lost, with an average loss rate of about 0.36%. As mentioned earlier, messages were usually lost in groups of consecutive messages. We observed 117 such groups, suggesting the occurrence of some network partition. The longest partition that we could observe lasted for a little less than 7 minutes, with 13 messages being lost. Notice also that the CPU load of both machines (EPFL and JAIST) was measured to be nearly constant during the whole experiment. All measurements were based on exactly the same measured sequence of heartbeat arrival times.

[5] In fact, it seems that most of the network traffic was in fact routed through the United States.

Table 2. φ-FD: wrong suspicions according to the threshold Φ_p.

Φ_p	0.5	1	5	9
Wrong suspicions	328	268	185	162
Mistake rate [%]	0.54	0.44	0.30	0.26

In the first part, we have measured the performance of the φ-failure detector, changing two parameters. We have first measured the effect of changing the threshold value for φ on both the mistake rate and the detection time. Then, we have observed the impact of the window size on the behavior of the failure detector.

In the second part, we have compared our failure detector with two different adaptive failure detectors [5, 2]. In particular, we have measured the performance of all three failure detector implementations by injecting each time the same three weeks sequence of measured heartbeat arrival times. With this approach, all three failure detectors are compared under exactly the same conditions, while the experiment is based on real traffic.

6.3 Environments

The sending host (at EPFL) was running Red Hat Linux 7.2 (kernel 2.4.9). The machine had an Intel Pentium III processor clocked at 766 MHz and was equipped with 128 MB of RAM.

The receiving host (at JAIST) was running Red Hat Linux 9.0 (kernel 2.4.20). The machine had an Intel Pentium II processor at 450 MHz and was equipped with 512 MB of RAM.

Both machines were connected to their respective local network, which was a simple Ethernet 100 Base-TX hub. All messages were transmitted using UDP/IP, hence the occurrence of message losses.

6.4 Tuning Parameters of the φ-Failure Detector

From the standpoint of an application, we consider that it sets a threshold Φ_p, and decides to suspect or not a process p based on the value φ_p returned by the failure detector; i.e., suspect p if and only if $\varphi_p > \Phi_p$. As far as the application is concerned, the threshold Φ_p plays the role of a timeout. A major difference is that thresholds are set on a per-application basis, and, within each application, can also be set on a per-channel basis. Also, the threshold need not remain constant over time.

We have studied the impact of Φ_p on our implementation of the φ-failure detector. In addition, we have also measured the impact of the size of the sliding window.

Influence of the Threshold Φ_p. Figure 4 shows how the threshold Φ_p impacts the detection time[6]. In these measurements, the window size is set to $5,000$ samples. In fact, the figure shows the time elapsed between the receipt of the last heartbeat and the beginning of the suspicion. It was not possible to measure the actual detection time as it depends on the propagation time of the last heartbeat message, which could not be measured reliably in the absence of synchronized clocks.

At the beginning of the experiment, the figure shows that the detection time jumps up. This is due to a message loss and the fact that only few samples have been gathered, thus making the failure detector more sensitive. After a while, the failure detector stabilizes. During the rest of the experiment, we can see that the detection is less sensitive to changes for small values of Φ_p than it is for large values. However, Table 2 that this comes at the expense of the accuracy, as a larger number of wrong suspicions are generated. In fact, the difference is even bigger if we consider that the 117 suspicions that are generated as the result of message losses, are independent of the chosen threshold (at least in the cases studied here).

[6]The estimated arrival time implies the detection time with certain threshold Φ_p in Fig. 4

