Definition and Specification of Accrual Failure Detectors

Xavier Défago[1,2], Péter Urbán[1], Naohiro Hayashibara[1], Takuya Katayama[1]

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

March 22, 2005

IS-RR-2005-004

Japan Advanced Institute of Science and Technology (JAIST)

School of Information Science
1-1 Asahidai, Nomi, Ishikawa 923-1292, Japan

http://www.jaist.ac.jp/

ISSN 0918-7553

Definition and Specification of Accrual Failure Detectors*

Xavier Défago[(a,b)], Péter Urbán[(a)], Naohiro Hayashibara[(a)],Takuya Katayama[(a)]
[(a)]School of Information Science,
Japan Advanced Institute of Science and Technology (JAIST),
1-1 Asahidai, Nomi, Ishikawa 923-1292, Japan.
[(b)]PRESTO, Japan Science and Technology Agency (JST).
Email: {defago,urban,nao-haya,katayama}@jaist.ac.jp

Abstract

For many years, people have been advocating the development of failure detection as a basic service, but, unfortunately, without meeting much success so far. We believe that this comes from the fact that important *system engineering* issues have not yet been addressed adequately, thus preventing the definition of a truly generic service. Ultimately, our goal is to define a service that is both simple and expressive, yet powerful enough to support the requirements of many distributed applications.

To this end, we consider an alternative interaction model between the service and the applications, called *accrual failure detectors*. Roughly, an accrual failure detector associates to each process a real value representing a *suspicion level*, instead of the traditional binary information (i.e., trust vs. suspect). In this paper, we provide a rigorous definition for accrual failure detectors, demonstrate that changing the interaction model leads to no loss in computational power, discuss quality of service issues, and present several possible implementations.

1 Introduction

Failure detection is an essential component for building reliable distributed systems. As such, it was proposed many times that failure detection ought to be provided as a generic service, shared among distributed applications (e.g., [13, 16, 29]). In spite of many ground-breaking advances made on failure detection, such a service still remains at a distant horizon.

We contend that the current obstacles to provide failure detection as a generic system service—in sharp contrast with the success of NTP for time synchronization—are due to the fact that several important *architectural* and *engineering* issues have been overlooked until now. To be genuinely ubiquitous, a failure detection service must be able to satisfy the requirements of a large variety of application classes without introducing unnecessary limitations. To this end, the following two major issues must be addressed properly. Firstly, at any time, the service must be able to provide various levels of quality of service (QoS) in order to meet the requirements of independent applications that may run simultaneously. Secondly, the service must support all reasonably common usage patterns as smoothly as possible.

Although the computational aspects of failure detectors are now well-established and several efficient implementations have been proposed, only few studies have been looking at the issues mentioned above. This paper addresses these issues by defining *accrual failure detectors*, a concept that allows for

*Part of this research was conducted for the program "Fostering Talent in Emergent Research Fields" in Special Coordination Funds for Promoting Science and Technology by the Japan Ministry of Education, Culture, Sports, Science and Technology; the Japan Society for the Promotion of Science; a Grant-in-Aid for JSPS Fellows from the Japan Ministry of Education, Culture, Sports, Science and Technology; and the Swiss National Science Foundation.

a cleaner decomposition of the behavior of the underlying system and the quality of service provided to the applications. In recent work, we have proposed a possible implementation of an accrual failure detector, called the φ failure detector [23].

This paper defines the generic notion of accrual failure detectors, and makes the link with the computational aspects of failure detection. More specifically, this paper complements our earlier work by (1) providing a precise definition for the concept of *accrual failure detection*, (2) establishing important properties of such failure detectors, and (3) presenting the characteristics of several useful implementations.

1.1 Failure detectors

In their seminal paper, Chandra and Toueg [7] have established the theoretical foundation of failure detection. Many important results stem from their work, such as minimal conditions, equivalences, transformations, metrics (e.g., [6, 8, 9, 15, 24, 28, 25, 26]). These studies concentrate on the *computational power* of failure detectors from an algorithmic perspective. Other studies have been aimed at implementing such failure detectors over small-scale (e.g., [8, 3]) and large-scale networks (e.g., [29, 4]). However, most failure detectors proposed in the literature are based on a binary interaction model,[1] whereby a monitored process is either trusted or suspected.[2]

1.2 Limitations of the binary model

The binary model has some limitations when it comes to providing failure detection as a generic service.

First, a binary interaction model makes it difficult to support several applications running simultaneously. To see this, one must realize that there is an inherent tradeoff between *conservative* (i.e., slow and accurate) and *aggressive* (i.e., fast but inaccurate) failure detection. Different applications are likely to have different requirements with respect to the QoS of the failure detection. Moreover, several levels of QoS can be useful even within the same application. For instance, an application can take precautionary measures against catastrophic failure 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.

Second, although binary failure detectors are well-adapted to meet the needs of many algorithms, their interaction model cannot easily cope with some usage patterns that arise in practice. The simple example below illustrates two such usage patterns.

1.3 Illustration: BoT computations

To further illustrate our point, we present a simple example taken from the execution of Bag-of-Tasks (BoT) computations in the OurGrid platform [10] (kindly suggested to us by Francisco V. Brasileiro). This example is particularly helpful as it shows two interesting usage patterns of failure detectors.

Consider a simplified environment with one master process and a collection of worker processes. The master holds a list of independent tasks that need to be executed, dispatches these tasks to available workers, and gathers results. For simplicity, assume that the master never fails but that some of the workers may crash. Clearly, the master must be able to detect the crash of a worker and reassign the tasks of the worker, or else the computation may never complete. Consider the following two situations, where the master needs to use information about the possible failure of workers.

First, when assigning tasks to the workers, the master must avoid sending them to workers that have crashed. Hence, the master needs to be able to sort workers according to how likely they are still operational.[3]

[1] Some notable exceptions (e.g., [18, 28]) are discussed in Section 6.

[2] This includes the eventual leader oracle, that can be expressed in terms of *trust* and *suspect* [6].

[3] Of course, other parameters, such as the load on the workers, may be equally important when assigning tasks to workers.

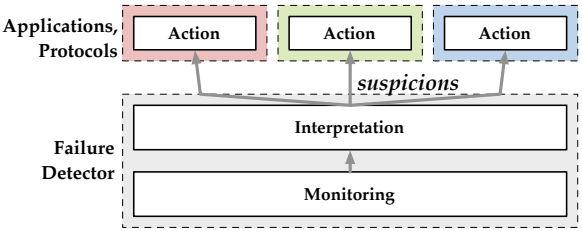

Figure 1: Binary failure detectors: monitoring and interpretation are combined.

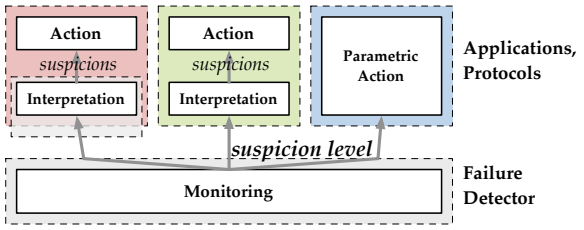

Figure 2: Accrual failure detectors: monitoring and interpretation are decoupled.

Second, when a task is being executed by a worker, the crash of this worker must be detected and the task restarted. However, let us consider the cost of making a wrong decision: if a task is wrongly aborted, all CPU cycles that were spent computing the task are wasted. Note that the cost of aborting the task due to a wrong suspicion *increases as time passes*.

The two situations described above are difficult to handle with binary failure detectors. While ad-hoc solutions certainly exist, a more suitable abstraction can simplify the design and thus improve the quality of the system. We know of one attempt at defining such an abstraction, called slowness oracles [28], that cope with the first situation by ordering processes according to their perceived speed. However, slowness oracles do not cope well with the second situation.

1.4 Accrual failure detectors

To cope with the situations described above, we advocate a more flexible interaction model for failure detectors, on top of which binary and other kinds of failure detectors can be constructed. More specifically, we define a family of failure detectors, called *accrual failure detectors*, whereby each monitoring process associates, to each of the monitored processes, a real number that changes over time. The value represents a *suspicion level*, where zero means that the process is not suspected at all, and the larger the value, the stronger the suspicion. Roughly speaking, accrual failure detectors ensure that the suspicion level associated with a monitored process p (1) accrues toward infinity if p is faulty, and (2) is bounded if p is correct.

