UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

ANUBIS GRACIELA DE MORAES ROSSETTO

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

Thesis presented in partial fulfillment
of the requirements for the degree of
Doctor of Computer Science

Advisor: Prof. Dr. Cláudio F. R. Geyer

Porto Alegre
November 2016

CIP – CATALOGING-IN-PUBLICATION

Rossetto, Anubis Graciela de Moraes

Impact FD: An Unreliable Failure Detector Based on Process Relevance and Confidence in the System / Anubis Graciela de Moraes Rossetto. – Porto Alegre: PPGC da UFRGS, 2016.

120 f.: il.

Thesis (Ph.D.) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2016. Advisor: Cláudio F. R. Geyer.

1. Fault tolerance. 2. Unreliable failure detector. 3. Impact factor. 4. Trust level of the system. 5. Process relevance. 6. Margin of failures. 7. Flexibility property. I. Geyer, Cláudio F. R.. II. Título.

"We know accurately only when we know little,
with knowledge doubt increases"
(Johann Wolfgang Von Goethe)

I dedicate this thesis to my son, Luiz Augusto

and to my husband, Luiz Valerio

ACKNOWLEDGMENT

I would like to extend my sincerest thanks and appreciation to those dear people who helped me accomplish this study.

I would also like to express my gratitude to my advisor Professor Claúdio Geyer, for his guidance, background knowledge and support during the development of this work.

A respectful thank you to Luciana Arantes, my co-advisor, for her encouragement on the subject of the thesis and guidance during the writing process.

I thank Pierre Sens for the feedback, direction, and assistance when I needed it.

I would also like to express my gratitude to my colleagues Carlos Rolim, Valderi Leithardt, Guilherme Borges, Julio Anjos, João Lopes, Rodrigo Souza and other colleagues of the room 205 of Institute of Informatics (UFRGS). They have provided, with kindness, their insight and suggestions, which are precious to me.

I also want to thank Federal Institute Sul-rio-grandense for supporting the development of my Ph.D study.

I would like to thank the Brazilian Research Agency (CNPq) for the partial support by the grant 012909/2013-00.

I warmly thank and appreciate my family for understanding my absence and encouragement at every moment.

Finally, I do not have enough words to thank the people who have closely followed all my days of this period and who often I could not give the deserved attention. Valerio and Luiz Augusto, thank you for your patience, for careful attention and unconditional love. I love you.

ABSTRACT

Traditional *unreliable failure detectors* are per process oracles that provide a list of processes suspected of having failed. This work proposes a new and flexible unreliable failure detector (FD), denoted the *Impact* FD, that outputs a *trust level* value which is the degree of confidence in the system. By expressing the relevance of each process by an impact factor value as well as a margin of acceptable failures of the system, the Impact FD enables the user to tune the failure detection configuration in accordance with the requirements of the application: in some scenarios, the failure of low impact or redundant processes does not jeopardize the confidence in the system, while the crash of a high impact process may seriously affect it. Either a softer or stricter monitoring strategy can be adopted. In particular, we define some *flexibility* properties that characterize the capacity of the Impact FD to tolerate a certain margin of failures or false suspicions, i.e., its capacity of providing 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. We also show that some classes of the Impact FD are equivalent to Ω and Σ which are fundamental FDs to circumvent the impossibility of solving the consensus problem in asynchronous message-passing systems in presence of failures. Additionally, based on different synchrony assumptions and message-pattern or timer-based approaches, we present three algorithms which implement the Impact FD. Performance evaluation results using real PlanetLab traces confirm the degree of flexible applicability of our failure detector and, due to the accepted margin of failures, that false responses or suspicions may be tolerated when compared to traditional unreliable failure detectors.

Keywords: Fault tolerance, unreliable failure detector, impact factor, trust level of the

system, process relevance, margin of failures, flexibility property.

Impact FD: Um Detector de Falhas Baseado na Relevância dos Processos e Confiança no Sistema

RESUMO

Detectores de falhas não confiáveis tradicionais são oráculos disponíveis localmente para processos de um sistema distribuído que fornecem uma lista de processos suspeitos de terem falhado. Este trabalho propõe um novo e flexível detector de falhas não confiável, chamado Impact FD, que fornece como saída um valor *trust level* que é o grau de confiança no sistema. Ao expressar a relevância de cada processo por um valor de fator de impacto, bem como por uma margem de falhas aceitáveis do sistema, o Impact FD permite ao usuário ajustar a configuração do detector de falhas de acordo com os requisitos da aplicação: em certos cenários, o defeito de um processo de baixo impacto ou redundante não compromete a confiança no sistema, enquanto o defeito de um processo de alto fator de impacto pode afetá-la seriamente. Assim, pode ser adotada uma estragégia de monitoramento com maior ou menor rigor. Em particular, definimos algumas propriedades de *flexibilidade* que caracterizam a capacidade do Impact FD para tolerar uma certa margem de falhas ou falsas suspeitas, ou seja, a sua capacidade de fornecer diferentes conjuntos de respostas que levam o sistema a estados confiáveis. O Impact FD é adequado para sistemas que apresentam redundância de nodos, heterogeneidade de nodos, recurso de agrupamento e permite uma margem de falhas que não degrada a confiança no sistema. Nós também mostramos que algumas classes do Impact FD são equivalentes a Σ e Ω, que são detectores de falhas fundamentais para contornar a impossibilidade de resolver o problema do consenso em sistemas de transmissão de mensagens assíncronas na presença de falhas. Adicionalmente, com base em pressupostos de sincronia e nas abordagens baseada em tempo e padrão de mensagem, apresentamos três algoritmos que implementam o Impact FD. Os resultados da avaliação de desempenho usando traces reais do PlanetLab confirmam o grau de aplicabilidade flexível do nosso detector de falhas e, devido

à margem aceitável de falhas, o número de falsas respostas ou suspeitas pode ser tolerado quando comparado a tradicionais detectores de falhas não confiáveis.

Palavras-chave: tolerância a falhas, detectores de falhas não-confiáveis, fator de impacto, nível de confiança do sistema, relevância dos processos, margem de falhas, propriedade de flexibilidade.

LIST OF ABBREVIATIONS AND ACRONYMS

λ_R	Average Mistake Rate
AFD	Autonomic failure detector
BS	Base station
CH	Cluster head
DI	Disturbance Index
ECG	Electrocardiogram
EDCC	European Dependable Computing Conference
FD	Failure Detector
FLP	Fischer-Lynch-Paterson
FM	Fault Manager
FS	Failure Suspector
IID	Independent and identically distributed
LAN	Local Area Network
LCV	Local Connectivity Vector
MANETs	Mobile Ad hoc NETworks
ms	Millisecond
QoS	Quality of service
P_A	Query Accuracy Probability
resp	respectively

SMNP	Simple Network Management Protocol
SN	Sensor node
SR	Sample rate
SRP	Stabilized responsiveness property
TCP	Transmission Control Protocol
T_D	Detection Time
T_G	Good period duration
T_{FG}	Forward good period duration
T_M	Mistake duration
T_{MR}	Mistake recurrent time
TE	Threshold Exceeded
TH	Threshold
UTC	Universal Time Coordinated
WS	Window Size
WSN	Wireless Sensor Network

LIST OF FIGURES

CONTENTS

1 INTRODUCTION

Unreliable failure detectors have been an important topic of research for the last two decades. Proposed by Chandra and Toueg (CHANDRA; TOUEG, 1996) in order to circumvent the impossibility of deterministically solving the consensus problem in distributed asynchronous systems subject to failures, an unreliable failure detector (FD) can be seen as an oracle that gives information, not always correct, about process failures. Since in asynchronous systems, a process or the network can be arbitrarily slow, the consensus impossibility comes from the fact that it is not possible to determine whether a process has really failed or if it is just slow (FISCHER; LYNCH; PATERSON, 1985). Thus, in (CHANDRA; TOUEG, 1996), Chandra and Toueg introduced the abstraction of unreliable failure detectors and proved that consensus can be solved deterministically in asynchronous systems in presence of failures, if they are enriched with an unreliable failure detector that present certain properties. The latter is unreliable because it can make mistakes by erroneously suspecting a correct process or by not suspecting a faulty process.

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 (BONNET; RAYNAL, 2013a), homonymous (ARÉVALO et al., 2012); *type of link* (AGUILERA et al., 2004) (LARREA; ANTA; ARÉVALO, 2013), (AGUILERA et al., 2003): lossy asynchronous, reliable, timely, eventually timely, etc.; *behavior properties* (MOSTÉFAOUI; MOURGAYA; RAYNAL, 2003), (AGUILERA et al., 2004), (MALKHI; OPREA; ZHOU, 2005); *type of network*: static (BERTIER et al., 2003) (LARREA; ANTA; ARÉVALO, 2013) or dynamic (ARANTES et al., 2013) (GÓMEZ-CALZADO et al., 2013); *type of failure*: crash, omission (FERNÁNDEZ-CAMPUSANO et al., 2016) (DELPORTE-GALLET; FAUCON-

NIER; FREILING, 2005), Byzantine (GREVE et al., 2012), etc. They can also have different implementation choices: timer-based (CHEN; TOUEG; AGUILERA, 2002) (LARREA; FERNÁNDEZ; ARÉVALO, 2004), message pattern (MOSTÉFAOUI; MOURGAYA; RAYNAL, 2003); and performance or quality of service (QoS) requirements (CHEN; TOUEG; AGUILERA, 2002). The type of problem can also define the properties of the FD: mutual exclusion (DELPORTE-GALLET et al., 2005), k-set agreement (BONNET; RAYNAL, 2011), register implementation (DELPORTE-GALLET et al., 2004), etc.

The majority of these FDs are based on a binary model, in which monitored processes are either “trusted” or “suspected”. Consequently, most of them output the set of processes that are currently suspected to have crashed.

However, current distributed environments are usually large scale and/or heterogeneous infrastructures composed from super computers and large data centers to thousands of small portable computers or embedded devices. Nodes have, therefore, different relevance and/or power capacities. On the one hand, these systems are often affected by unpredictable behavior which can lead to failures. On the other hand, the service rendered by them should not be interrupted even in the presence of failures, i.e., the system reliability is not a binary property and tolerates a certain margin of failures which depends on the relevance and number of the components of the systems. Hence, a failure detector that considers these features and can be configured in accordance with the needs of the environment is fundamental.

In view of such requirements, this thesis presents a new unreliable failure detector, denoted the *Impact* Failure Detector. 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 differ-

ent importance (relevance), power, or roles and, thus, their failures may have distinct impact on the system. Systems that present node redundancy, heterogeneity of nodes, clustering, and allow a margin of failures which does not degrade the confidence in the system can, therefore, benefit from the Impact FD and its configuration choices. They have motivated our work. Section 1.1 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 ${trust\_level_p}^S$ 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 ${trust\_level_p}^S$ 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 ${trust\_level_p}^S$ 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 (AGUILERA et al., 2004). 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. (2010) propose 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 *flexibility property* expresses the capability of the Impact FD 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 $PR(\Diamond IT)_p^S$, 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., $PR(\Diamond IT)_p^S$) 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. Furthermore, inspired in (MALKHI; OPREA; ZHOU, 2005), we also introduce the concept of $PS{-}accessibility$ and $\Diamond PS{-}accessibility$: a correct process p is $PS{-}accessible$ (resp., $\Diamond PS{-}accessible$) if every query broadcast by p obtains from the beginning (resp., eventually) a set Q of responses that satisfy the degree of confidence in S, i.e., the trust level of S is greater or equal than the $threshold$ value. Interestingly that the set Q of processes is not fixed, i.e., it can change at each query, which is in accordance with the *flexibility property* of the Impact FD.

Taking into account the problem of solving consensus in asynchronous message-passing systems enriched with failure detectors, we show in this thesis 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 (CHANDRA; HADZILACOS; TOUEG, 1996) and Σ FD (or Quorum FD) (DELPORTE-GALLET et al., 2004) are two fundamental classes of failure detectors; the Ω FD is the weakest one to solve consensus, provided that a majority of processes are correct, while the pair of FDs $<\Omega,\Sigma>$ 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 of failures and/or membership hold.

We also present different algorithms for the Impact FD with their respective proofs of correctness, showing, therefore, that the Impact FD can have distinct implementations. The latter depends on the characteristics of the system model and behavior properties, exploiting either a timer-based or message-pattern approach. Then, based on real trace files collected from nodes of PlanetLab (PLANETLAB, 2014), 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 heartbeats. 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.

1.1 Motivation

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: A system in the area of healthcare requires the use of several sensors to measure different kinds of information about the health status of a person, such as, vital signs, location, falls, gait patterns, and acceleration. From this perspective, this

is a critical situation since any faults in the components can put a patient's life at risk. For instance, consider a scenario with four sensors: q_1 - *body temperature;* q_2 - *pulse;* q_3 - *electrocardiogram (ECG); and* q_4 - *galvanic skin* as well as node p, responsible for collecting information from these sensors and taking appropriate action based on the output of the Impact FD. In this example, some sensors are not considered to be critical, such as the sensor q_1 which measures the temperature. On the other hand, q_3, the ECG sensor, is crucial which means that the impact factor assigned to q_3 should be higher than that of q_1. Furthermore, q_3 collects data about both the heartbeats and electrical activity of the heart while q_2 is a type of sensor that also collects data about heartbeats. Hence, there is redundancy of information, i.e., the failure of q_2 sensor is not critical enough to make the system vulnerable and endanger the life of the monitored person. We could then define a threshold as being equal to the sum of the impact factor of all the sensors minus the impact factor of the q_2 since, the failure of q_2 does not jeopardize the trustworthiness of the system.

Scenario 2: Another important scenario that underlies our proposal is Ubiquitous Wireless Sensor Networks (WSNs). These kinds of networks are deployed to monitor physical conditions in various places such as geographical regions, agriculture lands, battlefields, etc. In WSNs, there is a wide range of sensor nodes with different battery resources and communication or computation capabilities (ISHIBASHI; YANO, 2005). However, these sensors are prone to failures (e.g., battery failure, process failure, transceiver failure, etc.) (GEETA; NALINI; BIRADAR, 2013). Hence, it is necessary to provide failure detection and adaptation strategies to ensure that the failure of sensor nodes does not in any way 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 (ABBASI et al., 2014).

Let us take as example an ubiquitous WSN which is used to collect environmental data from within a vineyard (Figure 1.1 (a)), and is divided into management zones in accordance with different characteristics (e.g., soil properties). In the Figure 1.1 (a), the area is divided into five zones (Z1, Z2, Z3, Z4, Z5). 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. Furthermore, in some situations, there might be a need to dynamically reconfigure the density of the zones, for instance, when there is a stability of weather conditions. In this case, the management zones can be jointed and the density of sensor decreased, as illustrated in Figure 1.1 (b), where zones Z4 and Z5 were joined and some sensors disabled. Consequently, the *threshold* value would change.