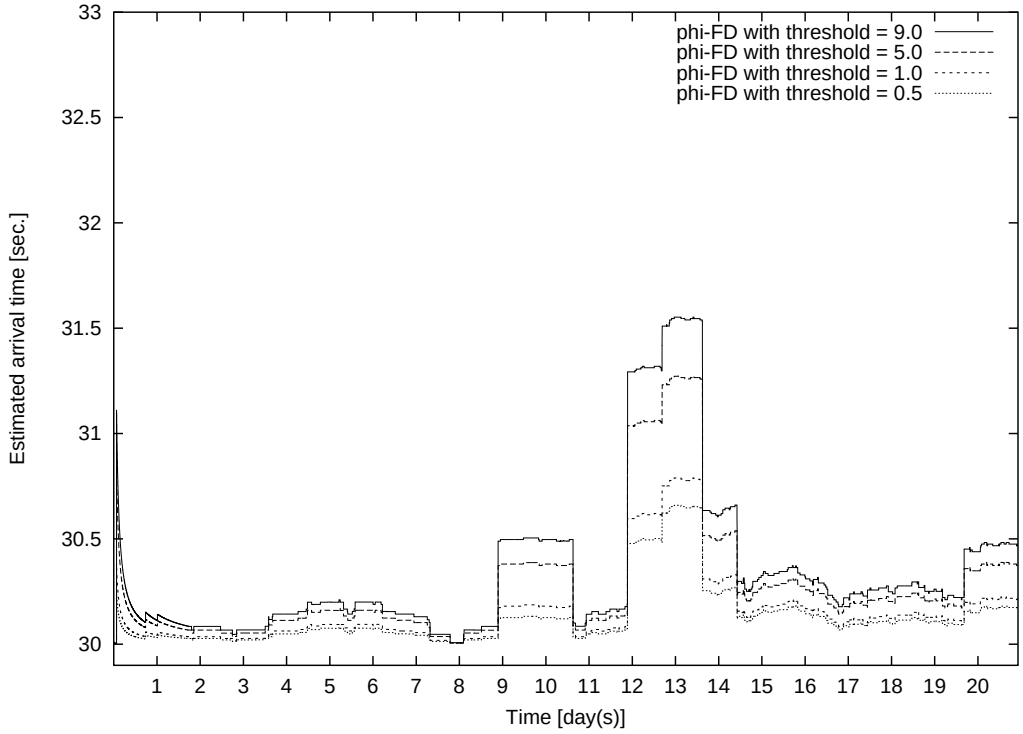

Figure 4. φ-FD: detection time according to the threshold Φ_p.

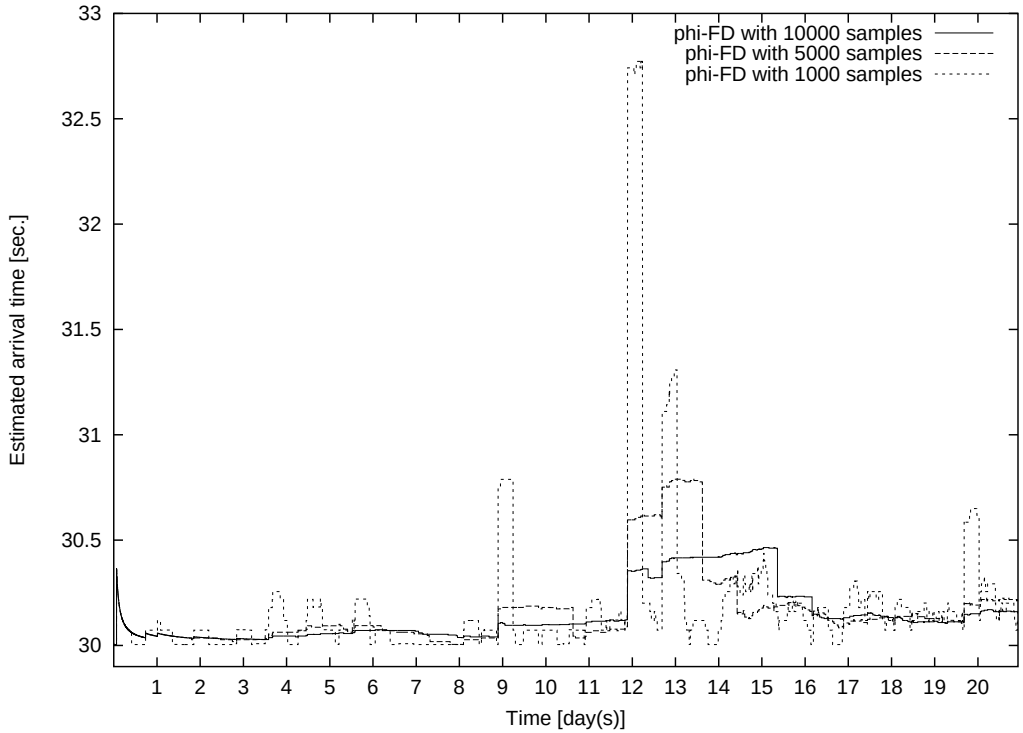

Figure 5. φ-FD: detection time according to the window size.