1.5 Architectural issues

Failure detection can be decomposed into three basic tasks. *Monitoring* allows the failure detector to gather information about other hosts and their processes. This is usually done through the network, by sampling heartbeat arrivals or query-response delays. *Interpretation* is necessary to make sense of the information obtained through monitoring. With binary failure detectors, this is often done by setting some timeout and generating suspicions. QoS parameters intervene at this stage. *Actions* are executed as a response to triggered suspicions. This is most often done within the applications.

For a service, one of the major advantages of providing an accrual failure detector over a binary one is that the former allows for a complete[4] *decoupling between monitoring and interpretation.* Indeed, binary failure detectors combine these two roles (see Fig. 1), and thus provide applications only with information that is already interpreted. Applications are left with how to react to suspicions. Unfortunately, suspicion tradeoffs largely depend on the nature of the triggered action, as well as its cost in terms of performance or resource usage.

[4]Notice that a common misconception considers heartbeat intervals as a parameter for setting the QoS of failure detectors. In practice, while heartbeat intervals indeed have an effect on the overall QoS, the parameter is actually *imposed* by the underlying system (i.e., its behavior as well as its administration). Refer to [23] for a more detailed argumentation on this issue.

In contrast, accrual failure detectors leave the task of interpreting the suspicion level to applications (see Fig. 2). Thus, different applications can set different thresholds to suspect processes according to their needs, or even directly use the suspicion level as a parameter to their actions. Note that this is an *architectural* consideration: a library can still provide the interface of a binary failure detector to applications that prefer that interaction model. However, there will be one interpretation module per application, not one interpretation module shared among all applications within the failure detector.

1.6 Contribution & structure

The main contribution of this paper is to provide a rigorous definition for accrual failure detectors. In particular, we focus on a class of accrual failure detectors that is computationally equivalent to an unreliable failure detector of class $\Diamond\mathcal{P}$ (i.e., one that stops making mistakes after some time). We identify important properties of accrual failure detectors in relation with the quality of service of failure detectors. Finally, we discuss several possible implementations of accrual failure detectors and explain how they are related.

The rest of the paper is structured as follows. Section 2 describes our system model, as well as some basic definitions. Section 3 defines accrual failure detectors and their basic properties. Section 4 states several important theorems related to particular classes of accrual failure detectors. Section 5 outlines several possible implementations of accrual failure detectors. Section 6 discusses how accrual failure detectors are related to previous work. Finally, Section 7 concludes the paper.

2 System model & definitions

System model. We consider a distributed system consisting of a set of processes $\Pi = \{p_1, \ldots, p_n\}$.

We assume the existence of some global time, unbeknownst to processes, the domain of which, denoted by $\mathbb{T}$, is an infinitely countable subset of real numbers with no upper bound. We assume that processes always make progress, and that at least $\delta > 0$ time units elapse between consecutive steps (the purpose of the latter is to exclude the case where processes take an infinite number of steps in finite time).

Failures. The failure model considered in this paper is based on the model of Chandra and Toueg [7]. A process can be correct or faulty. A process is *faulty* if its behavior deviates from its specification, and a process is *correct* if it is not faulty. We say that a process *fails* when its behavior starts deviating from its specification. Faulty processes never recover.

A failure pattern is a function $F : \mathbb{T} \mapsto 2^{\Pi}$, where $F(t)$ is the set of processes that have failed before or at time t. The function $correct(F)$ denotes the set of correct processes (processes that never belong to failure pattern F) while $faulty(F) = \Pi - correct(F)$ denotes the set of faulty processes.

Failure detectors. Chandra and Toueg [6] define failure detectors as a collection of failure detector modules, one attached to each process, that output information on the failure pattern that occurs in an execution.[5] A failure detector module outputs information from a range $\mathcal{R}$ of values. A failure detector history H with range $\mathcal{R}$ is a function $H : \Pi \times \mathbb{T} \mapsto \mathcal{R}$, where $H(p, t)$ is the value output by the failure detector module of process p at time t. H is only defined at times when the failure detector module provides an answer to a *query*; the failure detector module may be queried whenever process p takes a step, and each query eventually results in an answer. This follows the definition of oracles introduced

[5]The definition of failure detectors of Chandra and Toueg [7] restricts the output to a set of suspected processes. Accrual failure detectors are based on the definition of Chandra et al. [6], that allows values taken from an arbitrary range.

by Aguilera et al. [2].[6] The times at which queries $1, 2, \cdots$ are answered are denoted by the sequence $t_p^{query}(1), t_p^{query}(2), \cdots$. Correct processes query their failure detector modules infinitely-many times.

Binary failure detectors, such as those defined by Chandra and Toueg [7], output values from the range $\mathcal{R} = 2^{\Pi}$, that is, the power set of Π. If a process is part of the output set, it is *suspected* to have failed, otherwise it is *trusted*. An *S-transition* occurs when a trusted process becomes suspected and a *T-transition* occurs when a suspected process becomes trusted.

Chandra and Toueg [7] define a class hierarchy of unreliable binary failure detectors, of which we present only one, called $\Diamond\mathcal{P}$ (eventually perfect). The class is defined by the set of failure detector histories that it permits, as specified by the following two properties of *completeness* and *accuracy*.

(STRONG COMPLETENESS) Eventually every faulty process is permanently suspected by all correct processes.

(EVENTUAL STRONG ACCURACY) There is a time after which correct processes are never suspected by any correct process.

Quality of service metrics for failure detectors. Chen et al. [8] define metrics for the quality of service of failure detectors. Quality of service quantifies how fast a failure detector detects failures (completeness) and how well it avoids wrong suspicions (accuracy). All metrics are defined for a pair of processes p and q, with q monitoring p. The metrics used in this paper are summarized below.

- The *detection time* (T_D) is the time that elapses since p fails and until q starts suspecting p permanently (i.e., until the final S-transition).

The detection time is the only completeness metric, defined on runs in which p is faulty. In contrast, all others metrics (below) relate to the accuracy and are defined on runs in which p is correct.

- The *mistake recurrence time* (T_{MR}) measures the time elapsed between two consecutive mistakes, i.e., the time between two S-transitions.
- The *mistake duration* (T_M) measures the time it takes for the detector to correct a mistake, i.e., the time from an S-transition to the next T-transition.
- The *average mistake rate* (λ_M) measures the rate at which a failure detector make mistakes, i.e., the average number of S-transitions per time unit.
- The *query accuracy probability* (P_A) is the probability that the failure detector's output is correct at a random time.
- The *good period duration* (T_G) measures the length of a good period, i.e., the time from a T-transition to the next S-transition.

3 Definition of accrual failure detectors

In this section, we define what accrual failure detectors are. We begin by defining the notion of suspicion level between a pair of processes. Then, we define the notion of accrual failure detector for a distributed system with n processes. Finally, we define a class of accrual failure detectors of particular interest, called $\Diamond\mathcal{P}_{ac}$.

[6] Aguilera et al. [2] define oracles as a sequence of quadruples (p, t, i, o) where p is a process, t is a time instant, i is the query of p at time t and o is the answer of the oracle at time t. Both i and o may take the value $\perp$, meaning respectively that no query is made at time t and that no answer is available at time t.

3.1 Suspicion level

Consider two distinct processes p and q, with q monitoring p. Let $\mathbb{R}_0^+$ denote the real positive numbers and zero. The suspicion level of process q monitoring process p expresses the confidence of q in the statement that p is faulty.

Definition 1 (Suspicion level) *The suspicion level of process q with respect to process p is the function $sl_{qp} : \mathbb{T} \mapsto \mathbb{R}_0^+$. The function sl_{qp} has a finite resolution, i.e., it may only assume integer multiples of an (arbitrarily small, but non-infinitesimal) positive constant ϵ, where for all t, $\frac{sl_{qp}(t)}{\epsilon} \in \mathbb{Z}$*

Additionally, we consider that the suspicion level satisfies the following two properties.

