On the Quality of Service of Failure Detectors *

Wei Chen
Oracle Corporation
One Oracle Drive, Nashua, NH 03062, USA
weichen@us.oracle.com

Sam Toueg
DIX Departement d'Informatique
Ecole Polytechnique
91128 Palaiseau Cedex, France
sam@dix.polytechnique.fr

Marcos Kawazoe Aguilera
Department of Computer Science
Cornell University
Ithaca, NY 14853-7501, USA
aguilera@cs.cornell.edu

Abstract

We study the quality of service (QoS) *of failure detectors. By QoS, we mean a specification that quantifies (a) how fast the failure detector detects actual failures, and (b) how well it avoids false detections. We first propose a set of QoS metrics to specify failure detectors for systems with probabilistic behaviors, i.e., for systems where message delays and message losses follow some probability distributions. We then give a new failure detector algorithm and analyze its QoS in terms of the proposed metrics. We show that, among a large class of failure detectors, the new algorithm is optimal with respect to some of these QoS metrics. Given a set of failure detector QoS requirements, we show how to compute the parameters of our algorithm so that it satisfies these requirements, and we show how this can be done even if the probabilistic behavior of the system is not known. Finally, we briefly explain how to make our failure detector* adaptive, *so that it automatically reconfigures itself when there is a change in the probabilistic behavior of the network.*

1. Introduction

Fault-tolerant distributed systems are designed to provide reliable and continuous service despite the failures of some of their components. A basic building block of such systems is the *failure detector*. Failure detectors are used in a wide variety of settings, such as network communication protocols [8], computer cluster management [18], group membership protocols [5, 7, 21, 17, 16], etc.

Roughly speaking, a failure detector provides some information on which processes have crashed. This information, typically given in the form of a list of *suspects*, is not always up-to-date or correct: a failure detector may take a long time to start suspecting a process that has crashed, and it may erroneously suspect a process that has not crashed (in practice this can be due to message losses and delays).

Chandra and Toueg [9] provide the first formal specification of *unreliable failure detectors* and show that they can be used to solve some fundamental problems in distributed computing, namely, *consensus* and *atomic broadcast*. This approach was later used and generalized in other works, e.g., [15, 13, 1, 3, 2].

In all of the above works, failure detectors are specified in terms of their *eventual* behavior (e.g., a process that crashes is eventually suspected). Such specifications are appropriate for asynchronous systems, in which there is no timing assumption whatsoever.[1] Many applications, however, have some timing constraints, and for such applications, failure detectors with eventual guarantees are not sufficient. For example, a failure detector that starts suspecting a process one hour after it crashed can be used to solve asynchronous consensus, but it is useless to an application that needs to solve many instances of consensus per minute. Applications that have timing constraints require failure detectors that provide a *quality of service (QoS)* with some quantitative timeliness guarantees.

In this paper, we study the QoS of failure detectors in

*Research partially supported by NSF grant CCR-9711403 and an Olin Fellowship.

[1] Even though the *fail-aware* failure detector of [13] is implemented in the "timed asynchronous" model, its specification is for the asynchronous model.

systems where message delays and message losses follow some probability distributions. We first propose a set of metrics that can be used to specify the QoS of a failure detector; these QoS metrics quantify (a) how *fast* it detects actual failures, and (b) how *well* it avoids false detections. We then give a new failure detector algorithm and analyze its QoS in terms of the proposed metrics. We show that, among a large class of failure detectors, the new algorithm is optimal with respect to some of these QoS metrics. Given a set of failure detector QoS requirements, we show how to compute the parameters of our algorithm so that it satisfies these requirements, and we show how this can be done even if the probabilistic behavior of the system is not known. The QoS specification and the analysis of our failure detector algorithm is based on the theory of stochastic processes. To the best of our knowledge, this work is the first comprehensive and systematic study of the QoS of failure detectors using probability theory.

1.1. On the QoS Specification of Failure Detectors

We consider message-passing distributed systems in which processes may fail by crashing, and messages may be delayed or dropped by communication links.[2] A failure detector can be *slow*, i.e., it may take a long time to suspect a process that has crashed, and it can make *mistakes*, i.e., it may erroneously suspect some processes that are actually up (such a mistake is not necessarily permanent: the failure detector may later stop suspecting this process). To be useful, a failure detector has to be reasonably fast and accurate.

In this paper, we propose a set of metrics for the QoS specification of failure detectors. In general, these QoS metrics should be able to describe the failure detector's *speed* (how fast it detects crashes) and its *accuracy* (how well it avoids mistakes). Note that speed is with respect to processes that crash, while accuracy is with respect to processes that do not crash.

A failure detector's speed is easy to measure: this is simply the time that elapses from the moment when a process p crashes to the time when the failure detector starts suspecting p permanently. This QoS metric, called *detection time*, is illustrated in Fig. 1.

How do we measure a failure detector's accuracy? It turns out that determining a good set of accuracy metrics is a delicate task. To illustrate some of the subtleties involved, consider a system of two processes p and q connected by a lossy communication link, and suppose that the failure detector at q monitors process p. The output of the failure detector at q is either "I suspect that p has crashed" or "I trust that p is up", and it may alternate between these two outputs from time to time. For the purpose of measuring the accuracy of the failure detector at q, suppose that p does not crash.

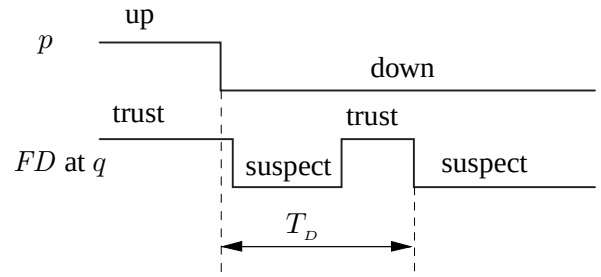

Figure 1. Detection time T_D

Consider an application that queries q's failure detector at random times. For such an application, a natural measure of accuracy is the probability that, *when queried at a random time*, the failure detector at q indicates correctly that p is up. This QoS metric is the *query accuracy probability*. For example, in Fig. 2, the query accuracy probability of FD_1 at q is $12/(12+4) = .75$.