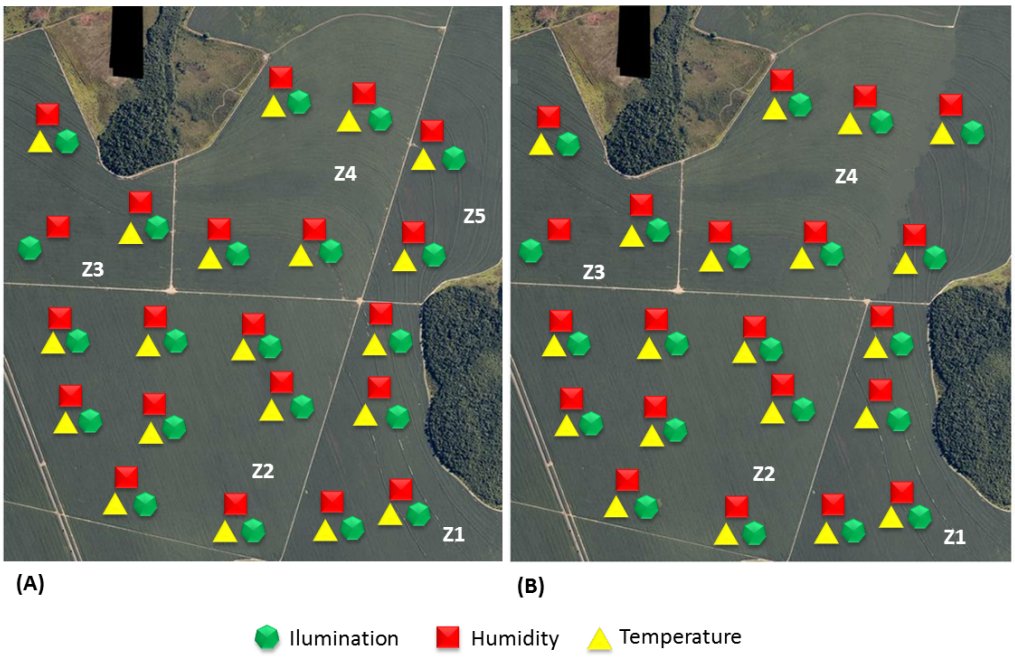

Figure 1.1: WSN in management zones

Scenario 3: 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 $cluster_i$ has a node, denoted cluster head (CH), which performs special tasks (for instance, 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 com-

munication 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 implies that the cluster is inaccessible, compromising network connectivity, therefore, the failure of CH leads to untrusted state 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 account, we could have: impact factor = 1 to SN, impact factor = n_i to the CH of $cluster_i$, and threshold for this cluster equal 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 4: A fourth example is a system with a main server that offers a certain quality of service X (bandwidth, response time, etc.). If it fails, N backup servers can replace it, since each backup offers the same service but with a X/N quality of service. In this scenario, both the impact factor of the main server and the *threshold* would have the value of $N * Iback$ where $Iback$ is the impact value of each backup server, i.e., the system becomes unreliable whenever the primary server and one or more of the N servers fail (or are suspected of being faulty).

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.

1.2 Main Objectives and Contributions of the Thesis

The main goal of this thesis is the conception of a new unreliable failure detector, denoted **Impact Failure Detector**, which exploits process relevance and the confidence degree in the system. We then seek to the following specific objectives:

- Formally define the *Impact* failure detector, which is based on process relevance and the confidence degree in the system;
- Define behavior properties that characterize the flexibility feature of the *Impact*

FD;

- Propose different algorithms that implement the *Impact* FD and show their proofs of correctness. Furthermore, if the above properties hold, the algorithms should ensure that the failure detector will always consider the system to be trusted;
- Show the equivalence between classes of the *Impact* FD to Ω and Σ failure detectors;
- Conduct experiments based on real traces for evaluating the effectiveness and QoS of the *Impact* FD.

As contributions of this thesis, we highlight the definition of a new unreliable failure detector, the *Impact FD*, which is based on process relevance and the confidence degree in the system. The proposed failure detector includes features which are not considered by traditional unreliable failure detectors: it provides as output a trust level of a set of processes (*trust level*); it allows processes to have different levels of importance (*impact factor*) and to be grouped in subsets; it tolerates a margin of failures (*threshold*). These features make the Impact FD very flexible and robust, providing an efficient solution for different applications with specific reliability requirements. The thesis also introduces the properties $PR(IT)_p^S$ and $PR(\Diamond IT)_p^S$, which characterize necessary conditions to ensure confidence (or eventual confidence) of the monitor process p in system S as well as the concept $PS{-}accessibility$ and $\Diamond PS{-}accessibility$. The latter enable Impact FD implementations (see Section 5.3.2) that do not depend on a maximum number of failures, which is a very useful feature in accordance with (JUNQUEIRA et al., 2010), as previously discussed in this chapter. Additionally, since the concept of Impact tolerates a margin of failures and false suspicions, it is expected that the Impact FD presents better QoS when the application is interested in the degree of confidence in the system (trust level) as a whole. Finally, we should point out that some classes of Impact FD can be used (see Section 4.5) to enrich message-passing asynchronous systems in order to circumvent the impossibility of the consensus problem.

1.3 Structure of the Document

The document is structured as follows. In Chapter 2 we provide a background of main concepts related to failures in distributed systems and unreliable failure detectors. Chapter 3 discusses relevant related work. Chapter 4 presents the *Impact* failure detector, its characteristics, some of its properties, and shows the equivalence of some classes of Impact FD in regard with Σ and Ω classes. Chapter 5 presents three algorithms (and their respective proofs of correctness) that implement the Impact FD, considering different synchrony assumptions, implementation choices (message-pattern and timer-based), and behavior properties. Chapter 6 presents evaluation results of experiments conducted with real traces on PlanetLab (PLANETLAB, 2014). Chapter 7 summarizes the conclusions of this thesis and highlights some future research directions.

2 BACKGROUND

In this chapter, we present some of the main concepts related to failures in distributed systems. First, we set out fundamental notions of dependability. Next, we discuss failures in distributed environments and provide an overview on some important concepts. Finally, we outline basic concepts of unreliable failure detectors and examine important features with regard to their implementation.

2.1 Dependability

Computing systems are subject to failures that may be from a variety of sources such as hardware failures, software bugs, malicious attacks, operation/maintenance human errors, and natural disasters. The capability of avoiding failures that can put services at risk is a guideline that must be followed. Therefore, dependability is a key issue in such systems. According to AVIZIENIS et al. (2004), dependability is the ability to deliver service that can justifiably be trusted, in spite of continuous changes.

As stated in AVIZIENIS et al. (2004), the dependability concept can be further extended to encompass mechanisms required to increase and maintain the dependability of a system. The authors add that the three main elements of dependability are attributes, threats, and means. Attributes are the qualities of a system that can be measured in terms of qualitative and quantitative features such as availability, reliability, safety, integrity, and maintainability. These attributes are differently emphasized, depending on the considered application.

On the other hand, threats are characterized by faults, errors, and failures. A computing system can be considered as a sum of components. The trustworthiness of the interaction between the system components can be seen in Figure 2.1. Denoted a

fault-error-failure chain by AVIZIENIS et al. (2001), the figure represents a component which can eventually return a fault, i.e., the component is not delivering an expected service. If this fault is activated, it can cause an error, which in turn (if propagated) can cause a failure. This failure will be translated by the next component as a fault.

The purpose of dependability is to reduce the number of failures experienced by the users of a system. To this well, the following well-known paradigms are applied: prevention, removal, forecasting, and tolerance. The prevention approach seeks to prevent faults from being incorporated into the systems; removal methods aim at reducing the number and seriousness of faults; forecasting techniques are used to predict or circumvent faults; fault tolerance is a paradigm characterized by the fact that it allows a system to continue delivering the required service in the presence of faults, although that service may be degraded. Therefore, the building of dependable systems is closely related to the control of failures.

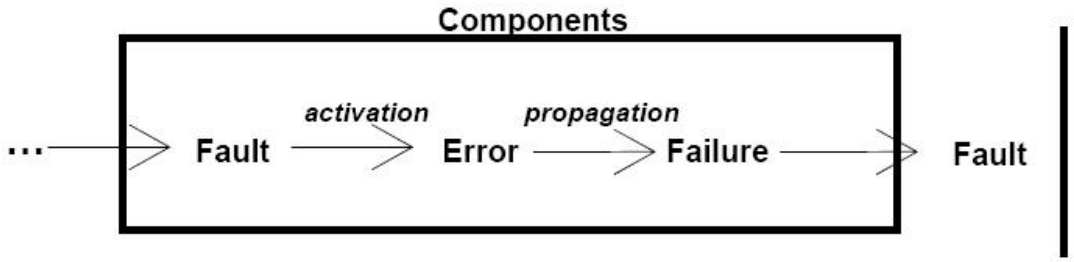

Figure 2.1: Fault-Error-Failure chain (AVIZIENIS et al., 2001)

2.2 Failures in Distributed Environments

According to COULOURIS; DOLLIMORE; KINDBERG (2005), a distributed system consists of a finite set of autonomous processes linked by a computer network which communicate with one another by exchanging messages. To refer to these interconnected elements the most common terms are node or process. We consider that there is one process by node (site) or sensor. Therefore, the word *process* can mean a node, sensor, or site.

Unfortunately, in distributed systems failures can occur. Several problems related to distributed systems requires some coordination among the various *processes* which communicate via communication channels/link. Both processes and communication channels are subject to failures. If a process fails during execution, it is said *faulty*,

otherwise the process is called *correct*. In this way, a key requirement for coordination among the processes is that they know each others states, so that they can take appropriate action in case of failure. In some types of distributed systems, this may be a difficult or impossible task, for instance in the case of asynchronous systems. With this in mind, in this section we take a closer look at some important concepts necessary to understand failures in distributed systems and their detections: failure models, synchrony models, communication channels/links, and the consensus problem in asynchronous message-passing systems in the presence of failures.

2.2.1 Fault Models

Faults are usually identified by the way they affect the behavior of the system components. The different ways in which a generic system component can fail are called failure models and were classified by Cristian (1991) *apud* (JALOTE, 1994) as:

- *Crash*: It occurs when the component stops working, ceasing to respond to any service request. This is the most common kind of failure in the system models related to the design of distributed systems. For instance, in a sensor node, the failure can occur due to at least one of the following reasons: a) battery is completely depleted, b) transceiver is faulty, or c) node is completely damaged. If nodes can recover without losing internal state, this model is called *crash-recovery*;
- *Omission*: It occurs when a component does not respond to some entries or requests, having an intermittent or transient behavior. For instance, in a *send omission* failure, a message which is sent by a process is never placed into the communication channel while in a *receive omission* failure, a message which arrives over the communication channel is never actually deliver to the destination process. Another example is a sensor node that does not respond to the sink node in time, fails to send a required message on time, or fails to relay the received message to its neighbor;
- *Timing/Performance*: It occurs when the response of component is functionally correct, but is provided outside the specified time interval. This kind of failure may either occur as an early response or through a late response. The timing failure is related to violations of delay bounds;
- *Byzantine*: Are the most general type of failures which can correspond to any ar-

bitrary behavior. For instance, a faulty process may change the content of messages, duplicate messages, send unsolicited messages, or even maliciously try to break down the whole system. This model includes the other models described above.

2.2.2 Synchrony Models

According to LARREA; FERNÁNDEZ; ARÉVALO (2004), distributed algorithms can be designed under different assumptions of system behaviors, i.e., system models. One of the main assumptions in which system models can differ is related to the timing aspects to which two attributes are considered: the time taken for message transmission across a communication channel and the time taken by a processor to execute a piece of code. Depending on whether these attributes are bounded or not, and on the knowledge of these bounds, they can be classified as synchronous, asynchronous, or partially synchronous (CHANDRA; TOUEG, 1996).

2.2.2.1 Synchronous System

A distributed system is synchronous if there are bounds on transmission delay and processing time and these bounds are known (VERISSIMO; RODRIGUES, 2012). In such systems, for instance, it is possible to distinguish crashed processes from slow processes or slow messages from omitted messages, which allows to deterministically detect failures. However, one problem of this type of system is the fact that bounds have to be defined to behave correctly in the worst case. Otherwise, the assumptions of the system model might be violated and, therefore, the algorithm may lose correctness (VERISSIMO; RODRIGUES, 2012).

In synchronous distributed systems, message transmission delays and process speed are bounded and known, such that a simple timeout mechanism can be used to surely assert if a node has failed or not.

2.2.2.2 Asynchronous System

A distributed system is asynchronous if there is no bound on message transmission delay nor the time to execute a processing step, i.e., there is no timing assumption (CRISTIAN; FETZER, 1999) (VERISSIMO; RODRIGUES, 2012). In asynchronous distributed systems, since there are not bounds on process speed neither message de-

lay, no mechanism can ensure the failure of a remote process since it is impossible to know whether the latter has actually crashed or whether its message transmissions are delayed for some reason (ARANTES; GREVE; SENS, 2010).

2.2.2.3 Partially Synchronous System

In general, a real system holds timing bounds most of the time, i.e, it behaves like a synchronous system almost always but can be unstable during some periods of time, behaving like an asynchronous system and thus exceeding normal bounds.

The partial synchrony was introduced by DOLEV; DWORK; STOCKMEYER (1987) aiming at defining an intermediate model between synchronous and asynchronous systems, namely partially synchronous systems. In particular, the authors introduced 32 models of partial synchrony by the combination of five parameters which can be set to ”favorable” or ”unfavorable”. However, an important definition about partially synchronous systems was presented by DWORK; LYNCH; STOCKMEYER (1988) who considered two interesting models: $M1$) There exist time bounds on message passing delays and relative speed of processes, but they are not known; $M2$) Bounds exist and are known, but hold only after an unknown time called Global Stabilization Time (GST). This system is also called eventually synchronous. Basically, before GST the system behaves like an asynchronous system and after GST it holds bounds like a synchronous system.

Another important definition was presented by CHANDRA; TOUEG (1996) who identified the previous models as $M1$ and $M2$ and introduced another partially synchronous model, namely $M3$, with the weakest assumptions of the previous ones: bounds exist but they are not known and they hold only after some unknown time GST.

In short, it is not necessary that an eventually synchronous system holds its bounds forever, i.e., behaves like a synchronous system for good. However, it is enough that stability lasts during the necessary time for the algorithm to finalize its execution (GUERRAOUI; RODRIGUES, 2006).

2.2.3 Communication Channels/Links

The behavior of the communication channels is an important issue in the definition of a distributed system model. Communication channels or links represent an abstraction of the network. Processes communicate among them by sending messages

though their corresponding links.

A unidirectional communication link from a process p to another process q allows p to send messages to q. When a link from p to q is bidirectional, it is assumed a communication link from p to q and another one from q to p.

