The φ Accrual Failure Detector

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

[1]School of Information Science, Japan Advanced Institute of Science and Technology
[2]PRESTO, Japan Science and Technology Agency (JST)

May 10, 2004
IS-RR-2004-010

The φ Accrual Failure Detector

Naohiro Hayashibara*, Xavier Défago*† (contact), Rami Yared* 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,r-yared,katayama}@jaist.ac.jp

Abstract

Detecting failures is a fundamental issue for fault-tolerance in distributed systems. Recently, many people have come to realize that failure detection ought to be provided as some form of generic service, similar to IP address lookup or time synchronization. However, this has not been successful so far. One of the reasons is the difficulty to satisfy several application requirements simultaneously when using classical failure detectors.

We present a novel abstraction, called accrual failure detectors, that emphasizes flexibility and expressiveness and can serve as a basic building block to implementing failure detectors in distributed systems. Instead of providing information of a boolean nature (trust vs. suspect), accrual failure detectors output a suspicion level on a continuous scale. The principal merit of this approach is that it favors a nearly complete decoupling between application requirements and the monitoring of the environment.

In this paper, we describe an implementation of such an accrual failure detector, that we call the φ failure detector. The particularity of the φ failure detector is that it dynamically adjusts to current network conditions the scale on which the suspicion level is expressed. We analyzed the behavior of our φ failure detector over an intercontinental communication link during several days. Our experimental results show that our φ failure detector performs equally well as other known adaptive failure detection mechanisms, with an improved flexibility.

I. Introduction

It is well-known that failure detection constitutes a fundamental building block for ensuring fault tolerance in distributed systems. For this reason, many people have been advocating that failure detection should be provided as a service [1]–[5], similar to IP address lookup (DNS) or time synchronization (e.g., NTP). Unfortunately, in spite of important technical breakthroughs, this view has met little success so far. We believe that one of the main reasons is that the conventional boolean interaction (i.e., trust vs. suspect) makes it difficult to meet the requirements of several distributed applications running simultaneously. For this reason, we advocate a different abstraction that helps decoupling application requirements from issues related to the underlying system.

It is well-known that there exists an inherent tradeoff between (1) *conservative* failure detection (i.e., reducing the risk of wrongly suspecting a running process), and (2) *aggressive* failure detection (i.e., quickly detecting the occurrence of a real crash). There exists a continuum of valid choices between these two extremes, and what defines an appropriate choice is strongly related to application requirements.

One of the major obstacles to building a failure detection service is that simultaneously running distributed applications with different quality-of-service requirements must be able to tune the service to meet their own needs without interfering with each other. Furthermore, some classes of distributed applications require the use of different qualities of service 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.

Accrual failure detectors: Failure detectors are traditionally based on a boolean interaction model wherein processes can only either trust or suspect the processes that they are monitoring. In contrast, we propose a novel abstraction, called accrual failure detector, whereby a failure monitor service outputs a value on a *continuous scale* rather than information of a boolean nature. Roughly speaking, this value

captures the degree of confidence that a corresponding monitored process has crashed. If the process actually crashes, the value is guaranteed to *accrue* over time and tend toward infinity, hence the name. It is then left to application processes to set an appropriate suspicion threshold according to their own quality-of-service requirements. A low threshold is prone to generate many wrong suspicions but ensures a quick detection in the event of a real crash. Conversely, a high threshold generates fewer mistakes but needs more time to detect actual crashes.

Example: Let us now illustrate the advantage of this approach with a simple example. Consider a distributed application in which one of the processes is designated as a master while all other processes play the role of workers. The master holds a list of jobs that needs to be computed and maintains a list of available worker processes. As long as jobs remain in its list, the master sends jobs to idle workers and collects the results after they have been computed. Assume now that some of the workers might crash (for simplicity, we assume that the master never crashes). Let some worker process p_w crash during the execution; the master must be able to detect that p_w has crashed and take appropriate actions, otherwise the system might block forever. With accrual failure detectors, this could be realized as follows. When the confidence level reaches some low threshold, the master simply flags the worker process p_w and temporarily stops sending new jobs to p_w. Then, when reaching a moderate threshold, the master cancels all unfinished computations that were running on p_w and resubmit them to some other worker processes. Finally, when reaching a high threshold, the confidence that p_w has crashed is high, so the master removes p_w from its list of available workers and releases all corresponding resources. Using conventional failure detectors to implement such a simple behavior would be quite a challenge.

Contribution: In this paper, we present the abstraction of accrual failure detectors and describe an adaptive implementation called the φ failure detector. Briefly speaking, the φ failure detector 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, similarly to conventional adaptive failure detectors [9], [10]. 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.

We have evaluated our failure detection scheme on a transcontinental link between Japan and Switzerland. Heartbeat messages were sent using the user datagram protocol (UDP) at a rate of about ten per second. The experiment ran uninterruptedly for a period of one week, gathering a total of nearly 6 million samples. Using these samples, we have analyzed the behavior of the φ failure detector, and compared it with traditional adaptive failure detectors [9], [10]. By providing exactly the same input to every failure detector, we could ensure the fairness of the comparison. The results show that the enhanced flexibility provided by our approach does not induce any significant overhead.

Structure: The rest of the paper is organized as follows. Section II recalls important concepts and definitions regarding failure detectors. Section III describes the abstraction of accrual failure detectors. Section IV presents an implementation of accrual failure detectors called the φ failure detector. The behavior of the φ failure detector is evaluated in Section V, where it is compared with other existing failure detector implementations on a wide-area network. Section VI discusses other related work. Finally, Section VII concludes the paper.