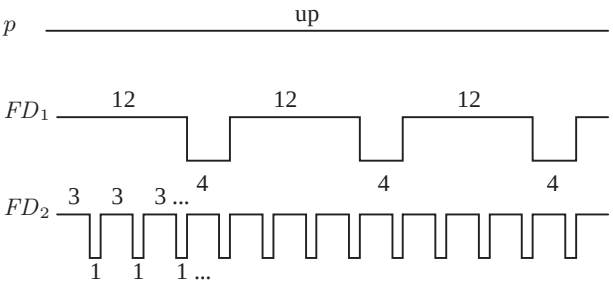

Figure 2. FD_1 and FD_2 have the same query accuracy probability of $.75$, but the mistake rate of FD_2 is four times that of FD_1

The query accuracy probability, however, is not sufficient to fully describe the accuracy of a failure detector. To see this, we show in Fig. 2 two failure detectors FD_1 and FD_2 such that (a) they have the same query accuracy probability, but (b) FD_2 makes mistakes more frequently than FD_1.[3] In some applications, every mistake causes a costly interrupt, and for such applications the *mistake rate* is an important accuracy metric.

Note, however, that the mistake rate alone is not sufficient to characterize accuracy: as shown in Fig. 3, two failure detectors can have the same mistake rate, but different query accuracy probabilities.

[2] We assume that process crashes are permanent, or, equivalently, that a process that recovers from a crash assumes a new identity.

[3] The failure detector *makes a mistake* each time its output changes from "trust" to "suspect" while p is actually up.

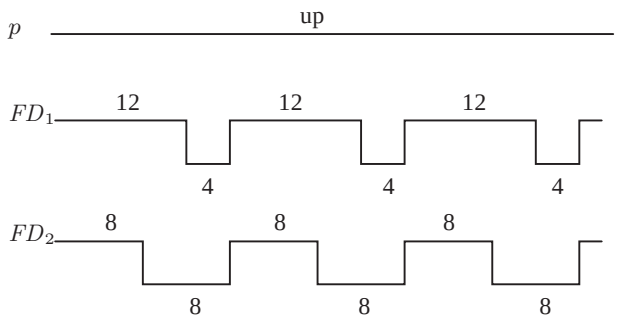

Figure 3. FD_1 and FD_2 have the same mistake rate $1/16$, but the query accuracy probabilities of FD_1 and FD_2 are $.75$ and $.50$, respectively

Even when used together, the above two accuracy metrics are still not sufficient. In fact, it is easy to find two failure detectors FD_1 and FD_2, such that (a) FD_1 is better than FD_2 in both measures (i.e., it has a higher query accuracy probability *and* a lower mistake rate), but (b) FD_2 is better than FD_1 in another respect: specifically, whenever FD_2 makes a mistake, it corrects this mistake faster than FD_1; in other words, the *mistake durations* in FD_2 are smaller than in FD_1. Having small mistake durations may be important to some applications.

As it can be seen from the above, there are several different aspects of accuracy that may be important to different applications, and each aspect has a corresponding accuracy metric.

In this paper, we identify six accuracy metrics (since the behavior of a failure detector is probabilistic, most of these metrics are random variables). We then use the theory of stochastic processes to quantify the relation between these metrics. This analysis allows us to select two accuracy metrics as the *primary* ones in the sense that: (a) they are not redundant (one cannot be derived from the other), and (b) together, they can be used to derive the other four accuracy metrics.

In summary, we show that the QoS specification of failure detectors can be given in terms of three basic metrics, namely, the detection time and the two primary accuracy metrics that we identified. Taken together, these metrics can be used to characterize and compare the QoS of failure detectors.

1.2. The Design and Analysis of a New Failure Detector Algorithm

In this paper, we consider a simple system of two processes p and q, connected through a communication link. Process p may fail by crashing, and the link between p and q may delay or drop messages. Message delays and message losses follow some probabilistic distributions. Process q has a failure detector that monitors p and outputs either "I suspect that p has crashed" or "I trust that p is up" ("suspect p" and "trust p" in short, respectively).

A Common Failure Detection Algorithm and its Drawbacks. A simple failure detection algorithm, commonly used in practice, works as follows: at regular time intervals, process p sends a heartbeat message to q; when q receives a heartbeat message, it trusts p and starts a timer with a fixed timeout value TO; if the timer expires before q receives a newer heartbeat message from p, then q starts suspecting p.

This algorithm has two undesirable characteristics; one regards its accuracy and the other its detection time, as we now explain. Consider the i-th heartbeat message m_i. Intuitively, the probability of a *premature timeout* on m_i should depend solely on m_i, and in particular on m_i's delay. With the simple algorithm, however, the probability of a premature timeout on m_i also depends on the heartbeat m_{i-1} that precedes m_i! In fact, the timer for m_i is started upon the receipt of m_{i-1}, and so if m_{i-1} is "fast", the timer for m_i starts early and this increases the probability of a premature timeout on m_i. This dependency on past heartbeats is undesirable.

To see the second problem, suppose p sends a heartbeat just before it crashes, and let d be the delay of this last heartbeat. In the simple algorithm, q would permanently suspect p only $d + TO$ time units after p crashes. Thus, the worst-case detection time for this algorithm is the *maximum* message delay plus TO. This is impractical because in many systems the maximum message delay is orders of magnitude larger than the average message delay.

The source of the above problems is that even though the heartbeats are sent at regular intervals, the timers to "catch" them expire at irregular times, namely the receipt times of the heartbeats plus a fixed TO. The algorithm that we propose eliminates this problem. As a result, the probability of a premature timeout on heartbeat m_i does *not* depend on the behavior of the heartbeats that precede m_i, and the detection time does *not* depend on the maximum message delay.

