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 FD: An Unreliable Failure Detector Based on Processes Relevance and Confidence in the System

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

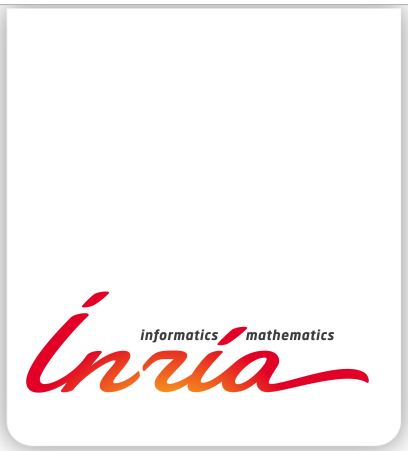
RESEARCH REPORT

N° hal-01136595

Aug 2016

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

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



Impact FD: An Unreliable Failure Detector Based on Processes Relevance and Confidence 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 — Aug 2016 — 41 pages

Abstract: This technical report presents a new unreliable failure detector, called the *Impact* failure detector (FD) that, contrarily to the majority of traditional FDs, outputs a *trust level* value which expresses the degree of confidence in the system. An *impact factor* is assigned to each node and the *trust level* is equal to the sum of the impact factors of the nodes not suspected of failure. Moreover, a *threshold* parameter defines a lower bound value for the *trust level*, over which the confidence in the system is ensured. In particular, we defined a $flexibility$ property that denotes the capacity of the Impact FD to tolerate a certain margin of failures or false suspicions, i.e., its capacity of considering different sets of responses that lead the system to trusted states. The Impact FD is suitable for systems that present node redundancy, heterogeneity of nodes, clustering feature, and allow a margin of failures which does not degrade the confidence in the system. The technical report also includes a timer-based distributed algorithm which implements a Impact FD, as well as its proof of correctness, for systems whose links are *lossy asynchronous* or for those whose all (or some) links are *eventually timely*. Performance evaluation results based on real PlanetLab [33] traces confirm the degree of flexible applicability of our failure detector and, due to the accepted margin of failure, the both failures and false suspicions are more tolerated when compared to traditional unreliable failure detectors. We also show the equivalence of some classes of Impact FD in regard with Σ and Ω classes, which are fundamental classes to circumvent the impossibility of consensus in asynchronous message-passing distributed systems.

Key-words: unreliable failure detector, impact factor, trust level of the system, processes' relevance, margin of failures, flexibility property.

* 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 FD: An Unreliable Failure Detector Based on Processes Relevance and Confidence 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

In distributed systems, failures can occur and the detection of them is a crucial task in the design of fault tolerant distributed systems or applications. On the other hand, in asynchronous systems there exist no bounds on message transmission neither on processes speed. Therefore, detection of crashed processes is particularly difficult in those systems since it is impossible to determine whether a process has really failed or if it and/or the network communication are just slow. Due to this lack of delay bounds, it is well-known that consensus problem cannot be solved deterministically in an asynchronous system subject to even a single crash failure [19].

To circumvent such an impossibility and give support to the development of fault tolerant distributed systems, Chandra and Toueg proposed in [11] the unreliable failure detector(FD) abstraction. An unreliable FD can be seen as an oracle that gives (not always correct) information about process failures. Many current FDs are based on a binary model, in which monitored processes are either "trusted" or "suspected". Thus, most of existing FDs, such as those defined in [11] [6], output the set of processes that is currently suspected to have crashed. According to the type and the quality of this information, several failure detector classes have been proposed.

This technical report presents a new unreliable failure detector, denoted the *Impact* failure detector. A preliminary proposal of it was presented in [34]. Contrarily to the majority of existing unreliable failure detectors, the Impact FD provides an output that expresses the trust of the FD with regard to the system (or set of processes) as a whole and not to each process individually. A system is considered "trusted" if it behaves correctly for a specific purpose even in the face of failures, i.e., the system is able to maintain the normal functionality.

The conception of the Impact FD was inspired on systems that have the following features: (1) applications that execute on them are interested on information about the reliability of the system as a whole and can tolerate a certain margin of failures. The latter may vary depending on the environment, situation, or context, such as the systems that provide redundancy of software/hardware; (2) systems that organize nodes with some common characteristic in groups; (3) systems where the nodes can have different importance (relevance) or roles and, thus, their failures may have distinct impact on the system. Systems that present node redundancy, heterogeneity of nodes, clustering feature, and allow a margin of failures which does not degrade the confidence in the system can, thus, benefit from the Impact FD and its configuration choices. They have motivated our work. Section 2 describes some examples of such systems, how the Impact FD can be applied and configured to them, and the advantages, in these cases, of using the Impact FD instead of traditional FDs.

The Impact FD outputs a *trust level* related to a given set of processes S of the monitored system. We, thus, denote FD (${I_p}^S$) the Impact failure detector module of process p that monitors the processes of S. When invoked in p, the Impact FD (${I_p}^S$) returns the $trust\_level$ value which expresses the confidence that p has in set S. To this end, an *impact* value, defined by the user, is assigned to each process of S and the $trust\_level$ is equal to the sum of the impact factors of the trusted nodes, i.e., those not suspected of failure by p. Furthermore, a *threshold* parameter defines a lower bound for the *trust level*, over which the confidence degree on S is ensured. Hence, by comparing the $trust\_level$ with the *threshold*, it is possible to determine whether S is currently "trusted" or "untrusted" by p. 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 in case 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 [3]. We should also point out that the Impact FD configuration allows nodes of S to be grouped into subsets and threshold values can be defined for each of these subsets. In addition, similarly to the traditional FD, several classes of Impact FDs can be defined depending on their capability of suspecting faulty processes (*completeness* property) and of not suspecting correct processes (*accuracy* property).

Arguing that traditional approaches which assume a maximum number of failures f may lead to suboptimal solutions, such as in replication protocols where the number of replicas depend on f, Junqueira et al. proposed in [27] the *survivor set* approach, i.e., the unique collection of minimal sets of correct processes over all executions, each set containing all correct processes of some execution. The principle of the Impact FD also follows the authors' argument: the *threshold* expresses certain margin of failures or false suspicions and the number of failures tolerated by the system is not necessarily fixed but depends on sets of correct processes, their respective impact factors, and *threshold* value. Therefore, the Impact FD presents, what we denoted, the *flexibility property*. The latter expresses its capacity of considering different sets of responses that lead S to trusted states. In this context, we also define in this work, two properties, $PR(IT)_p^S$ and $PR(\diamondsuit IT)_p^S$, which characterize the minimum necessary stability condition of S that ensures confidence (or eventual confidence) in it by the monitor process p. In other words, if $PR(IT)_p^S$ (resp., $PR(\diamondsuit IT)_p^S$) holds, the system S is always (resp., eventually always) trusted by the monitor process p. Note that the Impact FD *threshold/impact factor* approach is strictly more powerful than the maximum number of failures f approach since the latter can be expressed with the former but not the other way around.

Taking into account the problem of solving consensus in asynchronous message-passing systems enriched with failure detectors, we show that the Impact FD of class Impact Omega $I\Omega^U$ (resp., Impact Sigma $I\Sigma^U$) is equivalent to the Omega Ω (resp., Sigma Σ) FD. A failure detector is equivalent to another if there exist an algorithm that transforms the first one into the second one (i.e., the second is reducible to the first one) and an algorithm that transforms the latter into the former. Consequently, a problem that can be solved with one of the FDs can also be solved by the other. It is worth remembering that Ω FD [10] and Σ FD (or Quorum FD) [17] are two fundamental classes of failure detectors since the Ω FD is the weakest one to solve consensus, provided that a majority of processes are correct, while the pair of FDs <Ω,Σ> is the weakest one to solve consensus for any number of process failures. Furthermore, we also show that Σ is reducible to $\diamondsuit IP^U$ and $\diamondsuit IW^U$ FD is equivalent to Omega FD (Ω) if some conditions on the number or failures and/or membership hold.

We also present a *timer-based* distributed algorithm (and its proof of correctness) which implements a Impact FD. It uses the algorithm proposed by Chen et al.[12] to estimate heartbeat message arrivals from monitored processes. The implementation can be applied to systems whose links are *lossy asynchronous* or those whose all (or some) of them have eventually a bounded synchronous behavior ($\diamondsuit-timely$) [3]. Then, based on real trace files collected from nodes of PlanetLab [33], we conducted extensive experiments in order to evaluate the Impact FD. These trace files contained a large amount of data related to the sending and reception of heartbeat messages, including unstable periods of links and message, characterizing, therefore, distributed systems that use FDs based on heartbeat. The testbed of the experiments comprises various configurations with different threshold values, impact factor of nodes, and types of links. For evaluation sake, we used three of the QoS metrics proposed in [12]: *detection time*, *average mistake rate*, and *query accuracy probability*. The Impact FD implementation was also compared to a tradition timer-based FD one that outputs information about failure suspicions of each monitored process. Performance evaluation results confirm the degree of flexible applicability of the Impact FD, that both failures and false suspicions are more tolerated than in traditional FDs, and that the former presents better Qos than the latter if the application is interested in the degree of confidence in the system (trust level) as a whole.

The rest of this document is structured as follows. Section 2 describes some distributed systems for which the Impact FD is suitable. Section 3 outlines some basic concepts of unreliable failure detectors and equivalences between failure detectors. Section 4 presents the Impact failure detector, its characteristics, and some of its properties while Section 5 shows the equivalence of some classes of Impact FD in regard with Σ and Ω classes. In Section 6, we propose a timer-based algorithm that implements the Impact FD considering different systems, defined by the type of their links. The section also includes the proof of correctness of the algorithm. Section 7 presents a set of evaluation results

obtained from experiments conducted with real traces on PlanetLab [33]. Section 8 discusses some existing related work. Finally, Section 9 concludes the technical report and outlines some of our future research directions.

2 Motivation Scenarios

Our proposed approach can be applied to different distributed scenarios and is flexible enough to meet different needs. It is quite suitable for environments where there is node redundancy or nodes with different capabilities. We should point out that both the *impact factor* and the $threshold$ render the estimation of the confidence of S more flexible. Hence, there might be a situation where some processes in S are faulty or suspected of being faulty but S is still considered to be trusted. Furthermore, the Impact FD can easily be configured and adapted to the needs of the application or system requirements. For instance, the application may require a stricter monitoring of nodes during the night than during the day. For this kind of adaptation, it is only necessary to adjust the threshold.

The following examples show some scenarios to which the Impact FD can be applied

Scenario 1: Ubiquitous Wireless Sensor Networks (WSNs) are usually deployed to monitor physical conditions in various places such as geographical regions, agriculture lands, battlefields, etc. In WSNs, there is a wide range of sensor nodes with different battery resources and communication or computation capabilities [24]. However, these sensors are prone to failures (e.g., battery failure, process failure, transceiver failure, etc.) [20]. Hence, it is necessary to provide failure detection and adaptation strategies to ensure that the failure of sensor nodes does not affect the overall task of the network. The redundant use of sensor nodes, reorganization of the sensor network, and overlapping sensing regions are some of the techniques used to increase the fault tolerance and reliability of the network [1].

Let us take as example an ubiquitous WSN which is used to collect environmental data from within a vineyard and is divided into management zones in accordance with different characteristics (e.g., soil properties).

Each zone comprises sensors of different types (e.g., humidity control, temperature control, etc.) and the density of the sensors depends on the characteristics of each zone. That is, the number of sensors can be different for each type of sensor within a given zone. Furthermore, the redundancy of the sensors ensures both area coverage and connectivity in case of failure. Each management zone can thus be viewed as a single set which has sensors of the same type grouped into subsets. This grouping approach allows a threshold to be defined as being equal to the minimum number of sensors that each subset must have to keep the connectivity and application functioning all the time. Moreover, in some situations, there might be a need to dynamically reconfigure the density of the zones. In this case, the *threshold* value would change.

Scenario 2: In large-scale WSN environments, grouping sensor nodes into clusters has been widely adopted aiming the overall system scalability and reduction of resources consumption like battery power and bandwidth. Each $cluster_i$ is composed of a node, denoted cluster head (CH), which performs special tasks (e.g., routing, fusion, aggregation of messages, etc.), and several other sensor nodes (SN). The latter periodically transmit their data to the their corresponding CH node which aggregate and transmit them to the base station (BS) either directly or through the intermediate communication with other CH nodes. In this scenario, the concept of Impact FD can be applied considering each $cluster_i$ as a subset of the system S whose size is initially n_i. When defining the impact factor for the processes of $cluster_i$, two issues should be considered: 1) the failure of CH which implies that the cluster is inaccessible compromising, therefore, the network connectivity and leading to untrusted states of S; 2) When the number of alive SNs drops below a threshold, additional resources must be deployed to replenish the system to maintain its population density. Taking these constraints into ac-

count, we could have: impact factor = 1 to SNs, impact factor = n_i to the CH of $cluster_i$, and threshold for this cluster equals to $threshold_i = n_i + (n_i - f_i)$, where f_i is the maximum number of SN's failures of $cluster_i$, Thus, when either the CH fails or more than f_i SNs fail, the trust level will be below the threshold and the BS must be warned to take some decision.

Scenario 3: A third example might be a system consisting of 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 each backup server, i.e., the system becomes unreliable whenever both the primary server and one or more of the N servers fail (or are suspected of being faulty).

The Impact FD can be applied to all the above scenarios which have the following features: a) the grouping of nodes that have some common characteristics into subgroups (subsets); b) the possibility of having nodes with different levels of relevance and c) the flexibility of some systems in being able to tolerate a margin of failure.

3 Unreliable Failure Detectors

Proposed by Chandra and Toueg in [11], 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 [22], unreliable FDs are so named because they can make mistakes (1) by erroneously suspecting a correct process[1] (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. Although an unreliable FD can not accurately determine the real state of processes, its use increases knowledge about them and encapsulates the uncertainty of the communication delay between two processes [11].

Unreliable failure detectors are usually characterized by two properties: *completeness* and *accuracy*, as defined in [11]. *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 [11]. The combination of these properties yield eight classes of failure detectors. This approach allows the design of fault tolerant applications and proof of their correctness based only on these properties, without having to address, for example, low-level network parameters.

In this work, we are particularly interested in the following completeness and accuracy properties:

- *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.
- *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.

[1] A process is denoted *correct* if it does not crash during the whole execution.