In AGUILERA et al. (2004), the authors have defined the following properties:

- *integrity*: ensures that q receives a message m from p at most once, and only if p previously sent m to q. In other words, communication links cannot create or alter messages;
- *fairness*: messages carry a *type* in addition to its data. For every type, if p sends infinitely many messages of type *type* to q and q is correct, then q receives infinitely many messages of type *type* from p;
- *timeliness*: There exists a time interval δ such that if p sends a message m to q at time t and q is correct, then q receives m from p by time $t + \delta$. The maximum message delay δ is not know;
- $\diamondsuit-timeliness$: there exists δ and a time t such that if p sends a message m to q at time $t' \geq t$ and q is correct, then q receives m from p 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 consider the following types of links:

- $lossy\ asynchronous$: A link that satisfies the *integrity* property. 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;
- *reliable*: A link that satisfies the *integrity* property and does not lose messages. Every message sent by this link will be delivered and will not be lost.
- $fair\ lossy$: A link that satisfies the *integrity* property and the *fairness* property. Note that a reliable link is also fair lossy;
- *correct-restricted reliable*: Such a link is a variant of the reliable link that behaves as a reliable link only if the sender and the recipient do not crash. Otherwise, the link may behave as fair-lossy one;
- *timely*: A link that satisfies the *integrity* property and the *timeliness* property. A timely link is also reliable;

- $\Diamond-timely$: A link that satisfies the *integrity* property and the $\Diamond-timeliness$. Note that messages sent before time t can be lost and a $\Diamond-timely$ link is also a fair-lossy.

2.2.4 Consensus Problem in Presence of Failures

Consensus is a widely-studied fundamental problem in distributed computing, theory and practice. Intuitively speaking, it requires the processes of the system to agree on a common decision value. The consensus problem describes how all participants of a distributed system must decide on a common value: every process p_i proposes a value v_i and, eventually, every correct process calls the primitive $decide(d)$, where d is the same value for all correct processes and is chosen among the set of proposed values $v_1; v_2; ...; v_n$.

Formally, a Consensus implementation has to satisfy the following three properties:

- Validity: Every correct process has to decide a proposed value;
- Agreement: Every correct process has to decide the same value;
- Termination: Every correct process eventually decide.

Although many solutions have been proposed to solve consensus in synchronous systems, FISCHER; LYNCH; PATERSON (1985) presented an impossibility result, namely Fischer-Lynch-Paterson or FLP Impossibility, that states that it is impossible to reach consensus, deterministically, in an asynchronous systems subject to even a single crash failure. Intuitively, this impossibility stems from the following fact: in a purely asynchronous system there is no bound on the transmission delay, i.e., a message may be arbitrarily long in transit from a process p to another process q. On the other hand, processes may fail by crashing. Now, if p waits for a message from q, it can be never sure, whether this process has crashed or if just the process is slow. Whatever p does may be the wrong decision: if it waits without bound it may wait forever, since q may have crashed - violating the termination property. If it stops waiting for the message and continues without this message, the message might have been just slow, and p decides without the information from q, which may lead to a violation either of the agreement or the validity property.

The concept of partially synchrony emerged as a way for circumventing the FLP impossibility (Section 2.2.2.3). In this case, systems become synchronous eventually, so,

instead of using fixed value timeouts, algorithms should implement adaptive timeouts.

Also, aiming to overcome the impossibility of deterministically soving distributed consensus in fully asynchronous systems, CHANDRA; TOUEG (1996) introduced the concept of unreliable failure detectors. A failure detector is a module, available at every process, that informs, in a possibly unreliable way, about the operational state of processes in the system. The next sections of this chapter give a more detail description of failure detectors, their properties, and implementation.

Many other fundamental problems in fault-tolerant distributed computing also require agreement among the participants. Therefore, they also suffer from the same consensus impossibility result in asynchronous problem in presence of failures. Some examples of such problems are: *atomic broadcast* (LAMPORT, 1978), where messages are received by all processes in a total order; *group membership* (CHANDRA et al., 1996) where all members agree on the set of members; and *leader election* (LAMPORT; LYNCH, 1991) where all non-crashed processes agree on the same leader process.

We should also remember that *uniform consensus* (CHARRON-BOST; SCHIPER, 2004) is a variation of the consensus problem with the condition that no two processes (whether faulty or not) decide differently. Thus, the agreement property of the classical consensus is replaced by the *uniform agreement* property: every process has to decide the same value.

2.3 Unreliable Failure Detectors

An important abstraction for the development of fault-tolerant distributed systems is the unreliable failure detector (FD) (CHANDRA; TOUEG, 1996). As mentioned in the previous section, failure detectors were proposed by CHANDRA; TOUEG (1996) as an abstraction to encapsulate timing assumptions when solving consensus.

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. In addition, a failure detection system consists of local modules in which each machine may monitor a group or subgroup of system processes, or even be monitored by other detectors.

According to HAYASHIBARA; DÉFAGO; KATAYAMA (2003), unreliable FDs are so named because they can make mistakes (1) by erroneously suspecting a correct pro-

cess (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 the unreliable FDs can not accurately determine the real state of processes, using them increases knowledge about the processes of the system (CHANDRA; TOUEG, 1996). This means that an unreliable FD encapsulates the uncertainty of the communication delay between two distributed entities.

CHANDRA; TOUEG (1996) proposed failure detectors that output a list of suspected processes. On the other hand, HUTLE (2005) denotes the latter $suspicion\ based$ failure detectors and additionally characterizes failure detectors which output a list of non-suspected processes, i.e., $trusted$ processes, as $trust\ based$ ones.

Furthermore, CHANDRA; TOUEG (1996) presented some important formal definitions. They considered that processes can fail by crashing and assume the existence of a discrete global clock which is merely a fictional device:

- Failure patterns: A crash failure pattern is a function that defines a possible set of failures, and their corresponding time, that can occur in an execution. A failure pattern F is a function from $T \rightarrow 2^{\Pi}$, where $F(t)$ denotes the set of processes that have crashed through time t. Once a process crashes, it does not "recover", i.e., $\forall t : F(t) \subseteq F(t+1)$. They also defined $crashed(F) = \cup_{t \in T} F(t)$ and $correct(F), = \Pi - crashed(F)$. If $p \in crashed(F)$, it is said that $p\ crashes$ in F and if $p \in correct(F)$, it is said that p is $correct$ in F;
- Failure Detector History: It is a function that provides a list of the processes that a failure detector is suspecting at a given time. Formally, a failure detector history H is a function $\Pi \times T \rightarrow 2^{\Pi}$. $H(p,t)$ is the value of the failure detector module of process p at time t. If $q \in H(p,t)$ it is said that p suspect q at time t in H;
- Failure Detector: Failure detector D is a function that maps each failure pattern F to a set of failure detector histories $D(F)$. This is the set of all failure detector histories that could occur in executions with failure pattern F and failure detector D.

2.3.1 Classes of Failure Detectors

Failure detectors can be classified into classes based on properties they satisfy. This approach allows to design applications and prove their correctness based only on these properties, without referencing, for example, low-level network parameters.

According to CHANDRA; TOUEG (1996), unreliable failure detectors are characterized by two properties, *completeness* and *accuracy*. 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. Failure detectors are then classified according to two completeness properties and four accuracy properties (CHANDRA; TOUEG, 1996):

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

The combination of these properties yields eight classes of failure detectors as can be seen in Table 2.1.

For instance, the $\diamondsuit P$ (Eventually Perfect Failure Detectors) class includes all the failure detectors that, after some unknown but finite time, make no mistake. They satisfy the *strong completeness* and *eventual strong accuracy* properties.

2.3.2 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) (CHANDRA; HADZILACOS; TOUEG, 1996) and Sigma (or Quo-

Table 2.1: Failure detectors classification

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

rum - Σ) (DELPORTE-GALLET et al., 2004) failure detectors.

The Leader Failure Detector Omega (Ω): Together with Hadzilacos, Chandra and Toueg extended their work in (CHANDRA; HADZILACOS; TOUEG, 1996), 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 $\Diamond S$ and $\Diamond W$, the knowledge of membership of the system is not necessary (JIMÉNEZ; ARÉVALO; FERNáNDEZ, 2006). When it is known, a Ω FD trivially also implements a $\Diamond W$ or $\Diamond S$ 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, $LEADER_p = l$.

Note 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 con-

tains only correct processes.

According to DELPORTE-GALLET; FAUCONNIER; GUERRAOUI (2003), 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; FAUCONNIER; GUERRAOUI (2003), 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.

2.3.3 Reducibility and Equivalence

Failure detectors can be compared with each other through the notions of reducibility and equivalence. According to CHANDRA; TOUEG (1996), reducibility means that there is an algorithm $T_{D \to D'}$ which transforms a failure detector D into another failure detector D'. Algorithm $T_{D \to D'}$ uses D to maintain a variable $output_p$ at every p. Given a reduction algorithm $T_{D \to D'}$, any problem that can be solved using failure detector D', can be solved using D instead. Thus, if there is an algorithm $T_{D \to D'}$ that transforms D into D', we say that *D' is reducible to D*, noted $D \succeq D'$; we also say that *D' is weaker than D* ($\succeq$ is a transitive relation). Furthermore, if $T_{D \to D'}$ and $T_{D' \to D}$ both exist, then $D \cong D'$ and we 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.

A failure detector being weaker than another means that the stronger failure detector can provide an emulation of the weaker failure detector. Regarding the failure detectors classes proposed by CHANDRA; TOUEG (1996) (in Table 2.1), they provide that $P \cong Q, S \cong W$, $P \cong \diamondsuit Q$, and $\diamondsuit S \cong \diamondsuit W$. This means that the class of failure detectors on the same column of Table 2.1 are equivalent. They also prove that $P \succeq S, Q \succeq W, \diamondsuit P \succeq \diamondsuit S$, and $\diamondsuit Q \succeq \diamondsuit W$. That is, failure detectors with weak accuracy are weaker than the ones with strong accuracy.

Additionally, the authors prove that weakest class of failure detectors to solve consensus is $\diamondsuit W$, provided that there exist a majority of correct processes. Hence, any failure detector that solves consensus must be at least as strong as a $\diamondsuit W$.

2.3.4 Implementation of Failure Detectors

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

In the *timer-based* strategy, FD implementations make use of timers to detect failures in processes. There exist two mechanisms that can be used to implement the timer-based strategy: *heartbeat* and *pinging*. In the *heartbeat* mechanism every process q periodically sends a control message (*"I am alive" message*) to process p that is responsible for monitoring q. If p does not receive such a message from q after the expiration of a timer, it adds q to its list of suspected processes. If p later receives an *"I am alive"* message from q, p then removes q from its list of suspected processes.

An alternative approach uses the *pinging* mechanism which sends a query message *"Are you alive?"* from each process p to another process q periodically. Upon reception of such messages, the monitored process replies with an *"I am alive"* message. If process p times out on process q, it adds q to its list of suspected processes. If p later receives an *"I am alive"* message from q, p then removes q from its list of suspected processes. The *heartbeat* strategy have advantages over *pinging* since the former sends half of the messages pinging detectors send for providing the same detection quality. Furthermore, a heartbeat detector estimates only the transmission delay of *"I am alive"* messages, whereas the pinging detector must estimate the transmission delay of *"Are you alive?"* messages, the reaction delay, and the transmission delay of *"I am alive"* messages.

The *message-pattern* strategy does not use any timeout mechanism. In (MOSTEFAOUI; MOURGAYA; RAYNAL, 2003), the authors propose an implementation that uses a request-response mechanism. A process p sends a $QUERY$ message to n nodes that it monitors and then waits for responses ($RESPONSE$ message) from α processes ($\alpha \leq n$, traditionally $\alpha = n - f$, where f is the maximum number of failures). A query issued by p ends when it has received α responses. The other responses, if any, are discarded and the respective processes are suspected of having failed. A process sends $QUERY$ messages repeatedly if it has not failed. If, on the next request-response, p receives a response from a suspected process q, then p removes q from its list of suspects. This approach considers the relative order for the receiving of messages which always (or after a time) allow some nodes to communicate faster than the others.

2.3.5 QoS Metrics for Failure Detectors

A set of metrics have been proposed by CHEN; TOUEG; AGUILERA (2002) aiming at evaluating the quality of service (QoS) of failure detectors: how fast they detect actual failures and how well they avoid false detections, i.e., the speed and accuracy of failure detectors.

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

The proposed metrics are the following:

- Primary Metrics:
  - *Detection Time* (T_D): the time that elapses from the moment that process q crashes until the FD at p starts suspecting q permanently. More precisely, T_D measures the time that elapses from the moment that the crash of q occurs to the moment when the final $S-transition$ occurs (at p) and there are no further transitions (see figure 2.2);
  - *Mistake recurrency time* (T_{MR}): the time between two consecutive mistakes, i.e, it is a random variable representing the time that elapses from an $S-transition$ to the next one (see Figure 2.2);
  - *Mistake duration* (T_M): the time taken by the failure detector to correct a mistake, i.e., it is a random variable representing the time that elapses from an $S-transition$ to the next $T-transition$ (see Figure 2.2).
- Derived Metrics:
  - *Average Mistake Rate* (λ_R): measures the rate at which a failure detector makes mistakes, i.e., it is the number of $S-transitions$ per unit of time. This is an important metric for long-lived applications where a mistake results in a costly interrupt, such as group membership applications (SCHIPER; TOUEG, 2008) and cluster management (CORREIA; NEVES; VERÍSSIMO, 2006);

- *Query Accuracy Probability* (P_A): the probability that the failure detector's output is correct at a random time. This metric is useful for applications that interact with the failure detector by querying it at random times;
- *Good Period Duration* (T_G): measures the length of a good period, i.e., it is a random variable representing the time that elapses from a $T-transition$ to the next $S-transition$;
- *Forward Good Period Duration* (T_{FG}): a random variable representing the time that elapses from a random time at which p trusts q, to the time of the next $S-transition$.

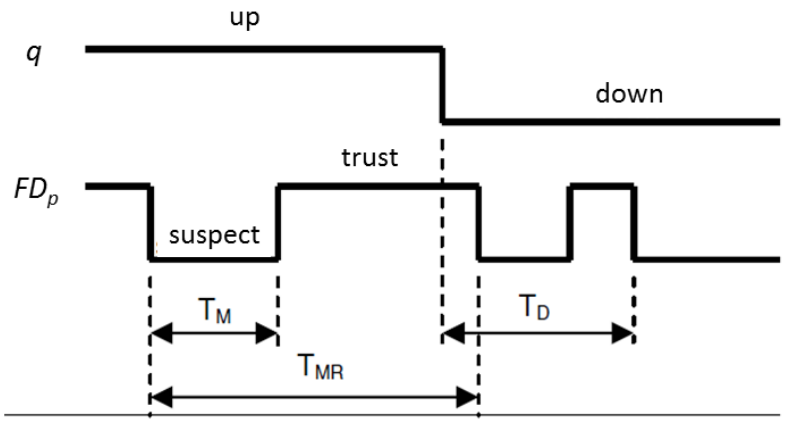

Figure 2.2: QoS Metrics

In order to make a proper comparison of the Impact FD, we used some of the QoS metrics described above in the evaluation presented in chapter 6.