A New Algorithm and its QoS Analysis. In the new algorithm, process p sends heartbeat messages $m_1, m_2, \ldots$ to q periodically every η time units (just as in the simple algorithm). To determine whether to suspect p, q uses a sequence $\tau_1, \tau_2, \ldots$ of fixed time points, called *freshness points*, obtained by shifting the sending time of the heartbeat messages by a fixed parameter δ. More precisely, $\tau_i = \sigma_i + \delta$, where σ_i is the time when m_i is sent. For any time t, let i be so that $t \in [\tau_i, \tau_{i+1})$; then q trusts p at time t if and only if q has received heartbeat m_i or higher.

Given the probabilistic behavior of the system (i.e., the probability of message losses and the distribution of mes-

sage delays), and the parameters η and δ of the algorithm, we determine the QoS of the new algorithm using the theory of stochastic processes. Simulation results given in [10] are consistent with our QoS analysis, and they show that the new algorithm performs better than the common one.

In contrast to the common algorithm, the new algorithm guarantees an upper bound on the detection time, and this bound depends only on the parameters η and δ of the algorithm — not on the probabilistic behavior of the heartbeats.

Moreover, the new algorithm is optimal in the sense that it has the best possible query accuracy probability with respect to any given bound on the detection time. More precisely, we show that among all failure detectors that send heartbeats at the same rate (they use the same network bandwidth) and satisfy the same upper bound on the detection time, the new algorithm has the best query accuracy probability.

The algorithm that we give here assumes that p and q have synchronized clocks. This assumption is not unrealistic, even in large networks. For example, GPS and Cesium clocks are becoming accessible, and they can provide clocks that are very closely synchronized (see, e.g., [23]). In [10], we show how to modify this algorithm so that it works even when synchronized clocks are not available.

Configuring our Algorithm to Meet the Failure Detector Requirements of an Application. Given a set of failure detector QoS requirements (provided by an application), we show how to compute the parameters of our algorithm to achieve these requirements. We first do so assuming that one knows the probabilistic behavior of the system (i.e., the probability distributions of message delays and message losses). We then drop this assumption, and show how to configure the failure detector to meet the QoS requirements of an application even when the probabilistic behavior of the system is not known.

1.3. Related Work

In [14], Gouda and McGuire measure the performance of some failure detector protocols under the assumption that the protocol stops as soon as some process is suspected to have crashed (even if this suspicion is a mistake). This class of failure detectors is less general than the one that we studied here: in our work, a failure detector can alternate between suspicion and trust many times.

In [22], van Renesse *et. al.* propose a scalable gossip-style randomized failure detector protocol. They measure the accuracy of this protocol in terms of the *probability of premature timeouts*.[4] The probability of premature timeouts, however, is not an appropriate metric for the specification of failure detectors in general: it is implementation-specific and it cannot be used to compare failure detectors that use timeouts in different ways.

In [19], Raynal and Tronel present an algorithm that detects member failures in a group: if some process detects a failure in the group (perhaps a false detection), then all processes report a group failure and the protocol terminates. The algorithm uses heartbeat-style protocol, and its timeout mechanism is the same as the simple algorithm that we described in Section 1.2.

In [23], Veríssimo and Raynal study *QoS failure detectors* — these are detectors that indicate when a service does not meet its quality-of-service requirements. In contrast, this paper studies the QoS *of* failure detectors, i.e., how well a failure detector works.

The probabilistic network model used in this paper is similar to the ones used in [11, 6] for probabilistic clock synchronization.

All proofs and many technical details are omitted here, and they can be found in [10].

2. On the QoS Specification of Failure Detectors

We consider a system of two processes p and q. We assume that the failure detector at q monitors p, and that q does not crash. Henceforth, real time is continuous and ranges from 0 to ∞.

2.1. The Failure Detector Model

The output of the failure detector at q at time t is either S or T, which means that q suspects or trusts p at time t, respectively. A *transition* occurs when the output of the failure detector at q changes: An *S-transition* occurs when the output at q changes from T to S; a *T-transition* occurs when the output at q changes from S to T. We assume that there are only a finite number of transitions during any finite time interval.

Since the behavior of the system is probabilistic, the precise definition of our model and of our QoS metrics uses the theory of stochastic processes. In particular, most of the metrics we proposed are random variables. To keep our presentation at an intuitive level, we omit the technical details related to this theory (they can be found in [10]).

2.2. Primary Metrics

We propose three primary metrics for the QoS specification of failure detectors. The first one measures the speed of a failure detector. It is defined with respect to the runs in which p crashes.

[4] This is called "the probability of mistakes" in [22].

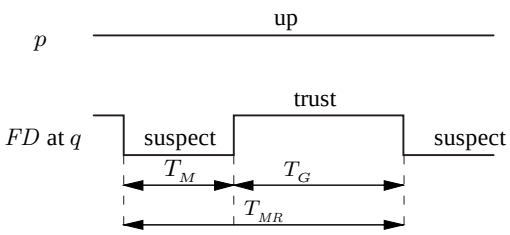

Figure 4. Mistake duration T_M, good period duration T_G, and mistake recurrence time T_{MR}

Detection time (T_D): Informally, T_D is the time that elapses from p's crash to the time when q starts suspecting p permanently. More precisely, T_D is a random variable representing the time that elapses from the time that p crashes to the time when the final S-transition (of the failure detector at q) occurs and there are no transitions afterwards (Fig. 1).[5]

We next define some metrics that are used to specify the accuracy of a failure-detector. Throughout the paper, all accuracy metrics are defined with respect to *failure-free* runs, i.e., runs in which p does not crash.[6] There are two primary accuracy metrics:

Mistake recurrence time (T_{MR}): this measures the time between two consecutive mistakes. More precisely, T_{MR} is a random variable representing the time that elapses from an S-transition to the next one (Fig. 4).