The class of the *eventually perfect* $\diamondsuit P$ (resp., *eventually strong* $\diamondsuit S$) failure detectors satisfies the strong completeness and the eventual strong (resp., eventual weak) accuracy properties; The class of eventually weak failure detectors ($\diamondsuit W$) failure detectors satisfies the weak completeness and the eventual weak accuracy properties. $\diamondsuit W$ is the weakest class allowing to solve consensus in an asynchronous distributed system with the additional assumption that a majority of processes are correct.

Note that the type of accuracy depends on the synchrony or stability of the network. For instance, an algorithm that provides eventual accuracy (strong or weak) may rely on partially synchronous systems which eventually ensure a bound for message transmission delays and processes speed.

From Chandra and Toueg's work, numerous other failure detector implementations and classes have been proposed in the literature. They usually differ in the system assumptions such as synchronous model, type of node(identifiable, anonymous [8], homonymous [5]), type of link [3] [28], [2] (lossy asynchronous, reliable, timely, eventually timely, etc.), behavior properties [31], [3]; type of network (static [6] [28], dynamic [4], [21]), etc. They can also have different implementation choices (timer-based [12],[29], message pattern [31]) and performance or quality of service (QoS) requirements [12]. The type of problem can also define the properties of the FD (mutual exclusion [18], k-set agreement [7], register implementation [17]), etc.

3.1 Omega and Sigma Failure Detectors

Many other classes of failures detectors have been defined in the literature. Two important ones, largely exploited by distributed algorithms and applications, are the classes of Omega (Ω) [10] and Sigma (or Quorum - Σ) [17] failure detectors.

The Leader Failure Detector Omega (Ω): Together with Hadzilacos, Chandra and Toueg extended their work in [10], proposing the leader FD Ω.

The specification of Ω states that eventually all the correct processes trust the same correct process, i.e., it provides an eventual leader election functionality. Ω is also the weakest failure detector to solve consensus in a distributed system, provided that a majority of processes are correct. Furthermore, contrarily to $\diamondsuit S$ and $\diamondsuit W$, the knowledge of membership of the system is not necessary [26]. When it is known, a Ω FD trivially also implements a $\diamondsuit W$ or $\diamondsuit S$ failure detectors.

At each process p, the failure detector module of Ω at p outputs the identity of a single process, denoted $LEADER_p$, such that the following property holds:

Eventual Leadership: There exists a correct process l and a time t after which, for every correct process p, $LEADER_p = l$.

We should point out that at any given time processes do not know if there is a leader; they only know that eventually a leader will be elected by all correct processes and will remain leader.

The Quorum Failure Detector Sigma ($Sigma$): A failure detector Sigma (Σ) outputs, at each correct process of the system and, at any time, a list of processes, called $trusted$ processes, such that:

- *Intersection* : Every two lists of trusted processes intersect;
- *Completeness*: Eventually, every list of processes trusted by a correct process contains only correct processes.

According to [16], the class of Sigma failure detectors is the weakest one to implement a register, in any environment.

The importance of Σ and Ω failures detectors was extended by Delporte-Gallet et. al.[16], proving that the pair <Ω,Σ> is the weakest FD to solve consensus (uniform or not) in asynchronous message-passing where all but one process may fail.

3.2 Reducibility and equivalence of failure detectors

According to [11], reducibility means that there exists an algorithm $T_{D\to D'}$ which transforms a failure detector D into another failure detector D' in an environment ε. 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' in ε, can be solved using D instead in ε. 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*.

Similarly, given two classes of failure detectors C and C', if for each failure detector $D \in C$ there is a failure detector $D' \in C$ such that $D \succeq D'$, we write $C \succeq C'$ and say that C' is weaker than C. So, if $C \succeq C'$, then if a problem is solvable using C', it is also solvable using C). If $C \succeq C'$ and $C' \succeq C$, we write $C \cong C'$ and say that C and C' are equivalent [11].

4 Impact Failure Detector

We consider a distributed system which consists of a finite set of processes $\Pi = \{q_1, \ldots, q_n\}$ with $|\Pi| = n$, $(n \geq 2)$ and that there is one process per node, site, or sensor. Therefore, the word *process* can mean a node, a sensor, or a site. Each process is uniquely identified ($id \mid 1 \leq id \leq n$) and identifiers are totally and consecutively ordered.

Processes can fail by crash and they do not recover. A process is considered correct if it does not fail during the whole execution. We consider the existence of some global time denoted T. A failure pattern is a function $F : T \to 2^{\Pi}$, 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.

A process $p \in \Pi$ monitors a set S of processes of Π. Every process in S is connected to p by a communication link and sends messages to it through this link. Notice that other links among processes of S can exist.

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 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 set S.

We note $correct(F_S) = correct(F) \cap S$ and $faulty(F_S) = faulty(F) \cap S$.

4.1 Impact Factor and Subsets

Each process $q \in S$ has an *impact factor* ($I_q | I_q > 0 : I_q \in \mathbb{R}$). Furthermore, set S can be partitioned into m disjoint subsets ($S = \{S_1, S_2, \ldots S_m\}$). Notice that the grouping feature of the Impact FD allows the processes 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 $S^* = \{S_1^*, S_2^*, \ldots S_m^*\}$ be the set S partitioned into m disjoint subsets where each S_i^* is a set composed of the tuple $\langle id, I \rangle$, where id is a process identifier and I is the value of the impact factor of the process in question.

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

4.2 Trust Level

We denote ${trusted_p}^S(t)$ the set that contains the processes of S that are not considered faulty by p at $t \in T$. The *trust level* at $t \in T$ of process $p \notin F(t)$ in relation to S is denoted ${trust\_level_p}^{S^*}$. We have then ${trust\_level_p}^{S^*}(t) =$
$Trust\_level({trusted_p}^S, S^*)$, where the function $Trust\_level({trusted_p}^S, S^*)$ returns, for each subset S_i^*, the sum of the impact factor of the elements $\langle id_q, I_q \rangle$ of S_i^* such that $id_q \in trusted$.

$$Trust\_level(trusted, S^*) = \{trust\_level_i \mid trust\_level_i = \sum_{\substack{j=1, \\ j \in (trusted \cap S_i^*)}}^{|S_i^*|} I_j \ , 1 \leq i \leq |S^*|\}$$

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 processes of S. Note that if all processes of S_i^* have failed $trust\_level_i = 0$.

4.3 Margin of Failures

An acceptable margin of failures, denoted ${threshold}^{S^*}$, 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 p 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 *untrusted* by p at t.

Three 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 is still trusted; (2) the Impact FD can be easily configured to adapt to the needs of the environment; (3) the ${threshold}^{S^*}$ can be tuned to provide a more restricted or softer monitoring. 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 different subset of S^*.

4.4 Examples

Table 1 shows several examples of sets and their respective thresholds. In the first example (a) there is just one subset with three processes. Each process has impact factor equal to 1 and the *threshold* defines that the sum of impact factor of non faulty processes must be at least equals to 2, i.e., the system is considered trusted whenever there are two or more correct processes. Example (b) shows a configuration where processes must be monitored individually. Each process is the only element of a subset and the threshold defines that if any of the processes fails, the system is not trusted anymore. In the third example (c), S has two sets with three processes each. The *threshold* requires at least two correct processes in each subset. The last example (d) has a single subset with five processes with different impact factors. The *threshold* defines that the set is trusted whenever the sum of impact factor of correct processes is at least equals to seven.

In Table 2, we consider a set S^* composed by three subsets: S_1^*, S_2^*, and S_3^* ($S^* = \{\{\langle q_1,1\rangle, \langle q_2,1\rangle, \langle q_3,1\rangle\}$, $\{\langle q_4,2\rangle, \langle q_5,2\rangle, \langle q_6,2\rangle\}$,
$\{\langle q_7,3\rangle, \langle q_8,3\rangle, \langle q_9,3\rangle\}\}$). The values of ${threshold}^{S^*} = \{1,4,6\}$ define that the subset S_1^* (resp., S_2^* and S_3^*) must have at least one (resp., two) correct process. The table shows several possible outputs for

Table 1: Examples of sets and threshold

	S^*	$threshold^{S^*}$
a	$\{\{\langle q_1,1\rangle,\langle q_2,1\rangle,\langle q_3,1\rangle\}\}$	{2}
b	$\{\{\langle q_1,1\rangle\},\{\langle q_2,1\rangle\},\{\langle q_3,1\rangle\}\}$	{1,1,1}
c	$\{\{\langle q_1,1\rangle,\langle q_2,1\rangle,\langle q_3,1\rangle\},\{\langle q_4,2\rangle,\langle q_5,2\rangle,\langle q_6,2\rangle\}\}$	{2,4}
d	$\{\{\langle q_1,1\rangle,\langle q_2,1\rangle,\langle q_3,1\rangle,\langle q_4,5\rangle,\langle q_5,5\rangle\}\}$	{7}

FD (${I_p}^S$) depending of process failures: the set S^* is considered trusted at t if, for each subset S_i^*, $trust\_level_i(t) \geq threshold_i$.

Table 2: Example of FD (${I_p}^S$) output: S^* has three subsets

t	**F(t)**	${trusted_p}^S(t)$	${trust\_level_p}^{S^*}(t)$	**Status at t**
1	$\{q_2\}$	$\{q_1,q_3,q_4,q_5,q_6,q_7,q_8,q_9\}$	{2,6,9}	Trusted
2	$\{q_1,q_2,q_5\}$	$\{q_3,q_4,q_6,q_7,q_8,q_9\}$	{1,4,9}	Trusted
3	$\{q_1,q_2,q_5,q_6\}$	$\{q_3,q_4,q_7,q_8,q_9\}$	{1,2,9}	Untrusted

$S^* = \{\{\langle q_1,1\rangle,\langle q_2,1\rangle,\langle q_3,1\rangle\},\{\langle q_4,2\rangle,\langle q_5,2\rangle,\langle q_6,2\rangle\},\{\langle q_7,3\rangle,\langle q_8,3\rangle,\langle q_9,3\rangle\}\}$
$threshold^{S^*} = \{1,4,6\}$

4.5 Flexibility of the Impact FD

The *flexibility* of the Impact FD characterizes its capability in accepting different set of responses that lead to a trusted state of S. We define PS as the set that contains all possible subsets of processes which satisfy a defined *threshold*:

$$PS = TPowerSet(S^*, threshold^{S^*}) | TPowerSet(S^*, threshold^{S^*}) = \times\, PowerSet({S_i}^*, {threshold_i}^{S^*})$$

where $\times S_i$ corresponds to the cartesian product of several sets.

Initially, the *TPowerSet* function generates the power set [2] 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 selected. That is, the output is the sets of possible trusted set that satisfy the threshold for each subset ${S_i}^*$. Following this, the cartesian product is applied to generate all possible combinations, i.e., all the generated subsets of processes satisfy the $threshold^{S^*}$.

Let's consider the following example:

$S^* = \{\{\langle q_1,1\rangle,\langle q_2,1\rangle\},\{\langle q_3,1\rangle,\langle q_4,1\rangle\},\{\langle q_5,1\rangle,\langle q_6,1\rangle\}\}$
threshold$^{S^*}$ = {1,1,1}
$PS = TPowerset(S^*, threshold^{S^*})$

PowerSet(S_1^*, $threshold_1$) = $\{\{q_1\},\{q_2\},\{q_1,q_2\}\}$
PowerSet(S_2^*, $threshold_2$) = $\{\{q_3\},\{q_4\},\{q_3,q_4\}\}$
PowerSet(S_3^*, $threshold_3$) = $\{\{q_5\},\{q_6\},\{q_5,q_6\}\}$
PS = PowerSet(S_1^*, threshold$_1$) × PowerSet(S_2^*, threshold$_2$) × PowerSet(S_3^*, threshold$_3$)

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

PS = $\{\{q_1,q_3,q_5\},\{q_1,q_3,q_6\},\{q_1,q_3,q_5,q_6\},$
$\{q_1,q_4,q_5\},\{q_1,q_4,q_6\},\{q_1,q_4,q_5,q_6\},$
$\{q_1,q_3,q_4,q_5\},\{q_1,q_3,q_4,q_6\},\{q_1,q_3,q_4,q_5,q_6\},\ldots\}$

For instance, if ${trusted_p}^S(t_1) = \{q_1,q_3,q_5\}$ and ${trusted_p}^S(t_2) = \{q_1,q_3,q_4,q_6\}$, ${trusted_p}^S(t_1)$ and ${trusted_p}^S(t_1) \in PS$, and, therefore, p considers that the system S is trusted at both t_1 and t_2.

We define now two properties, $PR(IT)_p^S$ and $PR(\Diamond IT)_p^S$, that characterize the stability condition that ensures the confidence (or eventual confidence) of p on S.

Impact Threshold Property - $PR(IT)_p^S$: For a failure detector of a correct process p, the set ${trusted_p}^S$ is always a subset of PS.

$$PR(IT)_p^S \equiv p \in correct(F), \forall t \geq 0, {trusted_p}^S(t) \in PS$$

Eventual Impact Threshold Property - $PR(\Diamond IT)_p^S$: For a failure detector of a correct process p, there is a time after which the set ${trusted_p}^S$ is always a subset of PS.

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

If $PR(IT)_p^S$ (resp., $PR(\Diamond IT)_p^S$) holds, the system S is always (resp., eventually always) trusted by p.