Property 1 (Accruement) *If process p is faulty, then eventually, the suspicion level $sl_{qp}(t)$ is monotonously increasing at a positive rate.*

$$p \in faulty(F) \Rightarrow$$
$$\exists K \exists Q \forall k \geq K : (sl_{qp}(t_q^{query}(k)) \leq sl_{qp}(t_q^{query}(k+1)) \wedge sl_{qp}(t_q^{query}(k)) < sl_{qp}(t_q^{query}(k+Q)))$$

Property 2 (Upper bound) *If process p is correct, then the suspicion level $sl_{qp}(t)$ is bounded.*

$$p \in correct(F) \Rightarrow \exists SL_{max} : \forall t \, (sl_{qp}(t) \leq SL_{max})$$

3.2 Accrual failure detectors

An accrual failure detector $\mathcal{D}_{ac}$ is a failure detector with range $\mathcal{R} = (\mathbb{R}_0^+)^\Pi$ (note the analogy to binary failure detectors for which the range is 2^Π). Its history is defined as $H(q,t)(p) = sl_{qp}(t)$. In other words, failure detector modules output non-negative real values, with each value corresponding to a process and representing the current suspicion level of that process.

We now define the class $\Diamond\mathcal{P}_{ac}$ of accrual failure detectors. We discuss other classes of accrual failure detectors in Section 4.3.

Definition 2 ($\Diamond\mathcal{P}_{ac}$ accrual f. d.) *For all pairs of distinct processes p and q, the properties of Accruement (Prop. 1) and Upper Bound (Prop. 2) both hold.*

3.3 Discussion on properties

We now discuss interesting characteristics of the suspicion level $sl_{qp}(t)$. This discussion may shed some light on the reasons behind the definition of the properties.

- *The constraints on $sl_{qp}(t)$ in the two properties are mutually exclusive.* If it were not the case, an accrual failure detector would make it impossible to distinguish between correct and faulty processes.

- *The upper bound in the Upper Bound property is unknown.* If the bound was known, the interpretation of the suspicion level would be trivial: applications could just compare the suspicion level to the known bound. This is contrary to our key concept that the interpretation of the suspicion level should be left to the application.

- *The condition on the positive rate of increase mentioned in the Accruement property is necessary.* It may be tempting to consider a weaker condition that, if process p is faulty, the suspicion level goes to infinity with time, i.e., $\lim_{t \to +\infty} sl_{qp}(t) = +\infty$

  Although attractive by its simplicity, it turns out that this condition is not sufficient because it allows situations where a correct and a faulty process can never be distinguished. This statement

is proved by presenting an adversary whereby a process can never make any permanent decision (proof in the appendix; Sect. A.5).

In practice, since the minimal rate is unknown and so is the period when it becomes effective, this is not a very stringent condition.

- *A positive rate of increase mentioned in the Accruement property allows for stationary periods.* If p is faulty, and even after stabilization ($t_q^{query}(K)$), $sl_{qp}(t)$ need not be strictly monotonous. In particular, it may remain constant for a bounded number of queries (Q). Furthermore, the bound on the number of queries Q is unknown.

  We could require that $sl_{qp}(t)$ become strictly monotonous eventually. In contrast, since the current definition allows for period during which $sl_{qp}(t)$ remains constant, this leaves more flexibility with respect to the implementation of accrual failure detectors. Indeed, implementations may find it difficult or inconvenient to return an increased value upon every query. This would possibly necessitate to either artificially update the suspicion level upon every query, or access some hardware clock with sufficiently fine resolution to compute the suspicion level (note that such a clock might not be available).

- *The increase rate of the Accruement property is time-free.* The rate is stated in terms of a number of queries to the failure detector. Thus, it avoids referring to any notion of physical time by imposing a time bound on how long $sl_{qp}(t)$ may remain constant. Otherwise, we would effectively impose some restriction on the model either (1) by imposing a minimal speed on processes, or (2) by assuming access to synchronized clocks.

- *The minimal rate of increase has a lower bound that depends on ϵ and Q.* In particular, the positive rate of increase can be bounded from below during the stable period by $\epsilon/2Q$, where Q is the maximal number of queries before the function is strictly required to increase, and ϵ is the difference between two consecutive values in the range of $sl_{qp}(t)$.

$$\frac{sl_{qp}(t_q^{query}(k')) - sl_{qp}(t_q^{query}(k))}{k' - k} \geq \frac{\epsilon}{2Q} \tag{1}$$

  for all $k \geq K$ and $k' \geq Q + k$ (where K is the query number when the function starts increasing monotonously). Since Q can be arbitrarily large, the rate of increase can be arbitrarily small.

4 Power & QoS of accrual failure detectors

This section proves that an accrual failure detector of class $\Diamond\mathcal{P}_{ac}$ and a binary one of class $\Diamond\mathcal{P}$ have the same computational power. This means that any problem that can be solved with a binary failure detector of class $\Diamond\mathcal{P}$ can also be solved with an accrual failure detector of class $\Diamond\mathcal{P}_{ac}$, and vice-versa. We show the equivalence by presenting algorithms that transform a failure detector of one class into a failure detector of the other class.We then introduce other classes of accrual failure detectors, and also investigate the relationship between thresholds on the suspicion level and the corresponding quality of service.

4.1 Transformation: accrual to binary

For simplicity, Algorithm 1 is expressed for a pair of processes p and q, where q monitors p. Shortly, the algorithm works as follows. The algorithm maintains the status of the binary failure detector (*trust* or *suspect*) in the variable *status*. It also maintains a dynamic threshold SL_{susp} on the suspicion level of the accrual failure detector, that triggers S-transitions (this is a common technique [14, 6, 17]). Similarly,

Algorithm 1 Transforming an accrual failure detector of class $\Diamond\mathcal{P}_{ac}$ into a binary one of class $\Diamond\mathcal{P}$.

```
1:  Initialization:
2:     status := trust                                   {current status (trust or suspect)}
3:     SL_susp := sl_qp                                  {threshold for suspecting}
4:     l := 1                                            {run length of period with constant suspicion level}
5:     L_trust := 1                                      {run length for trusting}
6:     sl_prev := sl_qp                                  {previous suspicion level}
7:  when queried about process p
8:     sl := sl_qp(t)                                    {get current suspicion level}
9:     if sl ≠ sl_prev then                              {update run length}
10:       l := 0
11:    l := l + 1
12:    if sl > SL_susp and status = trust then           {suspect if level beyond threshold}
13:       status := suspect
14:       SL_susp := sl                                 {increase threshold for suspecting}
15:    if (sl < sl_prev or l > L_trust) and status = suspect then   {trust if level decreasing or constant for a long time}
16:       status := trust
17:       L_trust := L_trust + 1                        {increase run length for trusting}
18:    sl_prev := sl
```

a second threshold L_{trust}, also dynamic, is used for T-transitions and tracks the number of consecutive queries during which the suspicion level does not increase.

Let us now sketch why the algorithm is correct (proofs in the appendix; Sect. A.1). Increasing the two thresholds SL_{susp} and L_{trust} is the key to ensuring the correctness of the algorithm. On the one hand, if p is correct, the algorithm ensures that SL_{susp} will grow beyond the bound SL_{max} for the suspicion level (see Property 2) and thus S-transitions stop occurring. On the other hand, if p is faulty, the threshold L_{trust} will grow beyond the maximum number of queries Q during which the suspicion level may stay constant (see Property 1) and thus T-transitions stop occurring. After, it is easy to show that the last transition is a T-transition in the first case and an S-transition in the second.