II. Failure Detectors: basic concepts & implementations

This section briefly reviews important results concerning failure detection. We first outline the basic concepts, describe important metrics, and discuss basic aspects of their implementations. At the end of the section, we describe two prior implementations of adaptive failure detectors that we later use as a reference to compare with our φ failure detector.

A. 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, such as Consensus, cannot be solved deterministically in asynchronous[1] systems if even a single process might crash [11]. 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 [12]. 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 $\Diamond\mathcal{P}$ (eventually perfect), which is sufficient to solve the Consensus problem:

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

Property 2 (Eventual strong accuracy): There is a time after which correct processes are not suspected by any correct process.

B. Quality of service of failure detectors

Chen et al. [10] propose a set of metrics to evaluate the quality of service (QoS) of failure detectors. For simplicity and without loss of generality, they consider a simple system as follows. The system consist of only two processes called p and q, where process q monitors process p. Process p can possibly be subject to crash failures, in which case the crash is permanent. In the sequel, we consider the same system, and use the following subset of Chen's metrics.

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 (Average mistake rate λ_M): This measures the rate at which a failure detector generates wrong suspicions.

Notice that the first definition relates to the completeness whereas the other one relates to the accuracy of the failure detector.

C. Heartbeat failure detectors

In this section, we present a brief overview of heartbeat-based implementations of failure detectors. Assume that processes have also access to some local physical clock giving them the ability to measure time. These clocks may or may not be synchronized.

Using heartbeat messages is a common approach to implementing failure detectors. It works as follows (see Fig. 1): 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 heartbeat message from p for a period of time determined by a timeout Δ_{to}, with $\Delta_{to} \geq \Delta_i$. A third value of importance is the network transmission delay of messages. For convenience, we denote by Δ_{tr} the average transmission time experienced by messages.

In the conventional implementation of heartbeat-based failure detection protocols, the timeout Δ_{to} is fixed as a constant value. Upon receiving a heartbeat, process q waits for the next heartbeat for at most Δ_{to} units of time, after which it begins to suspect process p if no new heartbeat has been received.

Obviously, the choice of a timeout value must be larger than Δ_i, and is dictated by 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 detection time.

An alternative implementation of heartbeat failure detectors sets a timeout based on the transmission time of the heartbeat. The advantage of this approach is that the maximal detection time is bounded, but

[1] An asynchronous distributed system is one in which there are no bounds on communication delays and on the speed of processes.

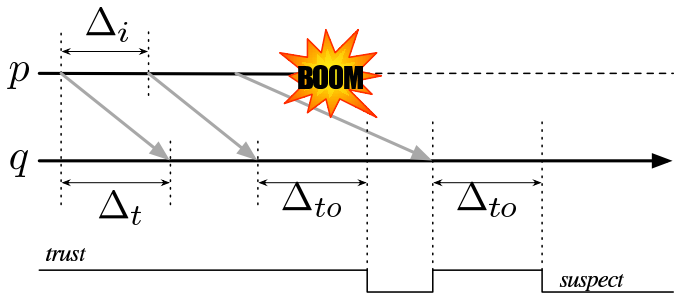

Fig. 1. Heartbeat failure detection and its main parameters.

its drawback is that it relies on clocks with negligible drift[2] and a shared knowledge of the heartbeat interval Δ_i. This last point can become a problem in practice, when the regularity of the sending of heartbeats cannot be ensured and a short interval makes the timing inaccuracies due to operating system scheduling take more importance (i.e., the actual interval differs from the target one as a result).

Each of the two approaches has its own merits that depend on the context, so we believe that there is no clearcut answer to the question of choosing one over the other.

D. Adaptive failure detectors

The goal of adaptive failure detectors is to adapt to changing network conditions. Most adaptive failure detectors presented in the literature are based on a heartbeat strategy (although nothing seems to preclude a query-response interaction style, for instance). The principal difference with using a fixed heartbeat strategy is that the timeout is modified dynamically according to network conditions.

1) Chen-FD: Chen et al. [10] 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 Δ_{to} is set according to this estimation and a constant safety margin α is added. The estimation of the next heartbeat arrival time is recomputed after each new heartbeat arrival. The safety margin is computed once, based on quality-of-service requirements. The authors propose two versions of their protocol; one that relies on synchronized clocks, and a second one that uses unsynchronized clocks with negligible drift. We have done our comparisons based on the second version of their protocol.

2) Bertier-FD: Bertier et al. [9] propose an adaptive failure detector based on the same approach, but using a different estimation function. Their estimation combines Chen's estimation with a dynamic estimation based on Jacobson's estimation of the round-trip time [13]. The resulting failure detector provides a shorter detection time, but generates more wrong suspicions than Chen's estimation, according to their measurements on a LAN.

3) Note on setting the heartbeat period: It is clear that the heartbeat period Δ_i is a factor that contributes to the detection time. However, in contrast to a common belief, Müller [14] shows that, on several different networks, Δ_i is not much determined by quality-of-service requirements, but rather by the characteristics of the underlying system.

An informal argument is as follows. Roughly speaking, the detection time is equally determined by three parameters: Δ_i, Δ_{tr}, and some additional margin α (with $\Delta_{to} \approx \Delta_i + \alpha$). Δ_{tr} is caused by the network and cannot really be tuned.

- On the one hand, if Δ_i is a lot smaller than Δ_{tr}, then reducing it will have little effect on reducing the detection time. Indeed, the detection time cannot possibly be shorter than the transmission time. In fact, reducing Δ_i further would generate both a larger amount of traffic on the network and a higher activity in the network stacks. This could in turn increase Δ_{tr}.

[2]A straightforward implementation requires synchronized clocks. Chen et al. [10] show how to do it with unsynchronized clocks, but this still requires a negligible drift between the clocks.

