Impact: an Unreliable Failure Detector Based on Processes' Relevance and the Confidence Degree in the System

Anubis G. M. Rossetto, Luciana Arantes, Pierre Sens, Claudio R. Geyer

Impact: an Unreliable Failure Detector Based on Processes' Relevance and the Confidence Degree in the System

Anubis G. M. Rossetto, Luciana Arantes, Pierre Sens , Cláudio F. R. Geyer

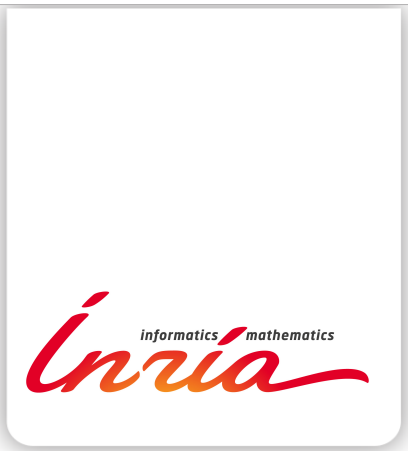
RESEARCH REPORT

N° hal-01136595

Mar 2015

Project-Teams Regal(INRIA) and GPPD (UFRGS University, Brazil)

ISSN 0249-6399 ISRN INRIA/RR--hal-01136595--FR+ENG



Impact: an Unreliable Failure Detector Based on Processes' Relevance and the Confidence Degree in the System

Anubis G. M. Rossetto*, Luciana Arantes†, Pierre Sens ‡, Cláudio F. R. Geyer§

Project-Teams Regal(INRIA) and GPPD (UFRGS University, Brazil)

Research Report n° hal-01136595 — Mar 2015 — 36 pages

Abstract: This technical report presents a new and flexible unreliable failure detector, denoted Impact Failure Detector (FD), whose output gives the trust level of a set of processes. By expressing the relevance of each node by an impact factor value as well as an acceptable margin of failure in the system, the Impact FD enables the user to tune the failure detection configuration in accordance with the requirements of the application: in some scenarios, the failure of low impact or redundant nodes does not jeopardize the confidence in the system, while the crash resulting from a high impact factor may seriously affect it. Either a softer or stricter monitoring is thus possible.

Key-words: unreliable failure detector, impact factor, trust level, flexibility, node redundancy, fault tolerance.

* Federal Institute of Education, Science and Technology Sul-rio-grandense - Campus Passo Fundo, Passo Fundo, Brazil
† Laboratoire d'Informatique de Paris 6 (LIP6), Université Pierre et Marie Curie, CNRS, INRIA, Paris, France
‡ Laboratoire d'Informatique de Paris 6 (LIP6), Université Pierre et Marie Curie, CNRS, INRIA, Paris, France
§ Institute of Informatics, Federal University of Rio Grande do Sul, Porto Alegre, Brazil

RESEARCH CENTRE
PARIS – ROCQUENCOURT

Domaine de Voluceau, - Rocquencourt
B.P. 105 - 78153 Le Chesnay Cedex

Impact:an Unreliable Failure Detector Based on Processes' Relevance and the Confidence Degree in the System

Résumé : Ce rapport technique présente un nouveau détecteur de défaillance non fiable, le *Impact Failure Detector (FD)*, dont la sortie correspond au niveau de confiance d'un ensemble de processus. En exprimant l'importance de chaque noeud par une valeur d'impact ainsi qu'une marge acceptable de défaillance du système, le détecteur de défaillance Impact permet á l'utilisateur d'ajuster la configuration de détection de défaillance selon les exigences de l'application : dans certains scénarios, la panne d'un noeud de faible impact ou des noeuds redondants ne compromette pas la confiance sur le système, tandis que la panne d'un noeud avec un facteur d'impact élevé peut sérieusement en compromettre. Par conséquent, une surveillance plus faible ou plus strictes est possible.

Mots-clés : détecteur de défaillance non fiable, valeur d'impact, niveau de confiance, redondancede noeuds, tolerance aux pannes.

Contents

1 Introduction

An unreliable FD can be seen as an oracle that gives (not always correct) information, about process failures. Most of them are based on a binary model, in which monitored processes are either "trusted" or "suspected". Thus, the majority of existing FDs, such as those defined in [7] [3], output the set of processes that is currently suspected to have crashed. A non-binary approach is adopted in [13], the accrual failure detector, which outputs a suspicion level on a continuous scale.

This paper presents a new unreliable failure detector, denoted *Impact Failure Detector* (*Impact FD*), whose output is a trust level concerning a set S of monitoring processes. The output can be considered as the degree of confidence in S, i.e., the confidence in the system as a whole. Hence, an *impact factor* value is assigned to each process of the set. Furthermore, a *threshold* parameter defines a limit value above which the confidence degree of the set is not compromised. The *impact factor* indicates the relative importance of the process in the set S, while the *threshold* offers a degree of flexibility for failures and false suspicions, thus allowing a higher tolerance of instability in the system. For instance, in an unstable network, although there might be many false suspicions, depending on the value assigned to the *threshold*, the system might remain trustworthy.

If the output value of the impact FD, denoted *trust level*, is below the limit defined by the *threshold*, the user decides which measures should be taken and whether the latter are urgent or not, with regard to the value of the trust level. In the concept of the Impact FD, the monitored processes can also be grouped, based on some criterion such as process type (e.g. nodes, sensors) or their relevance. It is worth remarking that, together with the *threshold*, the group feature of the Impact FD can characterize processes redundancy.

The Impact FD is easily configurable to the needs of the problem, type of accuracy, or system configuration. Moreover, its output does not depend on the identity of the processes. In section 5, we show two Impact FD implementations applied to anonymous[1] [4] and homonymous [2] [2] systems respectively.

2 Motivation Scenarios

The Impact FD can be applied to different distributed scenarios and it is flexible enough to meet different needs. It is quite suitable for environments where there is node redundancy or nodes with different capabilities. The following examples illustrate scenarios where the Impact FD could be useful.

2.1 Healthcare Area

A system in the area of healthcare requires the use of several sensors to measure different kinds of information about the health status of a person, such as, vital signs, location, falls, gait patterns, and acceleration. From this perspective, this scenario is critical since faults in the components can risk the patient's life. For instance, let's consider a scenario with four sensors: q_1 - *body temperature;* q_2 - *pulse;* q_3 - *electrocardiogram(ECG); and* q_4 - *galvanic skin* as well as node p, responsible for collecting information from these sensors and taking appropriate action based on the output of the Impact FD. In this example, some sensors are not considered to be critical, such as the sensor q_1 which measures the temperature. On the other hand, q_3, the ECG sensor,

[1] The anonymous processes cannot be distinguished one each other: they have no name and execute the same code

[2] Homonymous means that several processes may have the same identifier

is crucial. Therefore, the impact factor assigned to q_3 should be higher than q_1*'s*. Furthermore, q_3 collects data about both the heartbeats and electrical activity of the heart while q_2 is a type of sensor that also collects data about the heartbeats. Hence, there is redundancy of information, i.e., the failure of q_2 sensor is not critical enough to make the system vulnerable and endanger the life of the monitored person. We could then define a threshold equals to the sum of the impact factor of all the sensors minus q_2*'s* impact factor since, the failure of q_2 does not jeopardize the trustness of the system.

2.2 Ubiquitous Wireless Sensor Network

Another important scenario that motivates our proposal is Ubiquitous Wireless Sensor Network (WSNs). These kinds of networks are deployed to monitor physical conditions in various places such as geographical regions, agriculture lands, battlefields, etc. In WSNs, there are a variety of sensor nodes with different battery resources and communication or computation capabilities [14]. However, these sensors are prone to failures (e.g., battery failure, process failure, transceiver failure, etc.) [11]. Hence, it is necessary to provide failure detection and adaptation strategies in order to ensure as much as possible that the failure of sensor nodes does not affect the overall task of the network. Redundant use of sensor nodes, reorganization of sensor network, and overlapped sensing regions are some of the techniques used to increase the fault tolerance and reliability of the network [1].

Let us take as an example a ubiquitous WSN which collects environmental data within a vineyard, which is grouped into management zones according to different characteristics (e.g., soil properties). Each zone is composed of sensors of different types (e.g., humidity control sensors, temperature control sensors, etc.) and the density of sensors in each zone depends on the characteristics of the latter. That is, the number of sensors can be different for each type of sensor within a given zone. Furthermore, sensors' redundancy ensures both area coverage and connectivity in case of failure. We can thus consider each management zone as a single set whose sensors of the same type are grouped into subsets. Such grouping approach enables the definition of a threshold which is equal to the minimum number of sensors that each subset must have in order to keep connectivity and the application functioning all time. Moreover, in some situation, there might be a need to dynamically reconfigure the density of the zones. In this case, the *threshold* value would change.

2.3 Replicated Servers

A third example might be a system with a main server that offers a certain quality of service X (bandwidth, response time, etc.). If it fails, N backup servers can replace it, since each backup offers the same service but with a X/N quality of service. In this scenario, both the impact factor of the main server and the *threshold* would have the value of $N * Iback$ where $Iback$ is the impact value of the backup servers, i.e., the system becomes unreliable whenever the primary server and one or more of the N servers fail (or are suspected of being faulty).

3 Failure Detectors and Synchrony Models

An important abstraction for the development of fault tolerant distributed systems is the unreliable failure detector [7]. The aim of the latter is to encapsulate the uncertainty of the communication delay between two distributed entities.

An unreliable FD can be seen as an oracle that gives (not always correct) information about process failures (either trusted or suspected). It usually provides a list of processes suspected of

having crashed. According to [12], an unreliable FD module can make mistakes (1) by erroneously suspecting a correct process (false suspicion), or (2) by not suspecting a process that has actually crashed. If the FD detects its mistake later, it corrects it. For instance, a FD can stop suspecting at time $t + 1$, a process that it suspected at time t.

Unreliable failure detectors are characterized by two properties, *completeness* and *accuracy*, as defined in [7]. Completeness characterizes the failure detector's capability of suspecting faulty processes, while accuracy characterizes the failure detector's capability of not suspecting correct processes, i.e., restricts the mistakes that the failure detector can make. FDs are then classified according to two completeness properties and four accuracy properties [7]:

- Strong completeness: Eventually every process that crashes is permanently suspected by every correct process.
- Weak completeness: Eventually every process that crashes is permanently suspected by some correct process.
- Strong accuracy: No process is suspected before it crashes.
- Weak accuracy: Some correct process is never suspected.
- Eventual strong accuracy: There is a time after which correct processes are not suspected by any correct process.
- Eventual weak accuracy: There is a time after which some correct process is never suspected by any correct process.

The combination of these properties yield eight classes of failure detectors as can be seen in Table 2.

Table 1: Failure detectors classification

Completeness	**Accuracy**			
	Strong	**Weak**	**Eventual strong**	**Eventual weak**
Strong	Perfect P	Strong S	Eventual Perfect $\diamondsuit P$	Eventual Strong $\diamondsuit S$
Weak	Q	W	$\diamondsuit Q$	$\diamondsuit W$

3.1 Classes Sigma(Σ) and Omega (Ω)

Failure detectors of class Σ (Sigma or quorum) output, for any failure pattern, any time t, and any process p_i, a set of processes that are said to be *trusted* by p_i at time t, such that the two following properties are satisfied [10]:

- Every two sets of trusted processes always intersect;
- Eventually every trusted process is correct;

The failure detector of class Ω ensures that, eventually, each process in the system will be provided by an unique leader, elected among the set of correct processes, in spite of crashes. In other words, eventually, a single correct process is permanently trusted by all correct processes [10]. The Omega failure detector has been shown to be the weakest failure detector for solving consensus when a majority of processes are correct.

In [2], the authors define new classes of failure detectors suited to homonymous distributed systems where processes are prone to crash failures and have no initial knowledge of the system membership ("homonymous" means that several processes may have the same identifier). First, the paper introduce the classes $H\Sigma$ and $H\Omega$ (that are homonymous counterparts of the classes Σ and Ω, respectively) and shows how they can be implemented in homonymous systems without membership knowledge and different synchrony requirements.