4.6 Classes of Impact FD

Similarly to the *completeness* and *accuracy* properties defined in [11] (see Section 3), we define the following ones:
${Impact\ completeness_p}^S$: For a failure detector of a correct process p, there is a time after which p does not trust any crashed process of S;

$$\exists t \in T, p \in correct(F), \forall q \in faulty(F_S) : \forall t' \in T \geq t, q \notin {trusted_p}^S(t')$$

${Eventual\ impact\ strong\ accuracy_p}^S$: For a failure detector of a correct process p, there is a time after which all correct processes of S belong to ${trusted_p}^S$;

$$\exists t \in T, \forall t' \in T \geq t, p \in correct(F), \forall q \in correct(F_S) : q \in {trusted_p}^S(t')$$

${Eventual\ impact\ weak\ accuracy_p}^S$: For a failure detector of a correct process p, there is a time after which some correct process of S always belongs to ${trusted_p}^S$.

$$\exists t \in T, \forall t' \in T \geq t, p \in correct(F), \exists q \in correct(F_S) : q \in {trusted_p}^S(t')$$

Let consider that p in S and $S = \Pi$
We can then define some classes of Impact FD, similarly those defined in [11] and [17]:

- $\Diamond IP$ (*Eventual Perfect Impact Class*): For $S = \Pi$, $\forall p \in correct(F)$, ${impact\ completeness_p}^S$ and ${eventual\ impact\ strong\ accuracy_p}^S$ properties are satisfied;
- $\Diamond IS$ (*Eventual Strong Impact Class*): For $S = \Pi$, $\forall p \in correct(F)$, ${impact\ completeness_p}^S$ and ${eventual\ impact\ weak\ accuracy_p}^S$ properties are satisfied;
- $\Diamond IW$ (*Eventual Weak Impact Class*): For $S = \Pi$, $\exists p \in correct(F)$ such that ${impact\ completeness_p}^S$ property (*weak completeness*) is satisfied and $\forall p \in correct(F)$, *eventual impact weak* ${accuracy_q}^S$ property is satisfied;

- $I\Omega$: (*Impact Omega Class*): For $S = \Pi$, $\forall t \geq 0, |trusted(t)| = 1$, and $\exists l \in correct(F)$ such that $\exists t_1 \in T, \forall t_2 \in T \geq t_1, \forall p \in correct(F), {trusted_p}^S(t_2) = \{l\}$.
- $I\Sigma$: (*Impact Sigma Class*): For $S = \Pi$, the two following properties are satisfied:
  Intersection: $\forall t_1, t_2 \in T, \forall p_1, p_2 \in \Pi : {trusted_p}^S(t_1) \cap {trusted_p}^S(t_2) \neq \emptyset$
  Completeness: $\exists t \in T, \forall p \in correct(F) : \forall t' \in T \geq t, {trusted_p}^S(t') \subseteq correct(F)$

We point out that the trust level output of the failure detectors of the above classes depends on S^*, i.e., the impact factor assigned to the processes as well as how they are grouped in subsets.

5 Impact FD Equivalences

By considering that all processes of $S = \Pi$ have impact value equals to its identifier value and each process belongs to a different subset of S^* , we show that $I\Sigma^U$ (resp.$I\Omega^U$) FD is equivalent to Σ (resp., $Omega$) FD. In addition, Σ is reducible to $\diamondsuit IP^U$, provided there exist a majority of correct processes, and $\diamondsuit IW^U$ FD is equivalent to Omega FD (Ω), provided that the membership of the system is known. Both Σ and Ω FDs were defined in Section 3 while $I\Sigma^U$, $I\Omega^U$, $\diamondsuit IP^U$, and $\diamondsuit IW^U$ FDs will be defined in this section.

Let's assume that $\Pi = \{q_1, \ldots, q_n\}$ are uniquely identified by $\{1, 2, \ldots, n\}$ respectively with $|\Pi| = n$, ($n \geq 2$). For both cases, i.e., $Omega$'s and $Sigma$'s reductions, we consider that p in S, $S = \Pi$.

Furthermore, S^* requires the *unique identifier format*: $|S^*| = n, \forall S_i^* 1 \leq i \leq n, S_i^* = \{\langle \_, i \rangle\}$, i.e., each of the n subsets of S^* has just one process of S whose impact factor is equal to its identifier. This way, it is possible to deduce, by the output of the Impact FD of process p (*trust_level*), the processes that are trusted by p. For instance, consider the following configuration of S^* and a possible trust level output of the Impact FD of p at t (processes q_1 and q_4 suspected of failure):

$$S^* = \Pi^* = \{\{\langle q_1, 1 \rangle\}, \{\langle q_2, 2 \rangle\}, \{\langle q_3, 3 \rangle\}, \{\langle q_4, 4 \rangle\}, \{\langle q_5, 5 \rangle\}\}$$

$$trust\_level_p^S(t) = \{0, 2, 3, 0, 5\}$$

The set of processes trusted by p at t corresponds to those $trust\_level_i$ ($1 \leq n$) of $trust\_level(t)$ which are greater than 0.

$$trusted(t) = \{2, 3, 5\}$$

We denote I^U the set of failure detectors of Impact FD class that require the *unique identifier* format for S^*. Similarly, $\diamondsuit IP^U$, $\diamondsuit IS^U$, $I\Sigma^U$, $I\Omega^U$, and $\diamondsuit IW^U$ FDs are $\diamondsuit IP$, $\diamondsuit IS$, $I\Sigma$, $I\Omega$, and $\diamondsuit IW$ FDs respectively that also require the same S^* *unique identifier* format.

The following functions are used by the reductions algorithms:

- $Trust\_levelToProcs(trust\_level)$: returns the set of processes of the $trust\_level$ whose $trust\_level_i$ is greater than zero, i.e., the processes considered trusted:

$$Trust\_levelToProcs(trust\_level) = \{trust\_level_i \mid trust\_level_i > 0; 1 \leq i \leq |\Pi|\}$$

- $ProcsToTrust\_level(trusted)$: returns the set $trust\_level$ related to the *trusted* set, composed by the identifiers of processes that are trusted: $trust\_level_i$ is equal to i, if i belongs to $trusted$; otherwise it is equal to 0.

$$ProcsToTrust\_level(trusted) = \{trust\_level_i \mid trust\_level_i = i \text{ if } i \in trusted;$$
$$\text{else } trust\_level_i = 0; 1 \leq i \leq |\Pi|\}$$

We consider that the FDs and the algorithms presented in this section run on all nodes of Π. Note that the input of Ω and Σ FDs is Π while the input of $I\Sigma^U$, $I\Omega^U$, $\diamondsuit IP^U$, and $\diamondsuit IW^U$ is Π^*.

5.1 Equivalence between $I\Sigma^U$ FD and Σ FD

- Σ is reducible to $I\Sigma^U$ ($I\Sigma^U \succeq \Sigma$): Algorithm 1
- $I\Sigma^U$ is reducible to Σ ($\Sigma \succeq I\Sigma^U$): Algorithm 2

Algorithm 1 Transforming $I\Sigma^U$ to Σ

```
1:  Begin
    Input
2:      Π
    Init
3:      S* = ∅
4:      for i = 1 to Π do
5:          S* = S* ∪ {{⟨q_i, i⟩}}
6:      end for
    T1
7:      Upon invocation of Σ() do
8:          return Trust_levelToProcs(IΣ(S*))
9:      end
10: End
```

Algorithm 1 transforms the output of Impact $I\Sigma^U$ FD to the output of Σ FD. When invoked in p, $I\Sigma^U$ returns the trust level value of p in relation to processes of Π that p trusts. The function $Trust\_levelToProcs$ then transforms the trust level to the set of trusted processes (line 8).

Algorithm 2 Transforming Σ to $I\Sigma^U$

```
1:  Begin
    Input
2:      Π*
    Init
3:      S = ∅
4:      for i = 1 to Π* do
5:          S = S ∪ {i}
6:      end for
    T1
7:      Upon invocation of IΣ() do
8:          return ProcsToTrust_level(Σ(S))
9:      end
10: End
```

The Algorithm 2 transforms the output of Σ FD to the output of Impact $I\Sigma^U$ FD, i.e., the trust level. When invoked in p, the Sigma FD returns the set $trusted$ which contains the identifier of trusted processes. This set is then transformed in the trust level (line 8).

Sketch of Proof

Lemma 1. *Algorithm 1 transforms the output of $I\Sigma^U$ FD to the output of Σ FD.*

Proof. Immediate from the intersection and completeness properties of $I\Sigma^U$ and function $Trust\_levelToProcs$ that transforms a trust level value to a set of trusted processes identifiers. □

Lemma 2. *The Algorithm 2 transforms the output of* Σ *FD to the output of* $I\Sigma^U$ *FD.*

Proof. Immediate from the intersection and completeness properties of $I\Sigma$ FD and function $ProcsToTrust\_level$ that transforms a set of trusted processes identifiers to a trust level value. □

Theorem 1. Σ *FD is equivalent to* $I\Sigma^U$ *FD*

Proof. The theorem holds directly from Lemma 1 and Lemma 2. □

5.2 Equivalence between $I\Omega^U$ FD and Ω FD

- Ω is reducible to $I\Omega^U$ ($I\Omega^U \succeq \Omega$): Algorithm 3
- $I\Omega^U$ is reducible to Ω ($\Omega \succeq I\Omega^U$): Algorithm 4

Algorithm 3 Transforming $I\Omega^U$ to Ω

```
1:  Begin
    Input
2:      Π
    Init
3:      S* = ∅
4:      for i = 1 to Π do
5:          S* = S* ∪ {{⟨q_i, i⟩}}
6:      end for
    T1
7:      Upon invocation of Ω() do
8:          trusted = Trust_levelToProcs(IΩ(S*))
9:          return l such that l ∈ trusted
10:     end
11: End
```

Algorithm 3 transforms the output of Impact $I\Omega^U$ FD to the output of Ω FD. When invoked in p, $I\Omega^U$ returns the trust level value of p in relation to a process that p considers as leader. Then, the function $Trust\_levelToProcs$ returns a trusted set composed only by the leader process, which is returned by the function (line 8).

The Algorithm 4 transforms the output of Ω FD to the output of Impact $I\Omega^U$ FD, i.e., the trust level. When invoked in p, the $Omega$ FD returns a process that it considers as leader which is then included in the trusted set. This set is transformed in the trust level (line 8).

Sketch of Proof

Lemma 3. *Algorithm 3 transforms the output of* $I\Omega^U$ *FD to the output of* Ω *FD.*

Proof. Immediate from the leadership property of $I\Omega^U$ and function $Trust\_levelToProcs$. □

Lemma 4. *The Algorithm 4 transforms the output of* Ω *FD to the output of* $I\Omega^U$ *FD.*

Proof. Immediate from the leadership property of $I\Omega$ FD and function $ProcsToTrust\_level$. □

Theorem 2. Ω *FD is equivalent to* $I\Omega^U$ *FD*

Proof. The theorem directly holds from Lemma 3 and Lemma 4. □

Algorithm 4 Transforming Ω to $I\Omega^U$

```
1:  Begin
    Input
2:      Π*
    Init
3:      S = ∅
4:      for i = 1 to Π* do
5:          S = S ∪ {i}
6:      end for
    T1
7:      Upon invocation of IΩ() do
8:          trusted = {Ω(S)}
9:          return ProcsToTrust_level(trusted)
10:     end
11: End
```

5.3 Σ FD is reducible to $\diamondsuit IP^U$ with a majority of correct processes

We consider that every pair of processes in Π is connected by a bidirectional link which does not lose messages, neither corrupts them, nor generates spontaneous messages. In addition, there exist a majority of correct processes, i.e., the maximum number o failures $f < |\Pi|/2$. Then, Σ FD is reducible to $\diamondsuit IP^U$:

Algorithm 5 Transforming $\diamondsuit IP^U$ to Σ

```
1:  Begin
    Input
2:      Π
    Init
3:      trust_level = ∅ ; S* = ∅
4:      for i = 1 to |Π| do
5:          S* = S* ∪ {{⟨q_i, i⟩}}
6:      end for
    Task T1
7:      Upon invocation of Σ() do
8:          trusted = ∅
9:          repeat
10:             trust_level = ◇IP(S*)
11:             trusted = trusted ∪ Trust_levelToProcs(trust_level)
12:         until |trusted| > |Π|/2
13:         return trusted
14:     end
15: End
```

Algorithm 5 transforms the output of Impact $\diamondsuit IP^U$ FD to the output of Sigma FD. When invoked in p, $\diamondsuit IP^U$ returns the trust level value of p in relation to processes of Π (line 10). Then, the function $Trust\_levelToProcs(trust\_level)$ is called. The algorithm returns when there are a majority of process i whose $trust\_level_i$ is greater than zero, i.e., a majority of processes considered trusted by p (line 12).

Sketch of Proof

Lemma 5. *Let's consider that the call to $\diamondsuit IP()$ never blocks. The invocation of $\Sigma()$ by p (Algorithm 5) always returns a set of processes whose size is greater than $|\Pi|/2$.*

Proof. Since the call to $\diamondsuit IP()$ (line 10) never blocks by assumption, the only way for function $\Sigma()$ to not return from a call would be if it looped forever because the *until* condition of line 12 was never satisfied. However, since there is no message loss by assumption, p eventually receives every heartbeat message sent by other processes. Furthermore, since $f < |\Pi|/2$ by assumption, if all the f failures take place, the *eventual impact accuracy* of $\diamondsuit IP$ ensures that eventually the set trusted will contain a majority of processes (the correct ones) of the system, avoiding, thus, that the algorithm blocks permanently in line 12. Therefore, the condition of this line always becomes true and, by calling function *Trust_levelToProcs*, function $Sigma()$ returns a set of trusted processes whose size is greater than $|\Pi|/2$. □

Lemma 6. *Algorithm 5 ensures the intersection property of the Σ FD.*

Proof. By assumption, all processes of Π execute both $\diamondsuit IP^U$ FD and Algorithm 5. Therefore, the lemma holds directly from Lemma 5 since, when invoked by p, Algorithm 5 always outputs a set of at least $|\Pi|/2+1$ processes. □

Lemma 7. *Algorithm 5 ensures the completeness property of the Σ FD.*

Proof. By assumption, all processes of Π execute both $\diamondsuit IP^U$ FD and Algorithm 5. Thus, $\forall p \in correct(F)$, the lemma holds directly from the $completeness_p^{\Pi}$ property of $\diamondsuit IP$ FD. □

Theorem 3. *Algorithm 5 transforms $\diamondsuit IP^U$ to Σ FD.*

Proof. The theorem holds directly from Lemma 6 and Lemma 7. □

We should point out that $\diamondsuit IP^U$ FD is not reducible to Σ FD since the *eventual impact strong accuracy* of $\diamondsuit IP^U$ FD can not be ensured from the output of Σ FD.

5.4 Equivalence between $\diamondsuit IW^U$ FD and Ω FD

We consider that $f < n-1$. $\diamondsuit IW^U$ is equivalent to Ω. Note that the membership (Π) is known by all processes [26]:

- Ω is reducible to $\diamondsuit IW^U$ ($\diamondsuit IW^U \succeq \Omega$). The idea is to transform $\diamondsuit IW^U$ FD to $\diamondsuit W$ FD (Algorithm 6) and then use any algorithm of the literature, such as [10], [32], [13] (see Section 8), which transforms $\diamondsuit W$ to Ω.
- $\diamondsuit IW^U$ is reducible to Ω ($\Omega \succeq \diamondsuit IW^U$): Algorithm 7.