Mistake duration (T_M): this measures the time it takes the failure detector to correct a mistake. More precisely, T_M is a random variable representing the time that elapses from an S-transition to the next T-transition (Fig. 4).

As we discussed in the introduction, there are many aspects of failure detector accuracy that may be important to applications. Thus, in addition to T_{MR} and T_M, we propose four other accuracy metrics in the next section. We selected T_{MR} and T_M as the primary metrics because given these two, one can compute the other four (this will be shown in Section 2.4).

2.3. Derived Metrics

We propose four additional accuracy metrics:

Average mistake rate (λ_M): this measures the rate at which a failure detector make mistakes, i.e., it is the average number of S-transitions per time unit. This metric is important to long-lived applications where each failure detector mistake (each S-transition) results in a costly interrupt. This is the case for applications such as group membership and cluster management.

Query accuracy probability (P_A): this is the probability that the failure detector's output is correct at a random time. This metric is important to applications that interact with the failure detector by querying it at random times.

Many applications can make progress only during *good periods* — periods in which the failure detector makes no mistakes. This observation leads to the following two metrics.

Good period duration (T_G): this measures the length of a good period. More precisely, T_G is a random variable representing the time that elapses from a T-transition to the next S-transition (Fig. 4).

For short-lived applications, however, a closely related metric may be more relevant. Suppose that an application is started at a random time in a good period. If the *remaining part* of the good period is long enough, the short-lived application will be able to complete its task. The metric that measures the remaining part of the good period is:

Forward good period duration (T_{FG}): this is a random variable representing the time that elapses from a random time at which q trusts p, to the time of the next S-transition.

At first sight, it may seem that, on the average, T_{FG} is just half of T_G (the length of a good period). But this is incorrect, and in Section 2.4 we give the actual relation between T_{FG} and T_G.

2.4. How the Accuracy Metrics are Related

Theorem 1 below explains how our six accuracy metrics are related. We then use this theorem to justify our choice of the primary accuracy metrics. Henceforth, $Pr(A)$ denotes the probability of event A; $E(X)$, $E(X^k)$, and $V(X)$ denote the expected value (or mean), the k-th moment, and the variance of random variable X, respectively.

Parts (2) and (3) of Theorem 1 assume that in failure-free runs, the probabilistic distribution of failure detector histories is *ergodic*. Roughly speaking, this means that in failure-free runs, the failure detector slowly "forgets" its past history: from any given time on, its future behavior may depend only on its recent behavior. We call failure detectors satisfying this ergodicity condition *ergodic failure detectors*. Ergodicity is a basic concept in the theory of stochastic processes [20], but the technical details are substantial and outside the scope of this paper.

Theorem 1 *For any ergodic failure detector, the following results hold: (1)* $T_G = T_{MR} - T_M$. *(2) If* $0 < E(T_{MR}) < \infty$, *then* $\lambda_M = 1/E(T_{MR})$, *and* $P_A = E(T_G)/E(T_{MR})$. *(3) If* $0 < E(T_{MR}) < \infty$ *and* $E(T_G) = 0$, *then* T_{FG} *is always 0.*

[5] If there is no such final S-transition, then $T_D = \infty$; if such an S-transition occurs before p crashes, then $T_D = 0$. We henceforth omit the boundary cases of other metrics since they can be similarly defined.

[6] As explained in [10], these metrics are also meaningful for runs in which p crashes.

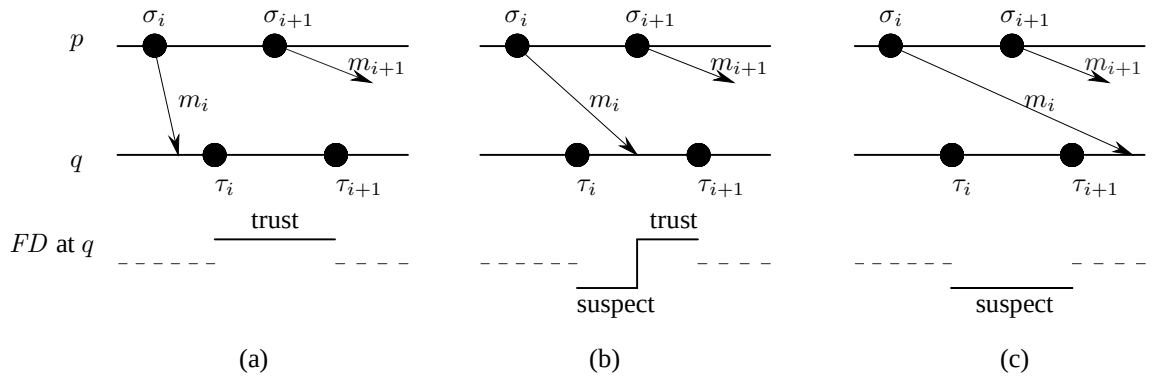

Figure 5. Three scenarios of the failure detector output in one interval $[\tau_i, \tau_{i+1})$

If $0 < E(T_{MR}) < \infty$ *and* $E(T_G) \neq 0$, *then (3a) for all* $x \in [0, \infty)$, $Pr(T_{FG} \leq x) = \int_0^x Pr(T_G > y)dy/E(T_G)$, *(3b)* $E(T_{FG}^k) = E(T_G^{k+1})/[(k+1)E(T_G)]$. *In particular, (3c)* $E(T_{FG}) = [1 + V(T_G)/E(T_G)^2]E(T_G)/2$.

The fact that $T_G = T_{MR} - T_M$ holds is immediate by definition. The proofs of parts (2) and (3) use the theory of stochastic processes. Part (2) is intuitive, while part (3), which relates T_G and T_{FG}, is more complex. In particular, part (3c) is counter-intuitive: one may think that $E(T_{FG}) = E(T_G)/2$, but part (3c) says that $E(T_{FG})$ is in general larger than $E(T_G)/2$ (this is a version of the "waiting time paradox" in the theory of stochastic processes [4]).