3 RELATED WORK

We can divide the state-of-the art related to this thesis into three groups: (1) failure detectors and failure handling systems, (2) heartbeat arrival estimation strategies, and (3) works which consider additional assumptions for asynchronous systems.

3.1 Failure Detectors and Failure Handling Systems

In this section we discuss some existing works that, similarly to the Impact FD, (a) provide an output that it is not a list of suspected processes and should be interpreted/ compared by the application ((HAYASHIBARA et al., 2004), (COSQUER; RODRIGUES; VERÍSSIMO, 1995), (GUPTA et al., 2013)); (b) assign some value (e.g. weight, probability of failure) to nodes ((YIM; CHOI, 2010) , (BRUN et al., 2011), (ARANTES et al., 2015), (VÉRON et al., 2015)): (c) consider a threshold value to define the confidence in the system or answers given by it ((ARANTES et al., 2015), (BRUN et al., 2011)); (d) flexibility in the number of failures ((ARANTES et al., 2015), (BRUN et al., 2011), (JUNQUEIRA et al., 2010)).

Accrual: The Accrual failure detector (DÉFAGO et al., 2005) adopts 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.

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

The aim of Accrual failure detectors is to decouple monitoring from interpretation. According to the authors, the Accrual failure detectors provide a lower level abstraction that avoids having to interpret monitoring information. The suspicion level is left for the applications to interpret. For instance, by setting an appropriate threshold, applications can trigger suspicions and take appropriate action. In addition, applications can directly use the value output by the detector as a parameter for their actions.

Cosquer et al.: This work proposes a group membership service which tunes its failure detection by monitoring several system parameters combined internally into a single value (COSQUER; RODRIGUES; VERÍSSIMO, 1995). To this end, the Failure Suspector (FS) is designed to evaluate a number of operational parameters as roundtrip delay, throughput, and error rate. The application can configure the failure suspector, by indicating which parameters must be measured and by giving their acceptable values.

Each parameter is characterized by a set of variables such as: the maximum interval at which the parameter needs to be evaluated; the maximum (threshold) acceptable value for the parameter; the parameter's relative importance (weight); the number of samples at which the parameter value has exceeded its threshold; the current value for the parameter, etc. The authors also define the disturbance index (DI) whose aim is to provide an index that measures the overall connectivity to a remote node and informs how often the operational parameters have exceeded the defined thresholds. A very low DI value means that the connectivity is satisfactory while a high value means very poor connectivity and, thus, the remote node is considered suspected.

Pigeon: Arguing that applications should have information about failures to take specific and suitable recovery actions, GUPTA et al. (2013) define a service, called Pigeon, for reporting faults to applications. Pigeon works within a single administrative domain (an enterprise, a data center, a campus network) and its architecture is geared toward extracting and exploiting the information about failures that is already available inside the system. The architecture of Pigeon has sensors, relays, and interpreters. A sensor is component-specific and tailored; it is embedded in the component and can detect faults in it. Relays communicate with sensors and propagate these sensors' fault information to end-hosts. Each end-host has an interpreter that receives information

about faults from the relays. The interpreters convey this information as failure conditions and estimate the expected duration of these conditions.

The Pigeon also encapsulates uncertainty which allows applications to proceed safely in the presence of doubt. The service, called *failure informer*, provides status reports related to fault detection with an abstraction that describes the degree of uncertainty. The *failure informer* interface discloses the conditions (shown in Figure 3.1) to the applications, where each condition abstracts a class of problems in a remote target process that affects the distributed application.

condition	occurred?	permanent?	description	example causes
stop	certain	certain	target stopped executing	core dump, machine reboot
unreachability	certain	uncertain	target unreachable	network link down
stop warning	expected; imminent	certain	target may stop executing	disk about to crash
unreachability warning	expected; imminent	uncertain	target may become unreachable	network link close to capacity, CPU overloaded

Figure 3.1: Conditions reported by Pigeon (GUPTA et al., 2013)

Yim et al.: In (YIM; CHOI, 2010) the authors present the concept of confidence levels of sensor nodes in an adaptive fault-tolerant event detection service for wireless sensor networks. After the analysis of the consistency of detecting results of nodes, the confidence levels of the nodes are dynamically adjusted as well as the threshold, which is used for decision making. The solution also exploits a filter for tolerating transient faults to correct some erroneous sensor readings. The approach models a sensor network as a weighted directed graph, *G(V, E)*, where V represents the set of sensor nodes and E represents the set of edges connecting the sensor nodes. Each node v_i is assigned a self-confidence level c_i. Each edge e_{ij} is also assigned a weight w_{ij}, which indicates the confidence level of v_j from the viewpoint of v_i. Confidence levels of nodes are used to isolate potentially faulty sensor nodes from the rest of the network as well as to reinstate an isolated node if the confidence levels associated to it satisfies required conditions. The confidence levels are updated each time a fault detection or event detection is performed.

RepFD: In (VÉRON et al., 2015), the authors propose the use of a reputation mechanism to implement failure detectors 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 in order to increase the quality of detection both in terms of completenness and accuracy. A reputation service based on the periodically heartbeat messages exchanged by nodes, classify nodes in terms of reputation: the reputation of a node dynamically increases if it sends its heartbeat on time, and decreases if some heartbeats get lost or arrive after the expected dates.

Survivor sets and cores: JUNQUEIRA et al. (2010) define a system model that can express correlated failures using cores and survivor sets. It is an alternative dependent failure model that captures the behavior of systems where failures are not necessarily independent and identically distributed. Giving as example replication protocols which use the traditional maximum number of f failure approach and can, therefore, be suboptimal by replicating more than necessary, the authors propose the *survivor sets*, the unique collection of minimal sets of correct processes over all executions, each set having all correct processes of some execution. Moreover, they also define *core* as a subset of processes that generalizes subsets of size $f+1$ in the maximum number of failures model. Hence, in every execution of the system, there is at least one process in every core that is correct. From the set of cores, one can obtain the survivor set system by creating all minimal subsets of processes that intersect every core.

Brun et al.: Assuming that each node has a probability of being Byzantine, a voting node redundancy approach is adopted in (BRUN et al., 2011) in order to improve the reliability of distributed systems. On the basis of these probability values, the authors estimate the minimum number of machines that the system should have to provide a degree of reliability which is equal to, or greater than, a threshold value (probability threshold). The threshold is defined by the probabilistic reliability that this answer will be correct, given the average reliability of the nodes in the system. The authors proposed an iterative redundancy approach whereby the system chooses a number of nodes to perform the computation necessary to reach the required reliability level. If consensus cannot be reached among the selected nodes, more nodes are allocated for the computation. The algorithm is repeated until the required degree of reliability is achieved.

Arantes et al.: In (ARANTES et al., 2015), the authors present a study on probabilistic reliability in Byzantine cloud computing environments. To this end, they propose to use reputation-based replication to mitigate Byzantine behaviors and its impact on the correctness of the computation. Each node has a given probability p of behaving in a Byzantine manner and, thus, a reputation $r = 1 - p$. The latter can dynamically change based on the history of good answers of tasks that run on the node. Furthermore, the answer given by the application must satisfy a given degree of correctness (probability threshold). Instead of fixing f (number of Byzantine nodes) like in the traditional solutions that tolerate this kind of failures, computation tasks are dynamically replicated over a minimal number of compute nodes until meeting the probabilistic thresholds of correctness.

3.2 Heartbeat Arrival Estimation Strategies

Timer-based failure detectors use time bounds (timeout) to wait for messages sent over the network. In order to both minimize false suspicions and not degrade quality of service (QoS), failure detectors dynamically adjust the timeout value, based on observed communication delays of the past heartbeat history. In this section, we review different estimation approaches that have been proposed in the literature to dynamically predict heartbeat arrivals.

Chen: Seeking to reduce both the number of false suspicions and the time needed to detect a failure, CHEN; TOUEG; AGUILERA (2002) propose a method for estimating the arrival of the next heartbeat which is based on the history of the arrival time of heartbeats and includes a safety margin (β). The timer is then set according to this estimation.

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

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

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

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

Bertier et al: In (BERTIER et al., 2003), the authors introduced a failure detector suitable for LAN environments. Their heartbeat arrival estimation approach combines of Chen's estimation with a dynamic estimation based on Jacobson's estimation (JACOBSON, 1988). 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 their measurements on a LAN.

ϕ **Accrual**: The ϕ Accrual failure detector (HAYASHIBARA et al., 2004) is based on inter-arrival estimation time which follows a normal distribution. The Accrual FD dynamically adapts current network conditions based on the suspicion level. The overall mechanism is described in Figure 3.2. The heartbeats arrival from the network and their arrival time are stored in a sampling window and past samples are used to estimate some arrival distribution. Thus, the time of the last arrival T_{last}, the current time t_{now} and the estimated distribution are used to compute the current value of ϕ. Similarly to the above discussed FDs (BERTIER et al., 2003) and (CHEN; TOUEG; AGUILERA, 2002), 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. Based on such information, it is possible to compute a value φ with a scale that changes dynamically to match recent network conditions.

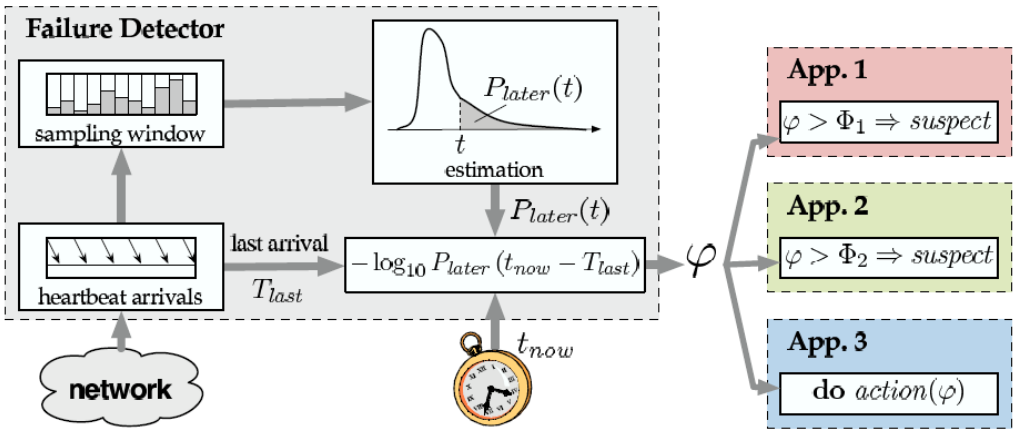

Figure 3.2: Information flow in the implementation of the Accrual failure detector (HAYASHIBARA et al., 2004)

Adaptive Accrual: In (SATZGER et al., 2007), the authors extended the Accrual FD by exploiting the 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 will arrive from a given process, i.e., it has failed.

ANNFD: The ANNFD presented in (ARAÚJO MACÊDO; LIMA, 2004) is a failure detector based on artificial neural networks. It 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 used to define the freshness point.

AFD: By observing the changes in the computing environment and exploiting both the feedback control theory and user-defined QoS constraints, the autonomic failure detector (AFD) proposed in (SÁ; ARAÚJO MACÊDO, 2010) dynamically configures the monitoring period and detection timeout value. AFD provides a crash detection service using pull monitoring style. It calculates the monitoring period and detection timeout based on the previously defined QoS (T_D, T_M, T_{MR}). A new metric, denoted failure detector availability (AV = $(T_{MR} - T_M)/T_{MR}$), is also defined. The AV is used to suggest a safety margin (α) in such a way to decrease failure detector mistakes and to

achieve the desired detection availability. If the detection service is inaccurate (i.e., AV is low), then the safety margin is increased to improve detection accuracy; otherwise, if AV is high, then α is decreased to improve the detection speed.

Furthermore, several works aim at improving the QoS of failure detectors which estimate the arrival time of the next heartbeat by varying some parameters such as window size, threshold, or safety margin ((TOMSIC et al., 2015), (XIONG et al., 2012), (NUNES; JANSCH-PORTO, 2004), (YANG et al., 2014)). In addition, there are studies that address the reduction of the number of sent heartbeats in the context of multiple concurrent applications with distinct QoS needs. One approach with this focus is presented in (TURCHETTI et al., 2016), which considers that different processes may require different heartbeat intervals to satisfy their respective QoS. To handle it, the authors propose the IFDS (Internet Failure Detector Service) that computes adaptive timeout intervals using two strategies: the first one, called η_{max}, seeks to maximize monitoring parameters to encompass the needs of all processes; the second one, called η_{GCD}, computes the greatest common divisor of the corresponding parameters.

3.3 Additional Assumptions for Asynchronous Systems

Several failure detector algorithms proposed in the literature rely on asynchronous distributed systems enriched with additional network behavior assumptions or properties. We present in this section some of them which are based on communication synchrony or message pattern.

Winning responses: The message pattern approach does not assume eventual bounds on process and communication delays. In (MOSTEFAOUI; MOURGAYA; RAYNAL, 2003), the authors consider that there is a correct process p and a set Q of f processes (with $p \notin Q$, moreover, Q can contain crashed processes) such that, each time a process $q \in Q$ broadcasts a query, it receives a response from p among the first $(n-f)$ corresponding responses (such a response is called a $winning$ response). Note that this assumption does not prevent message delays from always increasing without bound. This approach has been applied to the construction of a leader protocol.

$\diamond$**f-source**: Aguilera et al. introduce the $\diamond f-source$ assumption in (AGUILERA et al., 2004) aiming at providing communication-efficient leader and consensus protocol im-

plementations. In a system with n nodes and up to f process can crash, a $\diamondsuit f-source$ node p is a correct node with f outgoing links that are eventually timely, i.e., there exist t_0 and a bound δ, such that any message sent by p after t_0 on one of these links is received at most δ units of time after it has been sent.

$\diamondsuit$**f-accessibility**: In MALKHI; OPREA; ZHOU (2005), the authors define the concept of eventual $\diamondsuit f-accessibility$. A process p is eventual $\diamondsuit f-accessible$ if there is a time t_0 such that, at any time $t \geq t_0$, there is a set $Q(t)$ of f processes such that $p \notin Q(t)$ and a message broadcast by p at t receives a response from each process of $Q(t)$ by time $t+\delta$ (where δ is a bound known by the processes). This approach requires a majority of correct processes. Its interest lies in the fact that the set Q of processes whose responses have to be received in a timely manner is not fixed and can be different at distinct times. The paper also presents a protocol building Ω when there is a process that is $\diamondsuit f-accessible$ forever, and all other links are fair-lossy. The authors state that the assumptions are strongly motivated by practical needs, particularly those of the Paxos protocol.