$\diamondsuit IW^U$ FD can be trivially reduced to $\diamondsuit W$ FD (Algorithm 6) as well as Ω FD to $\diamondsuit IW^U$ FD (Algorithm 7).

Sketch of Proof

Lemma 8. *Algorithm 6 ensures the* completeness *property of the $\diamondsuit W$ FD.*

Algorithm 6 Transforming $\diamondsuit IW^U$ to $\diamondsuit W$

Begin
Input
  Π
Init
  $S^* = \emptyset$;
  for $i = 1$ *to* $|\Pi|$ **do**
    $S^* = S^* \cup \{\{\langle q_i, i \rangle\}\}$
  end for
Task T1
  Upon invocation of $\diamondsuit W$ **do**
    $trust\_level = \diamondsuit IW^U(S^*)$
    $suspected = \Pi - Trust\_levelToProcs(trust\_level)$
    return $suspected$
  end
End

Algorithm 7 Transforming Ω to $\diamondsuit IW^U$

Begin
Input
  Π^*
Init
  $trusted = \emptyset$; $S = \emptyset$;
  for $i = 1$ *to* $|\Pi^*|$ **do**
    $S = S \cup \{i\}$
  end for
Task T1
  Upon invocation of $\diamondsuit IW$ **do**
    $trusted = \{\Omega(S)\}$
    return $ProcsToTrust\_level(trusted)$
  end
End

Proof. At every invocation of $\diamondsuit W$ by p, Algorithm 6 outputs a set of suspected processes composed by all processes of Π which are not currently trusted by p (line 9). By assumption, $\diamondsuit IW^U$ and Algorithm 6 are executed by all nodes of Π. $\diamondsuit IW^U$ FD ensures that $\exists p \in correct(F)$ such $\forall q \in faulty(F) : \exists t \in T, \forall t' \in T \geq t, q \notin trusted_p{}^{\pi}(t')$. By Algorithm 6, all the faulty processes also belong to the $suspect$ set (line 9). Hence, $\exists p \in correct(F), \forall q \in faulty(F), : \exists t \in T, \forall t' \in T \geq t, q \in suspect$, and, thus, the *weak completeness* property of $\diamondsuit W$ FD is satisfied.

□

Theorem 4. *Algorithm 6 transforms $\diamondsuit IW^U$ FD to $\diamondsuit W$ FD.*

Proof. Lemma 8 ensures the *weak completeness* of $\diamondsuit W$ FD. Since $\diamondsuit IW^U$ and Algorithm 6 are executed by all nodes of Π by assumption, the *eventual impact weak* $accuracy_p^{\Pi}$ property of $\diamondsuit IW^U$ FD is satisfied $\forall p \in correct(F)$ and, therefore, the *eventual impact weak accuracy* of $\diamondsuit W$ is also satisfied. Thus, the theorem holds. □

Lemma 9. *Algorithm 7 executed by the correct process p ensures both the* impact $completeness_p^{\Pi}$ *and the* eventual impact weak $accuracy_p^{\Pi}$ *of* $\diamondsuit IW^U$.

Proof. The *eventual leadership* property of Ω FD ensures that there exists $t \in T$ and a correct process $l \in \Pi$ such that for all correct processes $\in \Pi$, $\forall t' \in T \geq t$, $Omega() = l$ and $trusted = \{l\}$ (line 8). Consequently, after t, no faulty processes belong to *trusted* set ($completeness_p^{\Pi}$ of $\diamondsuit IW^U$) and there exists a correct process l which is trusted by all $p \in correct(F)$ (*eventual impact weak* $accuracy_p^{\Pi}$ of $\diamondsuit IW^U$). □

Theorem 5. *Algorithm 7 transforms Ω FD to $\diamondsuit IW^U$ FD.*

Proof. The theorem holds directly from Lemma 9 and the call to *ProcsToTrust_level* (line 9) that transform a set of processes to a trust level value. □

6 Implementation 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) and the type of nodes. In this section, we present a *timer-based* implementation of the Impact FD (Algorithms 9 and 10).

The system S consists of n processes grouped in m subsets. The monitor process $p \notin S$.

Process synchrony: We consider that each process has a local clock that can accurately measure intervals of time, but the clocks of the processes are not synchronized. Processes are synchronous, i.e., there is an upper bound on the time required to execute an instruction. For simplicity, and without loss of generality, we assume that local processing time is negligible with respect to message communication delays.

Links and type of systems: For the current implementation, we consider that links are directed (either unidirectional or bidirectional) and there exists a link from q ($\forall q \in S$) to p.

Every link between p and q satisfies the following *integrity property*: p receives a message m from q at most once, and only if q previously sent m to p. In other words, communication links cannot create or alter messages. Links are not assumed to be FIFO. Concerning loss property and link synchrony, we consider the following types of links as defined in [3]:

- $lossy\ asynchronous$: A link that satisfies the *integrity* property and there is no bound on message delay. Note that, in this case, a message m sent over the link can be lost. However, if m is not lost, it is eventually received at its destination.
- $(Typed) fair\ lossy$: Assuming that each message has a type, link is *fair lossy* if, for every type infinitely many messages are sent, then infinitely many messages of each type are received (if the receiver process is correct).
- $\diamondsuit-timely$: A link that satisfies the *integrity* property and the following $\diamondsuit-timeliness$ property: there exists δ and a time t such that if q sends a message m to p at time $t' \geq t$ and p is correct, then p receives m from q by time $t' + \delta$. The maximum message delay δ and the time t after which it holds are not known. Note that messages sent before time t can be lost.

We then define the following types of system:

- *AS*: denotes a *lossy asynchronous* system with lossy asynchronous links;
- *F-AS*: denotes a *fair lossy asynchronous* system with fair lossy links;
- *W-ET*: denotes a *weak eventually timely* system: a system where some links are $\diamondsuit-timely$ while the others are lossy asynchronous;
- *S-ET*: denotes a *strong eventually timely* system: a system where all links are $\diamondsuit-timely$;
- *S-ET*-Π: A system which is a *S-ET* system such that p in S, $S = \Pi$, every pair of processes in S is connected either by a pair of directed links (with opposite directions) or bidirectional links, and all processes of Π executes the Impact FD algorithms.
- *W-ET*-Π: A system which is a *W-ET* system such that p in S, $S = \Pi$, every pair of processes in S is connected either by a pair of directed links (with opposite directions) or bidirectional links, and all processes of Π execute the Impact FD algorithms. Moreover, there exists a correct process q_1 in Π, such that, for all process q_2 in Π, $q_1 \neq q_2$, q_1 is connected to q_2 by a $\diamondsuit-timely$ link (similarly to the definition of $\diamondsuit - source$ of [2]).

Note that a *S-ET* is also a *W-ET* and *S-ET*-Π (resp. *W-ET*-Π) is also a *S-ET* (resp., *W-ET*). Our Impact FD implementation can be applied to all of these systems.

Figure 1 shows three types of system. The first one (a) represents an *AS* system where all links are $lossy\ asynchronous$ while system (b) shows a *W-ET* where some links are $\diamondsuit-timely$ and others are *lossy asynchronous*. Finally, the last one (c) shows a *W-ET*-Π where site q_1 is a $\diamondsuit - source$.

Our implementation (Algorithms 9 and 10) uses timers to detect failures of processes. Process q periodically sends (*heartbeat*) messages to process p, that is responsible for monitoring process q. If p does not receive such a message from q after the expiration of the timer, it removes q from its list of trusted processes.

Chen's heartbeat estimation arrival: Algorithm 9 uses the algorithm proposed by Chen et al.[12], denoted Chen's algorithm in this work, which computes the timeout value for waiting for a heartbeat message from each monitored process.

Chen's algorithm uses arrival times sampled in the recent past to compute an estimation of the arrival time of the next heartbeat. Then, timeout value is set according to this estimation and a safety margin (β). It is recomputed at each timer expiration.

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

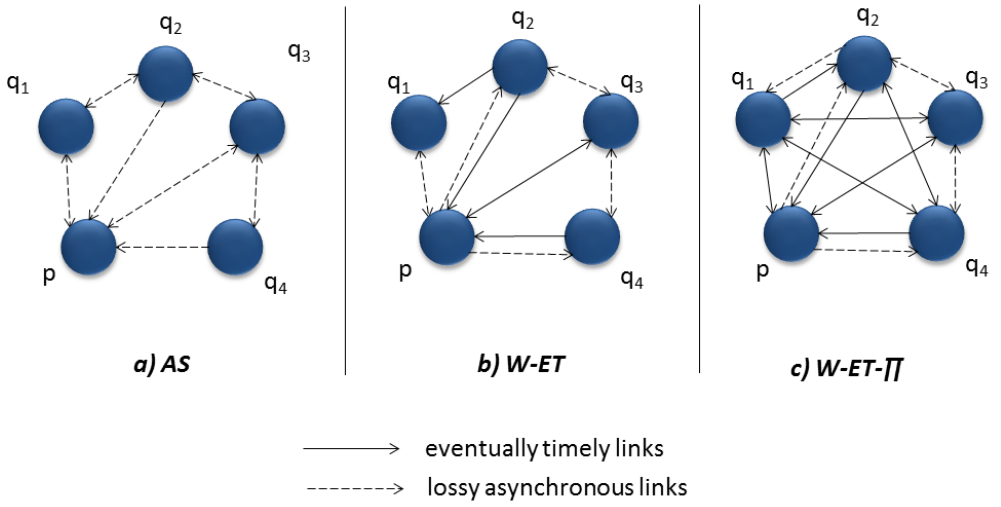


Figure 1: Scenarios of systems

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:

$$\tau_{(k+1)} = \beta + EA_{(k+1)} \; (next\ freshness\ point)$$

In Algorithm 9, Chen's algorithm is executed by the function $Timeout$ (Algorithm 8) which calculates the arrival estimation of the next heartbeat for process q. Furthermore, if the link is $\Diamond-timely$, a η value is added to the timeout. The η has an initial zero value and is incremented whenever p falsely suspects q (line 9 of Algorithm 9). Such an increment ensures that, if the link is $\Diamond-timely$ and stable, i.e., the delay bound δ verifies forever, the heartbeat arrival estimation time will be always equal or greater than the actual arrival time for every heartbeat and, therefore, there will be no more estimation mistakes and, therefore no more false suspicions.

Algorithm 8 Function Timeout

```
1: function TIMEOUT(q, η, model)
2:     if model = *−AS then                    ▷ AS or F-AS system
3:         τ_q = β + EA_q
4:     else
5:         τ_q = β + EA_q + η
6:     end if
7:     return τ_q
8: end function
```

Algorithm 9 is executed by the monitor process p while algorithm ,10 by all processes of S. The following local variables are used by the algorithm:

- *trusted*: set of processes considered not faulty py p;
- $\eta[]$: keeps the timeout increment of each process in S;
- $timer[]$: is set to the timeout value at each timer expiration.

Algorithm 9 Timer-based Impact FD Algorithm for p

```
1:  Begin
    Input
2:      S*, model, η
    Init
3:      trusted = S
4:      ∀q ≠ p : reset timer[q] → Timeout(q, 0, model); η[q] = 0

    Task T1 - Upon reception of ALIVE from q
5:      trusted = trusted ∪ {q}
6:      reset timer[q] → Timeout(q, η[q], model)

    Task T2 - When timer[q] expires
7:      trusted = trusted \ {q}
8:      if model = *−ET then                          ▷ W-ET or S-ET system
9:          η[q] = η[q] + η
10:     end if
11:     reset timer[q] → Timeout(q, η[q], model)

    Task T3
12:     Upon invocation of Impact() do
13:         return Trust_level(trusted, S*)
14:     end
15: End
```

Algorithm 10 Timer-based Impact FD Algorithm for q in S

```
1:  Begin
    Input
2:      p
    Task T1 - Repeat forever every Δ time unit
3:      send(ALIVE) to p

4:  End
```

In Algorithm 9, p receives as input the set S^*, the increment time η for the timeout estimation (used when occurs false suspicions in *W-ET* or *S-ET* systems), and the $model$ of the system (*AS, F-AS, W-ET* or *S-ET*). Note that by receiving S^*, the algorithm knows S, the impact factor of all processes of S, the number of subgroups m, and how processes are grouped.

At the initialization, $trusted$ is initialized with the set of processes. Then, for each process q in S ($q \neq p$), p initializes the timer that will control the arrival of heartbeat messages from q (line 4).

Upon reception of an ALIVE message from q (Task T1), q is added to the $trusted$ set (line 5) and the timeout related to q is recomputed (line 6).

In task T2, q is considered faulty by p and, therefore, removed from *trusted* (line 7). Furthermore, if the system is *W-ET* or *S-ET*, the timeout must be adjusted with a higher value (line 9). The timeout related to q is then recomputed ((line 11).

Task T3 handles the invocation of the *Impact()* function, which computes the *trust_level* of each subset and returns the trust level related to the current trusted processes which are trusted by p.

In Algorithm 10, every monitored process q of S sends periodically, every Δ units of time, an *ALIVE* message to its input observer p in order to inform the latter that it is alive (Task T1).

Note that if $p \in S$, like in *S-ET*-Π or *W-ET*-Π, all processes of Π execute the two algorithms behaving, thus, as both a monitor and a monitored process. In this case, the primitive $send$ in line 3 of Algorithm 10 is replaced by the primitive $broadcast$, i.e., every processes periodically sends a heartbeat to all processes of S.

6.1 Sketch of Proof

In this section, we prove the correctness of some properties of Algorithm 9 and 10.

Theorem 6. *If p is correct, Algorithms 9 and 10 satisfy the impact completeness property for p in relation to S.*

Proof. Let's consider that at t, $S_f = faulty(F_S)$ (i.e., all failures of processes in S took place) and that all the $ALIVE$ messages (heartbeats) sent by these faulty processes before they crashed were delivered to p. Thus, after t, p will receive no more $ALIVE$ messages from processes of S_f. Then, $\forall q \in S_f$, in the next expiration of the timer[q] after t, q will be removed from $trusted$ (line 7). Moreover, since p will receive no more $ALIVE$ messages from q, line 5 will never be executed for q anymore and, therefore, q will nevermore be included in $trusted$. Therefore, $\exists t' > t, \forall t'' \geq t', \forall q \in faulty(F_S) : q \notin trusted_p(t'')$.