We now explain how Theorem 1 guided our selection of the primary accuracy metrics. Parts (2) and (3) show that λ_M, P_A and T_{FG} can be derived from T_{MR}, T_M and T_G. This suggests that the primary metrics should be selected among T_{MR}, T_M and T_G. Moreover, since $T_G = T_{MR} - T_M$, it is clear that given the joint distribution of any two of them, one can derive the remaining one. Thus, two of T_{MR}, T_M and T_G should be selected as the primary metrics, but which two? By choosing T_{MR} and T_M as our primary metrics, we get the following convenient property that helps to compare failure detectors: if FD_1 is better than FD_2 in terms of both $E(T_{MR})$ and $E(T_M)$ (the expected values of the primary metrics) then we can be sure that FD_1 is also better than FD_2 in terms of $E(T_G)$ (the expected values of the other metric). We would not get this useful property if T_G were selected as one of the primary metrics.

3. The Design and QoS Analysis of a New Failure Detector Algorithm

3.1. The Probabilistic Network Model

We assume that processes p and q are connected by a link that does not create or duplicate messages, but may delay or drop messages. Processes p and q have access to synchronized clocks (the case where synchronized clocks are not available is treated in [10]).

We assume that the message loss and message delay behavior of any message sent through the link is probabilistic, and is characterized by the following two parameters: (a) *message loss probability* p_L, which is the probability that a message is dropped by the link; and (b) *message delay* D, which is a random variable with range $(0, \infty)$ representing the delay from the time a message is sent to the time it is received, under the condition that the message is not dropped by the link. We assume that the expected value $E(D)$ and the variance $V(D)$ of D are finite. Note that our model does not assume that the message delay time D follows any particular distribution, and thus it is applicable to many practical systems.

For simplicity we assume that the probabilistic behavior of the network does not change over time. In Section 6, we explain how to modify the algorithm so that it dynamically adapts to changes in the probabilistic behavior of the system.

3.2. The Algorithm

The new algorithm works as follows. The monitored process p periodically sends heartbeat messages $m_1, m_2, m_3, \ldots$ to q every η time units, where η is a parameter of the algorithm. Every heartbeat message m_i is tagged with its sequence number i. Henceforth, σ_i denotes the sending time of message m_i. The monitoring process q shifts the σ_i's forward by δ — the other parameter of the algorithm — to obtain the sequence of times $\tau_1 < \tau_2 < \tau_3 < \ldots$, where $\tau_i = \sigma_i + \delta$. Process q uses the τ_i's and the times it receives heartbeat messages, to determine whether to trust or suspect p, as follows. Consider time period $[\tau_i, \tau_{i+1})$. At time τ_i, q checks whether it has received some message m_j with $j \geq i$. If so, q trusts

Process p:

1 for all $i \geq 1$, at time $\sigma_i = i \cdot \eta$, send heartbeat m_i to q

Process q:

2 Initialization: $output = S$; {suspect p initially}

3 for all $i \geq 1$, at time $\tau_i = \sigma_i + \delta$:

4 **if** did not receive m_j with $j \geq i$ **then** $output \leftarrow S$; {suspect p if no fresh message is received}

5 upon receive message m_j at time $t \in [\tau_i, \tau_{i+1})$:

6 **if** $j \geq i$ **then** $output \leftarrow T$; {trust p when some fresh message is received}

Figure 6. Failure detector algorithm NFD-S with parameters η and δ

p during the entire period $[\tau_i, \tau_{i+1})$ (Fig. 5 (a)). If not, q starts suspecting p. If at some time before τ_{i+1}, q receives some message m_j with $j \geq i$ then q starts trusting p from that time until τ_{i+1}. (Fig. 5 (b)). If by time τ_{i+1}, q has not received any message m_j with $j \geq i$, then q suspects p during the entire period $[\tau_i, \tau_{i+1})$ (Fig. 5 (c)). This procedure is repeated for every time period. The detailed algorithm with parameters η and δ is denoted by NFD-S, and is given in Fig. 6.[7]

Note that from time τ_i to τ_{i+1}, only messages m_j with $j \geq i$ can affect the output of the failure detector. For this reason, τ_i is called a *freshness point*: from time τ_i to τ_{i+1}, messages m_j with $j \geq i$ are *still fresh* (useful). With this algorithm, q trusts p at time t if and only if q received a message that is still fresh at time t.

3.3. The QoS Analysis of the Algorithm

We now give the QoS of the algorithm (for a detailed analysis see [10]). We assume that the link from p to q satisfies the following *message independence* property: (a) the message loss and message delay behavior of any message sent by p is independent of whether or when p crashes; and (b) the behaviors of any two heartbeat messages sent by p are independent.[8] Henceforth, let $\tau_0 \stackrel{\text{def}}{=} 0$, and $\tau_i = \sigma_i + \delta$ for $i \geq 1$ (as in line 3 of the algorithm).

We first formalize the intuition behind freshness points and fresh messages:

[7] This version of the algorithm is convenient for illustrating the main idea and for performing the analysis. We have omitted some obvious optimizations.

[8] In practice, this holds only if consecutive heartbeats are sent more than some Δ time units apart, where Δ depends on the system. So assuming that the behavior of heartbeats are independent is equivalent to assuming that $\eta > \Delta$.

Lemma 2 *For all $i \geq 0$ and all time $t \in [\tau_i, \tau_{i+1})$, q trusts p at time t if and only if q has received some message m_j with $j \geq i$ by time t.*

The following definitions are for runs where p does not crashes.

Definition 1

(1) For any $i \geq 1$, let k be the smallest integer such that for all $j \geq i + k$, m_j is sent at or after time τ_i.

(2) For any $i \geq 1$, let $p_j(x)$ be the probability that q does not receive message m_{i+j} by time $\tau_i + x$, for every $j \geq 0$ and every $x \geq 0$; let $p_0 = p_0(0)$.

(3) For any $i \geq 2$, let q_0 be the probability that q receives message m_{i-1} before time τ_i.