4.2 Transformation: binary to accrual

Algorithm 2 Transforming a binary failure detector of class $\Diamond\mathcal{P}$ into an accrual one of class $\Diamond\mathcal{P}_{ac}$.

```
1:  Initialization:
2:     sl_prev := 0                                      {previous suspicion level}
3:  when queried about process p
4:     query the binary failure detector
5:     if p is suspected then
6:        sl_qp(t) := sl_prev := sl_prev + ε
7:     else
8:        sl_qp(t) := sl_prev := 0
```

Again, for simplicity, Algorithm 2 is expressed in terms of two processes p and q, where q monitors p. Upon each query to the accrual failure detector, the algorithm queries the binary failure detector and updates the suspicion level sl_{qp} the following way: (1) if p is suspected, sl_{qp} increases by the resolution ϵ; (2) if p is trusted, sl_{qp} is reset to zero.

It is easy to see that the algorithm is correct (details in the appendix; Sect. A.2) by looking at what happens after the binary failure detector stabilizes. If p is faulty, sl_{qp} increases by ϵ at each query and thus Accruement (Prop. 1) holds. In contrast, if p is correct, sl_{qp} remains zero and thus is bounded by the maximal value it took before the binary failure detector stabilized. Thus, Property 2 also holds.

4.3 Other classes of accrual failure detectors

In Section 3, we defined the properties of Accruement (Prop. 1) and Upper Bound (Prop. 2) and the $\Diamond\mathcal{P}_{ac}$ class of accrual failure detectors: the properties must hold for any pair of processes. In this section, we briefly introduce other classes of accrual failure detectors: $\mathcal{P}_{ac}$, $\Diamond\mathcal{S}_{ac}$ and $\mathcal{S}_{ac}$. Each of these failure detector classes is equivalent to the corresponding binary failure detector class $\mathcal{P}$, $\Diamond\mathcal{S}$ and $\mathcal{S}$ [7]. The formal definitions and the proofs (using slightly different properties) appear in [12].

$\mathcal{P}_{ac}$ The class $\mathcal{P}_{ac}$ is based on a stronger Upper Bound property for each pair of processes. The difference is that we require that a *known* bound holds for sl_{qp}, whereas Property 2 requires that an *unknown* bound holds.The transformation algorithm from $\mathcal{P}_{ac}$ to $\mathcal{P}$ is simply based on Algorithm 1 by initializing the suspicion threshold to the value of the known bound.

$\Diamond\mathcal{S}_{ac}$ We also define weaker failure detector classes. The class $\Diamond\mathcal{S}_{ac}$ differs from the class $\Diamond\mathcal{P}_{ac}$ in that Upper Bound (Prop. 2) only needs to hold for all processes with respect to one single correct process p (instead of all pairs). This is similar to the difference between the binary failure detector classes $\Diamond\mathcal{S}$ and $\Diamond\mathcal{P}$.

The simple implementation for $\Diamond\mathcal{P}_{ac}$ (sketched in Section 5.1 and presented in Section A.4) implements $\Diamond\mathcal{S}_{ac}$ as well. The transformation algorithms (Algorithms 1 and 2) remain the same, and the proofs need to be adapted only slightly.

$\mathcal{S}_{ac}$ The class $\mathcal{S}_{ac}$ is defined similarly. The Upper Bound property must however come with a known bound (see $\mathcal{P}_{ac}$ above), but that property only has to hold for some correct process p (as for $\Diamond\mathcal{S}_{ac}$).

4.4 Multiple thresholds for differentiated QoS

The introduction stated that one accrual failure detector can serve multiple applications with different quality of service requirements. This section explores that statement more concretely by expressing it in terms of quality of service. We consider applications that interpret the suspicion level by comparing it to a threshold. We show that using a lower threshold results in more *aggressive* failure detection, i.e., a better quality of service regarding the detection of actual failures but a worse quality of service regarding wrong suspicions. Conversely, a higher threshold results in more *conservative* failure detection, i.e., a better quality of service regarding wrong suspicions at the expense of detecting actual failures.

Consider two processes p and q, with q monitoring p. Let applications (running on q) interpret the suspicion level by comparing it to a given threshold (where the threshold is a function of time) and suspect the monitored process p if and only if the suspicion level is beyond the threshold:

$$\forall t \in \mathbb{T}, (p \text{ is suspected at } t) \Leftrightarrow sl_{qp}(t) > T(t) \tag{2}$$

where $T : \mathbb{T} \mapsto \mathbb{R}^+$ is a threshold function. The equation effectively describes a binary failure detector that we denote by $\mathcal{D}_T$.

Now, consider two applications that use two failure detectors D_{T_1} and D_{T_2} with different threshold functions $T_1(t)$ and $T_2(t)$. Let $T_1(t) \leq T_2(t)$ for any time t. We can state a number of interesting properties about the two failure detectors and their quality of service.

Theorem 1 *At all times, failure detector $\mathcal{D}_{T_2}$ suspects p only if failure detector $\mathcal{D}_{T_1}$ suspects p.*

Since the details of the proofs are not essential to convey the message, they are omitted here and presented in the appendix (Sect. A.3).
We can state the following simple corollaries in terms of quality of service metrics (see Section 2):

Corollary 2 *$\mathcal{D}_{T_1}$ detects failures at least as fast as $\mathcal{D}_{T_2}$: $T_D(\mathcal{D}_{T_1}) \leq T_D(\mathcal{D}_{T_2})$ where $T_D(\mathcal{D})$ is the detection time of failure detector D.*

Corollary 3 *At some random time, $\mathcal{D}_{T_2}$ is at least as likely to trust a process as $\mathcal{D}_{T_1}$: $P_A(\mathcal{D}_{T_1}) \leq P_A(\mathcal{D}_{T_2})$ where $P_A(\mathcal{D})$ is the query accuracy probability of failure detector $\mathcal{D}$.*

There is no such simple relationship stated with the quality of service metrics T_M, T_{MR}, λ_M and T_G. However, such relationships exist if the failure detectors interpret the suspicion level in a slightly different manner. Let $\mathcal{D}'_{T_1}$ and $\mathcal{D}'_{T_2}$ use $T_1(t)$ and $T_2(t)$ to trigger an S-transition, just as before, but let them use the *same* threshold function $T_0(t)$ to trigger T-transitions. $T_0(t) < T_1(t) \leq T_2(t)$ holds at any time t. With the new failure detectors $\mathcal{D}'_{T_1}$ and $\mathcal{D}'_{T_2}$, Theorem 1 and its corollaries still hold. Additionally, we have the following theorem:

Theorem 4 *If failure detector $\mathcal{D}'_{T_2}$ has T-transition at some time t, then failure detector $\mathcal{D}'_{T_1}$ also has a T-transition at time t.*

The following corollaries follow from Theorems 1 and 4.

Corollary 5 *$\mathcal{D}'_{T_2}$ generates wrong suspicions at most as frequently as $\mathcal{D}'_{T_1}$: $T_{MR}(\mathcal{D}'_{T_1}) \leq T_{MR}(\mathcal{D}'_{T_2})$ and $\lambda_M(\mathcal{D}'_{T_1}) \geq \lambda_M(\mathcal{D}'_{T_2})$ where $T_{MR}(\mathcal{D})$ and $\lambda_M(\mathcal{D})$ are the mistake recurrence time and average mistake rate of failure detector $\mathcal{D}$, respectively.*

Corollary 6 *$\mathcal{D}'_{T_2}$ rightly trusts a process for at least as long as $\mathcal{D}'_{T_1}$, both starting from an S-transition and a random time when the process is trusted: $T_G(\mathcal{D}'_{T_1}) \leq T_G(\mathcal{D}'_{T_2})$ where $T_G(\mathcal{D})$ is the good period duration of failure detector $\mathcal{D}$.*

Unfortunately, there is no simple derivation with the metric T_M. Informally, the reason is that the failure detector $\mathcal{D}_{T_1}$ can artificially obtain a better "score" by going through many brief periods of wrong suspicions during which $\mathcal{D}_{T_2}$ does not suspect.

5 Implementing accrual failure detectors

In this section, we present implementations of accrual failure detectors for systems in which processes can crash. We start with the simplest implementation, and then present three increasingly complex and versatile variants. For simplicity, the explanations consider just two processes p and q, with q monitoring p.

5.1 Simple implementation

In the simplest implementation, the monitored process p sends heartbeats at regular intervals to the monitoring process q. Upon a query, the accrual failure detector at q simply returns the time that elapsed since the reception of the last heartbeat.