Table 3. φ-FD: wrong suspicions according to the window size.

Window size	$1,000$	$5,000$	$10,000$
Wrong suspicions	$3,763$	268	190
Mistake rate [%]	6.24	0.44	0.31

Table 4. φ-FD vs. Chen's estimation.

	φ-FD	Chen's estimation
Parameters	$\Phi_p = 1.7$	$\alpha = 0.005$
Window size	$5,000$	$5,000$
Average detection time [s]	30.188	30.113
Wrong suspicions	230	230
Mistake rate [%]	0.38	0.38

Influence of the window size. Figure 5 shows the influence of the window size on the detection time. The threshold Φ_p is fixed and set to 1 for all trials. The figure clearly shows that a failure detector with a short window is more influenced by transient changes than one with a longer window. This is not surprising since, in the former case, a new sample has more relative weight than in the latter case. An indirect consequence of this is that a failure detector with a large window size generates significantly less wrong suspicions than one with a shorter window (see Table 3). In particular, a failure detector with a window size of $1,000$ samples generates nearly 20 times more wrong suspicions that one with a window size of $10,000$ samples.

6.5 Comparison with Other Adaptive Failure Detectors

In this section, we compare successively our φ-failure detector against Chen's estimation [5] and Bertier's dynamic estimation [2], respectively. The main reason why we do not compare all three together is that the chosen settings might handicap one implementation over the others. In contrast, we compare the failure detectors pairwise, and try to have them share as many characteristics as possible.

6.5.1 φ-FD vs. Chen's estimation

We compare the φ-failure detector with Chen's estimation [5], based on the detection time. We set the window of both failure detectors to the same size, that is, $5,000$ samples. Then, we tuned the threshold Φ_p of the φ-failure detector so that both failure detectors generate exactly the same number of wrong suspicions. The parameters are summarized in Table 4.

Figure 6 show that Chen's estimation has a slightly better detection time over the whole experiment. The difference is however almost negligible. Another interesting observation is that both failure detectors behave identically in the face of message losses; that is, message losses result in wrong suspicions.

The safety margin α of Chen's estimation is computed statically according to various QoS requirements. This makes it difficult to adjust in order to obtain better detection time and mistake rate. It is important to note that Chen's failure detector is tunable to meet application requirements. However, unlike the φ-failure detector, Chen's failure detector can only be tuned to meet the requirements of a single application.

6.5.2 φ-FD vs. Bertier's dynamic estimation

We compare the φ-failure detector and Bertier's dynamic estimation [2]. Unlike with Chen's estimation, we were unable to tune the failure detectors so that they have the same mistake rate. As in the previous case, the window