(4) For any $i \geq 1$, let $u(x)$ be the probability that q suspects p at time $\tau_i + x$, for every $x \in [0, \eta)$.

(5) For any $i \geq 2$, let p_S be the probability that an S-transition occurs at time τ_i.

The above definitions are given in terms of i, a positive integer. Proposition 3, however, shows that they are actually independent of i.

Proposition 3 *(1) $k = \lceil \delta/\eta \rceil$. (2) For all $j \geq 0$ and for all $x \geq 0$, $p_j(x) = p_L + (1 - p_L)Pr(D > \delta + x - j\eta)$. (3) $q_0 = (1 - p_L)Pr(D < \delta + \eta)$. (4) For all $x \in [0, \eta)$, $u(x) = \prod_{j=0}^{k} p_j(x)$. (5) $p_S = q_0 \cdot u(0)$.*

By definition, if $p_0 = 0$ then for every $i \geq 1$, the probability that q receives m_i by time τ_i is 1. Thus, if $p_0 = 0$ then, with probability one, q trusts p forever after time τ_1. Similarly, it is easy to see that if $q_0 = 0$ then, with probability one, q suspects p forever. So $p_0 = 0$ and $q_0 = 0$ are degenerated cases of no interest. We henceforth assume that $p_0 > 0$ and $q_0 > 0$.

The following theorem summarizes our QoS analysis of the new failure detector algorithm.

Theorem 4 *Consider a system with synchronized clocks, where the probability of message losses is p_L, and the distribution of message delays is $P(D \leq x)$. The failure detector NFD-S of Fig. 6 with parameters η and δ has the following properties.*

(1) The detection time is bounded:

$$T_D \leq \delta + \eta. \tag{3.1}$$

(2) The average mistake recurrence time is:

$$E(T_{MR}) = \frac{\eta}{p_S}. \tag{3.2}$$

(3) The average mistake duration is:

$$E(T_M) = \frac{\int_0^\eta u(x)\,dx}{p_S}. \tag{3.3}$$

From $E(T_{MR})$ and $E(T_M)$ given in the theorem above, we can easily derive the other accuracy measures using Theorem 1. For example, we can get the query accuracy probability $P_A = 1 - E(T_M)/E(T_{MR}) = 1 - 1/\eta \cdot \int_0^\eta u(x)\,dx$.

Theorem 4 (1) shows an important property of the algorithm: the detection time is bounded, and the bound does not depend on the behavior of message delays and losses.

In Section 4, we show how to use Theorem 4 to compute the failure detector parameters, so that the failure detector satisfies some QoS requirements (given by an application).

3.4. An Optimality Result

Among all failure detectors that send heartbeats at the same rate and satisfy the same upper bound on the detection time, the new algorithm provides the best query accuracy probability. More precisely, let $\mathcal{C}$ be the class of failure detector algorithms A such that in every run of A, process p sends heartbeats to q every η time units and A satisfies $T_D \le T_D^U$ for some constant T_D^U. Let A^* be the instance of the new failure detector algorithm NFD-S with parameters η and $\delta = T_D^U - \eta$. By part (1) of Theorem 4, we know that $A^* \in \mathcal{C}$. We can show that

Theorem 5 *For any $A \in \mathcal{C}$, let P_A be the query accuracy probability of A. Let P_A^* be the query accuracy probability of A^*. Then $P_A^* \ge P_A$.*

4. Configuring the Failure Detector to Satisfy QoS Requirements

Suppose we are given a set of failure detector QoS requirements (the QoS requirements could be given by the application that uses this failure detector). We now show how to compute the parameters η and δ of our failure detector algorithm, so that these requirements are satisfied. We assume that (a) the local clocks of processes are synchronized, and (b) one knows the probabilistic behavior of the messages, i.e., the message loss probability p_L and the distribution of message delays $Pr(D \le x)$. In Section 5, we consider the case when (b) does not hold, and in [10] we treat the case when both (a) and (b) do not hold.

We assume that the QoS requirements are expressed using the primary metrics. More precisely, a set of QoS requirements is a tuple (T_D^U, T_{MR}^L, T_M^U), where T_D^U is an upper bound on the detection time, T_{MR}^L is a lower bound on the average mistake recurrence time, and T_M^U is an upper bound on the average mistake duration. In other words, the requirements are that:[9]

$$T_D \le T_D^U,\ E(T_{MR}) \ge T_{MR}^L,\ E(T_M) \le T_M^U. \qquad (4.4)$$

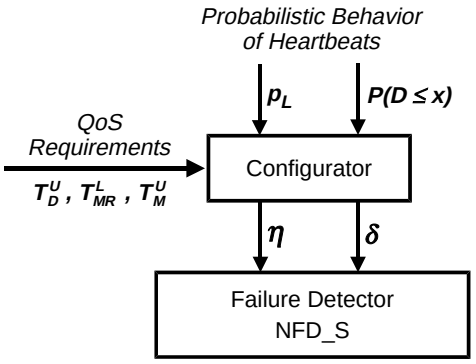

Figure 7. Meeting QoS requirements with NFD-S. The probabilistic behavior of heartbeats is given

Our goal, illustrated in Fig. 7, is to find a configuration procedure that takes as inputs (a) the QoS requirements, namely T_D^U, T_{MR}^L, T_M^U, and (b) the probabilistic behavior of the heartbeat messages, namely p_L and $Pr(D \le x)$, and outputs the failure detector parameters η and δ so that the failure detector satisfies the QoS requirements in (4.4). Furthermore, to minimize the network bandwidth taken by the failure detector, we want a configuration procedure that finds the largest intersending interval η that satisfy these QoS requirements.