Responsiveness property: In (ARANTES et al., 2013) and (GREVE et al., 2012), the authors propose a model to implement unreliable FDs in dynamic networks with suitable assumptions for such a scenario. The message pattern model establishes conditions on the logical time the messages are delivered by processes. They present a stabilized responsiveness property (SRP). The property states that there exists a time t after which all nodes of p_i's neighborhood receive, to every of their queries, a response from p_i which is always among the α_j responses to the query. That is, it denotes the ability of a node to reply to a query among the first nodes. Similarly to the winning channel approach, the response of p_i is always a winning response.

3.4 Discussion

Even if some of the above-cited works of group (1) are not failure detectors, they present interesting characteristics related to failure handling that the Impact FD also exploit, i.e., we selected these works taking into account potential similarity to the Impact FD features such as: (a) the non-boolean (suspected/trusted) output; (b) the grouping of processes in sets; (c) assignment of weight to processes to express their

relevance; (d) margin of failures. Note that, among them, there is no work that addresses all the Impact FD features and none of them tackle with subsets of processes associated with threshold values. Additionally, even if some of the works (VÉRON et al., 2015) (ARANTES et al., 2015) consider a reputation approach that provides information about each node reliability, they do not take into account the relevance of processes and are not easily configurable to the requirements of the system. Finally, only Accrual FD provides as output a suspicion level, however, related to each process and not to the system as a whole.

We should point out that some of these works such as (COSQUER; RODRIGUES; VERÍSSIMO, 1995) and (YIM; CHOI, 2010) were defined to specific systems or services. In (COSQUER; RODRIGUES; VERÍSSIMO, 1995), the authors put forward a failure detection mechanism based on application requirements of group membership service or view synchronous communication. In (YIM; CHOI, 2010), the concept of confidence level is applied to manage the status of sensors in wireless sensor networks. The other works target large scale distributed systems such as Clouds. For instance, Pigeon (GUPTA et al., 2013) addresses the question of how applications should be notified of failures and introduces an approach that encapsulates the degree of certainty in the failure report. In this way, the applications can use this information to take the most appropriate action needed for the failure handling, similarly to the Impact FD.

The probability threshold concept based on the degree of confidence in the answers, proposed in (BRUN et al., 2011) and (ARANTES et al., 2015), is in some extent close to the Impact FD threshold since it reflects some margin of failure. On the other hand, the threshold parameter of the Impact FD provides more flexibility in the number of failures because it considers the impact factor of processes and subsets.

It is worth remarking that, like the survivor sets (JUNQUEIRA et al., 2010), the Impact FD can be implemented without depending on the maximum number f of failures (see Section 5.3).

The works of group (2) present different approaches to estimate the timeout. As discussed in Section 2.3.5, the QoS of failure detectors depends on the choice of heartbeat arrival estimation strategy: a short timeout leads a FD to detect failures quickly, but may increase the number of false suspicions decreasing, consequently, its accuracy. This thesis proposes a new unreliable failure detector and its focus is not in heart-

beat arrival estimation strategies. However, implementations of Impact FD may use different approaches to estimate the timeout. In the case of the timer-based Impact FD implementation of Section 5.2 (Algorithm 13), we use the heartbeat arrival estimation proposed by CHEN; TOUEG; AGUILERA (2002). Note that to use another one, it is just necessary to change the code of the function *Timeout ()* (Algorithm 8) called by Algorithm 13. For the Chen's estimation algorithm, we consider the safety margin suggested by the authors, adding a dynamic increment for eventual timely links. The reason for the Chen et al's algorithm choice is that, although the estimation solutions proposed by the latter and Accrual FDs (HAYASHIBARA et al., 2004) (SATZGER et al., 2007) have similar performance (mistake rate X detection time) over a wide-area network (environment of our experiments), the Accrual FD estimation requires tuning of the threshold parameter for each process and depends on application characteristics. It is important also to point out that Bertier, AFD, and ANNFD estimations were designed to local area networks where messages are rarely lost.

The aim of the works of group (3) is to circumvent the impossibility of solving some distributed problems in pure asynchronous systems in the presence of failures (for instance, the consensus) by adding behavior properties on the underling system. Thus, the tailored assumptions, when satisfied, allow the implementation of the proposal solution in the target system. Similarly, we are interested in implementing the Impact FD on asynchronous systems enriched with additional assumptions. In Chapter 5, we propose algorithms for the Impact FD, considering different type of systems and behavior properties: $PR(IT)^{S}_{p}$, $PR(\diamondsuit IT)^{S}_{p}$, $PS{-}accessibility$, and $\diamondsuit PS{-}accessibility$.

4 IMPACT FD

In this chapter we describe the Impact Failure Detector. First, we present the definitions related to the Impact FD: Impact Factor, Subsets, Trust Level and Margin of Failures. After that, we define 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. In Section 4.3 we introduce the concept that a process can be $PS{-}accessible$ (or $\diamondsuit PS{-}accessible$). Section 4.4 we propose some properties and classes of FD that exploit the Impact FD concept. 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.

4.1 Definitions

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 crashing 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 $F : T \rightarrow 2^{\Pi}$, where $F(t)$ is the set of processes that have failed before or at time t. The function $correct(F)$ denotes the set of correct processes, i.e., those that have never belonged to a failure pattern (F), while function $faulty(F)$ 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 $faulty(F_S) = faulty(F) \cap S$.

4.1.1 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 ($S = \{S_1, S_2, ...S_m\}$). 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 $S^* = \{S_1^*, S_2^*, ...S_m^*\}$ be the set S partitioned into m disjoint subsets where each S_i^* is a set which each element is a tuple of the form $\langle id, I \rangle$, where id is a process identifier and I is the value of the impact factor of the process in question.

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

4.1.2 Trust Level

We denote ${trusted_p}^S(t)$ the set of 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 factors of the elements $\langle id_q, I_q \rangle$ of S_i^* such that $id_q \in trusted$.

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

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 $trust\_level_i = 0$.

4.1.3 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^*.

4.1.4 Examples

Table 4.1 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.

Table 4.1: 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	$\{\{\langle q_1,1\rangle,\langle q_2,1\rangle,\langle q_3,1\rangle,\langle q_4,5\rangle,\langle q_5,5\rangle\}\}$	$\{7\}$

In Table 4.2, 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 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 4.2: 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

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

4.2 Flexibility of the Impact FD

The *flexibility* of the Impact FD characterizes its capability of 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 [1] 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:

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

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

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

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 now define 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.

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

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

Eventual Impact Threshold Property - $PR(\diamondsuit IT)_p^S$: 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.

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

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

4.3 $PS{-}accessibility$ Assumptions

Inspired by the concept that a process is $\diamondsuit f{-}accessible$ proposed in MALKHI; OPREA; ZHOU (2005) (see Section 3.3), we also define the concept of a $PS{-}accessible$ and a $\diamondsuit PS{-}accessible$ process:

A process $p \in correct(F)$ is $PS{-}accessible$ (resp., $\diamondsuit PS{-}accessible$) if every $QUERY$ message broadcast by p obtains from the beginning (resp., eventually) a set Q of responses that satisfy the ${threshold}^{S^*}$.

Thus, if p is $PS{-}accessible$ (resp., $\diamondsuit PS{-}accessible$), $PR(IT)_p^S$ (resp., $PR(\diamondsuit IT)_p^S$) holds for p.

We have then the following definitions for the message pattern and timer-based approaches:

Message-pattern approach: Given a $QUERY$ issued by p, the set of the first $RESP$ messages received by p to this query are denoted *winning* responses (MOSTEFAOUI; MOURGAYA; RAYNAL, 2003), (RAYNAL, 2007).

A process $p \in correct(F)$ is $PS{-}accessible$ (resp., $\diamondsuit PS{-}accessible$) if for $\tau_0 = 0$ (resp., $\exists \tau_0$) $\forall \tau \geq \tau_0$, there exists a set $Q(\tau)$ of processes such that $Q(\tau) \in PS$ and $p \notin Q(\tau)$ and a $QUERY$ message broadcast by p at τ receives $RESP$ messages from processes of $Q(\tau)$ and these responses are always (resp., eventually always) *winning* responses.

Timer-based approach: A process $p \in correct(F)$ is $PS{-}accessible$ (resp., $\diamondsuit PS{-}accessible$) if for $\tau_0 = 0$ (resp., $\exists \tau_0$) $\forall \tau \geq \tau_0$, there exists a set Q of processes such that $Q(\tau) \in PS$ and $p \notin Q(\tau)$ and a $QUERY$ message broadcast by p at τ receives a $RESP$ message from each process of $Q(\tau)$ by time $\tau + \delta$.

We should point out that in both definitions of $PS-accessible$ and $\Diamond PS-accessible$, the set $Q(\tau)$ is not fixed and can be different at distinct instants of time which is in accordance with the *flexibility* property of the Impact FD.

4.4 Classes of Impact FD

Similarly to the *completeness* and *accuracy* properties defined in (CHANDRA; TOUEG, 1996) (see Section 2.3.1), we define the following properties for the Impact FD:
$Impact\ completeness_p{}^S$: For a failure detector of a correct process p, there is a time after which p does not trust any crashed process of S;

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

$Impact\ weak\ completeness_p{}^S$: For a failure detector of a correct process p, there is a time after which some p does not trust any crashed process of S;

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

$Eventual\ impact\ strong\ accuracy_p{}^S$: 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 is trusted by every correct process.

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

Lets consider that p in S and $S = \Pi$

We can then define some classes of Impact FD, similarly those defined in (CHANDRA; TOUEG, 1996) and (DELPORTE-GALLET et al., 2004):

- $\Diamond IP$ (*Eventually 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;
- $\Diamond IS$ (*Eventually 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;

- $\diamondsuit IW$ (*Eventually Weak Impact Class*): For $S = \Pi$, $\exists p \in correct(F)$ such that $impact\ weak\ completeness_p{}^S$ property 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 $\exists t_1 \in T, \forall t_2 \in T \geq t_1, \forall p \in correct(F), trusted_p{}^S(t_2) = \{l\}$ (*Impact leadership property*).
- $I\Sigma$: (*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.

4.5 Impact FD Equivalences

As we have seen in Section 2.3.3, failure detectors can be compared with each other through the notions of reducibility and equivalence. In (CHANDRA; HADZILACOS; TOUEG, 1996), 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 (CHU, 1998), 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 (MOSTÉFAOUI et al., 2007) 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 (CHU, 1998), requires each message to carry only one process identifier.

The most important results presented in those works is the fact that the classes Ω and $\diamondsuit W$ are equivalent.

By considering that all processes of $S = \Pi$ have impact value equals to its identifier 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 exists a majority of correct processes, and $\diamondsuit IW^U$ FD is equivalent to the Omega FD (Ω), provided that the membership of the system is known. Both Σ and Ω FDs were defined in Section 2.3.2 while $I\Sigma^U$, $I\Omega^U$, $\diamondsuit IP^U$, and $\diamondsuit IW^U$ FDs are 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 having failed):

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

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

The set of processes trusted by p at t corresponds to those $trust\_level_i$ $(1 \leq i \leq n)$ 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^*.

4.5.1 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
```

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. □

4.5.2 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).

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
```

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 9).

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. □

4.5.3 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 exists 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.

4.5.4 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 (JIMÉNEZ; ARÉVALO; FERNáNDEZ, 2006):

- 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 (CHANDRA; HADZILACOS; TOUEG, 1996), (MOSTÉFAOUI et al., 2007), (CHU, 1998), 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).

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

```
1:  Begin
    Input
2:      Π
    Init
3:      S* = ∅;
4:      for i = 1 to |Π| do
5:          S* = S* ∪ {{⟨q_i, i⟩}}
6:      end for
    Task T1
7:      Upon invocation of ◇W do
8:          trust_level = ◇IW^U(S*)
9:          suspected = Π − Trust_levelToProcs(trust_level)
10:         return suspected
11:     end
12: End
```

Sketch of Proof

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

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,

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

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

$\exists p \in correct(F), \forall q \in faulty(F), : \exists t \in T, \forall t' \in T \geq t, q \in suspect$, 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. □

5 IMPLEMENTATIONS OF IMPACT FD

The Impact FD can have different implementations in accordance with the characteristics of the system: the synchronization model, whether or not the process p has knowledge about the composition of S (membership) and the type of nodes.

CHANDRA; TOUEG (1996) state that the failure detector abstraction is a clean extension to the asynchronous model of computation that allows us to solve many problems that are otherwise unsolvable. It is important to point out that an asynchronous system is characterized by the absence of bounds on process speed and on message delay. It is impossible to distinguish with certainty a crashed process from a very slow process in a purely asynchronous distributed system (RAYNAL, 2007). In this context, we are interested in providing distributed algorithms which implement the Impact FD for asynchronous system models and enriching them with assumptions about synchrony of the links or the relative speed between them.

In this chapter we present three implementations of the Impact FD:

- 1) a *timer-based* implementation that can be applied to systems where either all links are *lossy asynchronous* or some or all links are $\diamond-timely$ while the others are *lossy asynchronous*;
- 2) an implementation based on a time-free message pattern approach which waits for responses from α processes or from a set Q of processes whose responses satisfy the $threshold^S$;
- 3) an implementation based on query-response message rounds that exploits the flexibility property and considers a set of bounded timely responses.

The three algorithms have S^* as input. Thus, they know S, the impact factor of all processes of S, the number of subgroups m, and how processes are grouped.

In the section 5.2 we present the implementation (1) of a *timer-based* distributed algorithm (and its proof of correctness) that uses the algorithm proposed by CHEN; TOUEG; AGUILERA (2002) 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 ($\diamondsuit-timely$) (AGUILERA et al., 2004).

Section 5.3 presents two different algorithms (implementations (2) and (3)) and their respective proofs of correctness, based on query-response message rounds, that implement the Impact FD. The implementations were tailored to satisfy the Impact FD's flexibility. In both implementations, if the process that monitors S is $PS-accessible$ or $\diamondsuit PS-accessible$, at every query round, it only waits (or eventually only waits) for a set of responses that satisfy the *threshold*.

5.1 Definitions

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.

Type of systems: For the current implementation, we consider that links are directed (either unidirectional or bidirectional) and there exists a link between p and $\forall q \in S$. Concerning loss property link synchrony, we consider the types of links as defined in section 2.2.3.

We then define the following types of system:

- *AS*: a system where all links are *lossy asynchronous*;
- *F-AS*: a system where all links are a *fair lossy asynchronous*;
- *R-AS*: a system with bidirectional *reliable* links;
- *R-W-δ*: a system which is a *R-AS* system such that there exists a known upper bound δ on the round-trip delay of messages, but it might not hold on all pairs

of processes at all times;

- *W-ET* (*weak eventually timely* system): a system where some links are $\diamondsuit-timely$ and the others are *lossy asynchronous* ;
- *S-ET* (*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 (AGUILERA et al., 2003)).

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*).