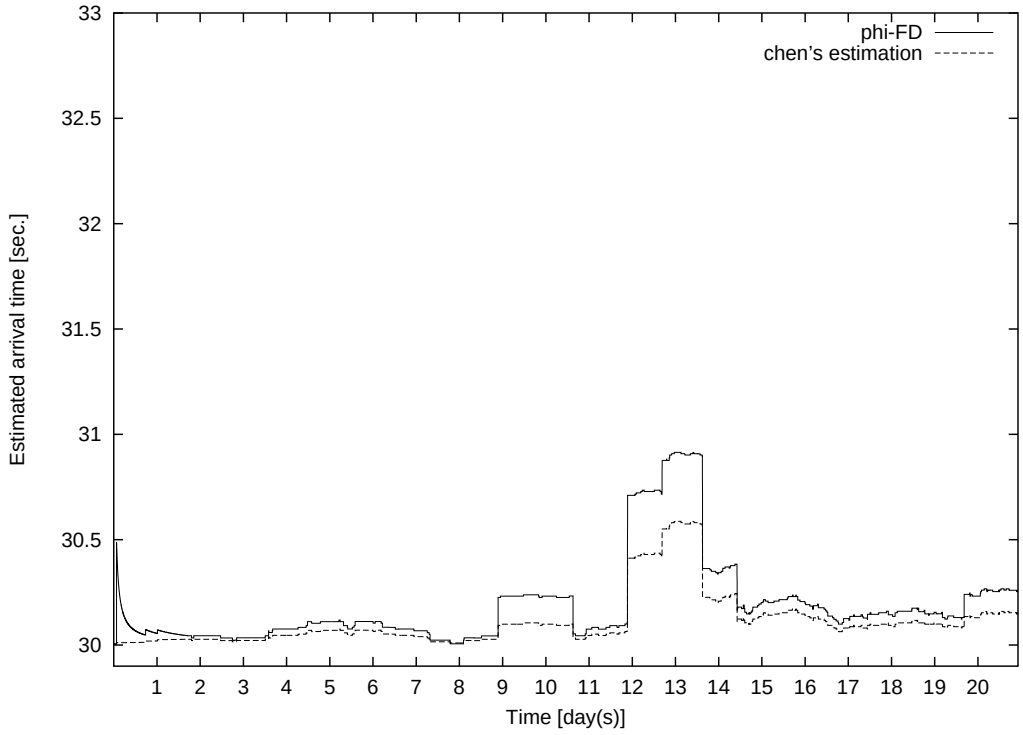

Figure 6. Detection time: φ-FD vs. Chen's estimation.

Table 5. φ-FD vs. dynamic estimation.

	φ-FD	Dynamic estimation
Parameters	$\Phi_p = 9.9$	$\gamma = 0.1, \beta = 1, \phi = 2$
Initialization	n/a	$\text{delay} = 0.007$
Window size	$5,000$	$5,000$
Average detection time [s]	30.354	32.576
Wrong suspicions	158	119
Mistake rate [%]	0.26	0.19

size was set to $5,000$ samples for both failure detectors. The parameters are summarized in Table 5. In particular, the threshold value $\Phi_p = 9.9$ of the φ-failure detector was chosen to match as closely as possible the mistake rate of the other failure detector.

In our experiments, the φ-failure detector had always a better detection time than the dynamic estimation, but it also had a higher mistake rate. Figure 7 illustrates the detection time of both failure detectors, and Table 5 shows their respective mistake rate. Again, both failure detectors generate wrong suspicions as the result of message losses.

7 Conclusion

We have presented the concept of the φ-failure detectors and described its implementation. We have analyzed the behavior of this failure detector over transcontinental Internet communication during a period of three weeks. Finally, we have compared the behavior of our failure detector with two important adaptive failure detectors; Chen's estimation [5] and Bertier's dynamic estimation [2].

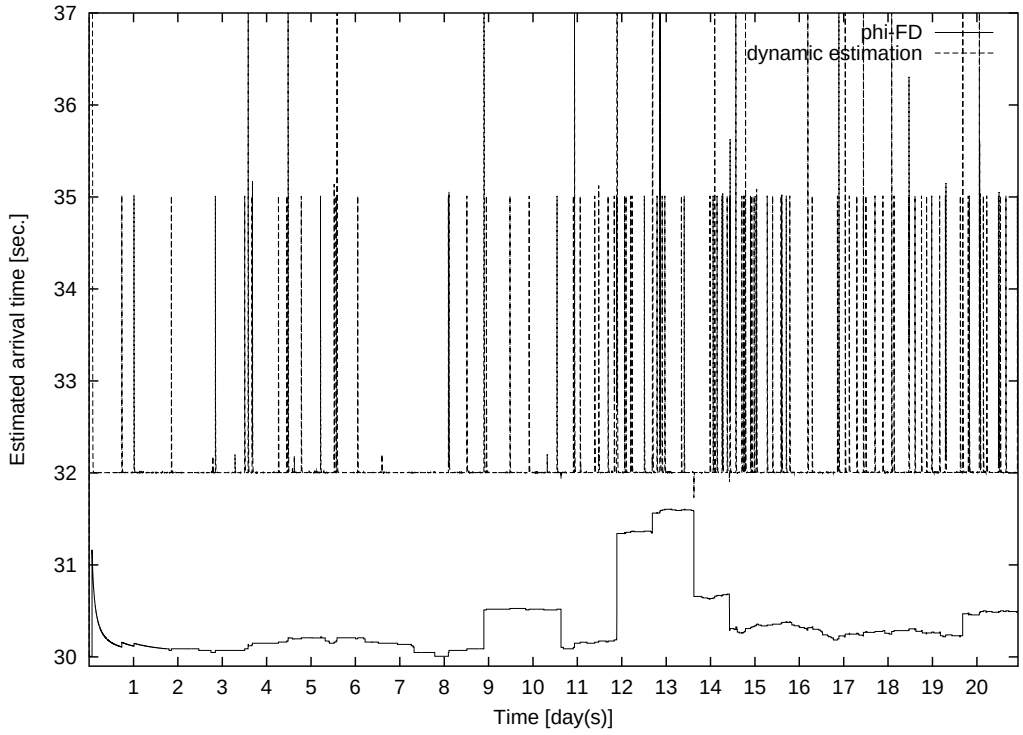

Figure 7. Detection time: φ-FD vs. dynamic estimation.