Using Theorem 4, our goal can be stated as a mathematical programming problem:

$$\begin{aligned} \text{maximize} \quad & \eta \\ \text{subject to} \quad & \delta + \eta \le T_D^U && (4.5) \\ & \frac{\eta}{p_S} \ge T_{MR}^L && (4.6) \\ & \frac{\int_0^\eta u(x)\,dx}{p_S} \le T_M^U && (4.7) \end{aligned}$$

where the values of $u(x)$ and p_S are given by Proposition 3. Solving this problem is hard, so instead we show how to find some η and δ that satisfy (4.5)–(4.7) (but the η that we find may not be the largest possible). To do so, we replace (4.7) with a simpler and stronger constraint, and then compute

[9] Note that the bounds on the primary metrics $E(T_{MR})$ and $E(T_M)$ also impose bounds on the derived metrics, according to Theorem 1. More precisely, we have $\lambda_M \le 1/T_{MR}^L$, $P_A \ge (T_{MR}^L - T_M^U)/T_{MR}^L$, $E(T_G) \ge T_{MR}^L - T_M^U$, and $E(T_{FG}) \ge (T_{MR}^L - T_M^U)/2$.

the optimal solution of this modified problem (see [10] for more details). We obtain the following procedure to find η and δ:

- *Step 1*: Compute $q_0' = (1 - p_L)Pr(D < T_D^U)$, and let $\eta_{\max} = q_0' T_M^U$.
- *Step 2*: Let $f(\eta) =$

$$\frac{\eta}{q_0' \prod_{j=1}^{\lceil T_D^U/\eta \rceil - 1} [p_L + (1 - p_L)Pr(D > T_D^U - j\eta)]}. \tag{4.8}$$

  Find the largest $\eta \leq \eta_{\max}$ such that $f(\eta) \geq T_{MR}^L$. To find such an η, we can use a simple numerical method, such as binary search (this works because when η decreases, $f(\eta)$ increases exponentially fast).
- *Step 3*: Set $\delta = T_D^U - \eta$.

Theorem 6 *Consider a system with synchronized clocks. With the parameters η and δ obtained by the above procedure, the failure detector algorithm NFD-S of Fig. 6 satisfies the QoS requirements given in (4.4).*

As an example of the configuration procedure of the failure detector, suppose we have the following QoS requirements: (a) a crash failure is detected within 30 seconds, i.e., $T_D^U = 30\,s$; (b) on average, the failure detector makes at most one mistake per month, i.e., $T_{MR}^L = 30 \text{ days} = 2\,592\,000\,s$; (c) on average, the failure detector corrects its mistakes within one minute, i.e. $T_M^U = 60\,s$. Assume that the message loss probability is $p_L = 0.01$, the distribution of message delay D is exponential, and the average message delay $E(D)$ is $0.02\,s$. By inputting these numbers into the configuration procedure, we get $\delta = 20.03\,s$ and $\eta = 9.97\,s$. With these parameters, our failure detector satisfies the given QoS requirements.

5. Dealing with Unknown Message Behavior

In Section 4, our procedure to compute the parameters η and δ of NFD-S to meet some QoS requirements assumed that one knows the probability p_L of message loss and the distribution $Pr(D \leq x)$ of message delays. This assumption is not unrealistic, but in some systems the probabilistic behavior of messages may not be known. In that case, it is still possible to compute η and δ, as we now explain. We proceed in two steps: (1) we first show how to compute η and δ using only p_L, $E(D)$ and $V(D)$ (recall that $E(D)$ and $V(D)$ are the expected value and variance of message delays, respectively); (2) we then show how to estimate p_L, $E(D)$ and $V(D)$.

Computing Failure Detector Parameters η and δ Using p_L, $E(D)$ and $V(D)$. With $E(D)$ and $V(D)$, we can bound $Pr(D > t)$ using the following *One-Sided Inequality* of probability theory (e.g., see [4], p.79): For any random variable D with a finite expected value and a finite variance,

$$Pr(D > t) \leq \frac{V(D)}{V(D) + (t - E(D))^2}, \text{ for all } t > E(D). \tag{5.9}$$

With this, we can derive the following bounds on the QoS metrics of algorithm NFD-S.

Theorem 7 *Consider a system with synchronized clocks and assume $\delta > E(D)$. For algorithm NFD-S, we have $E(T_{MR}) \geq \eta/\beta$, $E(T_M) \leq \eta/\gamma$, $P_A \geq 1 - \beta$, $E(T_G) \geq (1 - \beta)\eta/\beta$, and $E(T_{FG}) \geq (1 - \beta)\eta/(2\beta)$, where*

$$\beta = \prod_{j=0}^{k_0} \frac{V(D) + p_L(\delta - E(D) - j\eta)^2}{V(D) + (\delta - E(D) - j\eta)^2},$$

$$k_0 = \lceil (\delta - E(D))/\eta \rceil - 1,$$

and

$$\gamma = \frac{(1 - p_L)(\delta - E(D) + \eta)^2}{V(D) + (\delta - E(D) + \eta)^2}.$$

Theorem 7 can be used to compute the parameters η and δ of the failure detector NFD-S, so that it satisfies the QoS requirements given in (4.4). The configuration procedure is given below. This procedure assumes that $T_D^U > E(D)$, i.e., the required detection time is greater than the average message delay (a reasonable assumption).

- *Step 1*: Compute $\gamma' = (1 - p_L)(T_D^U - E(D))^2/(V(D) + (T_D^U - E(D))^2)$ and let $\eta_{\max} = \min(\gamma' T_M^U, T_D^U - E(D))$.
- *Step 2*: Let $f(\eta) =$

$$\eta \cdot \prod_{j=1}^{\lceil (T_D^U - E(D))/\eta \rceil - 1} \frac{V(D) + (T_D^U - E(D) - j\eta)^2}{V(D) + p_L(T_D^U - E(D) - j\eta)^2}. \tag{5.10}$$

  Find the largest $\eta \leq \eta_{\max}$ such that $f(\eta) \geq T_{MR}^L$.
- *Step 3*: Set $\delta = T_D^U - \eta$.

Notice that the above procedure does not use the distribution $Pr(D \leq x)$ of message delays; it only uses p_L, $E(D)$ and $V(D)$.