The algorithm assumes a partially synchronous system model in which processes fail by crashing permanently. Informally, we can see that the algorithm implements a failure detector of class $\Diamond\mathcal{P}_{ac}$ in that model. If p crashes, it stops sending heartbeats, and thus the suspicion level will increase forever, thus Property 1 (Accruement) is satisfied. In contrast, if p is correct, it is possible to calculate an upper bound on the maximal time elapsed between any two consecutive heartbeats, based on the characteristics of the execution. Then, Property 2 (Upper Bound) is satisfied.

Note that if one compares the suspicion level to a constant threshold T to suspect the process, the result is simply a binary heartbeat failure detector with timeout T.

This essentially shows that accrual failure detectors can be seen as a way to decompose binary failure detectors. The advantage is that accrual failure detectors can serve multiple applications with various qualities of service or applications with multiple thresholds or even more general adaptation policies.

5.2 Chen's failure detector as an accrual one

Chen et al. [8] have proposed a well-known implementation for a binary failure detector that adapts to changes in network conditions (unlike the simple implementation of Sect. 5.1). Briefly speaking, the failure detector monitors heartbeat arrivals to estimate the time EA when the next heartbeat is expected to arrive. The algorithm sets a timeout by taking EA and adding a constant safety margin α, computed from QoS requirements.

There is a simple way to transform their failure detector into an accrual one. The idea is that, when the next heartbeat is late, i.e., $t > EA$ where t is the current time, the suspicion level begins to increase linearly over time: $sl_{qp}(t) = t - EA$. Then, setting a constant suspicion threshold of α results in the original binary failure detector.

5.3 The φ adaptive accrual failure detector

The φ failure detector [23] adapts to changing network condition just like Chen's failure detector. However, whereas Chen et al. [8] only estimate the mean of the expected arrival time, φ estimates the full distribution. It does so by estimating both the mean and the variance, and supposing a distribution of a given shape [23] (e.g., a normal distribution for the inter-arrival time, or some Erlang distribution for the transmission time).

Let t_{last} be the arrival time of the last heartbeat, t be the current time, and $P_{later}(t)$ be the probability that a heartbeat will arrive more than t time units after the previous one; the latter is computed from the distribution estimated from past heartbeat arrivals. The suspicion level is computed as follows:

$$sl_{qp}(t) = -\log_{10}(P_{later}(t - t_{last})) \tag{3}$$

As $0 < P_{later} \leq 1$, sl_{qp} takes the full range of non-negative values. Using a threshold of T to suspect the monitored process p roughly means that the likelihood of a wrong suspicion is 10^{-T}, supposing that the behavior of the network is probabilistically stable.

5.4 The κ accrual failure detection framework

Finally, we briefly present the κ failure detector [22], which is a framework rather than a specific implementation. The motivation is based on the following observation: the failure detectors that estimate the arrival time of the next heartbeat do not cope well with lost heartbeats—good estimates for the variability could not prevent wrong suspicions due to bursts of message losses. The reason is that, in most systems, variability in arrival times and message losses are likely to have different reasons, hence a single random distribution cannot model all cases well enough.

The κ failure detector solves this problem in a different way. By design, its behavior changes from a fine-grained estimation at low suspicion levels (aggressive range) to a coarse-grained estimation based on counting missed heartbeats at high suspicion levels (conservative range). This change occurs *gradually* as the suspicion level increases. Experimental results [21] confirm that κ failure detectors cope well with message losses while still coping with variability in arrival times.

The κ failure detector works as follows. Each heartbeat that was not received contributes partly to the suspicion level of the failure detector. The contribution of a heartbeat H gradually increases from 0, meaning that H is not yet expected, to 1, meaning that H is considered lost. The suspicion level is calculated as the sum of all contributions.

The characteristics of the κ failure detector vary with the choice for the contribution function; this is why we consider κ to be a framework rather than a single implementation. A suitable contribution function is for instance the probability $P_{later}(t)$ of the φ failure detector, presented in the previous section. Another, simpler contribution function sets a timeout for each heartbeat; the contribution is simply 0 before the timeout and 1 after the timeout (i.e., a step function). Many other possibilities exist.

Finally, we present how the κ failure detector behaves under different conditions. When the network is stable, i.e., few messages are lost, only one single heartbeat contributes to the suspicion level significantly, and thus the suspicion level reflects the contribution function. If the contribution function adapts well to the variability in arrival times, so will the applications using the κ failure detector. On the other hand, when the network is unstable with a lot of message losses, or if the monitored process crashes, contributions for all missed heartbeats but one will likely be close to 1. In this case, the κ failure detector will give a count of missed heartbeats, and the shape of the contribution function will be nearly irrelevant.

6 Related work

We present existing work that, just like our approach, uses numeric and sometimes accruing values for failure detection or similar purposes.

Sampaio et al. [28] define slowness oracles as an oracle that outputs a list of processes ordered according to the perceived responsiveness of each process. Accrual failure detectors also quantify responsiveness, hence their output values could be used to establish (or estimate) this order.

Cosquer et al. [11] proposed a group membership service that allows the tuning of its failure detection (called suspectors) by applications. Applications do so by specifying interpretation conditions that are used by the failure detector to do the interpretation. The paper introduces many excellent ideas with respect to the tailoring of failure detectors, but does not address the issue of decoupling monitoring and interpretation. In contrast, our work focuses on the latter issue. This said, this should not be too difficult to adapt their system so that it implements an accrual failure detector.

More recently, Friedman [18] outlined in a position paper the idea of a *fuzzy group membership*, where a value called fuzziness level would be associated with each process to determine the extent to which the process belongs to the group. Technical issues were developed later by Friedman and Tcharny [19, 20]. Although the papers address different issues, the authors rely on some fuzzy failure detector that outputs some integer value and uses two thresholds to define three suspicion levels (*trusted*, *fuzzy*, or *suspected*). There are no details, however, because this is not the focus of their work. In particular, they provide no definition nor implementation of fuzzy failure detectors. We believe that, although developed independently, our works could nicely complement each other.

Aguilera et al. [1] propose the failure detector called $\mathcal{HB}$ (Heartbeat) that can be used together with an unreliable failure detector to solve Consensus in partitionable systems. Roughly speaking, the failure detector associates to each process an integer value that increases as long as the process remains reachable. This failure detector is used as a *complement* for other failure detectors, and not as a lower-level building block.

Several papers have proposed failure detectors that internally used some counters. However, these counters are used as an implementation technique and not as a means to separate decoupling system monitoring and interpretation. Bondavalli et al. [5] proposed the notion of α-count to distinguish between the transient and permanent/intermittent faults of system components. A value is associated with each component and incremented each time the component fails. When the value grows beyond a given threshold, the corresponding component is reported as permanently faulty. Chu [9], and later Mostefaoui et al. [27], present different algorithms to transform an unreliable failure detector into a leader oracle (also called Ω failure detector). The two algorithms are based on a similar approach. Briefly speaking, each process maintains a counter associated with each other process. The counters are incremented and processes exchange information on their values using gossiping. The process with the lowest value is deemed the most trustworthy and hence the most desirable candidate for a leader.

In recent work, Dunagan et al. [13] proposed a failure monitoring system called FUSE. They advocate providing failure detection as a global service and address several related engineering issues. In particular, they focus on providing consistent failure notifications in large-scale and wide-area networks.

Whereas FUSE addresses issues related to notifications about failures, accrual failure detectors provide a solution for detecting failures, and are thus useful components on top of which a system like FUSE could be built.

7 Conclusion

Failure detection constitutes a fundamental abstraction for fault tolerant distributed systems. However, from a more practical perspective, the binary model of classical failure detectors limit the development of failure detection as a generic service because this model combines monitoring and interpretation. The accrual failure detectors presented in this paper decouple these two tasks by outputting a suspicion level rather than a binary value, and leaving it to applications to interpret this value. Ideally, the monitoring is done by a single service running on each machine, while the interpretation of the suspicion level is left to each application process. Such a service can be implemented as a daemon, a linked library or a kernel service, depending on the desired tradeoff between intrusiveness and performance.