Figure 5.1 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 $\diamondsuit - source$.

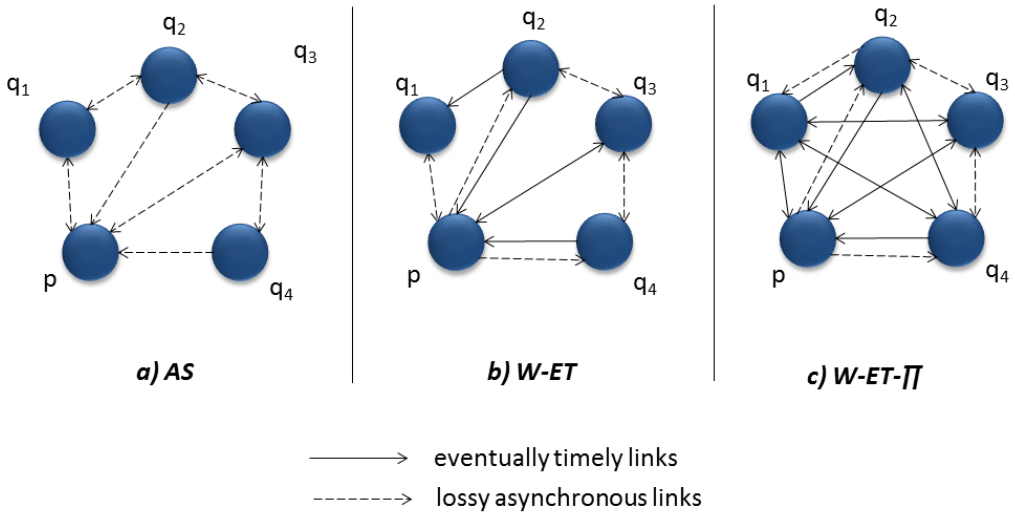

Figure 5.1: Examples of system types.

5.2 Asynchronous System with Additional Assumptions

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

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; TOUEG; AGUILERA (2002), denoted Chen's algorithm in this work, which computes the timeout value for waiting for a heartbeat message from each monitored process (see section 3.1).

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.

In Algorithm 9, Chen's algorithm is executed by the $Timeout$ function (Algorithm 8) which calculates the arrival estimation of the next heartbeat for process q. Furthermore, if the link is $\diamondsuit-timely$, 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 $\diamondsuit-timely$ 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 Timeout Function

```
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 time-out 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*).

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 the 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* (lines 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.

5.2.1 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 already 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 $\diamond-timely$, 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 $\diamond IS$.*

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 $\diamond-timely$ link. Thus, ∀q∈ $correct(F)$, 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 $\diamond IS$.

□

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

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

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

5.3 Implementation Under $PS{-}accessibility$ Assumptions

In this section we present two different implementations of the Impact FD: the first one is based on the message-pattern approach and the second one on the timer-based approach. Both of them use query-response message rounds and were conceived to exploit the flexibility capacity of the Impact FD.

In chapter 4 we introduced the concept that if process p is $PS{-}accessible$ (or $\Diamond PS{-}accessible$) the system S will always (or eventually always) be trusted by p, as well as two properties, $PR(IT)_p^S$ and $PR(\Diamond IT)_p^S$, that characterize the necessary stability condition of S that ensures confidence (or eventual confidence) on it.

A correct process p is $PS{-}accessible$ (resp., $\Diamond PS{-}accessible$) if every query broadcast by p obtains from the beginning (resp., eventually) a set Q of responses that satisfy the degree of confidence in S, i.e., the trust level of S is greater or equal than the $threshold$ value. In the case of the message pattern approach, this property implies that a query broadcast by p receives responses from a set of processes Q whose sum of impact factors satisfy the $threshold$ and these responses are always (resp., eventually always) *winning* responses, i.e., arrive before the other responses. In the case of the timer-based approach, there exists a set Q of processes whose sum of impact factors satisfies the $threshold$ and a query broadcast by p always (or eventually always) receives timely (i.e., within a known bounded delay) responses from each process of Q.

Note that in both implementations the set Q of processes is not fixed, i.e., can change at each round, which is in accordance with the *flexibility property* of the Impact FD.

We consider that the monitor process $p \in correct(F)$ and $p \notin S$. It repeatedly issues queries by calling the primitive *broadcast(m)* which sends a copy of the $QUERY$ message over every link from p to q, $\forall q \in S$. The time interval between two consecutive rounds of $QUERY$ messages is finite and arbitrary (resp., bounded) for the message pattern (resp., timer-based) implementation. The reception of the *QUERY* message is handled, for both implementations, by Algorithm 11 (Task T1), which is executed by every process $q \in S$. Upon the reception of *(QUERY,* r_p*)* message, where r_p is the round identifier (line 2), q responds to p with a *RESP* message, identified by the same round value r_p (line 3).

Algorithm 11 Impact FD Algorithm for $q \in S$