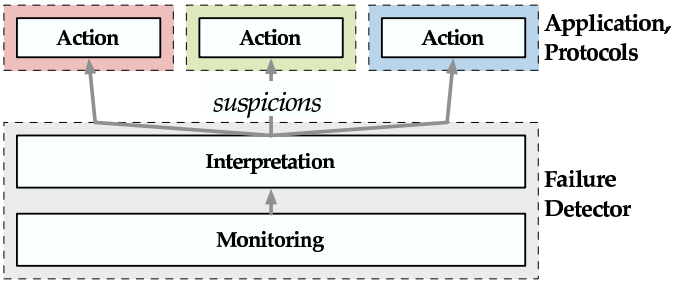

Fig. 2. Structure of traditional failure detectors. Monitoring and interpretation are combined. Interactions with applications and protocols is boolean.

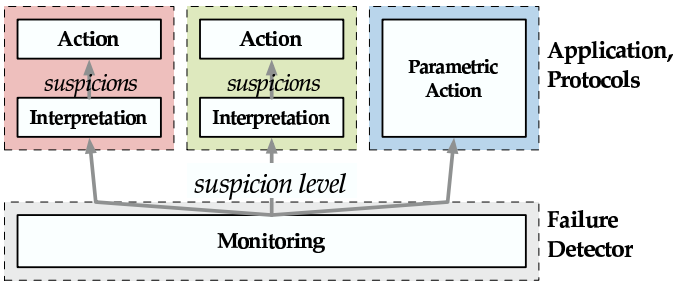

Fig. 3. Structure of accrual failure detectors. Monitoring and interpretation are decoupled. Applications interpret a common value based on their own interpretation.

- On the other hand, if Δ_i is a lot larger than Δ_{tr}, then Δ_i will almost entirely determine the detection time. Increasing it further will increase the detection time accordingly, but it will have nearly no effect in reducing the already low load on the network.

Hence, we can conclude that any reasonable value of Δ_i should be roughly equal to the average transmission time Δ_{tr}. The only exception that we could see is when an upper limit is set on the acceptable usage of network bandwidth for control messages.

Although the above argument is rather informal, it suggests that there exists, with every network, some nominal range for the parameter Δ_i with little or no impact on the accuracy of the failure detector. In other words, we can consider that the parameter Δ_i is given by the underlying system rather than computed from application requirements.

III. Accrual Failure Detectors

The principle of accrual failure detectors is simple. Instead of outputting information of a boolean nature, accrual failure detectors output suspicion information on a continuous scale. Roughly speaking, the higher the value, the higher the chance that the monitored process has crashed.

In this section, we first describe the use of accrual failure detectors from an architectural perspective, and put this in contrast with conventional failure detectors. Then, we give a more precise definition of accrual failure detectors. Finally, we conclude the section by showing the relation between accrual failure detectors and conventional ones. In particular, we show how an accrual failure detector can be used to implement a failure detector of class $\Diamond\mathcal{P}$.

A. Architecture overview

Conceptually, the implementation of failure detectors on the receiving side can be decomposed into three basic parts as follows.

1) *Monitoring*. The failure detector gathers information from other processes, usually through the network, such as heartbeat arrivals or query-response delays.
2) *Interpretation*. Monitoring information is used and interpreted, for instance to decide that a process should be suspected.
3) *Action*. Actions are executed as a response to triggered suspicions. This is normally done within applications.

The main difference between traditional failure detectors and accrual failure detectors is which component of the system does what part of failure detection.

In traditional timeout-based implementations of failure detectors, the monitoring and interpretation parts are combined within the failure detector (see Fig. 2). The output of the failure detector is of boolean nature; *trust* or *suspect*. An elapsing timeout is equated to suspecting the monitored process, that is, the

monitoring information is already being interpreted. Applications cannot do any interpretation, and thus are left with what to do with the suspicion. Unfortunately, suspicion tradeoffs largely depend on the nature of the triggered action, as well as its cost in terms of performance or resource usage.

In contrast, accrual failure detectors provide a lower-level abstraction that avoids the interpretation of monitoring information (see Fig. 3). Some value is associated with each process that represents a suspicion level. This value is then left for the applications to interpret. For instance, by setting an appropriate threshold, applications can trigger suspicions and perform appropriate actions. Alternatively, applications can directly use the value output by the accrual failure detector as a parameter to their actions. Considering the example of master/worker described in the introduction, the master could decide to allocate the most urgent jobs only to worker processes with a low suspicion level.

B. Definition

An accrual failure detector is defined as a failure detector that outputs a value associated with each of the monitored processes, instead of a set of suspected processes. In the simplified model considered in this paper (two processes p and q, where q monitors p), the output of the failure detector of q over time can be represented by the following function ("suspicion level of p").

$$susp\_level_p(t) \geq 0 \tag{1}$$

The values output by an accrual failure detector module define the function $susp\_level_p$ and must satisfy the properties below. The first two properties specify what the output of $susp\_level_p(t)$ should be if the process p is faulty, whereas the remaining two properties specify what the output should be if p is correct.

Property 3 (Asymptotic completeness): If process p is faulty, the suspicion level $susp\_level_p(t)$ tends to infinity as time goes to infinity.

Property 4 (Eventual monotony): If process p is faulty, there is a time after which $susp\_level_p(t)$ is monotonic increasing.

Property 5 (Upper bound): Process p is correct if and only if $susp\_level_p(t)$ has an upper bound over an infinite execution.

Property 6 (Reset): If process p is correct, then for any time t_0, $susp\_level_p(t) = 0$ for some time $t \geq t_0$.