The authors also presents two consensus algorithms for asynchronous homonymous systems where there is no initial knowledge of the membership. The consensus is implemented with the counterparts of the weakest failure detectors in classical message passing systems with unique processes' identities: Ω when a majority of processes are correct (its counterpart is called $H\Omega$, and $\langle \Omega\, \Sigma \rangle$ when a majority of processes can crash (its counterpart is called $\langle H\Omega, H\Sigma \rangle$.

3.2 Synchrony Models

Partial Synchronous System: A distributed system is partial synchronous if there are bounds on transmission delay, clock drift and processing time and these bounds are unknown

Asynchronous System: A distributed system is asynchronous if there is no bound on message transmission delay, clock drift, or the time to execute a processing step.

Synchronous System: A distributed system is synchronous if there are bounds on transmission delay, clock drift and processing time and these bounds are known.

3.3 Implementation of Failure Detectors

The literature has several proposals for implementing unreliable failure detectors which usually exploit either a *timer-based* or a *message-pattern* approach.

In the *timer-based* strategy, FD implementations make use of timers to detect failures in processes. Every process q periodically sends a control message (*heartbeat*) to process p that is responsible for monitoring q. If p does not receive such a message from q after the expiration of a timer, it adds q to its list of suspected processes. The use of *timeouts* assumes that the system is either synchronous or partial synchronous.

The *message-pattern* strategy does not use any mechanism of timeout. In [18], the authors propose an implementation that uses a request-response mechanism. A process p sends a $QUERY$ message to n nodes that it monitors and then waits for responses ($RESPONSE$ message) from α processes ($\alpha \leq n$, traditionally $\alpha = n - f$, where f is the maximum number of failures). A query issued by p ends when it has received α responses. The other responses, if any, are discarded and the respective processes are suspected of having failed. A process sends $QUERY$ messages repeatedly if it has not failed. If, on the next request-response, p receives a response from a suspected process q, then p removes q from its list of suspects. This approach considers the relative order for the receiving of messages which always (or after a time) allow some nodes to communicate faster than the others.

4 Related Work

Some important studies, that have been addressed to failure detectors, are reviewed in this section.

Aiming at reducing both the number of false suspicions and the time needed to detect a failure, Chen et al. [8] propose an approach to estimate the arrival of the next heartbeat which is based on the history of the arrival time of heartbeats and includes a safety margin (β). The timer is then set according to this estimation.

The estimation algorithm is the following: process p takes into account the n most recent heartbeat messages received from q, denoted by $m_1, m_2, \ldots, m_n$; $A_1, A_2, \ldots, A_n$ are their actual reception times according to p's local clock. When at least n messages have been received, the theoretical arrival time $EA_{(k+1)}$ for a heartbeat from q is estimated by:

$$EA_{(k+1)} = \frac{1}{n} \sum_{i=k-n}^{k} (A_i - \Delta_i * i) + (k+1)\Delta_i$$

where Δ_i is the interval between the sending of two q's heartbeats. The next timeout value which will be set in p's timer and will expire at the next freshness point $\tau_{(k+1)}$, is then composed by $EA_{(k+1)}$ and the constant safety margin β:

$$\tau_{(k+1)} = \beta + EA_{(k+1)}$$

Bertier et al. [3] introduced a failure detector principally intended for LAN environments. They propose a different estimation function, which combines Chen's estimation with another estimation, due to Jacobson [15] and developed in a different context. It adapts the safety margin each time it receives a message. The adaptation is based on the error in the last estimation. Its proposition provides a shorter detection time, but generate more wrong suspicions than when Chen's estimation.

The ϕ accrual failure detector [13] proposes a approach where the output is a suspicion level on a continuous scale, instead of providing information of a binary nature (trust or suspect). The suspicion level captures the degree of confidence with which a given process is believed to have crashed. If the process actually crashes, the value is guaranteed to accrue over time and tend toward infinity.

According authors, the accrual failure detectors provide a lower level abstraction that avoids the interpretation of monitoring information. The suspicion level is left for the applications to interpret. For instance, by setting an appropriate threshold, applications can trigger suspicions and perform appropriate actions. Besides, applications can directly use the value output by the detector as a parameter to their actions.

Considering two processes p and q, with q monitoring p, the suspicion level of process q monitoring process p expresses the confidence of q in the statement that p is faulty. So, the output of the failure detector of q over time can be represented by the function $susp\_level_p(t) \geq 0$ ("suspicion level of p").

The authors present a specific implementation which the accrual failure detector dynamically adapt current network conditions based on the suspicion level expressed. The overall mechanism is described in Fig. 1. Information flow in the proposed implementation of the ϕ failure detector, as seen at process q. Heartbeats arrival from the network and their arrival time are stored in the sampling window. Past samples are used to estimate some arrival distribution. The time of last arrival T_{last} , the current time t_{now} and the estimated distribution are used to compute the current value of ϕ. Applications trigger suspicions based on some threshold (Φ 1 for App. 1 and Φ 2 for App. 2), or execute some actions as a function of ϕ (App. 3).

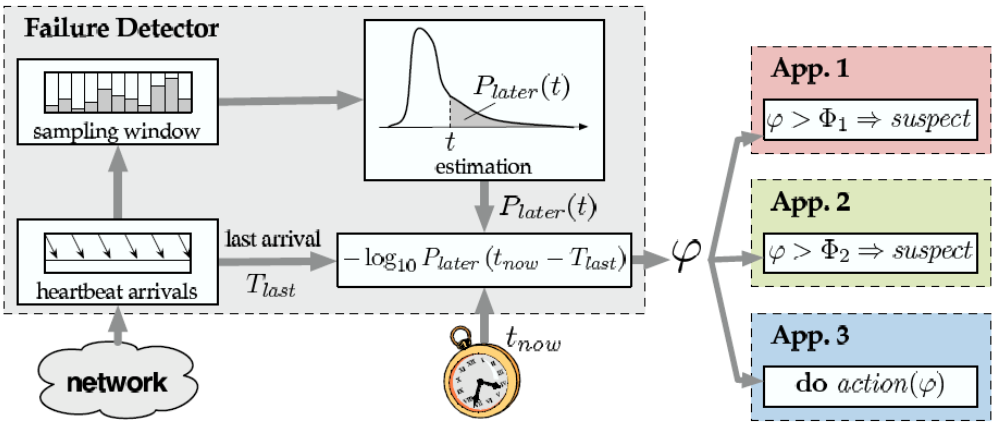


Figure 1: Information flow in the implementation of the accrual failure detector

Sung-Jib Yim and Yoon-Hwa Choi [21] propose the concept of sensor nodes' confidence levels in an adaptive fault-tolerant event detection for wireless sensor network. Analyzing the consistency of detecting results of nodes, confidence levels of nodes are dynamically adjusted, and they are used to adjust the threshold, which is used for decision making. Besides, it employs a filter for tolerating transient faults to correct some erroneous sensor readings. The approach models a sensor network as a weighted directed graph, G(V, E), where V represents the set of sensor nodes and E represents the set of edges connecting sensor nodes. Two nodes v_i and v_j are said to be connected if the distance between them dist(v_i,v_j) is less than or equal to r (transmission range). Each node vi is assigned a self-confidence level c_i. Each edge e_ij is also assigned a weight w_ij, indicating the confidence level of v_j from the viewpoint of v_i. The solution uses the confidence levels to isolate potentially faulty sensor nodes from the rest of the network. In addition, they are used to reinstate an isolated node if the confidence levels associated with it satisfy the required conditions to be addressed shortly. There are a range of the confidence level c_i (c_{min} and c_{max}) and a range of w_ij (w_{min} and w_{max}. Fig. 2 shows an illustration where six nodes are neighbors of the node v_3 (i.e. six nodes are located within the communication range of v_3) and confidence levels c_i and w_ij are assumed to be in the range of 0 to 1. From the viewpoint of node v_3, the highest confidence are nodes v_2 and v_4, while v_5 is a node with the lowest confidence. The confidence levels are updated each time a fault detection or event detections is performed. All the c_i and w_ij are initialized to 1 (i.e. c_max and w_max and they are increased or decreased by $\propto$ ($0 \propto < 1$) when the required conditions to be explained later are met. Node v5 is the most likely to be faulty and will be ignored from v_3 if $w_min = 0$.

Cosquer et al. [9] describe a group membership service that allows the tunning of its failure detection by monitoring several system parameters combined internally into a single value. The Failure Suspector (FS) is designed to evaluate a number of relevant operational parameters, such as: roundtrip delay, throughput and error rate. The application can configure the failure suspector, indicating which parameters must be measured ant their acceptable values. Each parameter is characterized by a set of variables, as illustrated in Fig. 3: SR (Sample Rate) - the maximum interval at which the parameter needs to be evaluated; TH(Threshold) - the parameter maximum acceptable value; W (Weight) - a measure of the parameter relative importance. A weight of 0 indicates that the parameter is not taken into account; TE (Threshold Exceeded) - the number of samples at which the parameter value has exceeded its threshold; Value (V) - the parameter current value. At every node, a complete set of variables for a given parameter is

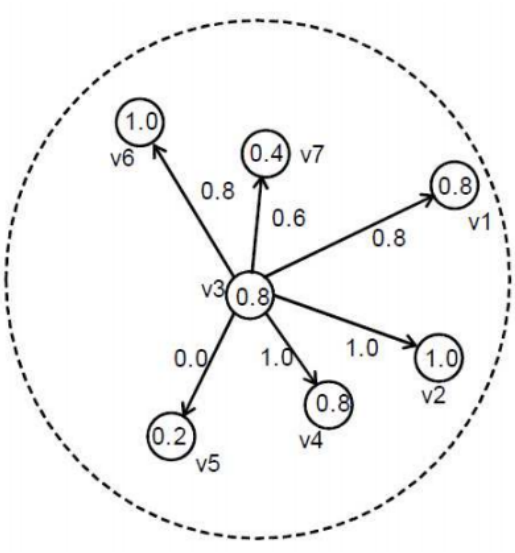


Figure 2: An illustration of confidence levels

kept for every other reachable node in the group. They denoted variable V is associated with P kept at local node l about some remote node r by $V_l{}^r(P)$. Disturbance index (DI) is a index to measure the overall connectivity to a remote node. DI is a sum of products of TE variables with their corresponding weight:

$$DI_l^r = \sum_j Weight_l^r(j) * TE_l^r(j)$$

The DI indicates how often operational parameters have exceeded the defined thresholds. A very low DI value means that the connectivity is satisfactory. A high value means very poor connectivity. The DI value is used to build a Local Connectivity Vector (LCV_l), an array of boolean that indicates whether the Disturbance Index has exceeded its threshold. Whenever the $\text{DI}_l{}^r$ threshold for given node r is exceeded, that node is suspected, and LCV_l[r] is set to false.

Operational Parameters Set-Up				**node**$_1$		**node**$_2$		...	**node**$_i$		...	**node**$_n$	
Parameter	**SR**	**TH**	**W**	**TE**	V_i^1	**TE**	V_i^2	...	**TE**	V_i^i	...	**TE**	V_i^n
roundtrip (s)	1m	10	2	1	12	2	25	...	-	-	...	0	0.3
throughput (Mbits/s)	5m	1	1	0	2	1	0.5	...	-	-	...	0	4
...	.	.	0	.	.	.	.	...	-	-	...	.	.
Disturbance Index (DI)	1m	3	-	(1*2)+(0*1)=2		(2*2)+(1*1)=5		...	-		...	(0*2)+(0*1)=0	
				↓		↓			↓			↓	
LCV$_{node_i}$ →				**1**		**0**		...	-		...	**1**	

Figure 3: Variables associated with operational parameters at $node_i$

Starting from the premise that applications should have information about failures to take

specific and suitable recovery actions, the work in [17] proposes a service to report faults to applications. The latter also encapsulates uncertainty which allows applications to proceed safely in the presence of doubt. The service provides status reports related to fault detection with an abstraction that describes the degree of uncertainty.

Considering that each node has a probability of being byzantine, a voting node redundancy approach is presented in [6] in order to improve reliability of distributed systems. Based on such probability values, the authors estimate the minimum number of machines that the system should have in order to provide a degree of reliability which is equal to or greater than a threshold value.

Considerations: In one way or another, the cited works have some characteristic that we seek in the detector that we are proposing. Now we discuss the main contributions of Impact Failure Detector with respect to the related work.