□

Lemma 10. *If S is a* W-ET *or* S-ET *system, p is correct, and $q \in correct(F_S)$ is linked to p by a $\diamond-timely$, there is a time t after which q is always trusted by p.*

Proof. Let's denote T_q the stabilization time of the link q from p, i.e., $\forall t \geq T_q$, if q sends a message m to p, then q receives m by time $t+\delta$. Then, when q sends a message to p at $t \geq T_q$, and p receives the message at $t_1 > t$, two cases may happen:

- the next timer of q expires after t_1 (Task T1). In this case, q will be added to $trusted$ (line 5) and the timer of q restarted;
- the current timer of q expires before t_1: p removes q from $trusted$ (line 7). Then, the timeout value of q is incremented (line 9) and the timer is restarted.

Since q keeps on sending $ALIVE$ messages to p and $timer[q]$ increases at every expiration of q's timer, there exists a time $t_2 > T_q$ such that $timer[q] \geq \delta$ and then Task 2 will nevermore be executed by p for q and, $\forall t_3 \geq t_2$, upon every q's message reception by p, task T1 will be executed for q. Therefore, q will remain forever in $trusted$. □

Theorem 7. *If S is a* S-ET *system and p is correct, then, for Algorithms 9 and 10, there is a time after which S is either always $trusted$ or always $untrusted$ for p.*

Proof. Since in *S-ET*, all links are $\diamondsuit-timely$, from Theorem 6 and Lemma 10, Algorithms 9 and 10 ensure both the *Impact* $completeness_p^S$ and the *Eventual impact weak* $accuracy_p^S$ properties. Thus, $\exists t, \forall t' \geq t, \forall q \in correct(F_S), q \in trusted$ and, $\forall q \in faulty(F_S), q \notin trusted$. Hence, $\forall t' \geq t$, $trusted$ never changes as well as the trust level value rendered by the FD. Consequently, if at t, the trust level output $\geq threshold_p^{S^*}$ (resp., trust level output $< threshold_p^{S^*}$), S is $trusted$ (resp., $untrusted$) for p at t, and it will remain forever $trusted$ (resp., $untrusted$) for p.

□

Theorem 8. *In* W-ET-Π *systems, Algorithms 9 and 10 implement a FD of class* $\diamondsuit IS$.

Proof. If the system is *W-ET*-Π, $S = \Pi$, all processes of Π execute Algorithms 9 and 10 and $\exists p \in correct(F)$ such that $\forall q \in \Pi, q \neq p$, p is linked to q by a $\diamondsuit-timely$ link. Thus, $\forall q \in correct(F)$, Lemma 10 holds for q and *Eventual impact weak* ${accuracy_q}^{\Pi}$ is satisfied. From Theorem 6, $\forall q \in correct(F)$, *Impact* $completenness_q^{\Pi}$ is also satisfied. Therefore, the algorithms implement a FD of class $\diamondsuit IS$.

□

Theorem 9. *In* S-ET-Π *systems, Algorithms 9 and 10 implement a FD of class* $\diamondsuit IP$.

Proof. If the system is *S-ET*-Π, $S = \Pi$ and all processes of Π execute Algorithm 9 and 10. Hence, since the system is a *S-ET*, from Theorem 6 and Lemma 10, $\forall p \in correct(F)$, both ${Impact\ completeness_p}^{\Pi}$ and ${Eventual\ impact\ weak\ accuracy_p}^{\Pi}$ are satisfied respectively. Therefore, the theorem holds.

□

Theorem 10. *If* $PR(IT)_p^S$ *(resp.,* $PR(\diamondsuit IT)_p^S$*) holds, the system S is always (resp., eventually always) trusted by p.*

Proof. if $PR(IT)_p^S$ (resp., $PR(\diamondsuit IT)_p^S$) holds, $\forall t \geq 0$ (resp., $\exists t_1, \forall t \geq t_1$), $trusted \in PS$ and, therefore, S is trusted by p. □

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 in different systems and configurations of node sets with regard to both the impact factor and the threshold.

Our goal is to evaluate the QoS of the Impact FD: how fast it detects failures and how well it avoids false suspicions. With this purpose, we exploit a set of metrics that have been proposed by [12] and we compare the results of Impact FD with an approach that monitors processes individually using Chen's FD [12]. We conducted a set of experiments, considering two different systems: 1)*AS*: a system where all links are lossy asynchronous; (b) *W-ET*: a system where some links are $\diamondsuit-timely$ and the others are lossy asynchronous.

7.1 Environment

Our experiments are based on real trace files, collected from ten nodes of PlanetLab [33], as summarized in Table 3. The PlanetLab 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, such traces can characterize any distributed system that uses FDs based on heartbeat. Furthermore, since our experiments were conducted using the PlanetLab traces, all of them reproduce exactly the same scenarios of sending and reception of heartbeats by the processes.

Table 3: Sites of Experiments

ID	Site	Local
0	planetlab1.jhu.edu	USA East Coast
1	ple4.ipv6.lip6.fr	France
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

For the evaluation of Impact FD, we defined $S = \{1,2,3,4,5,6,7,8,9\}$ and site *0* as the monitor node ($p \notin S$).

Table 4 gives some information about the heartbeat messages received by site *0* (the monitor node). We observe that the mean inter-arrival times of received heartbeats is very close to 100 ms. However, for some sites, the standard deviation is very high, like for site *5* which the standard deviation was 310.958 ms with a minimum inter-arrival time of 0.006 ms, and a maximum of 657,900.226 ms. Such deviation probably indicates 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 1,759,990 received messages.

The implementation of the Impact FD used in our evaluation experiments is based on Algorithms 9 and 10, presented in Section 6. For the estimation of the timeout value of Chen's estimation algorithm, 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.

7.1.1 Evaluation of sites' stability

We evaluated the stability of sites, considering that the traces could correspond to either an *AS* system or *W-ET* system. For the first case, the value *AS* was assigned to the $model$ parameter of Algorithm 9 while for the second case, the same parameter was set to *W-ET*. Each of the sites of S is considered individually and not as a whole system. The impact value of sites and the threshold values are not concerned for the experiments.

The β value of Chen's algorithm was set to 400ms. We chose such a value because it is an acceptable safety margin for detection time and is not too aggressive; otherwise the failure detector would be prone to too many mistakes. The stability of sites and the corresponding links to the monitor were evaluated during the whole trace period for the *AS* system and during just the first 24 hours of the trace period for the *W-ET* system.

***AS* system**:

Figure 2 shows the cumulative number of mistakes, i.e., false suspicions, made by the monitor site *0* for each site of S. We can observe that site or link periods of instability entail late arrivals or loss of heartbeats and, therefore, mistakes by the monitor site. For example, site *9* had a large number of

Table 4: Sites and heartbeat sampling

Site	Messages	Min (ms)	Max (ms)	Mean (ms)	Stand. Dev.(ms)
1	5,424,326	0.025	26,494.168	100.058	19.525
2	1,759,989	0.031	509.093	100.415	9.275
3	5,426,843	0.027	1,227.349	100.012	1.709
4	5,414,122	0.003	1,193.276	100.247	18.595
5	5,413,542	0.006	657,900.226	100.258	310.958
6	5,426,700	0.003	3,787.643	100.015	2.557
7	5,424,117	0.006	59,603.188	100.062	31.229
8	5,424,560	0.027	11,443.359	100.054	100.714
9	5,422,043	0.004	30,600.076	100.100	18.798

cumulative mistakes at hour 48. After that, there is a stable period with regard to this site. On the other hand, around this time, site *2* stopped sending messages since 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.

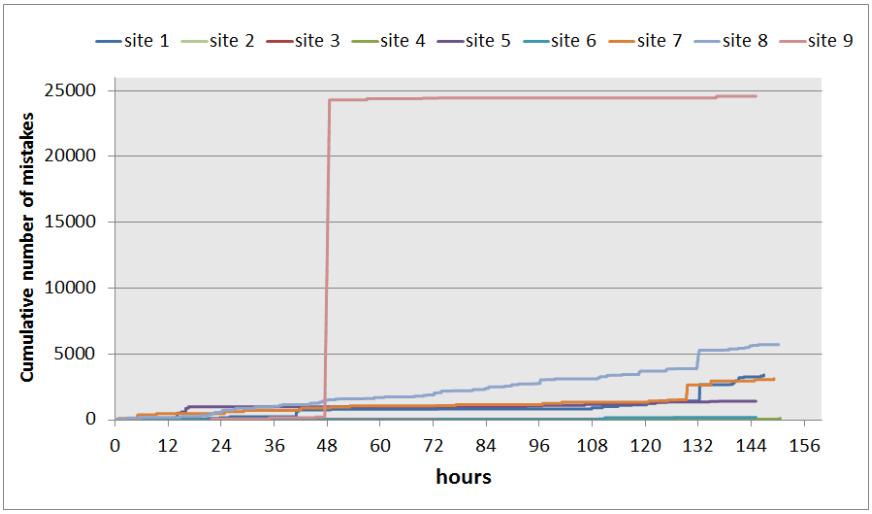


Figure 2: *AS* System: Cumulative number of mistakes of each site

***W-ET* system**: In Algorithm 8 (Task T2), when the system is *W-ET*, Chen's heartbeat arrival estimation value is incremented by η, whenever a false suspicion occurs. However, in order to prevent this estimation from increasing too fast when there is a period of high instability, which could increase the detection time considerably, we considered that the value of the timer (line 9) will be incremented by η at every μ heartbeat arrivals, provided that during the period of these μ heartbeat arrivals, one or more false suspicions took place. For the experiment, we considered μ equals to 10 and $\eta = 1ms$.

Note that when the heartbeat arrival estimation reaches a value which is greater than the transmission delay limit for links with $\diamond-timely$ behavior, the monitor site does not make anymore mistakes for the related sites. Moreover, for unstable sites, as the heartbeat arrival estimation value will also be incremented by η in case of false suspicions, such an increment will be responsible for decreasing the number of mistakes for these sites when compared to an *AS* system. However, in this case, at the expense of higher false suspicion detection time.

Figure 3 shows the cumulative number of mistakes that the monitor process made for each site in the first 24 hours of the traces. We can observe that there are links which behave $\diamondsuit-timely$ while the others are *lossy asynchronous*. The failure detector did not make mistakes related to site 4. For sites 2 and 3, it did only 1 and 2 mistakes, respectively, while for site 6, it did 99 mistakes during the first hour, and then no more mistakes. Although some sites have had some periods of stability (1, 5, 8 and 9), site 0 made mistakes related to them until almost the end of these execution. On the other hand, it did no mistakes for site 7 after hour 9. In summary, we can consider that site 0, the monitor site, is connected by $\diamondsuit-timely$ links to sites 2, 3, 4 and 6, and by *lossy asynchronous* links to 1, 5, 7, 8, and 9.

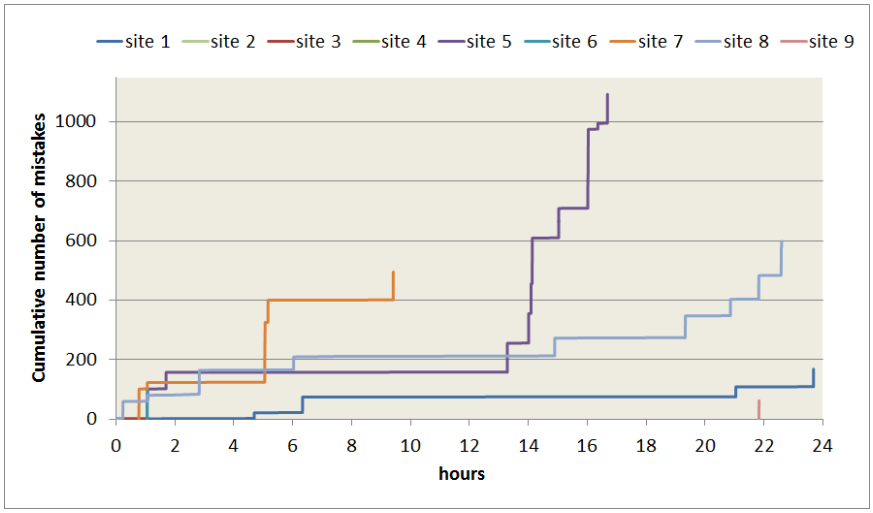


Figure 3: *W-ET* System: Cumulative number of mistakes of each site

7.1.2 Evaluation of heartbeat arrival times

The goal of this section is to show the behavior of the arrival times when the timer expires and the failure detector does not receive the heartbeat message. For the first 24 hours, we evaluated the behavior of the three arrival times in $site\ 0$ related to heartbeat messages of $site\ 1$ with two different values to β (100 and 400 ms). We chose $site\ 1$ because it has many periods of instability. We consider that $site\ 1$ and $site\ 0$ are either connected by *lossy asynchronous* or $\diamondsuit-timely$.

We evaluated three arrival times: 1) arrival of the heartbeat; 2) the estimated arrival time considering that the link is *lossy asynchronous*; 3) the estimated arrival time considering that the link is $\diamondsuit-timely$. In order to compute the latter, we set $\eta = 1ms$ and the number of heartbeats before incrementing the heartbeat arrival estimation value, in case of false suspicions, to 100 ($\mu = 100$). Figures 4 and 5 show the time difference between the arrival time of the previous heartbeat and the above three arrival ones (in ms): 1) milliseconds difference between the arrival time of the last heartbeat and the previous one; 2) milliseconds difference between the estimated arrival time ($\tau_q = \beta + EA_q$) and the arrival time of the previous heartbeat, considering the link lossy asynchronous (estimation LA); 3) milliseconds between the estimated arrival time ($\tau_q = \beta + EA_q + \eta$) and the arrival time of the previous heartbeat, considering the link $\diamondsuit-timely$ (estimation ET).

Figures 4 and 5 show the behavior of times when the timeout expires for $\beta = 100ms$ and $\beta = 400ms$ respectively till hour 24. Aiming at not overloading the figures, the points correspond only to the times where mistakes took place. Figure 5 has fewer points than Figure 4 because the number of mistakes drops considerably due to a higher β value.