C. Transformation into conventional failure detection

Given the definitions of accrual failure detectors, it easy to use them to construct existing failure detectors such as one of class $\Diamond\mathcal{P}$. The algorithm below is similar to one proposed by Fetzer et al. [15].

Consider the following transformation algorithm described for the situation where process q monitors process p. Process q maintains two dynamic thresholds T_{high} and T_{low}, initialized to the same arbitrary value greater than 0.

- *S-transition*: Whenever the value of $susp\_level_p$ crosses the upper threshold T_{high} upward, q updates the value of T_{high} to $T_{high}+1$, and begins to suspect p (or continues to suspect p if it does so already).
- *T-transition*: Whenever the value of $susp\_level_p$ crosses the lower threshold T_{low} downward, q updates the value of T_{low} to that of T_{high}, and stops suspecting p.

It is rather straightforward to prove that the above transformation satisfies the properties of $\Diamond\mathcal{P}$. Informally, strong completeness is ensured because the threshold T_{high} is always finite (consequence of Prop. 4) and must be eventually crossed (S-transition and Prop. 3). Similarly, eventually strong accuracy is ensured because, if p is correct, there is a time after which T_{high} is never crossed (S-transition and Prop. 5), and q does not suspect p (T-transition and Prop. 6).

Now, it is important to stress that the above result does not come in contradiction with the FLP impossibility of Consensus [11]. Accrual failure detectors merely define an abstraction, and are hence subject to the same restrictions as conventional failure detectors. It is well-known that it is impossible

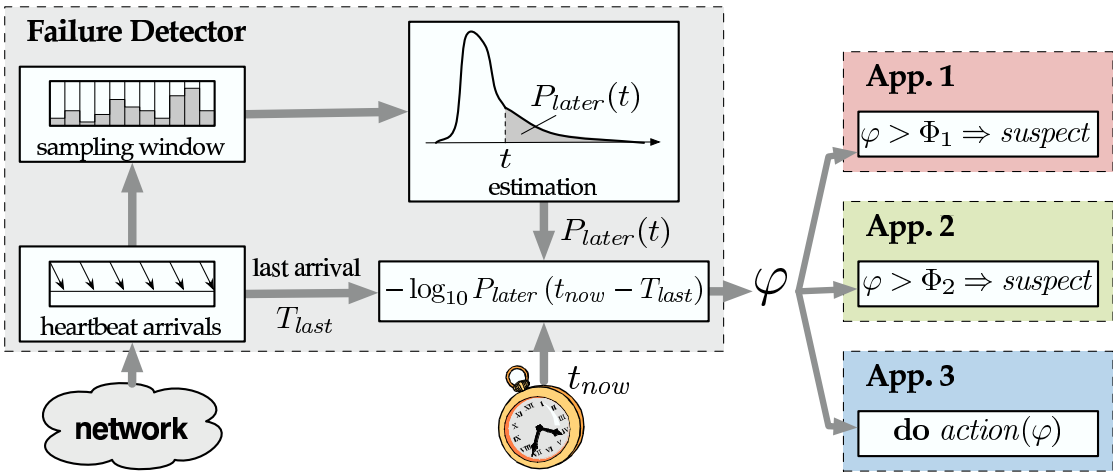

Fig. 4. Information flow in the proposed implementation of the φ failure detector, as seen at process q. Heartbeat arrivals arrive from the network and their arrival time are stored in the sampling window. Past samples are used to estimate some arrival distribution. The time of last arrival T_{last}, the current time t_{now} and the estimated distribution are used to compute the current value of φ. Applications trigger suspicions based on some threshold (Φ_1 for App. 1 and Φ_2 for App. 2), or execute some actions as a function of φ (App. 3).

to implement a failure detector of class $\Diamond\mathcal{P}$ deterministically in asynchronous systems. Likewise, accrual failure detectors cannot be implemented deterministically in all possible asynchronous systems. However, both kinds of failure detectors can be implemented *probabilistically*.

IV. Implementation of the φ accrual failure detector

In the previous section, we have presented the generic abstraction of accrual failure detectors. Accrual failure detectors can be implemented in many different ways. In this section, we present a practical implementation that we call the φ failure detector, and that we had outlined in earlier work [16].

A. Meaning of the value φ

As mentioned, φ failure detector implements the abstraction of an accrual failure detector. The suspicion level of accrual failure detector is given by a value called φ. The basic idea of the φ failure detector is to express the value of φ on a scale that is dynamically adjusted to reflect current network conditions.

Let T_{last}, t_{now}, and $P_{later}(t)$ denote respectively: the time when the most recent heartbeat was received (T_{last}), the current time (t_{now}), and the probability that a heartbeat will arrive more than t time units after the previous one ($P_{later}(t)$). Then, the value of φ is calculated as follows.

$$\varphi(t_{now}) \stackrel{\text{def}}{=} -\log_{10}(P_{later}(t_{now} - T_{last})) \tag{2}$$

Roughly speaking, with the above formula, φ takes the following meaning. Given some threshold Φ, and assuming that we decide to suspect p when $\varphi \geq \Phi = 1$, then the likeliness that we will make a mistake (i.e., the decision will be contradicted in the future by the reception of a late heartbeat) is about $10\,\%$. The likeliness is about $1\,\%$ with $\Phi = 2$, $0.1\,\%$ with $\Phi = 3$, and so on.

B. Calculating φ

The method used for estimating φ is in fact rather simple. This is done in three phases. First, heartbeat arrive and their arrival times are stored in a sampling window. Second, these past samples are used to determine the distribution of inter-arrival times. Third, the distribution is in turn used to compute the current value of φ. The overall mechanism is described in Figure 4.