This paper gives a rigorous definition for accrual failure detectors that is compatible with the seminal work of Chandra and Toueg [7]. In particular, we presented important conditions for the suspicion level under which an accrual failure detector ($\Diamond\mathcal{P}_{ac}$) is *computationally equivalent* to an eventually perfect binary failure detectors (i.e., of class $\Diamond\mathcal{P}$). This equivalence is important because it shows that accrual failure detectors do not hide any additional synchrony assumptions with respect to their binary counterparts. However, equivalence does not imply that accrual failure detectors cannot be more efficient or expressive than binary failure detectors. In fact, we argued extensively the architectural advantages of accrual failure detectors, and presented usage patterns that are very difficult to handle using a binary failure detector.

We have also outlined four different ways to implement accrual failure detectors and discussed their respective advantages. This is not exhaustive and there is room for developing many other implementations in the future.

Acknowledgments

The example discussed in the introduction was kindly suggested to us by Francisco V. Brasileiro. The authors are also grateful to the following persons for their insightful comments and suggestions: Adel Cherif, Matti Hiltunen, Michel Raynal, Robbert van Renesse, Richard D. Schlichting, Yoichi Shinoda, Makoto Takizawa, and Paulo Veríssimo.

References

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

[2] M. Aguilera, C. Delporte-Gallet, H. Fauconnier, and S. Toueg. Thrifty generic broadcast. In M. Herlihy, editor, *Proc. 14th Intl. Symp. on Distributed Computing (DISC'00)*, *LNCS* 1914, pages 268–282, Oct. 2000.

[3] M. Bertier, O. Marin, and P. Sens. Implementation and performance evaluation of an adaptable failure detector. In *Proc. IEEE Intl. Conf. on Dependable Systems and Networks (DSN'02)*, pages 354–363, June 2002.

[4] M. Bertier, O. Marin, and P. Sens. Performance analysis of a hierarchical failure detector. In *Proc. IEEE Intl. Conf. on Dependable Systems and Networks (DSN'03)*, pages 635–644, June 2003.

[5] A. Bondavalli, S. Chiaradonna, F. Di Giandomenico, and F. Grandoni. Discriminating fault rate and persistency to improve fault treatment. In *Proc. 27th Intl. Symp. on Fault-Tolerant Computing (FTCS-27)*, pages 354–362, June 1997.

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

[7] T. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *J. ACM*, 43(2):225–267, 1996.

[8] W. Chen, S. Toueg, and M. Aguilera. On the quality of service of failure detectors. *IEEE Trans. on Computers*, 51(5):561–580, May 2002.

[9] F. Chu. Reducing Ω to $\Diamond\mathcal{W}$. *Inf. Process. Lett.*, 67(6):289–293, Sept. 1998.

[10] W. Cirne, F. Brasileiro, J. Sauvé, N. Andrade, et al. Grid computing for Bag-of-Tasks applications. In *Proc. 3rd IFIP Conf. on E-Commerce, E-Business and E-Goverment*, Sept. 2003.

[11] F. Cosquer, L. Rodrigues, and P. Veríssimo. Using tailored failure suspectors to support distributed cooperative applications. In *Proc. 7th IASTED Intl. Conf. on Parallel and Distributed Computing and Systems (PDCS'95)*, pages 352–356, Oct. 1995.

[12] X. Défago, P. Urbán, N. Hayashibara, and T. Katayama. On accrual failure detectors. RR IS-RR-2004-11, JAIST, Ishikawa, Japan, May 2004.

[13] J. Dunagan, N. Harvey, M. Jones, D. Kostic, et al. FUSE: Lightweight guaranteed distributed failure notification. In *Proc. 6th Symp. on Operating Systems Design and Implementation (OSDI'04)*, Dec. 2004.

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

[15] R. Ekwall, A. Schiper, and P. Urbán. Token-based atomic broadcast using unreliable failure detectors. In *Proc. 23nd IEEE Intl. Symp. on Reliable Distributed Systems (SRDS'04)*, pages 52–65, Oct. 2004.

[16] P. Felber, X. Défago, R. Guerraoui, and P. Oser. Failure detectors as first class objects. In *Proc. 1st Intl. Symp. on Distributed-Objects and Applications (DOA'99)*, pages 132–141, Sept. 1999.

[17] C. Fetzer, M. Raynal, and F. Tronel. An adaptive failure detection protocol. In *Proc. 8th IEEE Pacific Rim Symp. on Dependable Computing (PRDC'01)*, pages 146–153, Dec. 2001.

[18] R. Friedman. Fuzzy group membership. In A. Schiper, A. Shvartsman, H. Weatherspoon, and B. Y. Zhao, editors, *Future Directions in Distributed Computing*, *LNCS* 2584, pages 114–118, Jan. 2003. Position paper.

[19] R. Friedman and G. Tcharny. Evaluating failure detection in mobile ad-hoc networks. TR CS-2003-06, Technion, Israel, Oct. 2003.

[20] R. Friedman and G. Tcharny. Stability detection in mobile ad-hoc networks. TR CS-2003-12, Technion, Israel, Nov. 2003.

[21] N. Hayashibara. *Accrual Failure Detectors*. PhD thesis, JAIST, Ishikawa, Japan, June 2004.

[22] N. Hayashibara, X. Défago, and T. Katayama. Flexible failure detection with κ-fd. RR IS-RR-2004-006, JAIST, Ishikawa, Japan, Feb. 2004.

[23] N. Hayashibara, X. Défago, R. Yared, and T. Katayama. The φ accrual failure detector. In *Proc. 23nd IEEE Intl. Symp. on Reliable Distributed Systems (SRDS'04)*, pages 66–78, Oct. 2004.

[24] M. Hurfin, A. Mostéfaoui, and M. Raynal. A versatile family of consensus protocols based on Chandra-Toueg's unreliable failure detectors. *IEEE Trans. on Computers*, 51(4):395–408, Apr. 2002.

[25] A. Mostéfaoui, E. Mourgaya, and M. Raynal. Asynchronous implementation of failure detectors. In *Proc. IEEE Intl. Conf. on Dependable Systems and Networks (DSN'03)*, pages 351–360, June 2003.

[26] A. Mostéfaoui, D. Powell, and M. Raynal. A hybrid approach for building eventually accurate failure detectors. In *Proc. 10th IEEE Pacific Rim Intl. Symp. on Dependable Computing (PRDC)*, pages 57–65, Mar. 2004.

[27] A. Mostéfaoui, M. Raynal, and C. Travers. Crash-resilient time-free eventual leadership. TR, IRISA, Rennes, France, Apr. 2004.

[28] L. Sampaio, F. Brasileiro, W. Cirne, and J. Figueiredo. How bad are wrong suspicions? towards adaptive distributed protocols. In *Proc. IEEE Intl. Conf. on Dependable Systems and Networks (DSN'03)*, pages 551–560, June 2003.

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

A Appendix

This appendix presents technical details that are not essential to convey the ideas presented in the paper, but that are important to justify the correctness of the argumentation (e.g., complete proofs of the theorems).

A.1 Proof for the transformation from accrual ($\Diamond\mathcal{P}_{ac}$) to binary ($\Diamond\mathcal{P}$)

Lemma 7 (Strong completeness) *Given an accrual failure detector $\mathcal{D}_{ac}$ of class $\Diamond\mathcal{P}_{ac}$, Algorithm 1 satisfies the property of Strong Completeness.*

PROOF. Strong Completeness requires that every faulty process is eventually suspected forever by every correct process.

Consider a faulty process p and some correct process q that monitors p. We must show that there is a time after which p is permanently suspected by q.

Consider the system after the stabilization, i.e., when sl_{qp} is increasing and increases by at least once every Q consecutive queries for some positive Q.

We first show that there is a last T-transition (if T-transitions occur at all). We show this by proving that no more than Q T-transitions may occur after stabilization. Suppose that stabilization and Q T-transitions have already occurred. As the run length for trusting L_{trust} increases by one at each T-transition, $L_{trust} > Q$ forever. As only run lengths shorter than Q will ever appear, the condition $l > L_{trust}$ will never hold. Moreover, as sl_{qp} is monotonously increasing, $sl < sl_{prev}$ will never hold, either. Hence we have shown that the condition to trigger a T-transition ($sl < sl_{prev}$ or $l > L_{trust}$) will never hold again.