Figure 4 summarizes the time differences for $\beta = 100ms$. The monitor $site$ 0 made 807 (resp., 592) mistakes when the link is *lossy asynchronous* (resp. $\Diamond-timely$). Note that at several points, the estimated arrival time for the *ET* estimation is higher than the arrival time of the heartbeat while, in the *LA* estimation, the difference between them is very small (1 or 2 ms), specially from time 6 to 21. Thus, both lines in the figure overlap but the estimation arrival time is often below the arrival one which explains the high number of mistakes. At times 1, 4, 6, 21, and 23, which correspond to periods of instability, the arrival time of the heartbeat is much higher than the estimation one for the *LA* estimation.

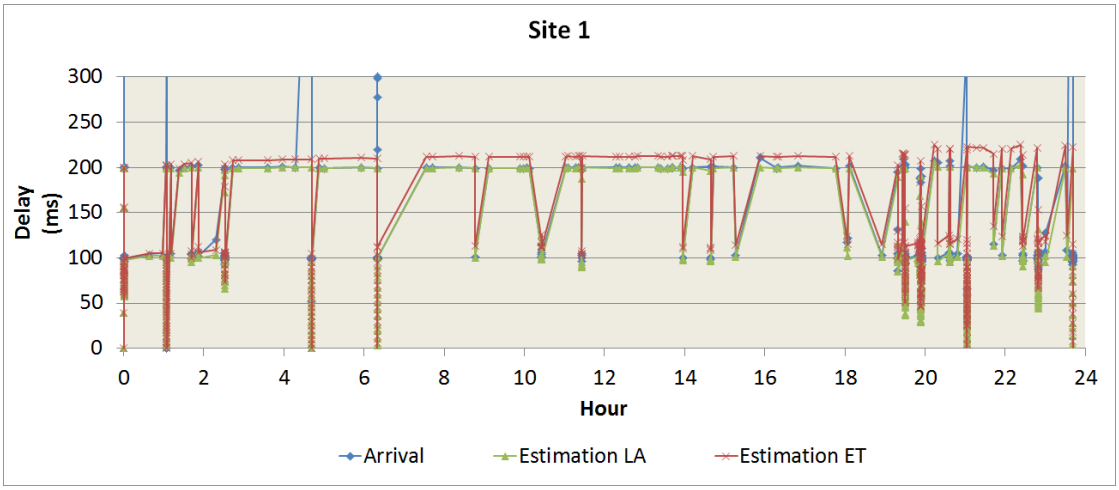


Figure 4: Behavior of times when timeout expires - $\beta = 100ms$, $\mu = 100$

Contrarily to Figure 4, the number of mistakes drops to 168 and 166 mistakes, for *ET* and *LA* estimations respectively as shown in Figure 5. Therefore, since they are almost equal, the estimated arrival times for the *lossy asynchronous* and $\Diamond-timely$ are also quite close. Similarly to Figure 4, the mistakes are concentrated in periods of great instability (1, 4, 6, 21, and 23).

7.2 QoS Metrics

Firstly, let's remember that the goal of the Impact FD is to inform if a system is "trusted" or "untrusted". This information can be deduced by comparing the output $trust\_level$ of the Impact FD with the $threshold$. Thus, we say that the output of the Impact FD of p is **correct** if either, for each subset of S^* ($1 \leq i \leq m$), $trust\_level_i >= threshold_i$ and S is actually trusted, or $\exists\ i$ such that $trust\_level_i < threshold_i$ and S is actually untrusted. Otherwise, the FD made a mistake.

For evaluating the Impact FD, we used three of the QoS metrics proposed in [12]: *detection time*, *average mistake rate*, and *query accuracy probability*. Considering that p monitors S, the QoS of the Impact FD at p must take into account the transitions between "trusted" to " untrusted" states of S.

- *Detection Time* (T_D): In [12], the T_D is defined as the time elapsed from the moment process q crashes until the FD at p starts suspecting q permanently.

  In the case of the Impact FD, the detection time (T_D) of p in relation to S is the time elapsed till the monitor process reports a suspicion that leads to a status transition in S from *trusted* to *untrusted*. To this end, for each freshness point of a process q in S, it is necessary to check which

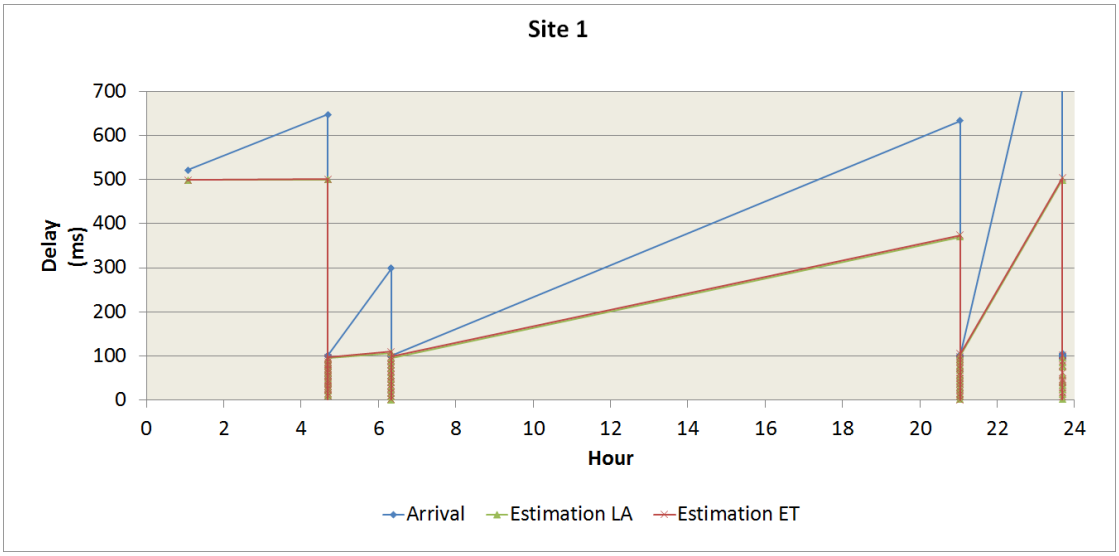


Figure 5: Behavior of times when timeout expires - $\beta = 400ms$, $\mu = 100$

process failures would lead to a state transition of S from *trusted* to *untrusted* and then compute the detection time T_D for each of these processes. The latter is the time elapsed between the current freshness (τ_{i+1}) and the last heartbeat arrival (A_i) with respect to the previous freshness point, i.e., $\tau_{i+1} - A_i$, from each of these processes. If there is more than one process $q \in S$ which could lead to the transition, i.e., $S_f = q \in trusted_i | (trust\_level_i - Impact(q)) < threshold_i$, the T_D in relation to S is the greatest of them: $T_D = max(\tau_{i+1} - A_i),\ \forall q \in S_f$.

Figure 6 shows an example where S^* has just one subset with three processes whose impact factor is 1. The $threshold^S$ defines that at least two processes must be correct. Note that at τ_{i+3}, process p did not receive the heartbeat message from q_1 and, therefore, p removes it from its trusted set ($trusted_p = \{\{\langle q_2, 1\rangle, \langle q_3, 1\rangle\}\}$). However, S remains trusted for p because the trust level satisfies the threshold. At freshness point τ_{i+5}, FD verifies if the failure of any of the processes of $trusted_p$ (q_2 and q_3) can lead to S transition ($trust\_level_1 < threshold_1$). For this purpose, p computes the T_D for each of the two processes. The T_D in relation to S is the greatest among T_D of q_2 and T_D of q_3. Since p did not receive heartbeat from q_3, S becomes *untrusted*.

- *Average Mistake Rate* (λ_R): represents the number of mistakes that the FD makes per unit of time, i.e., the rate at each the FD makes mistakes.

- *Query Accuracy Probability* (P_A): the probability that the FD output is correct at a random time.

7.3 Asynchronous System (AS)

For this evaluation we consider an *AS*, i.e., links are lossy asynchronous. Table 5 shows five configurations with regard to impact 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.

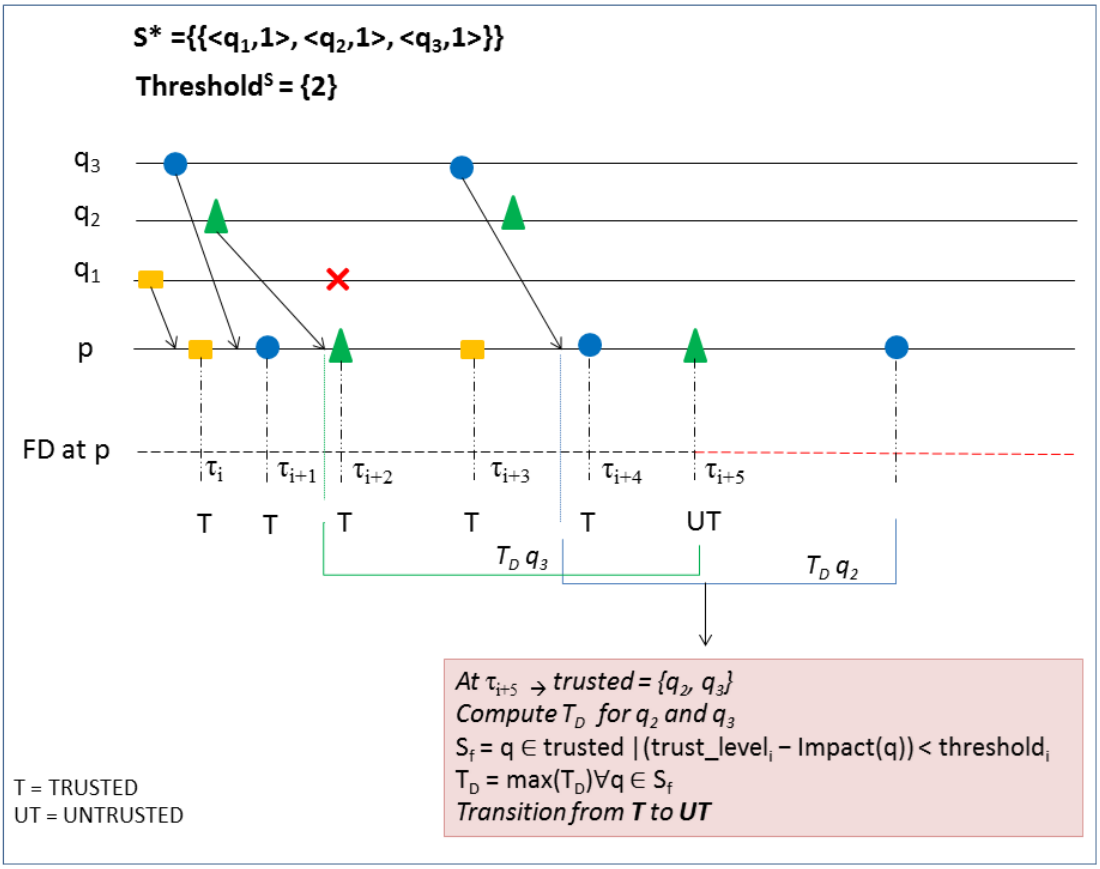


Figure 6: Transitions between "trusted" and "untrusted" states

Table 5: Set Configurations (S^*)

Config	Impact Factor of each site
S^* 0	$\{\{\langle q_1,7\rangle,\langle q_2,3\rangle,\langle q_3,20\rangle,\langle q_4,20\rangle,\langle q_5,3\rangle,\langle q_6,20\rangle,\langle q_7,3\rangle,\langle q_8,7\rangle,\langle q_9,7\rangle\}\}$
S^* 1	$\{\{\langle q_1,7\rangle,\langle q_2,20\rangle,\langle q_3,20\rangle,\langle q_4,3\rangle,\langle q_5,3\rangle,\langle q_6,20\rangle,\langle q_7,3\rangle,\langle q_8,7\rangle,\langle q_9,7\rangle\}\}$
S^* 2	$\{\{\langle q_1,20\rangle,\langle q_2,7\rangle,\langle q_3,3\rangle,\langle q_4,3\rangle,\langle q_5,7\rangle,\langle q_6,3\rangle,\langle q_7,7\rangle,\langle q_8,20\rangle,\langle q_9,20\rangle\}\}$
S^* 3	$\{\{\langle q_1,7\rangle,\langle q_2,3\rangle,\langle q_3,20\rangle,\langle q_4,3\rangle,\langle q_5,3\rangle,\langle q_6,20\rangle,\langle q_7,7\rangle,\langle q_8,20\rangle,\langle q_9,7\rangle\}\}$
S^* 4	$\{\{\langle q_1,10\rangle,\langle q_2,10\rangle,\langle q_3,10\rangle,\langle q_4,10\rangle,\langle q_5,10\rangle,\langle q_6,10\rangle,\langle q_7,10\rangle,\langle q_8,10\rangle,\langle q_9,10\rangle\}\}$

7.3.1 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 5). The safety margin was set to 400ms (β=400ms).

Figure 7 shows that in most cases 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 lower, the Query Accuracy Probability is higher.

On the one hand, except for threshold 83, *"S* 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, *"S* 2"* and *"S* 4"* have the lowest P_A since unstable sites have high impact factor values assignment. For instance, in *"S* 2"* the high impact factor value of unstable sites *8* and *9* with standard deviation of 100 and 18 ms respectively degrades the P_A of this set.

"S 4"* shows a sharp decline of the P_A curve 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 Query Accuracy Probability 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 *"S* 1"*, where the impact factor of site *2* is 20 (high), the P_A 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. For the latter, the metric is the average of the P_A value of all sites of S: $\overline{PA} = \frac{\sum_{x=1}^{n} PA_x}{n}$, for $n = 9$ and x equals to the index of each site in S. Thus, the obtained mean P_A ($\overline{PA}$) is equal to 0,979788. This result shows that, regardless of the set (S^*) configuration, the Impact FD has a higher P_A than Chen's 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.