By design, φ-failure detectors can adapt equally well to changing network conditions, and the requirements of any number of concurrently running applications. As far as we know, this is currently the only failure detector that addresses this problem, and provides the flexibility required for implementing a truly generic failure detection service. In particular, the two other failure detectors studied in this paper completely fail to address that problem.

In addition to interesting observations about transcontinental network communication, our experimental results show that our failure detector behave reasonably well if parameters are well-tuned. In particular, we see that the impact of the window size is significant. Our comparisons with other failure detectors show that the performance of the φ-failure detector are in the same order as that of Chen's and Bertier's estimations, while providing nearly limitless flexibility. Nevertheless, we believe that there is still room for improvement; Chen's estimation yields a slightly better detection time for the same mistake rate, and Bertier's dynamic estimation has a slower detection time but a lower mistake rate. The performance of the three failure detectors remain comparable, and hence we conclude that all three approaches are equally realistic with respect to their quality of service.

As we have observed, message losses account for a very significant number of wrong suspicions. In particular, with a well-tuned failure detector,[7] nearly all wrong suspicions come as the result of message losses and temporary network partitions. This means that (1) there is not much point in fine-tuning the failure detectors beyond a certain point, and (2) the failure detectors cannot meet the requirements of applications with need for very high-accuracy. The only way to address this problem is to somehow reduce the effect that message losses have on wrong suspicions. We believe that it is an important issue because it limits the flexibility of the failure detector. We are currently working on a way to make our φ-failure detectors more "loss-resistant."

[7]Note that this observation is true for all three failure detector implementations studied in this paper, since they are based on similar assumptions.

References

[1] A. Basu, B. Charron-Bost, and S. Toueg. Solving problems in the presence of process crashes and lossy links. Technical Report TR96-1609, Cornell University, USA, Sep. 1996.

[2] M. Bertier, O. Marin, and P. Sens. Implementation and performance evaluation of an adaptable failure detector. In *Proc. of the 15th Int'l Conf. on Dependable Systems and Networks (DSN'02)*, pages 354–363, Washington, D.C., USA, June 2002.

[3] T. D. Chandra, V. Hadzilacos, and S. Toueg. The weakest failure detector for solving consensus. *J. ACM*, 43(4):685–722, July 1996.

[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. *IEEE Transactions on Computers*, 51(5):561–580, May 2002.

[6] F. Cosquer, L. Rodrigues, and P. Verissimo. Using tailored failure suspectors to support distributed cooperative applications. In *Proc. 7th IASTED/ISMM Intl. Conf. on Parallel and Distributed Computing and Systems*, pages 352–356, Washington, D.C., USA, Oct. 1995.

[7] X. Défago, P. Felber, and A. Schiper. Optimization techniques for replicating CORBA objects. In *Proc. of the 4th IEEE Int'l Workshop on Object-oriented Real-time Dependable Systems(WORDS'99)*, pages 2–8, Santa Barbara, CA, USA, Jan. 1999.

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

[9] C. Dwork, N. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, Apr. 1988.

[10] 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)*, pages 132–141, Edinburgh, Scotland, Sept. 1999.

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

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

[13] 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)*, pages 404–409, Osaka, Japan, Oct. 2002.

[14] N. Hayashibara, X. Défago, and T. Katayama. Two-ways adaptive failure detection with the φ-failure detector. In *Proc. Intl. Workshop on Adaptive Distributed Systems*, Sorrento, Italy, Oct. 2003. *to appear*.

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

[16] I. Sotoma and E. R. M. Madeira. Adaptation - algorithms to adaptive fault monitoring and their implementation on CORBA. In *Proc. of the Third Int'l Symp. on Distributed-Objects and Applications (DOA'01)*, pages 219–228, Rome, Italy, Sept. 2001.

[17] 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*, pages 268–278, July 1998.

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