The works [9] and [21] are not in fact classes of failures detectors, but they address interesting aspects related to treatment failure. In [9], the authors presents an interesting proposal with regard the configuration according to application requirements, but with some limitations and maintaining the use of boolean output (trusted/suspected). [21] presents the concepts of confidence level to manage the status of sensor and a scheme based on cutting threshold allows each sensor to estimate locally if an failure event occurred. But they are applied to each single process and not to a set.

On the basis of this review, it is clear that none of others detectors deals with subsets and process relevance. In addition, only Accrual enables tuning the output. Moreover, the Impact FD outputs a set with trust level of each subset which render the estimation of the confidence in the monitored set. This means that a failure detector which covers these features, can represent a novel contribution for deal scenarios which it is easily configurable according to the needs of the system.

5 Impact FD

In this section we describe the Impact Failure Detector. Firstly, we give some definitions used by the Impact FD, including the concept of subset domains. Then, we present some implementations of the Impact FD, considering different synchronous model and type of processes (e.g. anonymous, homonymous, etc.)

5.1 Definitions

We consider a distributed system which consists of a finite set [3] of processes $\Pi = \{q_1, \ldots, q_n\}$ with $|\Pi| = n$. Failures are only by crash. Other types of failures (e.g. misbehavior, transient, etc) are the object of a study that will be carried out in the near future. A crashed process never recovers. We assume the existence of some global time denoted T. A failure pattern is a function $F : T \rightarrow 2^{\Pi}$, 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, i.e., those that have never belonged to a failure pattern (F), while $faulty(F)$ denotes the set of faulty processes, i.e., the complement of $correct(F)$ with respect to Π.

The Impact FD can be defined as an unreliable failure detector that provides an output related to the trust level with regard to a set of processes. If the trust level provided by the detector, is equal to, or greater than, a given threshold value, defined by the user, the confidence

[3] In this work, 'set' and 'multiset' are used interchangeably. Unlike a set, an element of a multiset can appear more than once. This allows different processes to have the same identity.

in the set of processes is ensured. We can thus say that the system is trusted. We denote FD (${I_p}^S$) the Impact failure detector module of process p and S is a set of processes of Π. When invoked in p, the Impact FD (${I_p}^S$) returns the ${trust\_level_p}^S$ value which expresses the confidence that p has in the set S.

Impact Factor and Subsets

Each process $q \in S$ has an *impact factor* value ($I_q | I_q > 0 : I_q \in \mathbb{R}$). Furthermore, the set S can be partitioned into m disjoint subsets. Notice that the grouping feature of the Impact FD allows the process of S to be partitioned into disjoint subsets, in accordance with a particular criterion. For instance, in a scenario where there are different types of sensors, those of the same type can be gathered in the same subset. Let then define $S^* = \{S_1^*, S_2^*, ...S_m^*\}$ as the set S partitioned into m disjoint subsets where each S_i^* is a set composed of the tuple $\langle id, impact \rangle$, where id is a process identifier and *impact* is the value of the impact factor of the process in question.

$S^* = \{S_1^*, S_2^*, ...S_m^*\}$ is a set of processes of such that $\forall i, j, i \neq j, S_i^* \cap S_j^* = \emptyset$ and $\bigcup \{q | \langle q, \_ \rangle \in S_i^*; 1 \leq i \leq m\} = S$

Trust Level

We denote ${trusted_p}^S(t) = \{trusted_1(t), \ldots, trusted_m(t)\}$, where each $trusted_i$ $(1 \leq i \leq m)$ contains the processes of S_i^* that are not considered faulty by p at $t \in T$. Similarly to S_i^*, each $trusted_i$ is composed of the tuple $\langle id, impact \rangle$.

The *trust level* at $t \in T$ of processes $p \notin F(t)$ of S is the denoted ${trust\_level_p}^S$ such that ${trust\_level_p}^S(t) = \{trust\_level_i | trust\_level_i = sum(trusted_i(t)); 1 \leq i \leq m\}$ where the function $sum(subset)$ returns the sum of the impact factor of all the elements of *subset*. In other words, the ${trust\_level_p}^S$ is a set that contains the trust level of each subset of S^* expressing the confidence that p has in the set S.

Margin of Failures

An acceptable margin of failures, denoted as the ${threshold}^S$, characterizes the acceptable degree of failure flexibility in relation to set S^*. The ${threshold}^S$ is adjusted to the minimum trust level required for each subset, i.e., it is defined as a set which contains the respective threshold of each subset of S^*: ${threshold}^S = \{threshold_1, \ldots, threshold_m\}$.

The ${threshold}^S$ is used by the application to check the confidence in the processes of S. If, for each subset of S^*, the $trust\_level_i(t) \geq threshold_i$, S is considered to be *trusted* at t by p, i.e., the confidence of p in S has not been compromised; otherwise S is considered *not trusted* by p at t.

Two points should be highlighted: (1) both the *impact factor* and ${threshold}^S$ render the estimation of the confidence in S flexible. For instance, it is possible that some processes in S might be faulty or suspected of being faulty but S can still be trusted; (2) the Impact FD can be easily configured to adapt to the needs of the environment. The ${threshold}^S$ can be tuned to provide a more restricted or softer monitoring. This kind of adaptability is essential in dynamic environments (such as the ubiquitous one). Note that the Impact FD can also be applied when the application needs individual information about each process of S. In this case, each process must be defined as a subset of S^*.

$S^* = \{\{\langle q_1, 1\rangle, \langle q_2, 1\rangle\}, \{\langle q_3, 3\rangle\}, \{\langle q_4, 4\rangle, \langle q_5, 4\rangle, \langle q_6, 4\rangle\}\}$

t	F(t)	$trusted_p^S(t)$	$trust\_level_p^S(t)$	Status
1	$\{\{\langle q_2, 1\rangle\}, \{\}, \{\}\}$	$\{\{\langle q_1, 1\rangle\}, \{\langle q_3, 3\rangle\}, \{\langle q_4, 4\rangle, \langle q_5, 4\rangle, \langle q_6, 4\rangle\}\}$	{1, 3, 12}	TRUSTED
2	$\{\{\langle q_2, 1\rangle\}, \{\}, \{\langle q_6, 4\rangle\}\}$	$\{\{\langle q_1, 1\rangle\}, \{\langle q_3, 3\rangle\}, \{\langle q_4, 4\rangle, \langle q_5, 4\rangle\}\}$	{1, 3, 8}	TRUSTED
3	$\{\{\langle q_2, 1\rangle\}, \{\}, \{\langle q_5, 4\rangle, \langle q_6, 4\rangle\}\}$	$\{\{\langle q_1, 1\rangle\}, \{\langle q_3, 3\rangle\}, \{\langle q_4, 4\rangle\}\}$	{1, 3, 4}	NOT TRUSTED
4	$\{\{\langle q_2, 1\rangle\}, \{\langle q_3, 3\rangle\}, \{\langle q_5, 4\rangle, \langle q_6, 4\rangle\}\}$	$\{\{\langle q_1, 1\rangle\}, \{\}, \{\langle q_4, 4\rangle\}\}$	{1, 0, 4}	NOT TRUSTED

$threshold^S = \{1, 3, 8\}$

Figure 4: Example of FD ($I_p{}^S$) output: S^* has three uniform subsets

5.2 Example

In Figure 4, we consider a set S, where S^* is composed by three subsets: S_1, S_2, and S_3. The values of $threshold^S$ define that the subsets S_1 and S_2 (resp. S_3) must have at least one (resp. 2) correct process(es). The figure shows several possible outputs for FD ($I_p{}^S$): the set S is considered trusted whenever, for each subset S_i^*, $trust\_level_i(t) \geq threshold_i$.

5.3 Impact Factor Domain of Subsets

Subsets can have different types of Impact factor configurations, denoted domains. To this end, we defined the following three domains : $\forall \langle \_, I\rangle \in S_i^1 \leq i \leq m \ : I > 0 \ and \ I \in \mathbb{R}$.

- U (Uniform): all processes of subset S_i^* of S^* have the same impact factor value.
  $S_i^* = \{\langle q, I_q\rangle \,|I_q = I_p, \forall \langle p, I_p\rangle \ and \ \forall \langle q, I_q\rangle \in S_i^*\}$ Example: see Figure 4
- H (Heterogeneous): the impact factor of processes of the subset can be of different values. It is defined according to the relevance of each process of subset S_i^* (default).
- D (Dominant): for each process $\langle q, I_q\rangle \in S_i^*$, its impact factor value is greater than the sum of the impact factor of the processes whose identity is smaller than q.
  $S_i^* = \{\langle q_j, I_{q_j}\rangle \,|I_{q_j} > \sum_{(i=1)}^{j-1} I_{q_i}, \forall \langle q_i, I_{q_i}\rangle, \langle q_j, I_{q_j}\rangle \in S_i^*\}$
  In this domain the concept of geometric progression can be applied to define the impact factor. A geometric progression is a sequence of numbers such that the quotient of any two successive members of the sequence is a constant called the common ratio of the sequence. A geometric sequence can be written as: a, ar, ar^2, ar^3, ar^4, ... where $r \neq 0$, r is the common ratio and a is a scale factor.

Example: The sequence 2, 6, 18, 54 is a geometric progression with common ratio 3 and $a = 2$.

$S_1^* = \{\langle q_1, 2\rangle, \langle q_2, 6\rangle, \langle q_3, 18\rangle, \langle q_4, 54\rangle\}$

5.4 Some implementations of Impact FD

The Impact FD can have different implementations in accordance with the characteristics of the system: the synchronization model, whether or not the process p has knowledge about the composition of S (membership), the type of nodes (identifiable, anonymous, homonymous), etc. We present in this section three different implementations. For the three algorithms, we consider that:

- the implementation of the algorithms is timer-based;
- *trust_ level*: set variable that contains the trust level of each subset of processes;
- *trusted*: set variable equals to $\{trusted_1, \ldots, trusted_m\}$, where each $trusted_i$ $(1 \leq i \leq m)$ contains the processes of S_i that are not considered faulty by p. Each $trusted_i$ is a set composed of the tuple $\langle id, impact\rangle$, where id is the process identifier and *impact* is the value of the impact factor of the process that belongs to the subset $trusted_i$.
- the algorithms use the functions (1) *sum(subset)* and (2) *Add(set, subset,* $\langle q, impact\rangle$*)* which respectively (1) returns the sum of the impact factor of all elements of *subset* and (2) adds the process q, with impact factor *impact* to the subset *subset* of the set *set*. Both functions are described in Section 6.

5.4.1 Asynchronous system, known membership, Chen estimation

Algorithm 1 shows an implementation of the Impact FD in an asynchronous system with message losses. It uses the Chen's heartbeat estimation. The membership of the system is known as well as its cardinality. Every process monitors the other ones ($p \in S$), i.e., it behaves like the monitoring process p that monitors all the other processes of S. $S*$ can contain several subsets whose domain is heterogeneous.

Process p receives as input its own impact factor (I_p), the subset to which it belongs ($subset_p$), the set S, the number of subsets m, and the heartbeat interval (Δ).

This algorithm also uses function *Remove(set, subset,* $\langle q, impact\rangle$*)* that removes process q, with impact factor *impact* from the subset *subset* of the set *set*;

At the initialization, process p creates, for each process q in S, a variable $timeout_q$ and initialize it (lines 6-9).

Each process p periodically sends to the processes in S an ALIVE message with its *id*, impact factor, and the subset to which it belongs (Task T1).

Upon reception of an ALIVE message from q (Task 2), if q does not belong to *trusted*, it is added (lines 11-13) to it and the timeout related to q is recomputed, based in Chen's estimation (line 14).

In task T3, if a process p does not receive an ALIVE message from q after $timeout_q$ time units, q is removed from *trusted* and the $timeout_q$ is recomputed, based in Chen's estimation (line 18).

Task T4 handles the invocation of the *Impact()* function, which computes the *trust_ level* of each subset and returns the *trust_ level* related to the current p's trusted processes.