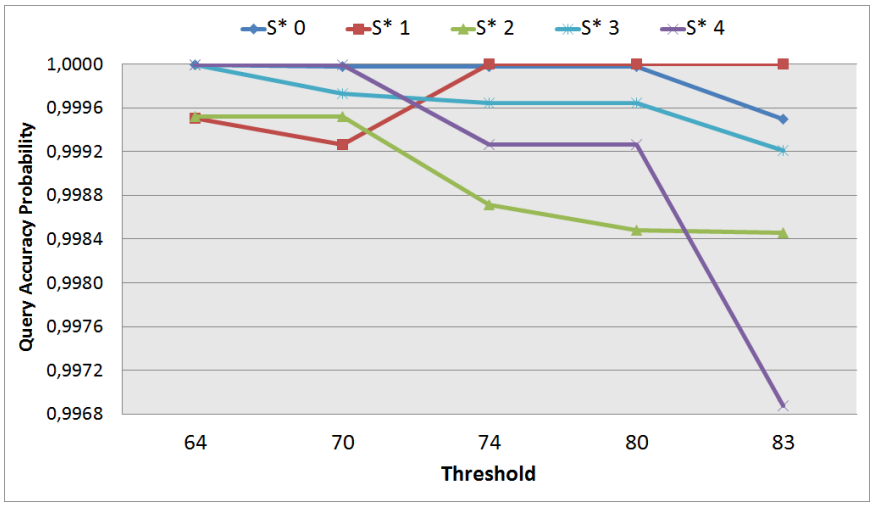


Figure 7: *AS* System: P_A vs. threshold with different set configurations (S^*)

7.3.2 Experiment 2 - Query Accuracy Probability X Detection time

In the second experiment, we evaluated the average Query Accuracy Probability (P_A) regarding the average detection time (T_D) for different threshold values (64, 70, 80, and 83). 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 *"S* 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. For the latter, the T_D is computed as the average of the individual T_D of all sites of S: $\overline{TD} = \frac{\sum_{x=1}^{n} TD_x}{n}$, for $n = 9$ and x equals to the index of each site in S.

Figure 8 shows that for a high threshold and detection time close to 200 ms, the P_A of the Impact FD is quite small, 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 false suspicions and mistake duration. However, when T_D is greater than 230 ms, the P_A of 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 [35].

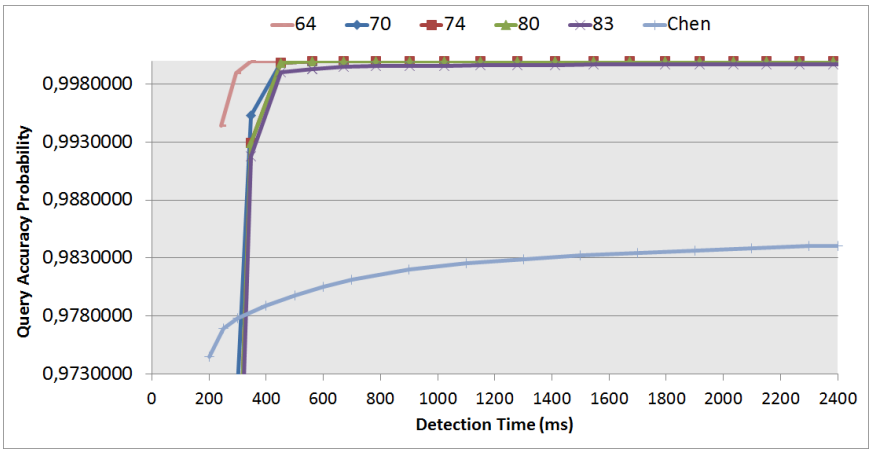


Figure 8: *AS* System: P_A vs. T_D with different thresholds

7.3.3 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). For Chen's algorithm, the λ_R is computed as the average of the individual λ_R of all sites of S: $\overline{\lambda_R} = \frac{\sum_{x=1}^{n} \lambda_R}{n}$, for $n = 9$ and x equals to the index of each site in S. We considered the *"S* 0"* configuration and the mistake rate is expressed in a logarithmic scale.

We can observe in Figure 9 that the mistake rate of the Impact FD is high when the detection time is low (i.e., smaller than 400 ms) and the threshold is high (i.e., 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 low or the time detection increases.

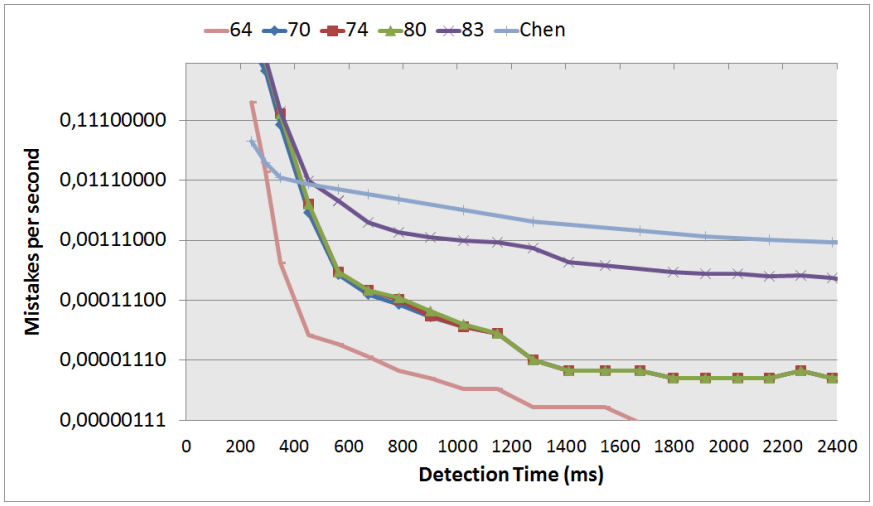


Figure 9: *AS* System: λ_R vs. T_D with different thresholds

7.3.4 Experiment 4 - Cumulative number of mistakes

Figure 10 shows the cumulative number of mistakes for *"S* 0"* during the whole trace period, considering β=400ms and threshold value equals either to 80 or 83.

We can observe in the figure that the cumulative number of mistakes is greater when the threshold value is equal to 83 (2754 mistakes) when compared to the threshold value equals to 80 (179 mistakes). The former makes few mistakes until approximately the hour 48 (when the site *2* crashed). After that, the number of cumulative mistakes significantly increases because, since the threshold is high (83) and the failure of site *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, there are fewer instability periods since the crash of site *2* does not have much impact in the confidence of the system. At hour 48, there is an increase in the cumulative number of mistakes due to the unstable period of site *9*, as shown in Figure 2. From hour 50 to 100, the FD makes fewer mistakes. Such a behavior can be explained since, as observed in the same figure, all sites, with exception of site *8*, also have this same period of stability. After hour 108, there is a greater number of mistakes which is related to the instability of sites *1*, *7*, and *8* (see Figure 2).

7.3.5 Experiment 5 - Query Accuracy Probability vs. Time

In this experiment, we divided the execution trace duration by fixed intervals of time and computed the average Query Accuracy Probability (P_A) for each of them. We chose the *"S* 0"* configuration, β=400ms, and the threshold values of 80 and 83. Similarly to the cumulative number of mistakes (Experiment 4), we observe in Figure 11 that instability periods have an impact in 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 *1*, *7*, and *8*.

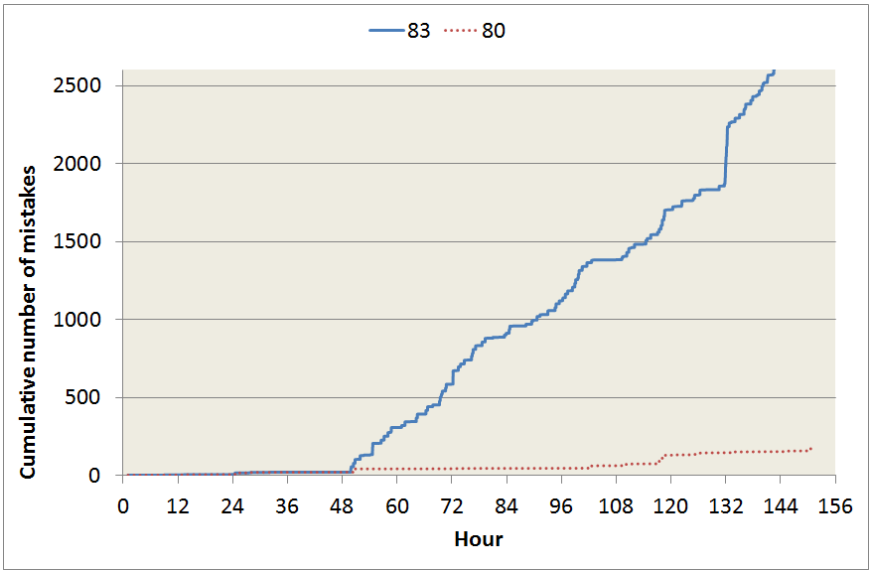


Figure 10: *AS* System: Cumulative number of mistakes for *"S* 0"* configuration

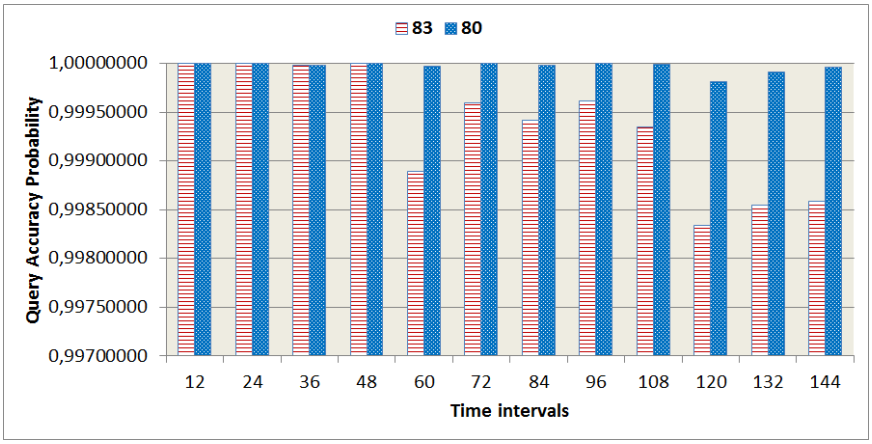


Figure 11: *AS* System: P_A vs. Time

7.4 Weak $\Diamond-timely$ System (W-ET)

In this section, we consider the *W-ET* system described in Section 7.1.1: site *0*, the monitor site, is connected by $\Diamond-timely$ links to sites 2, 3, 4 and 6 and by *lossy asynchronous* links to 1, 5, 7, 8, and 9.

We defined the set S^* with three subsets and all sites have the same impact factor (1) :

$$S^* = \{\{\langle q_1,1\rangle,\langle q_3,1\rangle,\langle q_4,1\rangle\},\{\langle q_2,1\rangle,\langle q_5,1\rangle,\langle q_6,1\rangle\},\{\langle q_7,1\rangle,\langle q_8,1\rangle,\langle q_9,1\rangle\}\}$$

The $threshold^S$ was defined as follows:

$$threshold^S = \{2,2,2\}$$

The $threshold^S$ defines that the subsets S_1, S_2 and S_3 must have at least two correct processes. As this experiment assigns *W-ET* to $model$ parameter, it uses the η value and the heartbeat arrival

Table 6: *W-ET* vs *AS* - $\beta = 50ms, \eta = 500m\mu$

μ	**Mistakes**	**Mistake rate**	$\mathbf{P}_A$	**Avg Mistake Duration (ms)**	**Time last mistake (min)**	**HB Number**	$\mathbf{T}_D$ (ms)
1	152	0.0017	0.99992	43.36	64 (1h)	349341	312.9
10	324	0.0037	0.99983	43.69	64 (1h)	349341	263.0
100	383	0.0044	0.99979	45.18	64 (1h)	349341	256.6
AS	4689	0.0542	0.99849	27.70	1438 (24h)	7749909	300.0

estimation value is incremented by η at every μ heartbeat arrivals, if false suspicions occurred during this period.

The experiments were carried out just for the first 24 hours of the traces, because after this time the failure detector does not make more mistakes for the set S^*.

7.4.1 Experiment 6 - Eventually Timely Links vs Asynchronous Links

In this experiment, we compare the results obtained taking into account the above S^* configuration and both systems *W-ET* and *AS*. The evaluation metrics are shown in Table 6. We set the value of safety margin β to $50ms$ and η to $50m\mu$. This safety margin value is quite aggressive, which, consequently, leads the failure detector prone to make mistakes. For the *W-ET* system, we also varied μ: 1, 10, and 100.

The first three rows of the table show the results for the *W-ET* system and the last row for the *AS* system. We can observe that the number of mistakes increases for different values of μ in the *W-ET*, but it is much smaller when compared to the *AS* (4689 mistakes). As a consequence, in the *AS*, the $mistake\ rate$ is higher and P_A is lower. In contrast, the average mistake duration in the *AS* (27.70 ms) is smaller than in the *W-ET* (around 43 ms). Such a difference occurs because the *AS* system has a lower timeout which induces false suspicions more often. Nevertheless, a heartbeat message may arrive immediately after the expiration of the timeout, generating a short mistake time. On the other hand, in the *W-ET*, the timeout value increases when there are false suspicions in periods of greater instability where messages take longer to arrive. For the *W-ET* system, we can observe that the time of the last mistake was at 64 minutes (heartbeat number 349,341) whereas in the *AS* there are mistake occurrence until the last hour (24h, heartbeat number 7749909). This happens because in the *W-ET* the heartbeat arrival estimation value is incremented by η when p falsely suspecting the process within a period of μ heartbeats, which allows p to eventually get every heartbeat message from a site before the timeout expires. It is worth remarking that the number of mistakes reduces drastically, but the T_D does not increase in the same rate.

Table 7 summarizes the results of the experiments considering $\beta = 100ms$ and $\eta = 500m\mu$. When comparing the two tables, we observe that with a less aggressive safety margin β, the number of mistakes reduces, especially in the *AS* system (231). Accordingly, the $mistake\ rate$ decreases and P_A increases in both systems. The last mistake is around 64 minutes in the *W-ET* while *AS* made mistakes until hour 24. The T_D of the *AS* reduces because it has a higher safety margin and makes fewer mistakes. For instance, with $\beta = 50ms$, two processes, whose maximum T_D is 300ms, that has the timeout expired, leads the set S^* to a state *untrusted*. However, with $\beta = 100$ only one of them is suspected which does not lead a transition of state from *trusted* to *untrusted*.

We also conducted the same experiment with $\beta = 100ms$ and $\eta = 1ms$ for the *W-ET* system (Table 8). We can note that the number of mistakes is reduced. On the other hand, with few mistakes, especially with $\mu = 1$, both the average mistake duration and T_D increase. Based on these results, we can conclude that setting μ with a value greater than 1 is more suitable for this scenario, achieving, therefore, a better trade-off between detection time and accuracy of the Impact FD.