```
1: Begin
   Task T1
2:     Upon reception of (QUERY, r_p) from p do
3:         send(RESP, r_p) to p
4:     end
5: End
```

Both algorithms (message-pattern and timer-based) use the following variables:

- *trusted,* $tmp\_trusted$: sets that keep those processes considered not faulty by the monitor process p;
- *PS*: set composed of all possible subsets of processes, whose sum of their impact factor values are equal or greater than $threshold_p^{S^*}$.

5.3.1 Message-pattern Implementation

Algorithm 12 presents the message pattern approach implementation of the Impact FD of process p with respect to S in a *R-AS* system (See Section 5.1).

Process p receives S^*, the *threshold* value of each subset of S^* (the set $threshold^{S^*}$), and the maximum number of messages to wait (α). The latter is a set $\alpha = \{\alpha_1, \ldots, \alpha_m\}$, where each α_i corresponds to a bound value on the number of messages to wait from the processes of subset S_i^*. For instance, if f_i denotes the maximum number of failures of processes of subset S_i^*, $\alpha_i \leq |S_i^*| - f_i$ (for $i = 1$ *to* m).

Algorithm 12 Message pattern implementation for p

1: **Begin**
Input
2: S^*, $threshold^{S^*}$, α
Init
3: $r_p = 0$
4: $trusted = \emptyset$
5: $tmp\_trusted = \emptyset$
6: **for** $i = 1$ *to* $|S^*|$ **do**
7: $trusted = trusted \cup \{q_i\}$
8: **end for**
9: $PS = TPowerSet(S^*, threshold^{S^*})$

Task T1
10: **loop**
11: $broadcast(QUERY, r_p)$
12: $wait\ until\ (|tmp\_trusted_i| \geq \alpha_i\ \forall i \in [1,m])\ or\ (tmp\_trusted \in PS)$
13: $trusted = tmp\_trusted$
14: $tmp\_trusted = \emptyset$
15: $r_p = r_p + 1$
16: **end loop**

Task T2
17: **Upon reception of** $(RESP, r_q)$ from q **do**
18: **if** $r_q = r_p$ **then**
19: $tmp\_trusted \cup \{q\}$
20: **end if**
21: **end**

Task T3
22: **Upon invocation of** $Impact()$ **do**
23: **return** $Trust\_level(trusted, S^*)$
24: **end**

25: **End**

The algorithm has three tasks. At the initialization, $trusted$ is initialized with the processes of S and both r_p and $tmp\_trusted$ are reset. Then, function $TPowerSet$ is carried out to generate the set PS which contains all possible subsets formed by processes of S that satisfy the $threshold^{S^*}$. The variable $tmp\_trusted$, at every query round, gathers the identifier and impact factor of those processes that answered to the current query.

Task T1 of p is composed by an infinite loop. First, p sends message *(QUERY,* r_p*)* to all processes of S (line 11). Then, at each round (r_p), p waits for at least α_i responses ($1 \leq i \leq m$) or until $tmp\_trusted$ is a subset of PS (i.e., contains processes whose sum of impact factor values satisfy the $threshold^{S*}$) (line 12). Finally, the round counter (r_p) is incremented (line 15).

Task T2 is responsible for the reception of messages *(RESP,* r_q*)* sent by a process q of S. If the round r_q value of the *RESP* message is equal to r_p, then q is added to the $tmp\_trusted_q$ set.

Task T3 handles the invocation of the *Impact()* function (line 22), which computes and return the trust level related to the trusted processes (line 23).

5.3.1.1 Sketch of Proof

Lemma 11. *Process p never blocks forever in a query-response round.*

Proof. The only point that p could block forever would be in the *wait* statement of Task T1 (line 12).

Let's consider round r_p and that the system is blocked in the *wait* statement. Let's also suppose that the system is trusted in round r and that the set $tmp\_trusted$ by Task T2 within round r_p is also included in PS. In this case, the second condition of the *wait* becomes true and T_1 will not block. Let's now suppose that p is blocked on the *wait* statement and that the second condition does not hold, i.e., p will not be unblocked because of it. However, for every subset S_i^*, p waits for α_i messages ($1 \leq i \leq m$), where f_i is the maximum number of processes of S_i^* that can fail and $\alpha_i \leq |S_i^*| - f_i$. Therefore, as the channels are reliable and, even if f_i nodes of each S_i^* have failed, p will receive α_i responses ($1 \leq i \leq m$) which will render the first condition true, and p will be unblocked. In other words, since α_i is bounded and no query or response messages are lost, such a condition always ensures the progress of the failure detector. □

Theorem 11. *Algorithm 12 satisfies the strong completeness property.*

Proof. From Lemma 11, task T1 never blocks. Let's consider a process q that crashes. Then, after some time t after the crash of q, all the $RESP$ messages sent by q before it crashed have been received or discarded by p. Thus, after t, p will no more receive any $RESP$ message from q and, therefore, q will never be included again in $tmp\_trusted$ after the latter was reset in a round at or after t (line 14). Consequently, there exists a time after t which q will permanently not in $tmp\_trusted$ and, consequently, not in $trusted$, i.e., q is permanently suspected. Since the same reasoning can be applied to all faulty processes, every process that crashed eventually permanently will not belong to $trusted$ and, thus, strong completeness is satisfied. □

Lemma 12. *At every query r_p issued by p, $trusted$ is updated with the identifier of winning processes which respond to query r_p.*

Proof. From Lemma 11, task T1 never blocks and, thus, line 13 is always executed. This line is the only point where $trusted$ is updated with $tmp\_trusted$ after initialization (line 7). Furthermore, $tmp\_trusted$ is set to empty at every query (line 14) and only processes whose response concerns query r_p are added to $tmp\_trusted$ at round r_p (lines 18 - 19) at task T2. Hence, at every query round r_p, $trusted$ is updated with the identifiers of winning processes which respond to query r_p. □

Lemma 13. *If p is $PS{-}accessible$ (resp., $\Diamond PS{-}accessible$), Algorithm 12 ensures that $PR(IT)_p^S$ (resp., $PR(\Diamond IT)_p^S$) holds for p.*

Proof. For $t = 0$, $trusted = S$. Let Q(r) be the set of winning responses at round r and r_0 be the first round such that $trusted = tmp\_trusted \in PS$. Let then consider that $\forall r \geq r_0$, for every $Q(r)$, $trusted \in PS$ which characterize the $PS{-}accessibility$ of p. In this case, since from Lemma 12, $trusted$ is updated at every query round with the identifiers of processes of $Q(r)$, $trusted = tmp\_trusted \in PS$. Let t be the time when $trusted$ is updated in round r_0 (line 7 or line 13). Therefore, $\forall t' \geq t$, when the impact FD is invoked, $PR(IT)_p^S$ (resp., $PR(\Diamond IT)_p^S$) holds.

□

Theorem 12. *Let $trust\_level_p^{S^*}(t)$ be the output value returned by Algorithm 12 at t. If p is $PS{-}accessible$ (resp., $\Diamond PS{-}accessible$), $\forall t' \geq 0$ (resp. $\exists t \in T, \forall t' \geq t$), $trust\_level_p^{S^*}(t') \geq threshold^{S^*}$.*

Proof. The proof follows directly from Lemma 13. □

5.3.2 Timer-based Implementation

Algorithm 13 shows a timer-based implementation of the Impact FD of process p with respect to S in a *R-W-*δ system (See Section 5.1). The latter includes the assumption that there exists a known upper bound δ on the round-trip delay of messages, but it might not hold on all pairs of processes at all times. Processes are considered synchronous and process p issues a query periodically at every Δ units of time ($\Delta > \delta$).

Process p monitors the processes of S and receives as input the set S^*, the $threshold^{S^*}$, the interval time Δ to send the broadcast message, and the upper bound δ on the round-trip delay of messages for timely links.

If $p \in correct(F)$ is $PS{-}accessible$ (resp., $\Diamond PS{-}accessible$), it always (resp. eventually always) receives $RESP$ messages, in a delay of time smaller than δ, from a set Q of processes whose impact factors ensure that the respective $trusted \in PS$.

The algorithm has four tasks. At the initialization, $trusted$ is initialized with the processes of S. Then, the function $TPowerSet$ is carried out to generate the set PS which contains all possible subsets formed by processes of S that satisfy the $threshold^{S*}$ (line 7). The variables r_p and timer $timeout$ are also initialized.

At every round r_p, Task T1 of p reset $tmp\_trusted$ and increments the round counter (lines 10-11). Periodically (interval of Δ time units), process p sends to the processes in S a $QUERY$ message (line 12) and starts the timer $timeout$ (line 13).

In Task T2, when process p receives a $RESP$ message sent by q, if round r_q is equal to r_p and q is not in $tmp\_trusted$, q is added to $tmp\_trusted$ (line 15). At this point, if $tmp\_trusted$ is a subset of PS (i.e., contains processes whose sum of impact factor values satisfy the $threshold^{S^*}$) (line 18), than the variable $tmp\_trusted$ is assigned to $trusted$ and the $timeout$ is stopped.

Task T3 is the same of Algorithm 12. It handles the invocation of the *Impact()* function by p, which returns the sum of impact factor of the trusted processes (line 24).

Upon the expiration of the timer $timeout$ (Task T4), p assigns its current knowledge about trusted processes (i.e., $tmp\_trusted$) to $trusted$.

5.3.2.1 Sketch of Proof

Algorithm 13 Timer-based implementation for p

1: **Begin**
Input
2: S^*, $threshold^{S^*}$, Δ, δ
Init
3: $trusted = \emptyset$
4: **for** $i = 1$ to $|S^*|$ **do**
5: $trusted = trusted \cup \{q_i\}$
6: **end for**
7: $PS = TPowerSet(S^*, threshold^{S^*})$
8: $r_p = -1$
9: $timeout = \delta$

Task T1 - Repeat forever every Δ time unit
10: $tmp\_trusted = \emptyset$
11: $r_p = r_p + 1$
12: $broadcast(QUERY, r_p)$
13: $start\ timeout$

Task T2
14: **Upon reception of** $(RESP, r_q)$ from q **do**
15: **if** $r_q = r_p$ **then**
16: $tmp\_trusted \cup \{q\}$
17: **end if**
18: **if** $tmp\_trusted \in PS$ **then**
19: $Stop\ timeout$
20: $trusted = tmp\_trusted$
21: **end if**
22: **end**

Task T3
23: **Upon invocation of** $Impact()$ **do**
24: **return** $Trust\_level(trusted, S^*)$
25: **end**

Task T4 - When timeout expires
26: **Upon** $expiration\ of\ timer$ **do**
27: $trusted = tmp\_trusted$
28: **end**
29: **End**

Theorem 13. *Algorithm 13 satisfies the strong completeness.*

Proof. The proof is the same of Theorem 11. □

Lemma 14. *At every query r_p issued by p, $trusted$ is updated with the identifiers of the processes which respond to query r_p within at most δ units of time.*

Proof. At every new query r_p of task T1, p starts a timer (line 13). Furthermore, $tmp\_trusted$ is set to empty at every query and only processes whose response is timestamped with r_p is added to $tmp\_trusted$ at round r_p at task T2. As $\Delta > \delta$, a new query will not be issued before the timer expires or is stopped. If the set of Q $RESP$ messages received by this query are such that $Q \in PS$, the timer is stopped and $trusted = tmp\_trusted = Q$ by Task T2 (lines 19 - 20). Otherwise, the timer will expires (Task T4) and $trusted$ will be updated with $tmp\_trusted$ set which contains the identifiers of processes that sent $RESP$ messages related to query r_p (line 27) within a delay of δ.

□

Lemma 15. *If p is $PS-accessible$ (resp., $\Diamond PS-accessible$), Algorithm 13 ensures that $PR(IT)_p^S$ (resp., $PR(\Diamond IT)_p^S$) holds for p.*

Proof. From Lemma 14, $trusted$ is updated at every query. Let $t_0 \in T$. For every query $QUERY$ message broadcast by p at $t \geq t_0$, it receives a $RESP$ message from processes of a set Q of processes such that $trusted \in PS$ within $t + \delta$. In this case, the timer started in every query at line 13 will never expire and, therefore, Task T4 will be never executed $\forall t' \geq t$. On the other hand, since at t, $trusted \in PS$, the test of line 18 is always true. Hence $\forall t' \geq t$, when the impact FD is invoked $PR(IT)_p^S$ (resp., $PR(\Diamond IT)_p^S$) holds for p. □

Theorem 14. *Let $trust\_level_p^{S^*}(t)$ be the output value returned by Algorithm 13 at t. If p is $PS-accessible$ (resp., $\Diamond PS-accessible$), $\forall t' \geq 0$ (resp. $\exists t \in T, \forall t' \geq t$), $trust\_level_p{}^{S^*}(t') \geq threshold^{S^*}$.*

Proof. The proof follows directly from Lemma 15. □

6 PERFORMANCE EVALUATION