Algorithm 1 Asynchronous system, known membership, Chen estimation

1: **Begin**
Input
2: I_p, $subset_p$, S, Δ,cm
Init
3: $trusted \leftarrow \emptyset$
4: $Add(trusted, subset_p, \langle p, I_p \rangle)$
5: $trust\_level \leftarrow \emptyset$
6: **for** *each* $q \in S$ **do**
7: *create* $timeout_q$
8: $timeout_q \leftarrow init(q)$
9: **end for**

Task T1 - Repeat forever every Δ time unit
10: $broadcast(ALIVE, p, I_p, subset_p)$

Task T2 - Upon reception of (ALIVE, q, I_q, $subset_q$) from q
11: **if** $\langle q, I_q \rangle \notin trusted_q$ **then**
12: $Add(trusted, subset_q, \langle q, I_q \rangle)$
13: **end if**
14: $timeout_q \leftarrow compute(q)$ ▷ using Chen's estimation

Task T3 - When $timeout_q$ expires
15: **if** $\langle q, \_ \rangle \in trusted_q$ **then**
16: $Remove(trusted, subset_q, \langle q, \_ \rangle)$
17: **end if**
18: $timeout_q \leftarrow compute(q)$ ▷ using Chen's estimation

Task T4
19: **Upon invocation of** $Impact()$ **do**
20: **for** $i = 1$ *to* m **do** ▷ for each subset
21: $trust\_level_i \leftarrow sum(trusted_i)$
22: **end for**
23: *return* $trust\_level$
24: **end**
25: **End**

5.4.2 Anonymous, synchronous system, unknown membership

Algorithm 2 presents an implementation of the Impact FD in a synchronous anonymous system, with reliable channels. The membership of the system is unknown as well as its cardinality. Every process monitors the other ones ($p \in S$) and the processes are grouped into one subset whose domain is either heterogeneous or uniform.

Process p receives as input its own impact factor (I_p), the timeout value and the heartbeat interval (Δ).

Initially, the variables are initialized (lines 3-6). We consider that all processes start at the same time.

Process p periodically sends to the other processes an ALIVE message with its impact factor value I_p and starts the *timeout* (Task 1).

In Task T2, when process p receives an ALIVE message from an anonymous processes, p includes the information about the impact value of this process in its *trusted* list. Notice that all processes are included in the subset 1, without identifier, since they are anonymous.

In task T3, after *timeout* time units, p saves the current *trusted*, and reinitialize the *trusted*.

Task T4 handles the invocation of the *Impact()* function, which returns the sum of impact factor of the trusted processes.

The followings points should be highlighted:

- Let δ_h be the maximum difference (delay) between the sending time of the heartbeats by all correct processes in a single round and δ_{lat}, be the maximum time for the transmission of a heartbeat message. The value of the *timeout* must, thus, be greater than $\delta_h + \delta_{lat}$ and Δ, interval between the sending of two heartbeats from a node, must be greater than the *timeout* value;

- Since the system is synchronous (no false suspicions), the trust level value returned by Impact FD at each round is the same or smaller than the previous value. If f is the maximum number of failures, when all of the f faults take place, the value returned by the Impact FD will always be the same;

- The algorithm can be applied to both homonymous and identifiable systems;

- If we consider a uniform domain with impact value equals to 1, a similar approach is proposed by the $\overline{AP}$ failure detector [5], which returns an estimation of the current number of processes in anonymous synchronous systems.

5.4.3 Homonymous, partially synchronous system, unknown membership

In this section, we present an implementation, Algorithms 3 and 4, of the Impact FD in a homonymous and partially synchronous systems, with reliable channels. The membership is unknown as well as the cardinality of the system. Processes of S can be grouped into several subsets and the domain is uniform in each subset. The algorithms are derived from the algorithm of Fig. 2 of [2].

We consider that there is a process p that monitors a set S of homonymous processes. Process p does not belong to S ($p \notin S$) and its id is different from all the processes in S, i.e., p is identifiable, while the processes in S are homonymous. Process p is the only process to execute Algorithm 3. It is a polling-based algorithm that executes in rounds. On the other hand, the processes of S execute Algorithm 4 responding to p's polling messages. As S is homonymous, different processes can have the same identity and, therefore, every subset is a multiset, i.e., an element can appear more than once.

Algorithm 2 Anonymous, synchronous system, unknown membership

1: **Begin**
   Input
2: I_p, $timeout$, Δ
   Init
3: $c\_trusted \leftarrow \emptyset$
4: $trusted \leftarrow \emptyset$
5: $Add(c\_trusted, 1, \langle \_, I_p \rangle)$
6: $Add(trusted, 1, \langle \_, I_p \rangle)$

   Task T1 - Repeat forever every Δ time unit
7: $broadcast(ALIVE, I_p)$
8: $start\ timeout$

   Task T2
9: **Upon reception of** $(ALIVE, I_q)$ from q **do**
10: $Add(trusted, 1, \langle \_, I_q \rangle)$
11: **end**

   Task T3 - When timeout expires
12: $c\_trusted \leftarrow trusted$
13: $trusted \leftarrow \emptyset$
14: $Add(trusted, 1, \langle \_, I_p \rangle)$

   Task T4
15: **Upon invocation of** $Impact()$ **do**
16: $return\ sum(c\_trusted_1)$
17: **end**
18: **End**

In Algorithm 3, process p receives as input its identity p, its own impact factor (I_p), the subset to which it belongs ($subset_p$), the number of subsets of S (m), and the initial timeout value (Δ).

At every round r_p, Task T1 of p assign the maximum value of set $t\Delta$ to *timeout*. Process p broadcasts $(POLLING, r_p)$ messages and waits for the expiration of the timer *timeout*. When it expires, p saves its current knowledge about trusted processes (in $c\_trusted$). Finally, the round counter (r_p) is incremented (line 14).

Task T2 handles the reception of messages $(P\_REPLY, r, q, I_q, subset_q)$, sent by q. If the process identity q is not yet known by p, then q is added to $local\_known$ and its $t\Delta$ is initialized (lines 17-20). If $r = r_p$, the tuple $\langle q, I_q, \rangle$ is added to the respective subset of q in *trusted*. When p receives an outdated message (i.e., a message whose round $r \neq r_p$), the timeout is increased (line 24).

Upon invocation of the *Impact()* function, Task T3 computes the $trust\_level$ of each subset and returns the $trust\_level$ of the trusted processes.

Algorithm 4 runs in every process $q \in S$. Task T1 is responsible for the reception of messages *POOLING*. When a process q receives a $(POLLING, r)$ message from process p, process q sends a P_REPLY message with its id, impact factor, and subset.

We point out that these algorithms could be also applied if S was composed by anonymous processes. In this case, we could also consider the impact factor of q ($q \in S$) as the subset of S^* to which $< q, I_q >$ belongs. In this way, redundancy could be exploited.

6 Some Classes derived from the Impact FD

As pointed out in the section 5, the Impact FD approach enables a more restricted or softer monitoring of the system to be carried out. Different classes of Failure detector, derived from the Impact FD can be defined, provided that different network and behavior properties are ensured. Hence, in this section, we propose some functions, properties and examples of FD that exploit the Impact FD concept.

6.1 Functions

Let $subset_i$ be a set of k_i elements of type $\langle q, I_{q_i} \rangle$ and set be a set of $subsets_i : 1 \leq i \leq m$.

$subset_i = \{\langle q, I_{q_i} \rangle \, | 1 \leq j \leq k_i\}$

$set = \{subset_i | 1 \leq i \leq m\}$

We then define, the following functions are defined:

- $sum(subset)$: returns the sum of impact factor of all elements of $subset$;

$$sum(subset) = \sum_{i=1}^{\#(set)} (\{I_{q_i} | \, \langle q_i, I_{q_i} \rangle \in subset\})$$

- *size(subset)*: function that returns the number of elements of $subset$;
- $Impact(q)$: returns the impact factor of q.
- *Add(set, subset,* $\langle q, impact \rangle$*)*: function that adds the process q, with impact factor $impact$ to the subset $subset$ of the set set;
- *Remove(set, subset,* $\langle q, impact \rangle$*)*: function that removes the process q, with impact factor $impact$ from the subset $subset$ of the set set;

Algorithm 3 Homonymous, partially synchronous system, unknown membership - process p

1: **Begin**
Input
2: I_p, $subset_p$, m, Δ
Init
3: $r_p \leftarrow 1$ ▷ round
4: $c\_trusted \leftarrow \emptyset$
5: $trust\_level \leftarrow \emptyset$
6: $local\_known \leftarrow \emptyset$ ▷ set of process identifiers
7: $t\Delta_p \leftarrow \Delta$ ▷ set of timeouts

Task T1
8: **loop**
9: $trusted \leftarrow \emptyset$
10: $timeout \leftarrow max(t\Delta)$
11: $broadcast(POLLING, r_p)$
12: *wait timeout time*
13: $c\_trusted \leftarrow trusted$
14: $r_p \leftarrow r_p + 1$
15: **end loop**

Task T2
16: **Upon reception of** $(P\_REPLY, r, q, I_q, subset_q)$ from q **do**
17: **if** $\langle q, I_q \rangle \notin local\_known$ **then**
18: $Add(local\_known, subset_q, \langle q, I_q \rangle)$
19: $t\Delta_q \leftarrow \Delta$
20: **end if**
21: **if** $r = r_p$ **then**
22: $Add(trusted, subset_q, \langle q, I_q \rangle)$
23: **else**
24: $t\Delta_q \leftarrow t\Delta_q + 1$
25: **end if**
26: **end**

Task T3
27: **Upon invocation of** $Impact()$ **do**
28: **for** $i = 1$ *to* m **do** ▷ for each subset
29: $trust\_level_i \leftarrow sum(c\_trusted_i)$
30: **end for**
31: *return trust_level*
32: **end**
33: **End**

Algorithm 4 Homonymous, partially synchronous system, unknown membership - process $q \in S$

```
1: Begin
   Input
2:     I_q, subset_q
   Task T1
3:     Upon reception of (POLLING, r) do
4:         send(P_REPLY, r, q, I_q, subset_q) to p
5:     end Upon reception

6: End
```

- $PMax(subset)$: returns the process with the greatest impact factor of *subset*. The domain is considered dominant, i.e., the processes have different impact factor value and, therefore, it is possible to uniquely identify them.

$$Pmax(subset) = q | \langle q, I_q \rangle \in subset \; and \; \forall \langle p, I_p \rangle \in subset, I_q \geq I_p$$

- $PMaxImpact(val, subset)$: returns the id. of the process of *subset* with the greatest impact factor value of *subset*, which is smaller than *val*. The domain is considered dominant.

$$PMaxImpact(val, subset) = q_i | \langle q_i, I_{q_i} \rangle \in subset \; and \; val > I_{q_i} \; and \; (i = size(subset) \; or \; val < I_{q_i+1}))$$

Example: $S_1^* = \{\langle q_1, 1\rangle, \langle q_2, 2\rangle, \langle q_3, 8\rangle, \langle q_4, 16\rangle, \langle q_5, 32\rangle\}$. $PmaxImpact(S_1^*, 15) = q_3$.

- PSet(val, subset): returns a set of processes of *subset* whose sum of respective impact failure is equal to *val* (Algorithm 5). The domain must be dominant.

Algorithm 5 Function PSet

```
1: function PSET(val, subset)
2:     aux ← ∅
3:     while val > 0 do
4:         q ← PMaxImpact(val, subset)
5:         aux ← aux ∪ {q}
6:         val ← val − Impact(q)
7:     end while
8:     return aux
9: end function
```

- $TPowerSet(set, threshold)$: returns a set comprising of all subsets of *set* which the sum of its elements is greater than or equal to the threshold. In mathematics, the power set of any set S is the set of all subsets of S, including the empty set and S itself.

Initially, the *TPowerSet* function generates the power set [4] for each subset (S_i^*) of S^*. Then, only the subsets of S_i^* whose sum of their parts is greater than, or equal to, $threshold_i$ are

[4] the power set of any set S is the set of all subsets of S, including the empty set and S itself

selected. Finally, the Cartesian product is applied to generate all possible combinations [5], i.e., all the subsets generated satisfy the $threshold^S$.