1) Sampling heartbeat arrivals: The monitored process (p in our model) adds a sequence number to each heartbeat message. The monitoring process (q in our model) stores heartbeat arrival times into a sampling window of fixed size WS. Whenever a new heartbeat arrives, its arrival time is stored into the window, and the data regarding the oldest heartbeat is deleted from the window. Arrival intervals are easily computed. In addition, to constantly determine the mean μ and the variance σ^2, two other variables are used to keep track of the sum and sum of squares of all samples in the window.

2) Estimating the distribution and computing φ: The estimation of the distribution of inter-arrival times assumes that inter-arrivals follow a normal distribution. The parameter of the distribution are estimated from the sampling window, by determining the mean μ and the variance σ^2 of the samples. Then, the probability $P_{later}(t)$ that a given heartbeat will arrive more than t time units later than the previous heartbeat is given by the following formula.[3]

$$P_{later}(t) = \frac{1}{\sigma\sqrt{2\pi}} \int_{t}^{+\infty} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \mathrm{d}x \quad (3)$$

$$= 1 - F(t) \quad (4)$$

where $F(t)$ is the cumulative distribution function of a normal distribution with mean μ and variance σ^2.

Then, the value of φ at time t_{now} is computed by applying Equation 2 described in Section IV-A.

V. Experimental Results

In this section, we study the behavior of the φ failure detector when used over a wide-area network. The measurements have been taken in a rather extreme environment (wide area network, short heartbeat interval) to assess both the robustness and the scope of applicability of the failure detector.

First, we describe the environment in which the experiments have been conducted. Second, we study the effect of several parameters on the behavior of the φ failure detector. Third, we compare the results obtained using the φ failure detector with that of Chen and Bertier (see §II-D).

A. Environment

Our experiments involved two computers, with one located in Japan and the other located in Switzerland. The two computers were communicating through a normal intercontinental Internet connection. One machine was running program sending heartbeats (thus acting like process p) while the other one was recording the arrival times of each heartbeat (thus acting like process q). Neither machine failed during the experiment.

1) Hardware/software/network:

- *Computer p (monitored; Switzerland):* 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).
- *Computer q (monitoring; Japan):* The receiving host was located in Japan, at the Japan Advanced Institute of Science and Technology (JAIST). The machine was equipped with a Pentium III processor at 1 GHz and 512 MB of memory. The running operating system was Red Hat Linux 9 (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.

[3]The formula is simplified assuming that crashes are rare events.

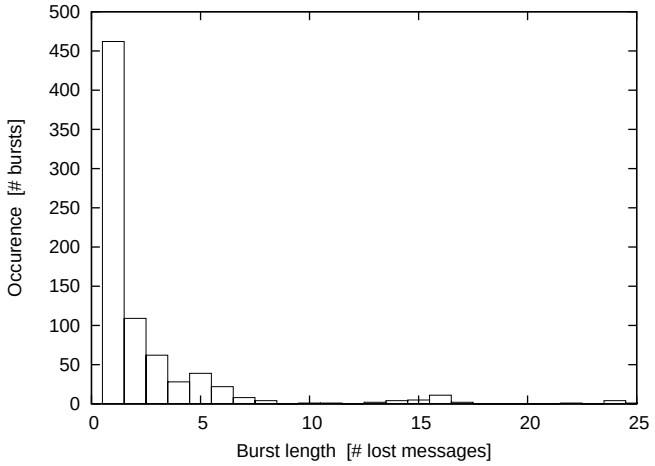

Fig. 5. Distribution of the length of loss bursts. A burst is defined by the number of consecutive messages that were lost.

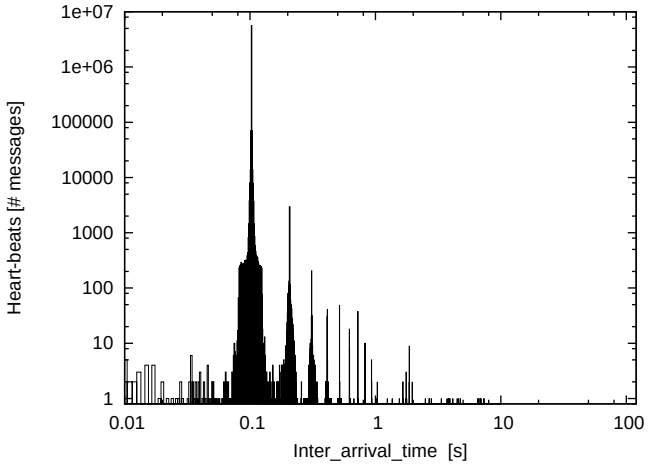

Fig. 6. Distribution of heartbeat inter-arrival times as measured by the receiving host. Horizontal and vertical scales are both logarithmic.

In addition, we have monitored the CPU load average on the two machines during the whole period of the experiments. We observed that the load was nearly constant throughout, and well below the full capacity of the machines.

2) Heartbeat sampling: The experiment started on April 2, 2004 at 17:56 UTC, and finished exactly one full week later. During the one week that the experiment lasted, heartbeat messages were generated at a target rate of one heartbeat every 100 ms. The average sending rate actually measured was of one heartbeat every 103.5 ms (standard deviation: 0.19 ms; min.: 101.7 ms; max.: 234.3 ms). In total, $5,845,712$ heartbeat messages were sent among which only $5,822,521$ were received (about $0.4\,\%$ of message loss).

We observed that message losses tended to occur in bursts, the longest of which was 1093 heartbeats long (i.e., it lasted for about 2 minutes). We observed 814 different bursts of consecutively lost messages. The distribution of burst lengths is represented on Figure 5. Beyond 25, there is a flat tail of 48 bursts that are not depicted on the figure. After 25, the next burst is 34 heartbeats long, and the lengths of the five longest bursts were respectively 495, 503, 621, 819, and 1093 heartbeats.