Now, consider a time after which both stabilization and the last T-transition have occurred. If p is suspected at this time, it will be suspected forever, and the proof is complete. If p is trusted at this time, an S-transition will occur, as sl_{qp} goes to infinity, and thus $sl > SL_{susp}$ will eventually hold forever, whatever the value of SL_{susp} may be. After the S-transition, p will be suspected forever, and the proof is complete. $\square_{\text{Lemma 7}}$

Lemma 8 (Eventual strong accuracy) *Given an accrual failure detector $\mathcal{D}_{ac}$ of class $\Diamond\mathcal{P}_{ac}$, Algorithm 1 satisfies the property of Eventual Strong Accuracy.*

PROOF. Eventual Strong Accuracy requires that, after some time, no correct process is suspected by a correct process. We prove the lemma by considering two correct processes p and q chosen arbitrarily, and such that q monitors p. We must show that there is a time after which p is never suspected. We can rely on the existence of a bound SL_{max} on the suspicion level sl_{qp}.

We first show that there is a last S-transition (if S-transitions occur at all). We show this by proving that no more than $\lceil SL_{max}/\epsilon \rceil$ S-transitions occur, where ϵ is the resolution of the suspicion level (see Definition 1). Suppose that $\lceil SL_{max}/\epsilon \rceil$ S-transitions have already occurred. As S_{susp} increases by at least ϵ upon every S-transition, $SL_{susp} \geq SL_{max}$, and hence $sl > SL_{susp}$ will never hold again. We have shown that the condition to trigger an S-suspicion will never hold again.

Now, consider the system after the last S-transition (or any time if no S-transition occurs). If p is trusted at this time, it will be trusted forever. If p is suspected at this time, a T-transition will occur, as we show below. After the T-transition, p will be trusted forever, and the proof is complete.

We must now prove that a T-transition occurs. Let s_i denote the value that s takes during the i-th query. If the sequence s_i is not monotonously increasing, the condition $sl < sl_{prev}$ will hold at least once, and thus a T-transition occurs. If s_i is monotonously increasing, there will be a run made of identical values of length $L_{trust} + 1$, as s_i is an infinite sequence whose elements are from a finite set $(0, \epsilon, 2\epsilon, \ldots, \lfloor SL_{max}/\epsilon \rfloor \cdot \epsilon)$. This implies that the condition $l > L_{trust}$ will hold at least once, and thus

a T-transition occurs. As p and q were chosen arbitrarily, we have shown that Eventual Strong Accuracy holds. $\square_{\text{Lemma 8}}$

Theorem 9 *Algorithm 1 transforms an accrual failure detector of class $\Diamond\mathcal{P}_{ac}$ into one of class $\Diamond\mathcal{P}$.*

PROOF. Follows from Lemma 7 (Strong completeness) and Lemma 8 (Eventual strong accuracy). $\square_{\text{Theorem 9}}$

A.2 Proof for the transformation from binary ($\Diamond\mathcal{P}_{ac}$) to accrual ($\Diamond\mathcal{P}$)

Lemma 10 (Accruement) *Given a binary failure detector $\mathcal{D}$ of class $\Diamond\mathcal{P}$, Algorithm 2 satisfies the property of Accruement (Prop. 1).*

PROOF. Consider a faulty process p and a correct process q. Since p is faulty, $\mathcal{D}$ ensures that p is eventually suspected permanently (Strong completeness). Consider the system after the last S-transition (or from the start if p is never trusted). $sl_{qp}(t)$ is obviously increasing by ϵ upon every query. We have thus shown that Accruement holds. $\square_{\text{Lemma 10}}$

Lemma 11 (Upper Bound) *Given a binary failure detector $\mathcal{D}$ of class $\Diamond\mathcal{P}$, Algorithm 2 satisfies the property of Upper Bound (Prop. 2).*

PROOF. Consider two correct processes p and q. Since p is correct, $\mathcal{D}$ ensures that p is eventually trusted permanently (Strong eventual accuracy). Consider a point in time after the last T-transition (or any time if p is never suspected). Every query after this point in time returns 0. A suitable upper bound for sl_{qp} is thus the highest suspicion level returned up to this point in time. We have thus shown that Upper bound holds. $\square_{\text{Lemma 11}}$

Theorem 12 *Algorithm 2 transforms $\Diamond\mathcal{P}$ into $\Diamond\mathcal{P}_{ac}$.*

PROOF. Follows directly from Lemma 10 (Accruement), and Lemma 11 (Upper bound). $\square_{\text{Theorem 12}}$

A.3 Proofs for the theorems about quality of service

In this section, we present the proofs for the theorems in Section 4.4. We start by presenting Algorithm 3, an unambiguous description of the failure detector $\mathcal{D}'_T$ (where T is a threshold function).

Algorithm 3 Transforming an accrual failure detector into a binary one using two thresholds.

```
1: Initialization:
2:     status := trust                                  {current status (trust or suspect)}
3: when queried about process p
4:     sl := sl_qp(t)                                   {get current suspicion level}
5:     if sl > T(t) and status = trust then             {suspect if level beyond high threshold}
6:         status := suspect
7:     if sl ≤ T_0(t) and status = suspect then         {trust if level below low threshold (or equal)}
8:         status := trust
```

PROOF. (for Theorem 1). The proof is straightforward for $\mathcal{D}_{T_1}$ and $\mathcal{D}_{T_2}$:

$$\begin{aligned} \mathcal{D}_{T_2} \text{ suspects } p \text{ at } t &\Rightarrow sl_{qp}(t) > T_2(t) \\ &\Rightarrow sl_{qp}(t) > T_1(t) \Rightarrow \mathcal{D}_{T_1} \text{ suspects } p \text{ at } t \end{aligned}$$

The proof is more complicated for $\mathcal{D}'_{T_1}$ and $\mathcal{D}'_{T_2}$. Consider a time t when $\mathcal{D}'_{T_2}$ suspects p. Let t_0 be the time of the preceding S-transition (or the time of the first query if there is no preceding S-transition) and $t_1 < t_2 < \cdots < t_K = t$, the times when the failure detectors are queried between t and the preceding S-transition. We know that both failure detectors suspect p at t_0:

$$\begin{aligned} \mathcal{D}'_{T_2} \text{ suspects } p \text{ at } t_0 &\Rightarrow sl_{qp}(t_0) > T_2(t_0) \\ &\Rightarrow sl_{qp}(t_0) > T_1(t_0) \Rightarrow \mathcal{D}'_{T_1} \text{ suspects } p \text{ at } t_0 \end{aligned}$$

We then prove by induction that both failure detectors suspect p at $t_k = t$ as well. The induction step is the following, for any $k = 1, \ldots, K$:

$$\begin{aligned} (\mathcal{D}'_{T_2} \text{ suspects } p \text{ at } t_{k-1} \Rightarrow \mathcal{D}'_{T_2} \text{ suspects } p \text{ at } t_k) &\Rightarrow sl_{qp}(t_k) > T_0(t_k) \\ &\Rightarrow (\mathcal{D}'_{T_1} \text{ suspects } p \text{ at } t_{k-1} \Rightarrow \mathcal{D}'_{T_1} \text{ suspects } p \text{ at } t_k) \end{aligned}$$

As t was chosen arbitrarily, this completes the proof. $\square_{\text{Theorem 1}}$

PROOF. (for Theorem 4)

Let the query preceding the query at time t happen at time t_0. We know that $\mathcal{D}'_{T_2}$ trusts p at t but suspects p at t_0, hence $sl_{qp}(t) \leq T_0(t)$. Theorem 1 ensures that also $\mathcal{D}'_{T_1}$ suspects p at t_0. This and $sl_{qp}(t) \leq T_0(t)$ implies that also $\mathcal{D}'_{T_1}$ makes a T-transition at t. $\square_{\text{Theorem 4}}$

A.4 Simple implementation in a partially synchronous model

We now describe, in further details, the simple implementation outlined in Section 5.1, and show that it implements an accrual failure detector of class $\Diamond\mathcal{P}_{ac}$ in a partially synchronous system model. Notice that the algorithm described in this section is intended as a simple illustration.