Let's consider the following example:

$S^* = \{\{\langle q_1, 2\rangle, \langle q_2, 4\rangle, \langle q_3, 3\rangle\}, \{\langle q_4, 1\rangle, \langle q_5, 2\rangle, \langle q_6, 3\rangle\}\}$
threshold^S = {3, 3} $X = TPowerset(S^*, threshold^S)$
PowerSet(S_1^*, $threshold_1$) = $\{\{\langle q_2, 4\rangle\}, \{\langle q_3, 3\rangle\}, \{\langle q_1, 2\rangle, \langle q_2, 4\rangle\}, \{\langle q_1, 2\rangle, \langle q_3, 3\rangle\}, \{\langle q_2, 4\rangle\ \langle q_3, 3\rangle\}, \{\langle q_1, 2\rangle, \langle q_2, 4\rangle, \langle q_3, 3\rangle\}\}$
PowerSet(S_2^*, $threshold_2$) = $\{\{\langle q_6, 3\rangle\}, \{\langle q_4, 1\rangle, \langle q_5, 2\rangle\}, \{\langle q_4, 1\rangle, \langle q_6, 3\rangle\}, \{\langle q_5, 2\rangle, \langle q_6, 3\rangle\}, \{\langle q_4, 1\rangle, \langle q_5, 2\rangle, \langle q_6, 3\rangle\}\}$
X = PowerSet(S_1^*, threshold$_1$) × PowerSet(S_2^*, threshold$_2$)
X = $\{\{\{\langle q_2, 4\rangle\}, \{\langle q_6, 3\rangle\}\}, \{\{\langle q_2, 4\rangle\}, \{\langle q_4, 1\rangle, q_5\}\}, \{\{\langle q_2, 4\rangle\}, \{\langle q_4, 1\rangle, \langle q_6, 3\rangle\}\}, \{\{\langle q_2, 4\rangle\}, \{q_5, \langle q_6, 3\rangle\}\}, \{\{\langle q_2, 4\rangle\}, \{\langle q_4, 1\rangle, q_5, \langle q_6, 3\rangle\}\}, \{\{\langle q_3, 3\rangle\}, \{\langle q_6, 3\rangle\}\}, \{\{\langle q_3, 3\rangle\}, \{\langle q_4, 1\rangle, \langle q_5, 2\rangle\}\}, \{\{\langle q_3, 3\rangle\}, \{\langle q_4, 1\rangle, q_6\}\}, \{\{\langle q_3, 3\rangle\}, \{q_5, \langle q_6, 3\rangle\}\}, \{\{q_3\}, \{\langle q_4, 1\rangle, \langle q_5, 2\rangle, \langle q_6, 3\rangle\}\}, \{\{q_1, \langle q_2, 4\rangle\}, \{\langle q_6, 3\rangle\}\}, \{\{q_1, \langle q_2, 4\rangle\}, \{\langle q_4, 1\rangle, \langle q_5, 2\rangle\}\}, \{\{q_1, q_2\}, \{\langle q_4, 1\rangle, q_6\}\}, \{\{q_1, \langle q_2, 4\rangle\}\}, \{\langle q_5, 2\rangle, q_6\}\}, \{\{\langle q_1, 2\rangle, \langle q_2, 4\rangle\}, \{\langle q_4, 1\rangle, \langle q_5, 2\rangle, \langle q_6, 3\rangle\}\}, \{\{\langle q_1, 2\rangle, \langle q_3, 3\rangle\}, \{\langle q_6, 3\rangle\}\}, \ldots\}$

6.2 Properties

The *flexibility* of the Impact FD characterizes its capacity of accepting different set of responses that lead to a trusted state of S.

Let X be the set that contains all possible subsets which satisfy a defined *threshold*:

$$X = TPowerSet(S, threshold) | TPowerSet(S, threshold) = \times\ PowerSet(S_i, threshold_i)$$

Prop. 1 $PR(\Diamond IT)$ *(Impact Threshold)*: For a failure detector of a correct process p, there is a time after which the set ${trusted_p}^S$ is a subset of X.

$$PR(\Diamond IT) \equiv \exists t \in T, p \in correct(F), \forall t' \geq t, {trusted_p}^S(t') \subset X$$

Prop. 2 *Impact strong completeness*${}_p{}^S$: For a failure detector of a correct process p, there is a time after which p does not trust any crashed process of S.

$$\exists t \in T, p \in correct(F), \forall q \in (faulty(F) \cap S), \forall t' \geq t, \langle q, \_\rangle \notin {trusted_p}^S(t')$$

Prop. 3 *Eventual impact strong accuracy*${}_p{}^S$: For a failure detector of a correct process p, there is a time after which all processes of ${trusted_p}^S$ are correct.

$$\exists t \in T, \forall t' \geq t, p \in correct(F), \forall \langle q, \_\rangle \in {trusted_p}^S(t'), q \in (correct(F) \cap S)$$

Prop. 4 *(Max Correct* ${trusted_p}^S$*)*: For a failure detector of a correct process p, there is a time after which p always trust the same correct process $\langle q, \_\rangle \in trusted$ (with the highest impact factor). In this case ${trusted_p}^S$ must be composed of only one subset of dominant domain. Let $trusted_1$ be such a subset:

$$\exists t, p \in S, \forall t' \geq t, q = PMax(trusted_1(t)) = PMax(trusted_1(t'))\ and\ q \in correct(F)$$

[5] The $\times S_i$ is a means of abbreviating the Cartesian Product when there are several sets, e.g. S_1^*, S_2^*, S_3^* $(X = (S_1^*) \times (S_2^*) \times (S_3^*))$

Prop. 5 (*Trust level leader*$_p{}^S$): For a failure detector of a correct process p, there is a time after which all correct processes output the same correct process (with highest impact factor of trusted). We denote this process the leader. In this case $trusted_p{}^S$ must be composed of only one subset of dominant domain. Let $trusted_1$ be such a subset:

$$\forall p, q \in correct(F), \exists t, \forall t' \geq t, PMax(trusted_1(t')) = Pmax(trusted_1(t'))$$

Prop. 6 (*Greater impact factor completeness*$_p{}^S$): If p, which monitors the set S, is correct, there is a time after which p does not trust any crashed process of S that the impact factor is greater than the *leader*. In this case $trusted_p{}^S$ must be composed of only one subset of dominant domain. Let $trusted_1$ be such a subset:

Note that, in this case, the processes whose impact failure is greater than the leader's one are faulty and eventually their respective failure are detected.

$$\exists t, p \in correct(F), \forall q : I_q > Impact(Pmax(trusted_1(t))), \forall t' \geq t, \langle q, \_ \rangle \notin trusted_1(t')$$

6.3 Examples of some Classes derived from the Impact FD

Considering the definitions of both the *impact factor* and the *threshold*, when one or more of the above properties are satisfied, we can define the following classes of Impact FD:

- $\diamondsuit IT$ (*Eventual Impact Threshold Class*): this class can be implemented when the underlying system satisfies the Prop. 1 (PR ($\diamondsuit IT$)).

  Definition 1 ($\diamondsuit IT$): For a failure detector of a correct process p, there is a time after which the $trust\_level_i{}^S(t)$ is greater than or equal to $threshold_i{}^S$, for all subsets of S.

  $$\exists t \in T, p \in correct(F), \forall t' \geq t, trust\_level_i(t') \geq threshold_i, \forall 1 \leq i \leq m$$

- $\diamondsuit ICT$ (*Eventual Correct Impact Class*): For the correct process p, the set S and a *threshold* defined, Prop. 1, Prop. 2 are satisfied.

- $\diamondsuit IPT$ (*Eventual Perfect Impact Class*): For the correct process p and the set S, Prop. 1, Prop. 2, Prop. 3 are satisfied.

- I^{Σ} (*Impact Failure Detector Sigma Class*): $S = \Pi$ and S^* is composed by just one subset with a dominant domain. Π is known by all processes. Prop. 1, Prop. 2 and Prop. 3 are satisfied.

- I^{Ω} (*Impact Failure Detector Omega Class*): $S = \Pi$ and S^* is composed by just one subset with a dominant domain. Π is known by all processes. Prop. 4, Prop. 5, and Prop. 6 are satisfied.

6.3.1 Implementation of a FD of Class $\diamondsuit IT$

In this section, we present an algorithm that implements the class $\diamondsuit IT$ based on a query-response approach on top of an asynchronous system where property $PR(\diamondsuit IT)$ is satisfied. Note that there exist other possible implementation as, for instance, a timer-based one with *heartbeat* on top of a partially synchronous system.

We consider that neither the membership, nor the number of processes of S are initially known by the processes. The network is assumed to be fully connected, i.e., $\Lambda = (p, q)|p, q \in \Pi$. Processes can invoke the primitive *broadcast(msg)* to send a message msg to all the processes

of the system. The primitive *broadcast(msg)* invoked by the correct process p sends one copy of message msg over the link from p to q, for each $q \in \Pi$. If p crashes while broadcasting msg, the latter is received by an arbitrary subset of processes. The links are reliable, i.e., they do not lose, duplicate or corrupt messages and they never generate spurious messages. Each process p repeatedly issues queries until it possibly crashes. At each new round, if p does not fail, it broadcasts a *QUERY* message. The time interval between two consecutive rounds is arbitrary but finite. When process q receives a *QUERY* message from p, q confirms its reception with a *RESP* message.

Algorithm 6 describes the implementation of the $\diamondsuit IT$ FD in process p with respect to S. We consider that every process behaves like a monitoring node, i.e., $p \in S$. Process p receives as input its own impact factor (I_p), the subset to which it belongs ($subset_p$), the *threshold* value of each subset of S^* (the set $threshold^S$), the number of subsets of S^* (m), and the maximum number of messages to wait (α). The latter is a set $\alpha = \{\alpha_1, \ldots, \alpha_m\}$, where each α_i corresponds to a threshold value for the number of messages to wait from the processes of subset S_i^*. For instance, if f_i denotes the maximum number of failures of processes of subset S_i^*, $\alpha_i \leq |S_i^*| - f_i$ (for $i \neq subset_p$) and $\alpha_i \leq |S_i^*| - f_i - 1$ (for $i = subset_p$), since p answers to its own message *QUERY*.

The following notation is used:

- r_p: round counter of process p;
- *trusted*: it is the set formed by $\{trusted_1, \ldots, trusted_m\}$, where each $trusted_i$ $(1 \leq i \leq m)$ contains the processes of S_i that are not considered faulty by p. Each $trusted_i$ is a set composed of the tuple $\langle id, impact \rangle$, where id is the process identifier and $impact$ is the value of the impact factor of the process that belongs to the subset $trusted_i$.
- *local_known*: current knowledge of p about the membership of S, impact factor of processes and their subsets. It is a set formed by $\{subset_1, \ldots, subset_m\}$. Each $subset_i$ $(1 \leq i \leq m)$ consists of the tuple $\langle id, impact \rangle$, where id is the process identifier and $impact$ is the value of the impact factor of the process;
- *trust_level*: the set that contains the trust level of each subset of processes;
- X: the set comprising all possible subsets formed by processes known by p, where the sum of its elements is greater than, or equal to, the $threshold^S$. Initially it is empty because the composition of the system is not known;

The algorithm has four tasks. Task T1 of p has an infinite loop. Firstly it sends the message *(QUERY, r_p)* to all the processes (line 13). Then, at each round (r_p), p waits for at least α_i responses $(1 \leq i \leq m)$ or until *trusted* is a subset of X (i.e., contains processes that satisfy the $threshold^S$) (line 14). Finally, the round counter (r_p) is incremented (line 18).

Task T2 handles the reception of messages *(RESP, r, I_q $subset_q$)* sent by q. The message contains the round, the impact factor and the subset to which q belongs. If round r is equal to r_p, then q is added to the $trusted_q$ set. If q is not yet known by p, then q is added to *local_known* in the subset $subset_q$ (line 25). In this case, when a new process is added, X is updated (line 26).

Task T3 is responsible for the reception of messages *QUERY*. When a process p receives a *(QUERY, r)* message from process q (line 28), p must respond with a *RESP* message containing the round, its impact factor, and the number of its subset (line 30).