The mean of inter-arrival times of received heartbeats was 103.9 ms with a standard deviation of about 104.1 ms. The distribution of the inter-arrival times is represented on Figure 6.

A different view of inter-arrival times is given in Figure 7. The figure relates arrival intervals (vertical axis) with the time when the second heartbeat of the interval arrived (horizontal), over the whole duration of the experiment. Very long intervals are not depicted. The first (thick) line of points at the bottom of the graph represents heartbeat that arrived normally within about 100 ms. The second (thinner) line represents intervals obtained after a single heartbeat was lost, and so on with the other lines above it. At that frequency, losing a single heartbeat seems to be a normal situation. There is a period (April 6 and 7) where more messages were lost.

3) Round-trip times: During the experiment, we have also measured the round-trip time (RTT), albeit at a low rate. We have measured an average RTT of 283.3 ms with a standard deviation of 27.3 ms, a minimum of 270.2 ms, and a maximum of 717.8 ms.

4) Experiment: To conduct the experiments, we have recorded heartbeat sending and arrival times using the experimental setup described above. We have used the sending times to compute the statistics mentioned above. Then, we replayed the receiving times recorded for each different failure detector implementation and every different value of the parameters. As a result, the failure detectors were compared based on *exactly* the same scenarios, thus resulting in a fair comparison.

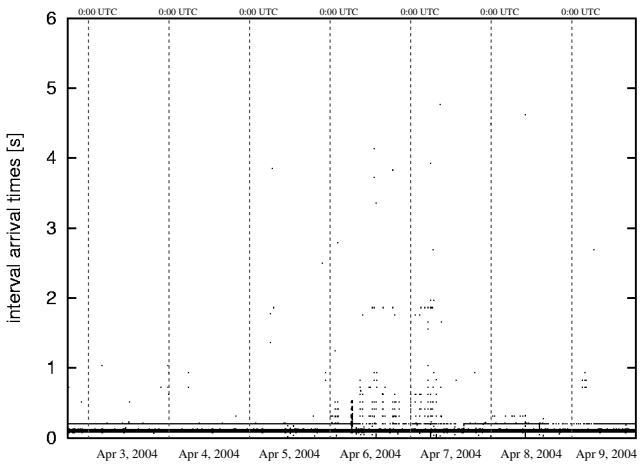

Fig. 7. Arrival intervals and time of occurrence. Each dot represents a received heartbeat. The horizontal position denotes the time of arrival. The vertical coordinate denotes the time elapsed since the reception of the previous heartbeat.

All three failure detectors considered in these experiments rely on a window of past samples to compute their estimations. Unless stated otherwise, the failure detectors were set using the same window size of $1,000$ samples. As the behavior of the failure detectors is stable only after the window is full, we have excluded from the analysis all data obtained during the warmup period—i.e., the period before the window is full.

B. Experiment 1: average mistake rate

In the first experiment, we have measured the average mistake rate λ_M obtained with the φ failure detector. In particular, we have measured the evolution of the mistake rate when the threshold Φ, used to trigger suspicions, increases.

Figure 8 shows the results obtained when plotting the mistake rate on a logarithmic scale. The figure shows a clear improvement in the mistake rate when the threshold increases from $\Phi = 0.5$ to $\Phi = 2$. This improvement is due to the fact that most late heartbeat messages are caught by a threshold of two or more. The second significant improvement comes when $\Phi \in [8; 12]$. This corresponds to the large number of individually lost heartbeat messages (i.e., loss bursts of length 1). As those messages no longer contribute to generating suspicions, the mistake rate drops significantly.

C. Experiment 2: average detection time

In the second experiment, we have measured the average detection time obtained with the φ failure detector, and how it evolves when changing the threshold Φ.

We have computed an *estimation* for the average detection time T_D as follows. Assuming that a crash would occur exactly after successfully sending a heartbeat,[4] we measure the time elapsed until the failure detector reports a suspicion. With the φ failure detector, we consider the threshold Φ and reverse the computation of φ to obtain the equivalent timeout. We compute this equivalent timeout each time a new heartbeat is received and take the mean value $\Delta_{to,\Phi}$. We estimated the mean propagation time Δ_{tr} based

[4]This is a worst case situation because any crash that would occur later (but before sending the next heartbeat) would be detected at the same time, and any crash that would occur earlier would actually prevent the last heartbeat from being sent. Either case would result in a shorter detection time.

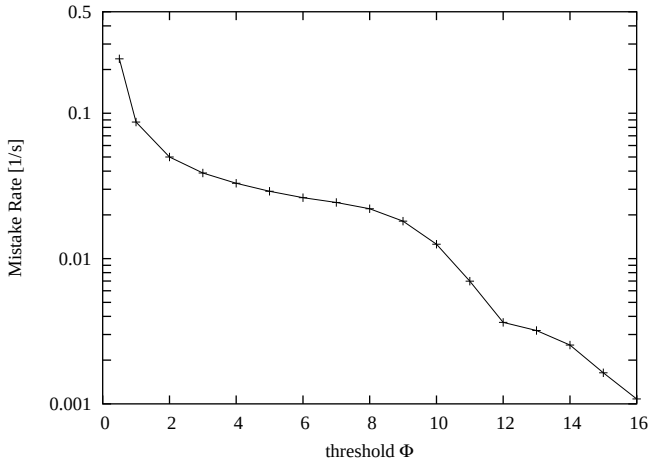

Fig. 8. Exp. 1: average mistake rate as a function of threshold Φ. Vertical axis is logarithmic.