Table 7: *W-ET* vs *AS* - $\beta = 100ms, \eta = 500m\mu$

μ	**Mistakes**	**Mistake rate**	$\mathbf{P}_A$	**Avg Mistake Duration (ms)**	**Time last mistake (min)**	**HB Number**	$\mathbf{T}_D$ (ms)
1	84	0.00097	0.99995	48.35	64 (1h)	349341	339.4
10	121	0.00140	0.99993	48.28	64 (1h)	349341	264.0
100	135	0.00156	0.99993	44.53	64 (1h)	349341	262.5
AS	231	0.00267	0.99989	37,56	1431 (24h)	7708057	240.0

Table 8: *W-ET* vs *AS* - $\beta = 100ms, \eta = 1ms$

μ	**Mistakes**	**Mistake rate**	$\mathbf{P}_A$	**Avg Mistake Duration (ms)**	**Time last mistake (min)**	**HB Number**	$\mathbf{T}_D$ (ms)
1	6	0.000069	0.999990	140.00	64 (1h)	349339	910.0
10	45	0.000520	0.999972	53.07	64 (1h)	349341	460.0
100	98	0.001133	0.999945	47.99	64 (1h)	349341	291.0
AS	231	0.002672	0.999899	37.56	1431 (24h)	7708057	240.0

8 Related Work

We can divide related studies of the literature into three groups: (1) unreliable failure detectors, (2) heartbeat arrival estimation strategies, and (3) reducibility of failure detectors.

Unreliable failure detectors: Most of the unreliable failure detectors in the literature are based on a binary model and provide as output a set of process identifiers, which usually informs the set of processes currently suspected of having failed ([11] [6]). However, in some detectors, such as class Σ (resp., Ω) (see Section 3.1) [17], the output is the set of processes (resp., one process) which are (resp., is) not suspected of being faulty, i.e., $trusted$.

The ϕ Accrual failure detector [23] proposes an approach where the output is a suspicion level on a continuous scale, rather than providing information of a binary nature (trusted or suspected). 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 tends toward infinity. The aim of Accrual failure detectors is to decouple monitoring from interpretation.

Starting from the premise that applications should have information about failures to take specific and suitable recovery actions, the work in [30] 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 [9] 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.

In [36], the authors propose the use of a reputation mechanism to implement a failure detector for large and dynamic networks. The reputation mechanism allows node cooperation through the sharing of views about other nodes. The proposed approach exploits information about the behavior of nodes to increase its quality in terms of detection. When classifying the behavior of the nodes, the FD includes a reputation service where the nodes periodically exchange heartbeat messages.

Heartbeat arrival estimation strategies: In the timer-based FD algorithms presented in section 6, we

used the heartbeat arrival estimation proposed by Chen et. al. [12]. With the same aim of Chen's algorithm, i.e., minimize false suspicions and failure detection time, several other estimation approach have been proposed in the literature. They dynamically predict new heartbeat arrivals based on observed communication delays of the past heartbeat history.

In [6], Bertier et. al introduced a failure detector that was mainly intended for LAN environments. Their heartbeat arrival estimation approach combines of Chen's estimation with a dynamic estimation based on Jacobson's estimation [25]. The latter is used in the protocol TCP to estimate the delay after which a node retransmits its last message. Basically, the estimation of the next heartbeat arrival is calculated by adding Chen's estimation to a safety margin given by Jacobson's algorithm. Their approach provides a shorter detection time, but generates more false suspicions than Chen's estimation, according to the authors' measurements on a LAN.

The ϕ Accrual failure detector is based [23] on inter-arrival estimation time, assuming that the latter follow a normal distribution. The Accrual FD dynamically adapts current network conditions based on the suspicion level. Similarly to the above FD [6] and [12], the estimation protocol samples the arrival time of heartbeats and maintains a sliding window of the most recent samples. The distribution of past samples is then used as an approximation for the probabilistic distribution of future heartbeat messages. With this information, it is possible to compute a value φ with a scale that changes dynamically to match recent network conditions

In [35], the authors extended the Accrual FD by exploiting histogram density estimation. Taking into account a sampled inter-arrival time and the time of the last received heartbeat, the algorithm estimates the probability that no further heartbeat messages arrive from a given process, i.e., it has failed.

In [14] is presented the failure detector ANNFD, based on Artificial Neural Networks. The detector uses as input parameters variables collected by the Simple Network Management Protocol (SMNP) that characterize the network traffic at each time instant. After training the neural network, it must compute the message arrival time estimation EA_{k+1}, which is utilized to define the freshness point.

The mechanism proposed in [15] follows an approach based on the feedback control theory which is able to dynamically configure the monitoring period and detection timeout following the observe changes in the computing environment and according to user-defined QoS constraints.

Reducibility of failure detectors: As we have discussed in Section 3.2, failure detectors can be compared with each other through the notions of reducibility and equivalence. In [10], Chandra and Toueg defined several failure detector classes which are sufficient to solve Consensus and showed that some pairs are equivalent while others are distinct. The authors proved that the weakest failure detector needed to solve Consensus is $\diamondsuit W$. Furthermore, they introduced the Ω class as an intermediate step in their proof, and showed that $\diamondsuit W$ and Ω are equivalent. Thus, any failure detector of one of these classes can be transformed into a failure detector of the other class.

In [13], Chu presents two transformations from $\diamondsuit W$ to Ω. The first one requires each message to carry an array of counters, some of them growing indefinitely. In the second one, each message keeps a sequence number plus a set of processes identities. Since the sequence number stops increasing, this transformation is quiescent, i.e., the processes eventually stop sending message in any run.

The authors in [32] present a communication efficient transformation from $\diamondsuit W$ to Ω for asynchronous message-passing systems equipped with a reliable broadcast communication primitive. The transformation is also quiescent and, contrarily to [13], requires each message to carry only one process identity.

The most important results presented in those works is the fact that the classes Ω and $\diamondsuit W$ are equivalent, provided that the membership of the system is known [26].

9 Conclusion and Future Work

This technical report introduced the *Impact* failure detector that provides an output that expresses the trust of the FD with regard to the system (or set of processes) as a whole. It is configured by the *impact factor* and the *threshold* which enable the user to define the importance (e.g., degree of reliability) of each node and an acceptable margin of failures respectively. It is thus suitable for environments where there is node redundancy or nodes with different capabilities. Both the *impact factor* and the *threshold* render the estimation of the confidence in the system (or a set of processes S) more flexible. In some scenarios, the failure of low impact or redundant nodes does not jeopardize the confidence in S, while the crash of a high impact factor one may seriously affect it. Either a softer or a stricter monitoring is, therefore, possible.

We have defined two properties, $PR(IT)^S_p$ and $PR(\diamondsuit IT)^S_p$, which denote the capacity of the Impact FD of accepting different set of trusted processes that lead to the confidence in S. Then, we presented a timer-based implementation of the Impact FD, which can be applied to systems whose links are *lossy asynchronous* or those whose all (or some) are $\diamondsuit - timely$. Performance evaluation results, based on real PlanetLab traces, showed 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. Furthermore, we observed that the Impact FD might weaken the rate of false suspicions when compared with the traditional Chen's unreliable failure detector. Additionally, in the experiments carried out considering a W-ET system, it was observed that the number of mistakes reduce drastically when compared with the AS system, however the *detection time* does not increase in the same rate. Therefore, such results confirm the degree of flexible applicability of the Impact FD, that both failures and false suspicions are more tolerated than in traditional FDs, and that the former presents better Qos than the latter if the application is interested in the degree of confidence in the system (trust level) as a whole.

The technical report also shows that the Impact FD of class $I\Omega$ (resp., $I\Sigma$) is equivalent to Ω (resp., Σ) FD. Both are two important weakest FD classes to circumvent the impossibility of consensus in asynchronous distributed systems in presence of failures. In addition, it shows that, if there exist a majority of correct processes, Σ is reducible to $\diamondsuit IP^U$ and $\diamondsuit IW^U$ FD is equivalent to Omega FD (Ω), provided that membership is known.

In the near future, we intend to generalize the trust level calculation as well as its comparison with the threshold. To this end, the $Trust\_level(trusted, S^*)$ function can perform an operation over the impact factor of the trusted processes other than the sum (e.g., multiplication, average, etc.) and the threshold will not necessary be a lower bound (e.g., upper bound, equality, etc.). For instance, suppose that the impact factor of a node corresponds to the probability that it behaves maliciously. The trust level, in this case, would express the probability that all nodes of the system behave maliciously. Thus, the $trust\_level$ sum operation would be replaced by multiplication operation and should be smaller than a reliability threshold value.

Another research direction is to render the impact factor dynamic, i.e., the impact factor of a node can vary during execution, depending on the current degree of reliability of the node or its current reputation, its past history of stable/unstable periods, etc. Finally, we also aim at extending performance experiments to other networks such as MANET or LAN, comparing the performance of Impact FD with other well-known failure detectors.

10 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] Marcos K. Aguilera, Carole Delporte-Gallet, Hugues Fauconnier, and Sam Toueg. On implementing omega with weak reliability and synchrony assumptions. In *Proceedings of the Twenty-second Annual Symposium on Principles of Distributed Computing*, pages 306–314, 2003.

[3] Marcos K Aguilera, Carole Delporte-Gallet, Hugues Fauconnier, and Sam Toueg. Communication-efficient leader election and consensus with limited link synchrony. In *Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing*, pages 328–337. ACM, 2004.

[4] Luciana Arantes, Fabíola Greve, Pierre Sens, and Véronique Simon. Eventual leader election in evolving mobile networks. In *Principles of Distributed Systems - 17th International Conference, OPODIS 2013, Nice, France, December 16-18, 2013. Proceedings*, pages 23–37, 2013.

[5] 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, 2012.

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

[7] François Bonnet and Michel Raynal. On the road to the weakest failure detector for k-set agreement in message-passing systems. *Theor. Comput. Sci.*, 412(33):4273–4284, 2011.

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

[9] 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.

[10] Tushar Deepak Chandra, Vassos Hadzilacos, and Sam Toueg. The weakest failure detector for solving consensus. *Journal of the ACM (JACM)*, 43(4):685–722, 1996.

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

[12] 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.

[13] Francis Chu. Reducing ω to $\diamond W$. *Information Processing Letters*, 67(6):289–293, 1998.

[14] R de Araújo Macêdo and F Ramon Lima e Lima. Improving the quality of service of failure detectors with snmp and artificial neural networks. In *Anais do 22o. Simpósio Brasileiro de Redes de Computadores*, pages 583–586, 2004.

[15] Alirio Santos de Sá and Raimundo José de Araújo Macêdo. Qos self-configuring failure detectors for distributed systems. In *IFIP International Conference on Distributed Applications and Interoperable Systems*, pages 126–140. Springer, 2010.

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

[17] Carole Delporte-Gallet, Hugues Fauconnier, Rachid Guerraoui, Vassos Hadzilacos, Petr Kouznetsov, and Sam Toueg. The weakest failure detectors to solve certain fundamental problems in distributed computing. In *Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing*, pages 338–346. ACM, 2004.

[18] Carole Delporte-Gallet, Hugues Fauconnier, Rachid Guerraoui, and Petr Kouznetsov. Mutual exclusion in asynchronous systems with failure detectors. *J. Parallel Distrib. Comput.*, 65(4):492–505, 2005.

[19] Michael J Fischer, Nancy A Lynch, and Michael S Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)*, 32(2):374–382, 1985.

[20] 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.

[21] Carlos Gómez-Calzado, Alberto Lafuente, Mikel Larrea, and Michel Raynal. Fault-tolerant leader election in mobile dynamic distributed systems. In *IEEE 19th Pacific Rim International Symposium on Dependable Computing, PRDC*, pages 78–87, 2013.

[22] 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.

[23] 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.

[24] 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.

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

[26] Ernesto Jiménez, Sergio Arévalo, and Antonio Fernández. Implementing unreliable failure detectors with unknown membership. *Inf. Process. Lett.*, 100(2):60–63, October 2006.

[27] Flavio Paiva Junqueira, Keith Marzullo, Maurice Herlihy, and Lucia Draque Penso. Threshold protocols in survivor set systems. *Distributed Computing*, 23(2):135–149, 2010.

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

[29] Mikel Larrea, Antonio Fernández, and Sergio Arévalo. On the implementation of unreliable failure detectors in partially synchronous systems. *Computers, IEEE Transactions on*, 53(7):815–828, 2004.

[30] Joshua B Leners, Trinabh Gupta, Marcos K Aguilera, and Michael Walfish. Improving availability in distributed systems with failure informers. In *Proc. of NSDI*, pages 427–441, 2013.

[31] Achour Mostéfaoui, Eric Mourgaya, and Michel Raynal. Asynchronous implementation of failure detectors. In *2003 International Conference on Dependable Systems and Networks (DSN 2003), 22-25 June 2003, San Francisco, CA, USA, Proceedings*, pages 351–360, 2003.

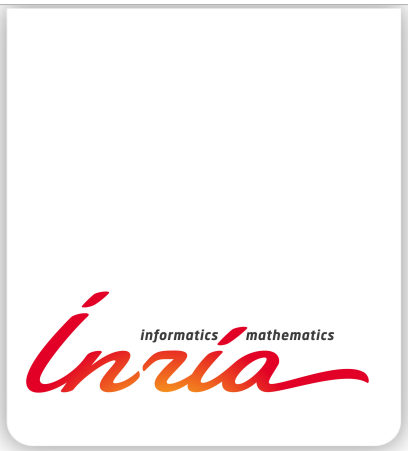
[32] Achour Mostéfaoui, Sergio Rajsbaum, Michel Raynal, and Corentin Travers. From W to omega: A simple bounded quiescent reliable broadcast-based transformation. *J. Parallel Distrib. Comput.*, 67(1):125–129, 2007.

[33] PlanetLab. Planetlab. `http://www.planet-lab.org`, 2014. Online. Access date: September 16, 2014.

[34] Anubis GM Rossetto, Cláudio FR Geyer, Luciana Arantes, and Pierre Sens. A failure detector that gives information on the degree of confidence in the system. In *2015 IEEE Symposium on Computers and Communication*, pages 532–537, 2015.

[35] 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.

[36] Maxime Véron, Olivier Marin, Sébastien Monnet, and Pierre Sens. Repfd-using reputation systems to detect failures in large dynamic networks. In *Parallel Processing (ICPP), 2015 44th International Conference on*, pages 91–100. IEEE, 2015.



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