Algorithm 6 $\diamondsuit IT$ FD class in process p

```
1:  Begin
    Input
2:      I_p, subset_p, threshold^S, m, α

    Init
3:      r_p ← 0
4:      local_known ← ∅
5:      trust_level ← ∅
6:      X ← ∅
7:      trusted ← ∅
8:      c_trusted ← ∅
9:      Add(local_known, subset_p, ⟨p, I_p⟩)
10:     Add(trusted, subset_p, ⟨p, I_p⟩)
11:     Add(c_trusted, subset_p, ⟨p, I_p⟩)

    Task T1
12:     loop
13:         broadcast(QUERY, r_p)
14:         wait until (size(trusted_i) ≥ α_i |1 ≤ i ≤ m) or (trusted ⊆ X)
15:         c_trusted ← trusted
16:         trusted ← ∅
17:         Add(trusted, subset_p, ⟨p, I_p⟩)
18:         r_p ← r_p + 1
19:     end loop

    Task T2 - Response
20:     Upon reception of (RESP, r, I_q, subset_q) from q do
21:         if r = r_p then
22:             Add(trusted, subset_q, ⟨q, I_q⟩)
23:         end if
24:         if ⟨q, I_q⟩ ∉ susbset_q of local_known then
25:             Add(local_known, subset_q, ⟨q, I_q⟩)
26:             X ← TPowerSet(local_known, threshold^S)
27:         end if
28:     end

    Task T3 - Query
29:     Upon reception of (QUERY, r) from q do
30:         send(RESP, r, I_p, subset_p) to q
31:     end

    Task T4
32:     Upon invocation of Impact() do
33:         for i = 1 to m do                          ▷ for each subset
34:             trust_level_i ← sum(c_trusted_i)
35:         end for
36:         return trust_level
37:     end
38: End
```

Task T4 handles the invocation of the *Impact()* function (line 32), which computes the *trust_level* of each subset and returns the *trust_level* of the trusted processes (line 36).

Sketch of Proof

Lemma 1. *No correct process blocks forever in a query-response round.*

Proof. The only point that p could block forever would be in the *wait* statement of Task T1 (line 14).

Let's suppose that p is blocked on the *wait* statement and that the $PR(\diamondsuit IT)$ property does not hold yet. Thus, the second condition may not be true and p will not be unblocked because of it. However, for every subset S_i^*, p waits for α_i messages ($1 \leq i \leq m$), where f_i is the maximum number of processes of S_i^* that can fail and $\alpha_i \leq |S_i^*| - f_i$ (resp., $\alpha_i \leq |S_i| - f_i - 1$) for $i \neq subset_p$ (resp., $i = subset_p$, since p is always included in its own *trusted* variable (lines 7 and 17)). Therefore, as the channels are reliable and, even if f_i nodes of each S_i have failed, p will receive α_i responses which will render the first condition true, and p will be unblocked. In other words, since α_i is bounded by n and f_i and no query or response messages are lost, such a condition always ensures the progress of the failure detector. □

Lemma 2. *Let's consider that task T2 is indivisible and executes at one unit of time* $\in T$. $\forall t, t \geq 0, trusted \subset local\_known$.

Proof. $\forall t \geq 0$, $\langle p, I_p \rangle$ always belongs to both *trusted* (lines 7 and 17) and $local\_known_p$ (line 4) in subset $subset_p$. For every process q which responds at t to a query of p, $\langle q, I_q \rangle$ is also included at t in both *trusted* (line 22) and *local_known* (line 25) int the subset $subset_q$. Furthermore, every process which has been included in *local_known* stays forever in it and in every round $subset_p$ in *trusted* is initialized to $\langle p, I_p \rangle$. Thus, $\forall t, t \geq 0, \forall i \in trusted, i \in local\_known$. □

Theorem 1. *Algorithm 6 implements an unreliable failure detector of class* $\diamondsuit IT$.

Proof. From Lemma 1, Algorithm 1 never blocks. Let's $t \in T$ be the time where $PR(\diamondsuit IT)$ property holds. From Lemma 2, all processes in *trusted* take part in the computation of all possible subsets which satisfy $threshold^S$ (X) since, $trusted \subset local\_known$ and *local_kown* is a parameter of the function *TPowerSet* (line 26). Therefore, after t, the second condition of the *wait* statement of Task T1 will always be true for every round and, thus, *trust_level*, which is computed from *trusted*, will always be greater than $thershold^S$. Let $t' > t$ be the first time line 15 is executed ($c\_trusted = trusted$) after $PR(\diamondsuit IT)$ property holds. Hence, $\forall t'' > t'$, whenever the $Impact()$ function is invoked (line 32), $trust\_level \geq threshold^S$. □

6.4 Some FD reducibility

According to [7], reducibility means an algorithm $T_{D \to D'}$ which transforms a failure detector D into another failure detector *D'*. Algorithm $T_{D \to D'}$ uses D to maintain a variable $output_p$ at every p. Given a reduction algorithm $T_{D \to D'}$, any problem that can be solved using failure detector *D'*, can be solved using D instead. Thus, if there is an algorithm $T_{D \to D'}$ that transforms D into *D'*, we say that *D' is reducible to D*, noted $D \succeq D'$; we also say that *D' is weaker than D* ($\succeq$ is a transitive relation). Furthermore, if $T_{D \to D'}$ and $T_{D \to D'}$, we write $D \cong D'$ and say that D and *D' are equivalent.*

6.4.1 I^{Σ}(Sigma)

We show in this section that the Impact Failure Detector Sigma class is equivalent to Sigma Failure Detector class (see section 3.1), provided that there exist a majority of correct processes. Thus, any distributed algorithm problem that can be solved with a Sigma Failure Detector class can also be solved with the I^{Σ} (Impact Failure Detector Sigma).

Since the domain in this case is the dominant one (e.g., Geometric Progression), processes have different impact factor value and, therefore, it is possible to uniquely identify them. There is just one subset.

I^{Σ} is equivalent to Σ:

- Σ is reducible to I^{Σ} ($I^{\Sigma} \succeq \Sigma$) : Algorithm 7
- I^{Σ} is reducible to Σ ($\Sigma \succeq I^{\Sigma}$): Algorithm 8

Algorithm 7 Transforming a I^{Σ} to Σ

```
1: Begin
   process p:
2:     when queried about set S at time t do
3:         repeat
4:             subset = PSet(I_p^S(t), S_1)
5:         until #(subset) > (n/2)
6:         return subset
7:     end when
8: End
```

Algorithm 8 Transforming Σ to I^{Σ}

```
1: Begin
   process p:
2:     when queried about set S at time t do
3:         trust_level_p^S(t) = sum(trusted1)
4:         return trust_level_p^S(t)
5:     end when
6: End
```

6.4.2 I^{Ω}(Omega)

In Section 3.1, we have presented the definition of the Omega Ω failure detector. The Impact Failure Detector Ω is reducible to I^{Ω}, i.e., $I^{\Omega} \succeq \Omega$ (Algorithm 9). However, I^{Ω} is not reducible to Ω.

Note: The implementation of Ω in [16] applies the same approach of the Greater Impact Factor Completeness: if p_l is the leader, the failures of $p_i \ldots p_l - 1$ is eventually detected.

Since Ω is weaker than Ω, any algorithm that solves consensus in Ω, also solves consensus in I^{Ω}.

Algorithm 9 Transforming a I^{Ω} to Ω

1: **Begin**
process p:
2: **when** queried about set S at time t **do**
3: $leader = PMaxImpact(trust\_level_p{}^{S}(t), S)$
4: **return** $leader$
5: **end when**
6: **End**

Table 2: Sites of Experiments

ID	Site	Local
0	ple4.ipv6.lip6.fr	France
1	planetlab1.jhu.edu	USA East Coast
2	planetlab2.csuohio.edu	USA, Ohio
3	75-130-96-12.static.oxfr.ma.charter.com	USA, Massachusetts
4	planetlab1.cnis.nyit.edu	USA, New York
5	saturn.planetlab.carleton.ca	Canada, Ontario
6	PlanetLab-03.cs.princeton.edu	USA, New Jersey
7	prata.mimuw.edu.pl	Poland
8	planetlab3.upc.es	Spain
9	pl1.eng.monash.edu.au	Australia

7 Performance Evaluation

In this section, we firstly describe the environment in which the experiments were conducted and the QoS metrics used for evaluating the results. Then, we discuss some of the results concerning the execution of Algorithm 1 of Section 5.4.1 with different configurations of node sets with regard to both the impact factor and the threshold. We also compared the latter with some results related to Chen FD [8], whose output is a list of suspected processes.

7.1 Environment

We used realistic trace files collected from ten nodes of PlanetLab [19], as summarized in Table 2. The experiment started on July 16, 2014 at 15:06 UTC, and ended exactly a week later. Each site sent heartbeat messages to other sites at a rate of one heartbeat every 100 ms (the sending interval). We should point out that these traces of PlanetLab contain a large amount of data concerning the sending and reception of heartbeats, including unstable periods of links and message loss which induce false suspicions. Thus, they can characterize any distributed system that uses FDs based on heartbeat. Furthermore, since the sending and arrival times of each heartbeat are recorded in the trace files, all the experiments were conducted with exactly the same scenarios and history of heartbeats.

Table 3 shows information about the heartbeat messages received by site number *1* (the monitor node). We observed that the mean inter-arrival times of received heartbeats was very close to 100 ms. However, in some sites, the standard deviation is very high, like in site *5* which the standard deviation was 310.958 ms with a minimum of 0.006 ms, and a maximum

Table 3: Sites and heartbeat sampling

Site	Messages	Mean (ms)	Min (ms)	Max (ms)	Stand. Dev.(ms)
0	5424326	100.058	0.025	26494.168	19.525
2	1759989	100.415	0.031	509.093	9.275
3	5426843	100.012	0.027	1227.349	1.709
4	5414122	100.247	0.003	1193.276	18.595
5	5413542	100.258	0.006	657900.226	310.958
6	5426700	100.015	0.003	3787.643	2.557
7	5424117	100.062	0.006	59603.188	31.229
8	5424560	100.054	0.027	11443.359	100.714
9	5422043	100.100	0.004	30600.076	18.798

of 657900.226 ms. This points out that, for a certain time interval during execution, the site stopped sending heartbeats and started again afterwards. Note also that Site *2* stopped sending messages after approximately 48 hours and, therefore, there are just 1759990 received messages.

7.2 QoS Metrics

For evaluating the Impact FD, we used three of the QoS metrics proposed by [8]: detection time, average mistake rate, and query accuracy probability. Considering two processes, q and p, where p monitors q, the QoS of the FD at p can be evaluated from the transitions between the "trusted" and "not trusted" states with respect to q.

- Detection Time (T_D): the time that elapses from the moment that process q crashes until the FD at p starts suspecting q permanently.
- Average Mistake Rate (λ_R): represents the number of mistakes that FD makes in a unit time, i.e., the rate at each a FD makes mistakes.
- Query Accuracy Probability (P_A): the probability that the FD output is correct at a random time.

7.3 Estimation of heartbeat arrivals

Aiming at reducing both the number of false suspicions and the time needed to detect a failure, Chen et al. [8] propose an approach to estimate the arrival of the next heartbeat which is based on the history of the arrival time of heartbeats and includes a safety margin (β). The timer is then set according to this estimation. The estimation algorithm is described in Section 4.

For the estimation of the arrival time, the authors suggest that the safety margin β should range from 0 to 2500 ms. For all experiments, we set the window size to 100 samples, which means that the FD only relies on the last 100 heartbeat message samples for computing the estimation of the next heartbeat arrival time.