Partially synchronous system model. We extend the model of Section 2, by considering that processes communicate only by message-passing. We assume that processes have their own memory space.[7] Also, channels are reliable, and we consider only crash failures of processes. We assume a partially synchronous model, as defined by Chandra and Toueg [7], where some *unknown* bounds on process speed and message delays hold after some *unknown* time called GST (stands for global stabilization time).[8] We also assume a local clock that has a bounded drift from global time after GST. The local clock is accessed by the function now. More precisely, the bounded drift means that $now(t') - now(t) > \theta \cdot (t' - t)$ for all $t' > t > GST$ and some $\theta > 0$.

Algorithm. The algorithm (Algorithm 4) is based on heartbeats and is actually quite simple. The code of the algorithm, identical for all processes, is expressed for some arbitrary process $q \in \Pi$. A monitored process sends heartbeat messages on a regular basis, according to its own local clock. Heartbeats are sequence numbered. A monitoring process q keeps track of the time of arrival $T_{last}(p)$ (according to its own local clock) of the most recent heartbeat message from a monitored process p. The value of the function $sl_{qp}(t)$ is given by the time elapsed since the arrival of the most recent heartbeat (according to the local clock of the monitoring process).

[7] This means that variables are *not* shared between processes. Although the same variable name (say, T_{last}) may be employed by two different processes (say, p and q), this always refers to two *distinct* variables (that is, T_{last} of p and T_{last} of q).

[8] This model is in fact a simple variation over the definitions of partial synchrony due to Dwork et al. [14].

Algorithm 4 Simple implementation of an accrual failure detector.

code of some process $q \in \Pi$:

```
1:  Initialization:
2:     start := now
3:     next_sn := 1                                  {Sequence number for the next heartbeat}
4:     forall p in Π − {q} do
5:        T_last(p) := start                         {Arrival time of the last heartbeat from each process}
6:        SN_last(p) := 0                            {Seq. number of the last heartbeat received}
7:  when receive (heartbeat, sn) from p              {receive heartbeat with sequence number sn}
8:     if sn > SN_last(p) then
9:        T_last(p) := now
10:       SN_last(p) := sn
11: periodically do
12:    broadcast (heartbeat, next_sn)
13:    next_sn := next_sn + 1
14: when queried about process p at time t
15:    if p ≠ q then
16:       sl_qp(t) := t − T_last(p)                  {rounded to the precision ε}
17:    else
18:       sl_qp(t) := 0
```

Lemma 13 *Algorithm 4 satisfies Prop. 1 (Accruement) for sl_{qp} , where p and q are two distinct processes in* Π.

PROOF. Let process p be faulty. It is sufficient to prove that after some stabilization time, $sl_{qp}(t)$ is increasing, and that it only remains constant during a bounded number of queries.

Since p crashes, it can send only a finite number of heartbeat messages. Let t_0 be the time when the heartbeat with the greatest sequence number ever received arrives. At this time, the algorithm updates $T_{last}(p)$ to $now(t_0)$ (i.e., t_0 in local time) and this value never changes again. It follows that, for any time t after t_0, $sl_{qp}(t) = now(t) - now(t_0)$.

This function is increasing after t_0. We still have to find a number of queries Q such that sl_{qp} increases at least once every Q queries (after t_0):

$$sl_{qp}(t_q^{query}(k+Q)) - sl_{qp}(t_q^{query}(k)) \geq \epsilon$$

Using that the drift of the local clock is bounded by θ, and that at least δ time elapses between queries (see Section 2), we obtain a lower bound for $sl_{qp}(t_q^{query}(k+Q)) - sl_{qp}(t_q^{query}(k))$:

$$now(t_q^{query}(k+Q)) - now(t_q^{query}(k+Q)) > \theta \cdot (t_q^{query}(k+Q) - t_q^{query}(k)) \geq \theta \cdot (Q \cdot \delta)$$

We need to ensure that this lower bound is at least ϵ. Thus $Q = \lceil \epsilon/\delta\theta \rceil$ is a suitable choice for ensuring that $sl_{qp}(t)$ increases at least once every Q queries.

$\square_{\text{Lemma 13}}$

Lemma 14 *Algorithm 4 satisfies Prop. 2 (Upper Bound) for sl_{qp} , where p and q are two distinct processes in* Π.

PROOF. Let process p be correct. We must prove that $sl_{qp}(t)$ is bounded. All times that appear are in local time.

Let t_1 be the time when the first heartbeat message H_1 sent after GST arrives. Clearly, until t_1, $sl_{qp}(t)$ is bounded by $t_1 - start$.

After t_1, only heartbeat messages with a higher sequence number than H_1, hence sent *after* H_1, are taken into account. It follows that they are subject to the synchrony assumptions of the model. Let Δ be the end-to-end upper bound on transmission time, and Δ' the maximal interval between the sending of two consecutive heartbeats. It follows that the largest interval that elapses between receiving two consecutive heartbeats is $\Delta + \Delta'$.

Combining the two parts, we obtain that $sl_{qp}(t)$ is bounded by $\max(t_1 - start, \Delta + \Delta')$.

$\square_{\text{Lemma 14}}$

Theorem 15 *Algorithm 4 implements an accrual failure detector of class $\Diamond\mathcal{P}_{ac}$.*

PROOF. The proof follows directly from Lemma 13 and Lemma 14, as these lemmas hold for an arbitrarily chosen pair of processes. $\square_{\text{Theorem 15}}$

A.5 Discussion on the Accruement property

When defining the suspicion level function in Section 3.1, we mentioned that there was a simple alternative to Accruement (Property 1):

Property 3 (Weak Accruement) *If process p is faulty, then eventually, the suspicion level $sl_{qp}(t)$ is monotonously increasing and goes to infinity:*

$$\lim_{t \to +\infty} sl_{qp}(t) = +\infty$$

We now prove that this property is not strong enough for our purposes. In particular, it does not allow the implementation of a $\Diamond\mathcal{P}$ binary failure detector on top of a $\Diamond\mathcal{P}_{ac}$ accrual failure detector (defined with Weak Accruement).

For simplicity, we present the proof for two processes p and q, with q monitoring p. Consider any algorithm A that implements $\Diamond\mathcal{P}$, and an adversary that controls the suspicion level sl_{qp}. Let the adversary reply to queries using the following strategy:

1. If the algorithm A suspects p, then keep sl_{qp} constant, i.e., return the result of the previous query.
2. If the algorithm A trusts q, then increase sl_{qp} by ϵ with respect to the result of the previous query.

At any time t, the history of the accrual failure detector produced by the adversary satisfies both Properties 2 and 3. Nevertheless, algorithm A does not implement a $\Diamond\mathcal{P}$ failure detector. The proof is indirect: suppose that A implements a $\Diamond\mathcal{P}$ failure detector.

1. If p is faulty, A must eventually suspect p forever. Suppose that A suspects p from time t on. sl_{qp} will be constant after t, and thus it does not go to infinity. Hence Weak Accruement (Property 3) does not hold, and thus p must be correct.
2. If p is correct, A must eventually trust p forever. Suppose that A trusts p from time t on. After t, sl_{qp} will grow by ϵ upon every query, hence it goes to infinity. Hence Upper Bound (Property 2) does not hold, and thus p must be faulty.

We have come to a contradiction whether p is correct or faulty, hence we have shown that no algorithm A can implement a $\Diamond\mathcal{P}$ binary failure detector on top of our adversary.

Finally, note why this adversary does not work with $\Diamond\mathcal{P}_{ac}$ accrual failure detectors defined by Accruement (Property 1). The reason is that it keeps sl_{qp} constant for an arbitrarily long period if p is faulty, whereas the Accruement property disallows this: sl_{qp} may only remain constant during a limited number of queries if p is faulty (although the limit is unknown).