Theorem 8 *Consider a system with synchronized clocks. With parameters η and δ computed by the above procedure, the failure detector algorithm NFD-S of Fig. 6 satisfies the QoS requirements given in (4.4), provided that $T_D^U > E(D)$.*

Estimating p_L, $E(D)$ and $V(D)$. It is easy to estimate p_L, $E(D)$ and $V(D)$ using heartbeat messages. For example, to estimate p_L, one can use the sequence numbers of the heartbeat messages to count the number of "missing" heartbeats, and then divide this count by the highest sequence number received so far. To estimate $E(D)$ and $V(D)$, we use the synchronized clocks as follows: When p sends a heartbeat m, p timestamps m with the sending time S, and when q receives m, q records the receipt time A. In this way, $A - S$ is the delay of m. We then compute the average and variance of $A - S$ for multiple past heartbeat messages, and thus obtain accurate estimates for $E(D)$ and $V(D)$.

6. Concluding Remarks

An Adaptive Failure Detector. In this paper, we assumed that the probabilistic behavior of heartbeat messages does not change. In some networks, this may not be the case. For instance, a corporate network may have one behavior during working hours (when the message traffic is high), and a completely different behavior during lunch time or at night (when the system is mostly idle): During peak hours, the heartbeat messages may have a higher loss rate, a higher expected delay, and a higher variance of delay, than during off-peak hours. Such networks require a failure detector that *adapts* to the changing conditions, i.e., it dynamically reconfigures itself to meet some given QoS requirements.

It turns out that we can use the configuration procedure given in Section 5 to make our failure detector adaptive. The idea is to periodically estimate the *current* values of p_L, $E(D)$ and $V(D)$ using the n most recent heartbeats. These estimates are then fed into the configuration procedure to recompute new failure detector parameters η and δ.

The above adaptive algorithm forms the core of a failure detection service that is currently being implemented and evaluated [12]. This service is intended to be shared among many different concurrent applications, each with a different set of QoS requirements. The failure detector in this architecture dynamically adapts itself not only to changes in the network condition, but also to changes in the current set of QoS demands (as new applications are started and old ones terminate).

References

[1] M. K. Aguilera, W. Chen, and S. Toueg. On quiescent reliable communication. *SIAM Journal on Computing*. To appear.

[2] M. K. Aguilera, W. Chen, and S. Toueg. Using the heartbeat failure detector for quiescent reliable communication and consensus in partitionable networks. *Theoretical Computer Science*, 220(1):3–30, June 1999.

[3] M. K. Aguilera, W. Chen, and S. Toueg. Failure detection and consensus in the crash-recovery model. *Distributed Computing*, 2000. To appear.

[4] A. O. Allen. *Probability, Statistics, and Queueing Theory with Computer Science Applications*. Academic Press, 2nd edition, 1990.

[5] Y. Amir, D. Dolev, S. Kramer, and D. Malkhi. Transis: a communication sub-system for high availability. In *Proceedings of the 22nd Annual International Symposium on Fault-Tolerant Computing*, pages 76–84, Boston, July 1992.

[6] K. Arvind. Probabilistic clock synchronization in distributed systems. *IEEE Transactions on Parallel and Distributed Systems*, 5(5):475–487, May 1994.

[7] O. Babaoğlu, R. Davoli, L.-A. Giachini, and M. G. Baker. Relacs: a communications infrastructure for constructing reliable applications in large-scale distributed systems, 1994. BROADCAST Project deliverable report, Department of Computing Science, University of Newcastle upon Tyne, UK.

[8] R. Braden, editor. *Requirements for Internet Hosts-Communication Layers*. RFC 1122, Oct. 1989.

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

[10] W. Chen. *On the Quality of Service of Failure Detectors*. PhD thesis, Cornell University, May 2000.

[11] F. Cristian. Probabilistic clock synchronization. *Distributed Computing*, 3(3):146–158, 1989.

[12] B. Deianov and S. Toueg. Personal communication, 2000.

[13] C. Fetzer and F. Cristian. Fail-aware failure detectors. In *Proceedings of the 15th Symposium on Reliable Distributed Systems*, pages 200–209, Oct. 1996.

[14] M. G. Gouda and T. M. McGuire. Accelerated heartbeat protocols. In *Proceedings of the 18th International Conference on Distributed Computing Systems*, May 1998.

[15] R. Guerraoui, M. Larrea, and A. Schiper. Non blocking atomic commitment with an unreliable failure detector. In *Proceedings of the 14th IEEE Symposium on Reliable Distributed Systems*, pages 41–50, Sept. 1995.

[16] M. G. Hayden. *The Ensemble System*. PhD thesis, Department of Computer Science, Cornell University, Jan. 1998.

[17] L. E. Moser, P. M. Melliar-Smith, D. A. Argarwal, R. K. Budhia, and C. A. Lingley-Papadopoulos. Totem: A fault-tolerant multicast group communication system. *Commun. ACM*, 39(4):54–63, Apr. 1996.

[18] G. F. Pfister. *In Search of Clusters*. Prentice-Hall, Inc., 2nd edition, 1998.

[19] M. Raynal and F. Tronel. Group membership failure detection: a simple protocol and its probabilistic analysis. *Distributed Systems Engineering Journal*, 6(3):95–102, 1999.

[20] K. Sigman. *Stationary Marked Point Processes, an Intuitive Approach*. Chapman & Hall, 1995.

[21] R. van Renesse, K. P. Birman, and S. Maffeis. Horus: a flexible group communication system. *Commun. ACM*, 39(4):76–83, Apr. 1996.

[22] R. van Renesse, Y. Minsky, and M. Hayden. A gossip-style failure detection service. In *Proceedings of Middleware'98*, Sept. 1998.

[23] P. Veríssimo and M. Raynal. Time, clocks and temporal order. In S. Krakowiak and S. K. Shrivastava, editors, *Recent Advances in Distributed Systems*, chapter 1. Springer-Verlag, 2000. to appear.