7.4 Set Configuration

We defined a set composed of sites 0, 2, 3, 4, 5, 6, 7, 8 and 9 (*S= {{0, 2, 3, 4, 5, 6, 7, 8, 9}}*). Site *1* was the monitor node (p). Table 4 shows the five configurations with regard to impact

Table 4: Set Configurations

Config	Impact Factor of each site
Set 0	I_0=7; I_2=3; I_3=20; I_4=20; I_5=3; I_6=20; I_7=3; I_8=7; I_9=7;
Set 1	I_0=7; I_2=20; I_3=20; I_4=3; I_5=3; I_6=20; I_7=3; I_8=7; I_9=7;
Set 2	I_0=20; I_2=7; I_3=3; I_4=3; I_5=7; I_6=3; I_7=7; I_8=20; I_9=20;
Set 3	I_0=7; I_2=3; I_3=20; I_4=3; I_5=3; I_6=20; I_7=7; I_8=20; I_9=7;
Set 4	I_0=10; I_2=10; I_3=10; I_4=10; I_5=10; I_6=10; I_7=10; I_8=10; I_9=10;

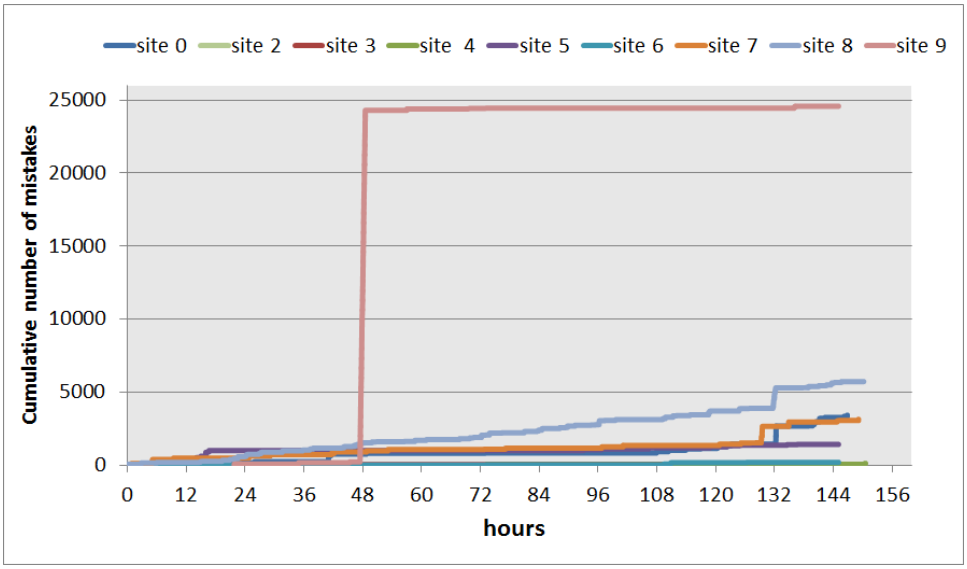


Figure 5: Cumulative number of mistakes of each site

factor values that have been considered for S in the experiments. The sum of the impact factor of the processes is 90 for all configurations.

In order to decide about the impact factor value to assign to each site, we have evaluated the stability of the sites. To this end, we monitored the sites individually using Chen's algorithm with β=400ms. Figure 5 shows the cumulative number of mistakes for each site during the whole trace period. We can observe that site or link periods of instability entail late arrivals or loss of heartbeats and, therefore, mistakes by the monitor node. For example, site *9* had a large number of cumulative mistakes at hour 48. After that, there is a stable period with regard to this site. It should also be noted that site *2* stopped sending messages after approximately 48 hours (it crashed) and, consequently, the monitor node made no more mistakes about it after this time. Finally, we can say that, considering the whole period, sites *3* and *6* (resp., *8* and *9*) are, in average, the most stable (resp., unstable) sites.

7.5 Experiments

7.5.1 Experiment 1 - Query Accuracy Probability

The aim of this experiment is to evaluate the Query Accuracy Probability (P_A) with different threshold values (64, 70, 74, 80, and 83) and different impact factor configurations (Table 4). We considered the safety margin as being β=400ms.

Figure 6 shows that the P_A decreases when the threshold increases. It should be remembered that the *threshold* is a limit value defined by the user and if the FD trust level output value is equal to, or greater than, the threshold, the confidence in the set of processes is ensured. Hence, the results confirm that when the threshold is more flexible, the Query Accuracy Probability is higher.

On the one hand, except for threshold 83, *"Set 0"* configuration has the highest P_A for most of the *thresholds* due to the assignment of high (resp., low) impact factors for the most stable (resp., unstable) sites. On the other hand, *"Set 2"* and *"Set 4"* have the lowest P_A since unstable sites have high impact factor values assignment. For instance, the hight impact factor value of sites *8* and *9* degrades the P_A of these sets.

"Set 4" shows a sharp decline when the *threshold* = 83. This behavior can be explained since, in this set configuration, all sites have the same impact factor (10) which implies that every false suspicion renders the *trust_level* smaller than the *threshold* (83), increasing the mistake duration. Therefore, the P_A decreases.

Notice that site *2* failed after approximately 48 hours. Thus, after its crash, the FD output, which indicates *trust_level* smaller than the *threshold*, is not a mistake, i.e. it is not a false suspicion. Hence, in *"Set 1"*, where the impact factor of site *2* is 20 (high), the *PA* is constant for a *threshold* greater than 70: after the crash of site *2*, the FD output is always smaller than the *threshold* and false suspicions related to other sites do not alter it. The average mistake duration in the experiment is thus smaller after the crash, which improves the P_A.

Finally, we have compared the P_A of the Impact FD and a FD approach that monitors processes individually by applying Chen's algorithm with *WS=100* and β=400ms. The mean P_A obtained was 0,979788. This result shows that, regardless of the set configuration, the Impact FD has a higher P_A than Chen FD since the former has enough flexibility to tolerate failures, i.e., the mistake duration only starts to be computed when the *trust_level* provided by Impact FD is smaller than the *threshold*, in contrast with individual monitoring, such as that by Chen FD, where every false suspicion increases the mistake duration.

The results of this experiment highlight the fact that the assignment of heterogeneous impact factors to nodes can degrade the performance of the failure detector, especially when unstable sites have a high impact factor.

7.5.2 Experiment 2 - Detection time

In the second experiment, we evaluated the average Query Accuracy Probability (P_A) regarding the average detection time (T_D). In order to obtain different values for the detection time, we varied the safety margin (Chen's estimation) with intervals of 100 ms, starting at 100 ms. For this experiment, we chose the *"Set 0"* configuration since it presented the best P_A in Experiment 1. We also evaluated the P_A and T_D for Chen's algorithm, which outputs the set of suspected nodes.

Figure 7 shows that for high threshold and detection time close to 200 ms, the P_A of the Impact FD is smaller, independently of the threshold, because the safety margin (used to compute the expected arrival times) is, in this case, equals to 100 ms, which increases both the number of failure suspicion and mistake duration. However, when T_D is greater than 230 ms, the P_A of

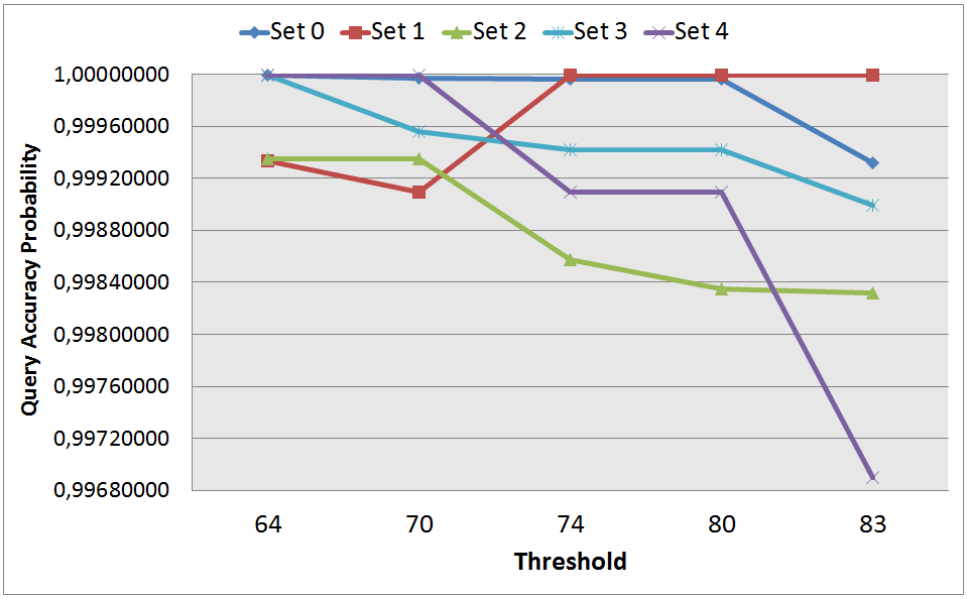


Figure 6: P_A vs. threshold with different set configurations

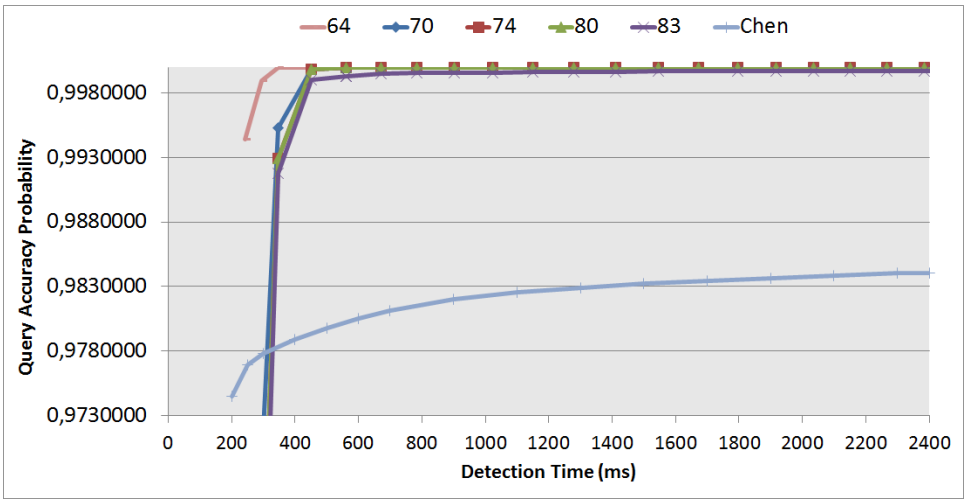


Figure 7: P_A vs. T_D with different thresholds

Impact FD is considerably higher than that of Chen. After a detection time of approximately 400 ms, the P_A of Impact FD becomes constant regardless of the detection time and threshold, and gets close to 1. Such a behavior can be explained since the higher the safety margin, the smaller the number of false suspicions, and the shorter the mistake duration which confirms that when the timeout is short, failures are detected faster but the probability of having false detections increases [20].

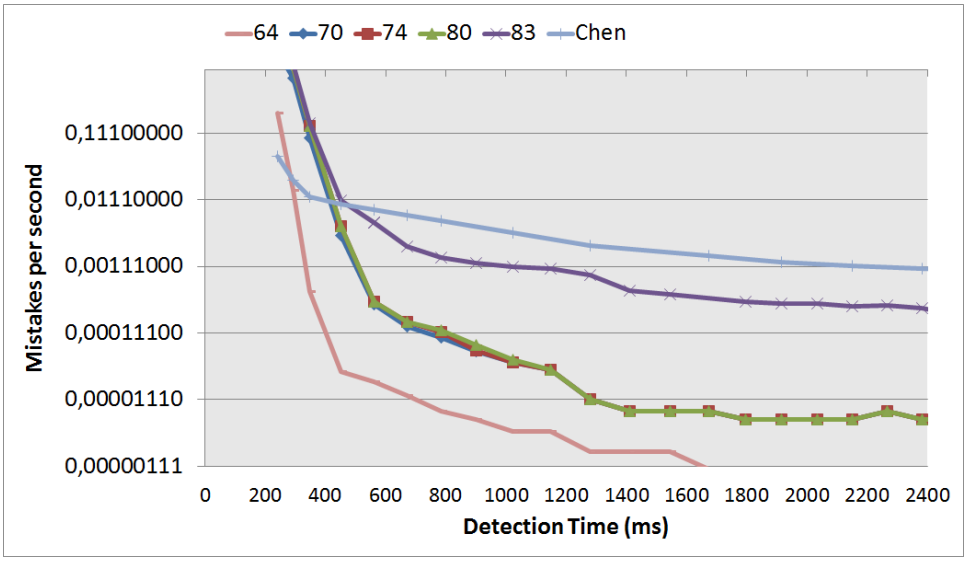


Figure 8: λ_R vs. T_D with different thresholds

7.5.3 Experiment 3 - Average mistake rate

In this experiment, we evaluated the average detection time (T_D) vs. the mistake rate (λ_R) (mistakes per second). We considered the *"Set 0"* configuration and the mistake rate is expressed in a logarithmic scale. It can be observed in Figure 8 that the mistake rate of the Impact FD is higher when the detection time is low (smaller than 400 ms) and the threshold is high (from 23 to 25). Such a result is in accordance with Experiment 2: whenever the safety margin is small and *threshold* tolerates fewer failures, the Impact FD makes mistakes more frequently. In other words, the mistake rate decreases when the threshold is more flexible or the time detection increases.