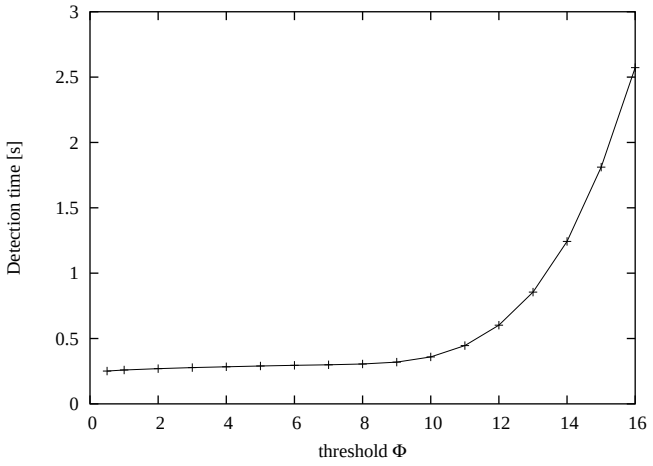

Fig. 9. Exp 2: average detection time as a function of threshold Φ.

on our measurements of the round-trip time. Then, we have estimated the average (worst-case) detection time simply as follows.

$$T_D \approx \Delta_{tr} + \Delta_{to,\Phi} \tag{5}$$

Figure 9 depicts the evolution of the detection time as the suspicion threshold Φ increases. The curve shows a sharp increase in the average detection time for threshold values beyond 10 or 11.

D. Experiment 3: effect of window size

The third experiment measures the effect of the window size on the mistake rate of the φ-failure detector. We have set the window size from very small (20 samples) to very large ($10,000$ samples) and measured the accuracy obtained by the failure detector when run during the full week of the experiment. We have repeated the experiment for three different values of the threshold Φ, namely $\Phi = 1$, $\Phi = 3$, and $\Phi = 5$. Figure 10 shows the results, with both axes expressed on a logarithmic scale.

The experiment confirms that the mistake rate of the φ failure detector improves as the window size increases (see Fig. 10). The curve seems to flatten slightly for large values of the window size, suggesting that increasing it further yields only little improvement. A second observation is that the φ failure detector seems to be affected equally by the window size, regardless of the threshold.

E. Experiment 4: comparison with Chen-FD and Bertier-FD

In this fourth experiment, we compare the φ-failure detector with two well-known adaptive failure detectors, namely the failure detector of Chen et al. [10] and that of Bertier et al. [9]. The goal of the comparison is to show that the additional flexibility offered by the φ failure detector does not incur any significant performance cost.

The three failure detectors do not share any common tuning parameter, which makes comparing them difficult. To overcome this problem, we measured the behavior of each of the three failure detectors using several values of their respective tuning parameters. We have then plotted the combinations of QoS metrics (average mistake rate, average worst-case detection time) obtained with each of the three failure detectors.

The tuning parameter for the φ failure detector was the threshold Φ (values are also represented in Fig. 8 and 9). The tuning parameter for Chen's failure detector was the safety margin α; this is simply an additional period of time that is added to the estimate for the arrival of the next heartbeat. Unlike the other two failure detectors, Bertier's has no tuning parameter. For this reason, its behavior is plotted as a

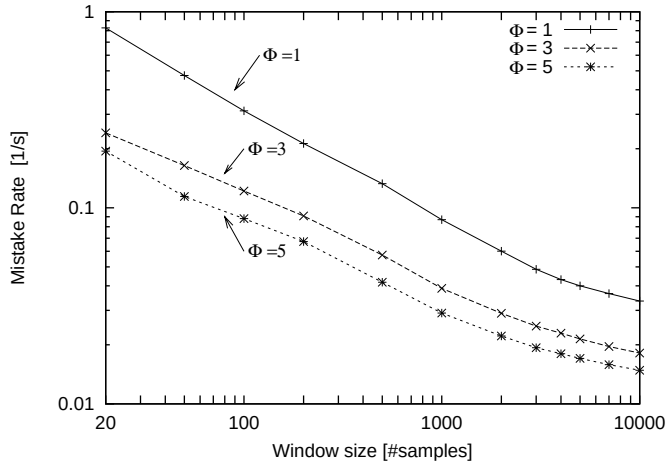

Fig. 10. Exp. 3: average mistake rate as a function of the window size, and for different values of the threshold Φ. Horizontal and vertical axes are both logarithmic.

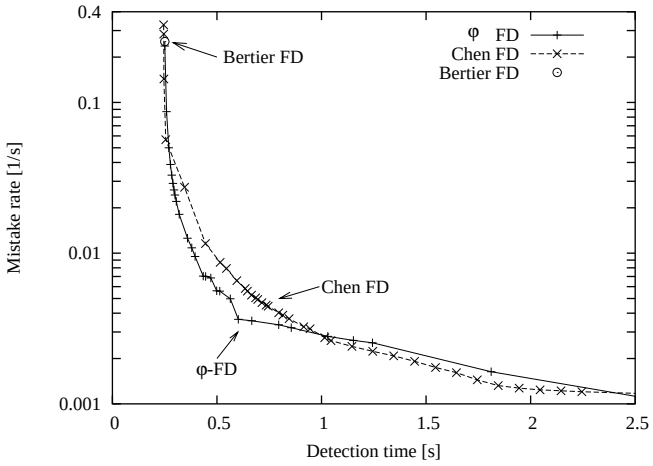

Fig. 11. Exp. 4: comparison of failure detectors. Mistake rate and detection time obtained with different values of the respective parameters. Most desirable values are toward the lower left corner. Vertical axis is logarithmic.

single point on the graph. Finally, as already mentioned, the window size for all three failure detectors was set to the same value of $1,000$ samples.