7.5.4 Experiment 4 - Cumulative number of mistakes

In this experiment, we evaluated the cumulative number of mistakes for the *"Set 0"* configuration during the whole trace period, considering β=400ms and the threshold value of 80 and 83.

We can observe in Figure 9 that the cumulative number of mistakes is greater for the threshold value 83 (2754 mistakes) when compared to threshold value 80 (179 mistakes). The former made few mistakes until approximately the hour 48 (when the site *2* crashed). After that, the number of cumulative mistakes significantly increased because, as the threshold is high (83) and the failure of *2* was detected, false suspicions of any other site induce a *trust_level* value smaller than 83 in most cases. For instance, site *8* is highly unstable and has impact factor value of 7. Whenever there is a false suspicion about it, after the crash of site *2*, the *trust_level* value is 80. On the other hand, for the threshold 80, we can observe that there were fewer instability periods since the crash of site *2* does not have much impact on the confidence of the system. At hour 48, there was an increase in the cumulative number of mistakes due to the unstable period of site *9*, as shown in Figure 5. From hour 50 to 100, the FD made fewer mistakes. Such a behavior can be explained since, as observed in the same figure, all sites, with exception of site *8*, also had this same period of stability. After hour 108, there was a greater number of mistakes which is related to the instability of sites *0*, *7*, and *8* (see Figure 5).

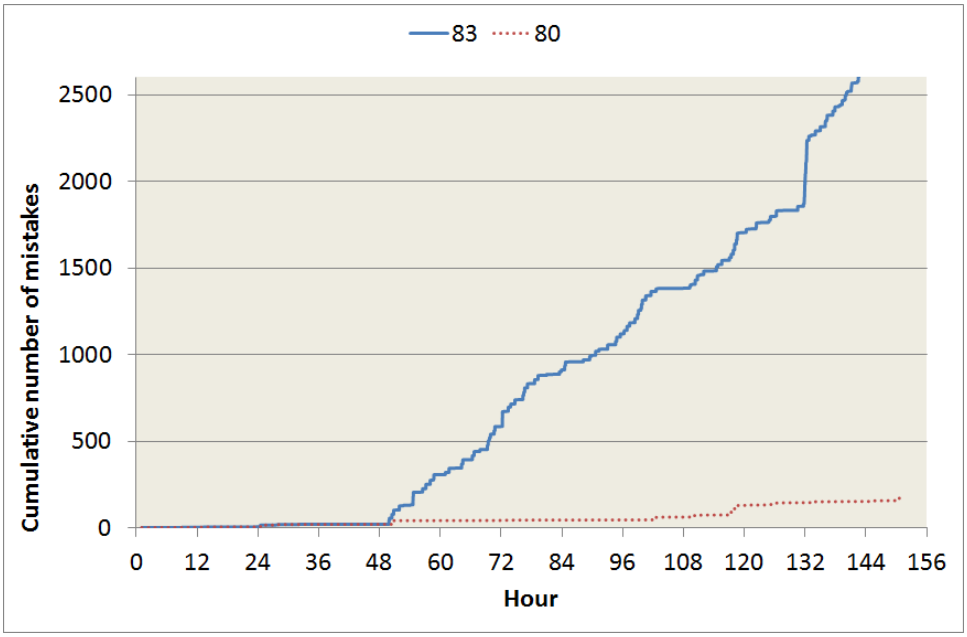


Figure 9: Cumulative number of mistakes for *"Set 0"* configuration

7.5.5 Experiment 5 - Query Accuracy Probability vs. Time

In this experiment, we divided the execution trace time by fixed intervals and computed the average Query Accuracy Probability (P_A) for each of them. We chose the *"Set 0"* configuration, β=400ms, and the threshold values of 80 and 83. Similarly to the cumulative number of mistakes (Experiment 3), we can note in Figure 10 that instability periods have an impact on the P_A. For instance, for the threshold = 80, from hour 108, the cumulative number of mistakes increases very fast. Consequently, the PA decreases. The period of instability of site 9 is the responsible for the important reduction of the P_A at hour 60 (i.e., from hour 48 to 60) when threshold = 83. A new degradation of the P_A happens at hour 120 (i.e., from hour 108 to 120), due to unstable periods of the sites *0*, *7*, and *8*.

8 Conclusion

In this technical report, we have proposed a new unreliable failure detector, the **Impact FD**, which provides a single output value related to a set of processes and not to each one individually. Processes can be divided in different subgroups based on some criterion. Both its *impact factor* and *threshold* concepts offer a degree of flexibility since they enable the user to tune the Impact FD in accordance with the specific needs and acceptable margin of failures of the application. In some scenarios, they also might weaken the rate of false responses when compared to traditional unreliable failure detectors. Performance evaluation results show that the assignment of a high (resp. low) impact factor to more stable (resp., unstable) nodes increases the Query Accuracy Probability of the failure detector.

We have also presented three possible implementations of the Impact FD, with different synchronous models and process types (e.g. anonymous, homonymous). Furthermore, by defining some behavior properties, we showed some examples of classes that exploit the concept of Impact

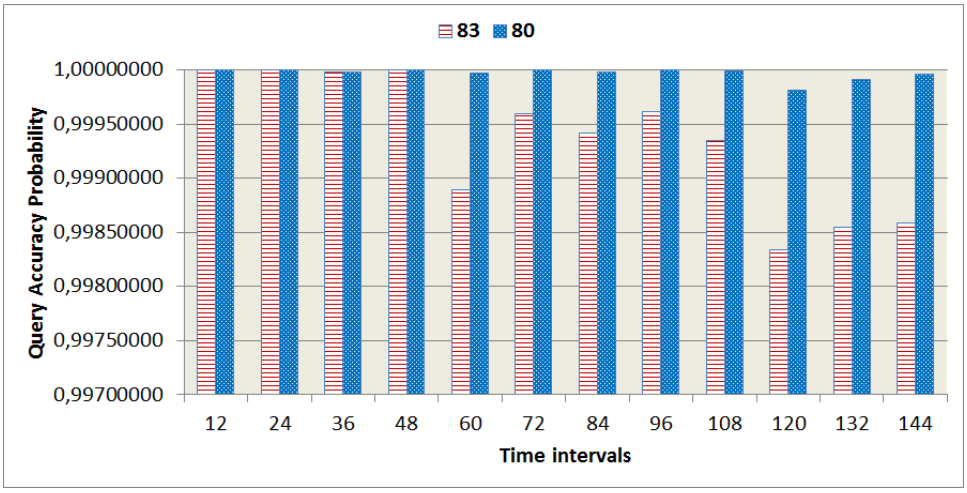


Figure 10: P_A vs. Time

FD, and some equivalences of the latter with existing failure detectors (e.g. Sigma, Omega).

9 Acknowledgment

This work was partially supported by grant 012909/2013-00 from the Brazilian Research Agency (CNPq).

References

[1] Abu Zafar Abbasi, Noman Islam, Zubair Ahmed Shaikh, et al. A review of wireless sensors and networks' applications in agriculture. *Computer Standards & Interfaces*, 36(2):263–270, 2014.

[2] Sergio Arévalo, Antonio Fernández Anta, Damien Imbs, Ernesto Jiménez, and Michel Raynal. Failure detectors in homonymous distributed systems (with an application to consensus). In *Distributed Computing Systems (ICDCS), 2012 IEEE 32nd International Conference on*, pages 275–284. IEEE, 2012.

[3] Marin Bertier, Olivier Marin, Pierre Sens, et al. Performance analysis of a hierarchical failure detector. In *DSN*, volume 3, pages 635–644, 2003.

[4] François Bonnet and Michel Raynal. The price of anonymity: Optimal consensus despite asynchrony, crash, and anonymity. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 6(4):23, 2011.

[5] François Bonnet and Michel Raynal. Anonymous asynchronous systems: The case of failure detectors. *Distributed computing*, 26(3):141–158, 2013.

[6] Yuriy Brun, George Edwards, Jae Young Bang, and Nenad Medvidovic. Smart redundancy for distributed computation. In *Distributed Computing Systems (ICDCS), 2011 31st International Conference on*, pages 665–676. IEEE, 2011.

[7] Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM (JACM)*, 43(2):225–267, 1996.

[8] Wei Chen, Sam Toueg, and Marcos Kawazoe Aguilera. On the quality of service of failure detectors. *Computers, IEEE Transactions on*, 51(5):561–580, 2002.

[9] François JN Cosquer, Luís Rodrigues, and Paulo Veríssimo. Using tailored failure suspectors to support distributed cooperative applications. In *Parallel and Distributed Computing and Systems*, pages 352–358. Citeseer, 1995.

[10] Carole Delporte-Gallet, Hugues Fauconnier, and Rachid Guerraoui. Shared memory vs message passing. *Technical Report*, 77, 2003.

[11] DD Geeta, N Nalini, and Rajashekhar C Biradar. Fault tolerance in wireless sensor network using hand-off and dynamic power adjustment approach. *Journal of Network and Computer Applications*, 36(4):1174–1185, 2013.

[12] Naohiro Hayashibara, Xavier Défago, and Takuya Katayama. Two-ways adaptive failure detection with the ϕ-failure detector. In *Workshop on Adaptive Distributed Systems (WADiS03)*, pages 22–27. Citeseer, 2003.

[13] Naohiro Hayashibara, Xavier Defago, Rami Yared, and Takuya Katayama. The φ accrual failure detector. In *Reliable Distributed Systems, 2004. Proceedings of the 23rd IEEE International Symposium on*, pages 66–78. IEEE, 2004.

[14] Koichi Ishibashi and Masatsugu Yano. A proposal of forwarding method for urgent messages on an ubiquitous wireless sensor network. In *Information and Telecommunication Technologies, 2005. APSITT 2005 Proceedings. 6th Asia-Pacific Symposium on*, pages 293–298. IEEE, 2005.

[15] Van Jacobson. Congestion avoidance and control. In *ACM SIGCOMM Computer Communication Review*, volume 18, pages 314–329. ACM, 1988.

[16] Mikel Larrea, Antonio Fernández Anta, and Sergio Arévalo. Implementing the weakest failure detector for solving the consensus problem. *IJPEDS*, 28(6):537–555, 2013.

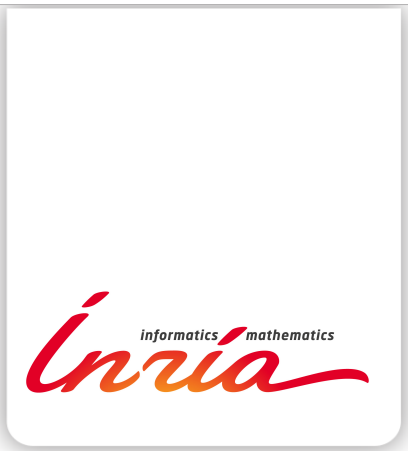
[17] Joshua B Leners, Trinabh Gupta, Marcos K Aguilera, and Michael Walfish. Improving availability in distributed systems with failure informers. In *Proc. of NSDI*, 2013.

[18] Achour Mostefaoui, Eric Mourgaya, and Michel Raynal. Asynchronous implementation of failure detectors. In *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 351–351. IEEE Computer Society, 2003.

[19] PlanetLab. Planetlab. `http://www.planet-lab.org`, 2014.

[20] Benjamin Satzger, Andreas Pietzowski, Wolfgang Trumler, and Theo Ungerer. A new adaptive accrual failure detector for dependable distributed systems. In *Proceedings of the 2007 ACM symposium on Applied computing*, pages 551–555. ACM, 2007.

[21] Sung-Jib Yim and Yoon-Hwa Choi. An adaptive fault-tolerant event detection scheme for wireless sensor networks. *Sensors*, 10(3):2332–2347, 2010.



RESEARCH CENTRE
PARIS – ROCQUENCOURT

Domaine de Voluceau, - Rocquencourt
B.P. 105 - 78153 Le Chesnay Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399