The results of the experiment are depicted on Figure 11. The vertical axis, representing the mistake rate, is expressed on a logarithmic scale. The horizontal axis, representing the estimated average detection time, is on a linear scale. Best values are located toward the lower left corner because this means that the failure detector provides a short detection time while keeping a low mistake rate.

The results show clearly that the φ failure detector does not incur any significant performance cost. When compared with Chen's failure detector, both failure detectors follow the same general tendency. In our experiment, the φ failure detector behaves a little better in the aggressive range of failure detection, whereas Chen's failure detector behaves a little better in the conservative range.

Quite interestingly, Bertier's failure detector did not perform very well in our experiments. By looking at the trace files more closely, we observed that this failure detector was more sensitive than the other two (1) to message losses, and (2) to large fluctuations in the receiving time of heartbeats. It is however important to note that, according to their authors [9], Bertier's failure detector was primarily designed to be used over *local* area networks (LANs), that is, environments wherein messages are seldom lost. In contrast, these experiments were done over a wide-area network.

Putting too much emphasis on the difference between Chen and φ would not be reasonable as other environments might yield to other conclusions. It is however safe to conclude that the flexibility of φ does not come with any drop in performance, especially when used over wide-area networks.

VI. Related Work

A. Other adaptive failure detectors

There exists other adaptive failure detectors in addition to Chen's and Bertier's described in Section II-D.

Fetzer et al. [15] have proposed a protocol using 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 [17], 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. The proposed algorithm adapts only very slowly as this is not a focus of that paper.

Sotoma et al. [4] propose an implementation of an adaptive failure detector with CORBA. Their algorithm computes the timeout based on the average time for arrival intervals of heartbeat messages, and some ratio between arrival intervals.

B. Flexible failure detectors

As far as we know, there exists only a few failure detector implementations that allow non-trivial tailoring by applications, let alone the requirements of *several* applications running simultaneously.

Cosquer et al. [18] 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. [10] 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], [19] 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 (it supports only two applications).

C. 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 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. [20]. 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 [21] 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 accrual failure detectors.

Similarly, accrual failure detectors could also be useful as a low-level building block for implementing a partitionable group membership, such as Moshe [22]. Such a group membership must indeed distinguish between message losses, network partitions, and actual process crashes. For instance, Keidar et al. [22] decide that a network partition has occurred after more than three consecutive messages have been lost. Typically, this could be done by using accrual failure detector and setting an appropriate threshold.

VII. Conclusion

We have presented the concept and the implementation of the φ-failure detector, an instance of the more general abstraction of accrual failure detectors. We have analyzed the behavior of the φ failure detector over a transcontinental Internet link, based on nearly 6 million heartbeat messages. Finally, we have compared the behavior of our failure detector with two important adaptive failure detectors, namely, Chen's [10] and Bertier's [9] failure detectors.

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 both problems and provides the flexibility required for implementing a truly generic failure detection service. In particular, the two other failure detectors studied in this paper do not address both problems.[5]

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 the other failure detectors show that the φ-failure detector does not induce any significant overhead as performance are similar. Nevertheless, we believe that there is still room for improvement. In particular, we are investigating techniques and mechanisms to (1) improve the estimation of the distribution when computing φ, (2) reduce the use of memory resources, and (3) better cope with message losses for highly conservative failure detection.

Concerning accrual failure detectors, we are currently working on a more thorough formalization of the abstraction. Some of those results have been briefly summarized in this paper, but will be further developed in the future.

ACKNOWLEDGMENTS

We are grateful to André Schiper for allowing us to remotely use a machine in his laboratory at EPFL, so that we could conduct our experiments. We are also thankful to Michel Raynal, Richard D. Schlichting, Yoichi Shinoda, Makoto Takizawa, and Péter Urbán for their insightful comments and numerous suggestions that greatly helped us improve the quality of this paper.

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. Shnayderman, and A. Schiper, "Comparison of failure detectors and group membership: Performance study of two atomic broadcast algorithms," in *Proc. IEEE Intl. Conf. on Dependable Systems and Networks (DSN'03)*, San Francisco, CA, USA, June 2003, pp. 645–654.

[9] 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.

[10] 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.

[11] 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.

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

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

[14] M. Müller, "Performance evaluation of a failure detector using SNMP," Master's thesis, École Polytechnique Fédérale de Lausanne, Switzerland, Feb. 2004. [Online]. Available: http://lsewww.epfl.ch/Publications/ById/366.html

[5] These two failure detectors were aimed at different problems, that they both solve admirably well.

[15] 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.
[16] N. Hayashibara, X. Défago, and T. Katayama, "Two-ways adaptive failure detection with the φ-failure detector," Printouts distributed to participants, pp. 22–27, Oct. 2003, presented at Workshop on Adaptive Distributed Systems, as part of the 17th Intl. Conf. on Distributed Computing (DISC'03).
[17] C. Dwork, N. Lynch, and L. Stockmeyer, "Consensus in the presence of partial synchrony," vol. 35, no. 2, pp. 288–323, Apr. 1988.
[18] 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*, Washington, D.C., USA, Oct. 1995, pp. 352–356.
[19] X. Défago, P. Felber, and A. Schiper, "Optimization techniques for replicating CORBA objects," in *Proc. 4th IEEE Intl. Workshop on Object-oriented Real-time Dependable Systems(WORDS'99)*, Santa Barbara, CA, USA, Jan. 1999, pp. 2–8.
[20] G. Chockler, I. Keidar, and R. Vitenberg, "Group communication specifications: A comprehensive study," vol. 33, no. 4, pp. 427–469, May 2001.
[21] 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.
[22] 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.