In this chapter, we first 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 CHEN; TOUEG; AGUILERA (2002) and we compare the results of Impact FD with an approach that monitors processes individually using Chen's FD (CHEN; TOUEG; AGUILERA, 2002). 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.

6.1 Environment

Our experiments are based on real trace files, collected from ten nodes of PlanetLab (PLANETLAB, 2014), as summarized in Table 6.1. 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 receiving of heartbeats by the processes.

Table 6.1: 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 6.2 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.

Table 6.2: 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

The implementation of the Impact FD used in our evaluation experiments is based on Algorithms 9 and 10, presented in Section 5.2. 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.

6.1.1 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 6.1 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 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.

***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

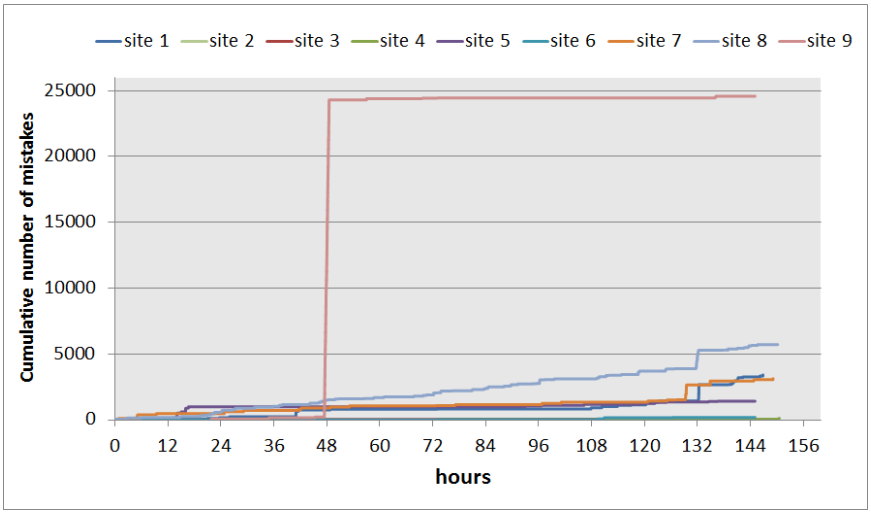

Figure 6.1: *AS* System: Cumulative number of mistakes of each site.

than the transmission delay limit for links with $\diamondsuit-timely$ 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 6.2 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.

6.1.2 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 at $site$ 0 related to heartbeat messages of $site$ 1 with two different values to β (100 and 400 ms). We chose

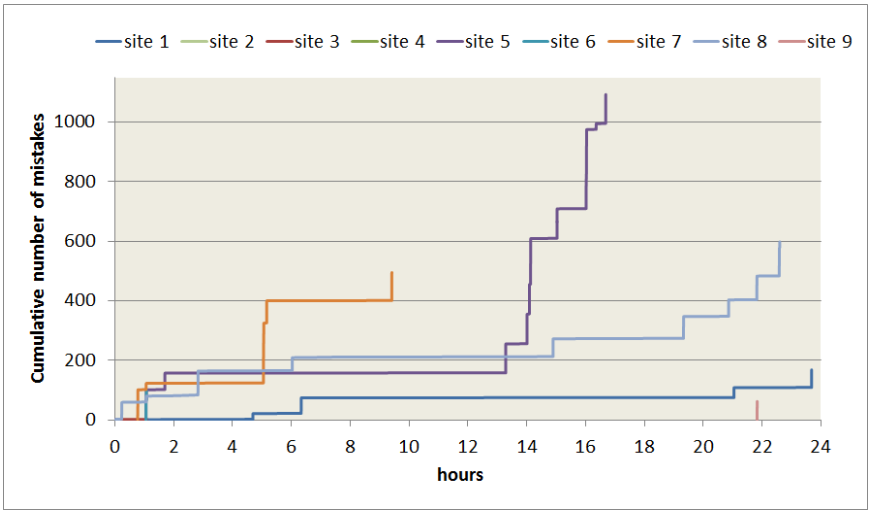

Figure 6.2: *W-ET* System: Cumulative number of mistakes of each site.

$site\,1$ because it has many periods of instability. We consider that $site\,1$ and $site\,0$ are alternately connected by *lossy asynchronous* or $\diamondsuit-timely$ links.

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 6.3 and 6.4 show the time difference between the arrival time of the previous heartbeat and the above three arrival ones (in ms): 1) the difference in milliseconds between the arrival time of the last heartbeat and the previous one; 2) the difference in milliseconds 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) the number of milliseconds elapsed 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 6.3 and 6.4 show the behavior of times when the timeout expires for $\beta = 100ms$ and $\beta = 400ms$ respectively till hour 24. In order to simplify at not overloading the figures, the points correspond only to the times where mistakes took place. Figure 6.4 has fewer points than Figure 6.3 because the number of mistakes drops considerably due to a higher β value.

Figure 6.3 summarizes the time differences for $\beta = 100ms$. The monitor $site\,0$ made 807 (resp., 592) mistakes when the link is *lossy asynchronous* (resp. $\diamondsuit-timely$).

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.

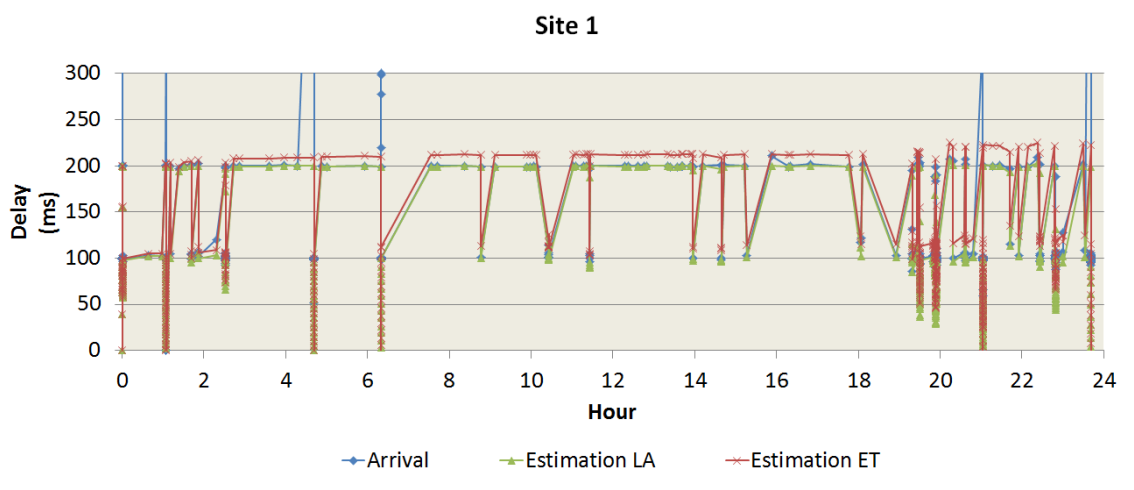

Figure 6.3: Behavior of the arrival times when timeout expires - $\beta = 100ms$, $\mu = 100$.

Contrarily to Figure 6.3, the number of mistakes drops to 168 and 166 mistakes, for *ET* and *LA* estimations respectively as shown in Figure 6.4. Therefore, since they are almost equal, the estimated arrival times for the *lossy asynchronous* and $\diamond-timely$ are also quite close. Similarly to Figure 6.3, the mistakes are concentrated in periods of great instability (1, 4, 6, 21, and 23).

6.2 QoS Metrics

First, 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 (CHEN;

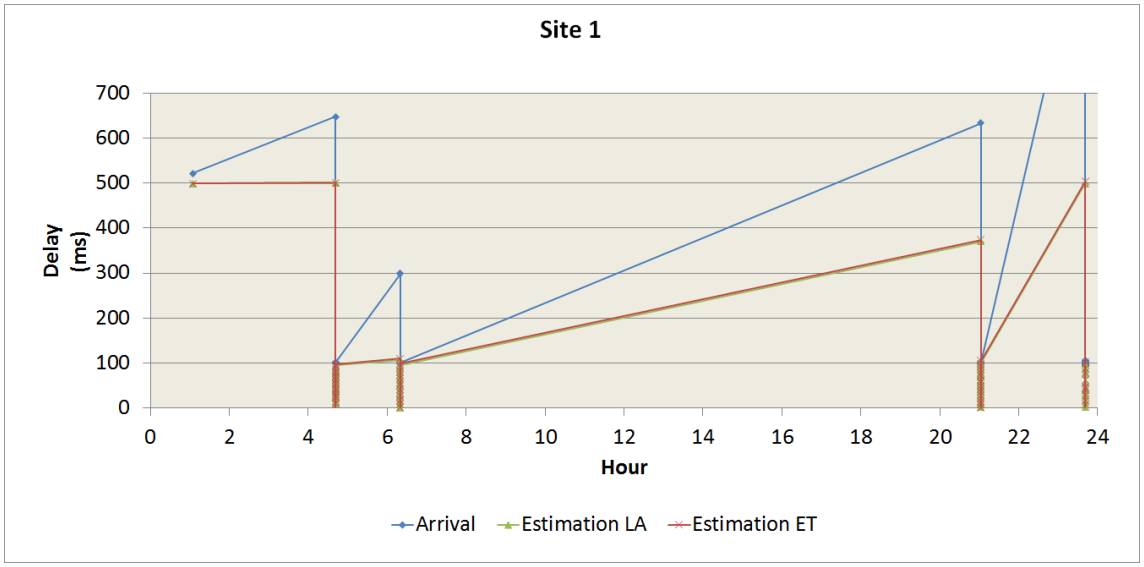

Figure 6.4: Behavior of the arrival times when timeout expires - $\beta = 400ms$, $\mu = 100$.

TOUEG; AGUILERA, 2002): *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.

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 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 (Hb_i) with respect to the previous freshness point, i.e., $\tau_{i+1} - Hb_i$, 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: $T_D = max(\tau_{i+1} - Hb_i),\ \forall q \in S_f$.

Figure 6.5 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 of

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 a heartbeat from q_3, S becomes *untrusted*.

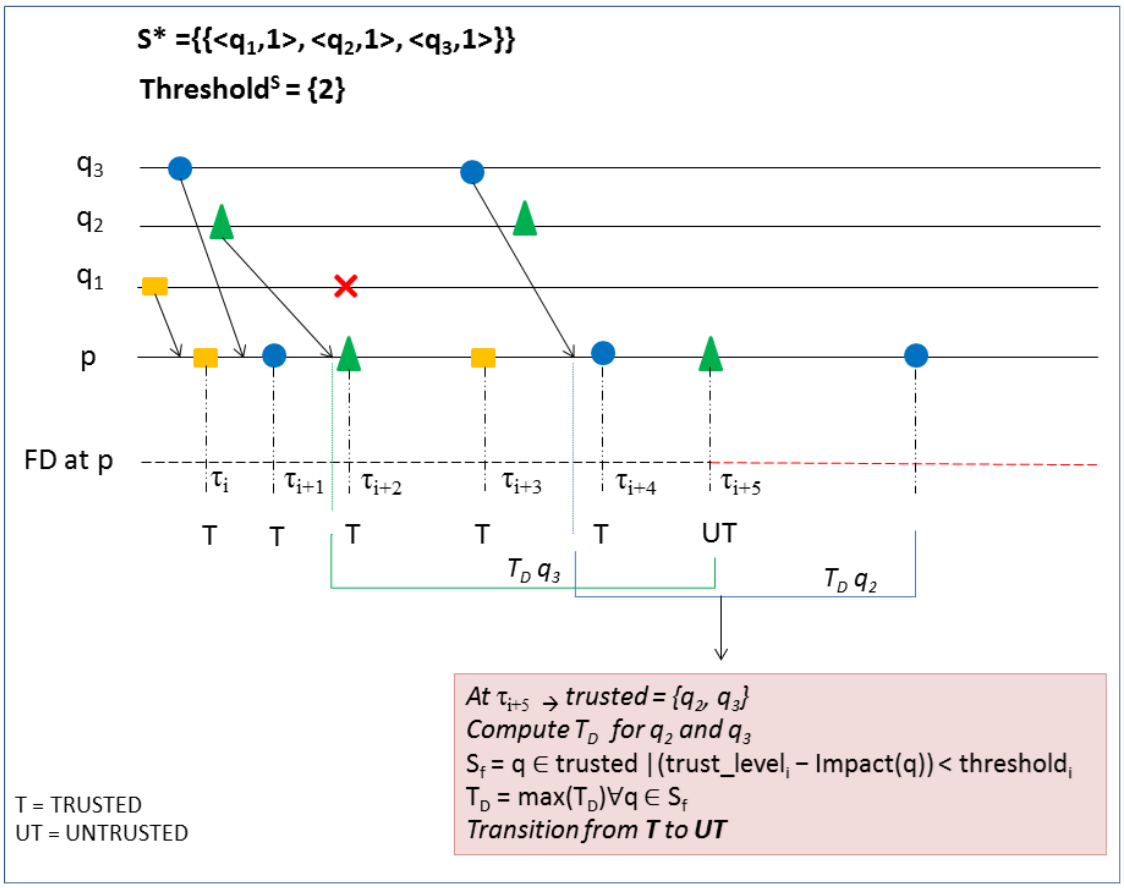

Figure 6.5: Transitions between “trusted” and “untrusted” states.

6.3 Asynchronous System (AS)

For this evaluation we consider an *AS*, i.e., links are lossy asynchronous. Table 6.3 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.

6.3.1 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 6.3). The safety margin was set to 400ms (β=400ms).

Figure 6.6 shows that in most cases the P_A decreases when the threshold increases.

Table 6.3: 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	$\{\{\langle q_1,10\rangle,\langle q_2,10\rangle,\langle q_3,10\rangle,\langle q_4,10\rangle,\langle q_5,10\rangle,\langle q_6,10\rangle,\langle q_7,10\rangle,\langle q_8,10\rangle,\langle q_9,10\rangle\}\}$

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 on the set of processes is ensured. Hence, the results confirm that when the threshold is lower, the Query Accuracy Probability is higher.

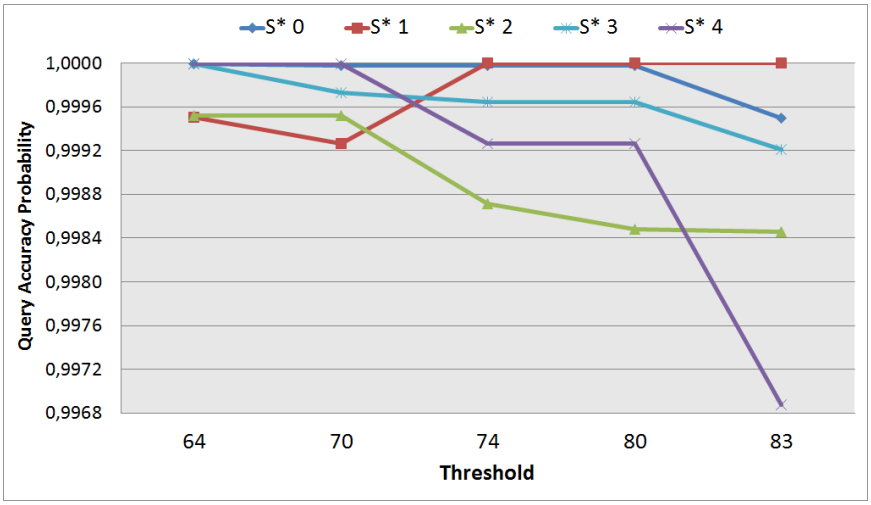

Figure 6.6: *AS* System: P_A vs. threshold with different set configurations (S^*).

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 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.

6.3.2 Experiment 2 - Query Accuracy Probability vs. 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 6.7 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, equal 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 (SATZGER et al., 2007).

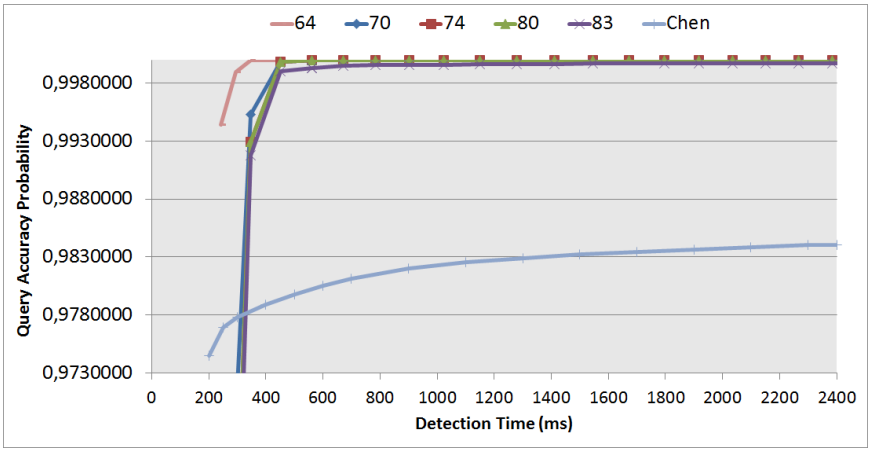

Figure 6.7: *AS* System: P_A vs. T_D with different thresholds.

6.3.3 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 6.8 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 detection time increases.

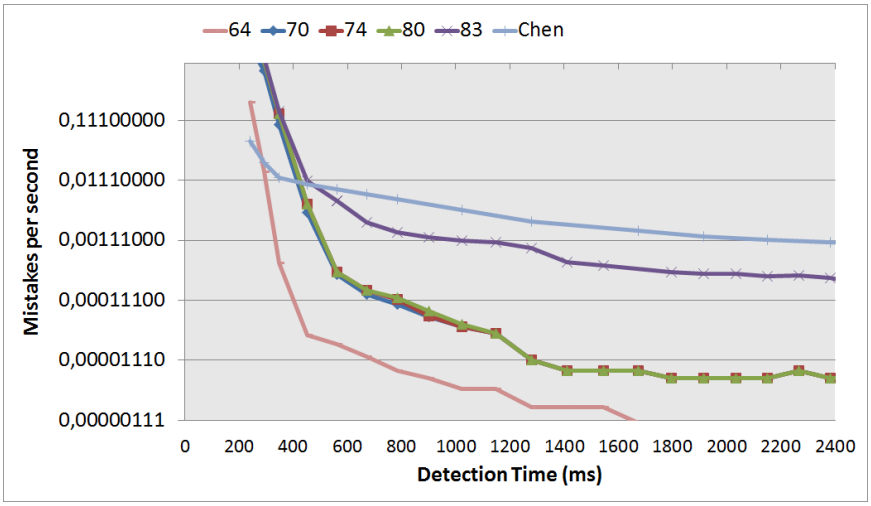

Figure 6.8: *AS* System: λ_R vs. T_D with different thresholds.

6.3.4 Experiment 4 - Cumulative Number of Mistakes

Figure 6.9 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.

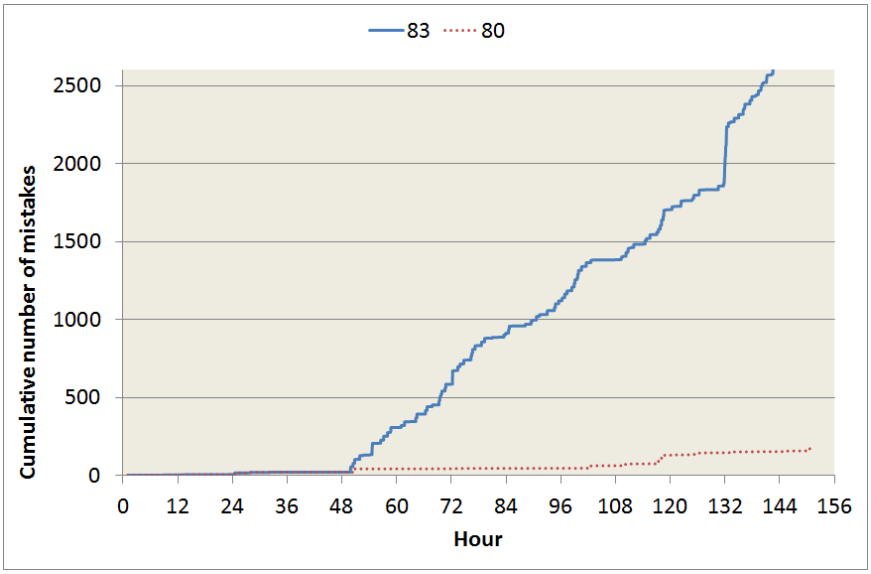

Figure 6.9: *AS* System: Cumulative number of mistakes for *"S* 0"* configuration.

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 on 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 6.1. 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 6.1).

6.3.5 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 6.10 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*.

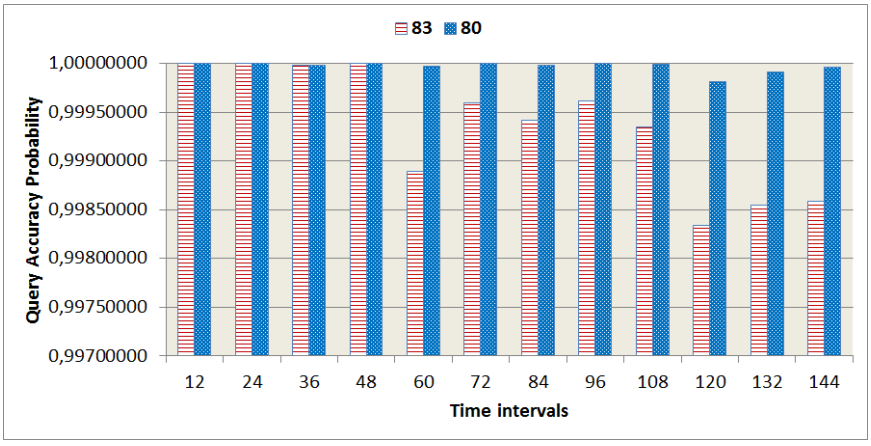

Figure 6.10: *AS* System: P_A vs. Time.

6.4 Weak $\diamondsuit-timely$ System (W-ET)

In this section, we consider the *W-ET* system described in Section 6.1.1: 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.

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 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^*.

6.4.1 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 6.4. 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 6.4: *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**	**MAX($\mathbf{T}_D$)** (ms)	$\mathbf{T}_D$ (ms)
1	152	0.0017	0.99992	43.36	64 (1h)	349341	312.9	234.5
10	324	0.0037	0.99983	43.69	64 (1h)	349341	263.0	182.0
100	383	0.0044	0.99979	45.18	64 (1h)	349341	256.6	173.9
AS	4689	0.0542	0.99849	27.70	1438 (24h)	7749909	300.0	151.7

Table 6.5 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*.

Table 6.5: *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**	**MAX($\mathbf{T}_D$)** (ms)	$\mathbf{T}_D$ (ms)
1	84	0.00097	0.99995	48.35	64 (1h)	349341	339.4	273.7
10	121	0.00140	0.99993	48.28	64 (1h)	349341	264.0	224.0
100	135	0.00156	0.99993	44.53	64 (1h)	349341	262.5	219.6
AS	231	0.00267	0.99989	37,56	1431 (24h)	7708057	240.0	208.0

We also conducted the same experiment with $\beta = 100ms$ and $\eta = 1ms$ for the *W-ET* system (Table 6.6). 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 6.6: *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**	**MAX($\mathbf{T}_D$)** (ms)	$\mathbf{T}_D$ (ms)
1	6	0.000069	0.999990	140.00	64 (1h)	349339	910.0	689.5
10	45	0.000520	0.999972	53.07	64 (1h)	349341	460.0	383.9
100	98	0.001133	0.999945	47.99	64 (1h)	349341	291.0	243.9
AS	231	0.002672	0.999899	37.56	1431 (24h)	7708057	240.0	226.7

7 CONCLUSION

In this thesis, we presented a new unreliable failure detector, the Impact FD, that provides an output that expresses the trust of the failure detector with regard to the system (or set of processes) as a whole. The trust 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 exist applications which require information on the reliability of the system as a whole, processes can be grouped into different sets based on some criterion, nodes have different capabilities, and the applications tolerate a certain margin of failure. Both the *impact factor* and the *threshold* render the estimation of the confidence in the system (or a set of processes S) more flexible. For instance, 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 further defined two properties, $PR(IT)_p^S$ and $PR(\diamondsuit IT)_p^S$, which denote the capacity of the Impact FD of process p of accepting different sets of responses that lead to a trusted state of the system S as well as the concept of a $PS{-}accessibility$ and $\diamondsuit PS{-}accessibility$: a correct process p is $PS{-}accessible$ (resp., $\diamondsuit PS{-}accessible$) if every query broadcast by p obtains from the beginning (resp., eventually) a set Q of responses that satisfy the degree of confidence in S ($threshold^{S^*}$). These properties can be ensured by a query-based and timer-based approaches. Interestingly that, in both cases, set Q is not fixed and can be different at distinct times which further improves the *flexibility* property of the Impact FD.

This thesis has also shown that the Impact FD of class $I\Omega$ (resp., $I\Sigma$) is equivalent to Ω (resp., Σ) FD. These equivalences are extremely important since Ω (resp., the pair

$\langle\Omega,\Sigma\rangle$) is the weakest failure detector that solves the consensus impossibility problem in asynchronous distributed systems with a majority of correct processes (resp., any number of failures). Consequently, $I\Omega$ and/or $I\Sigma$ can be these detectors. In addition, it has been shown that, if there exists 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.

The Impact FD can have distinct implementations in accordance with the characteristics of the system model and behavior properties. We have provided three algorithms which implement the Impact FD: the first one is a timer-based implementation for systems where either all links are lossy asynchronous or some or all links are $\diamondsuit-timely$ while the others are lossy asynchronous. The second and third ones exploit the $PS-accessibility$ or $\diamondsuit PS-accessibility$ properties of the monitor process p. The former is based on a time-free message pattern approach where p waits for responses from α processes or from a set Q of processes whose responses satisfy the $threshold^{S^*}$. The latter is based on query-response message rounds where p can receive responses to its broadcast query from a set of processes within a bounded delay (timely responses) and the responses of this set of processes (or eventually) satisfy the *threshold*.

Based on real trace files collected from nodes of PlanetLab (PLANETLAB, 2014), we conducted extensive experiments aiming at evaluating 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 messages, characterizing, therefore, distributed systems that use FDs based on heartbeats. The testbed of the experiments comprises various configurations with different threshold values, impact factor of nodes, and types of links. For evaluation sake, we basically used three of the QoS metrics proposed in (CHEN; TOUEG; AGUILERA, 2002): *detection time*, *average mistake rate*, and *query accuracy probability*. The Impact FD implementation was also compared to the tradition Chen's timer-based FD that outputs information about failure suspicions of each monitored process. Performance evaluation results showed that the assignment of a high (resp. low) impact factor to more stable (resp. unstable) nodes increases the Query Accuracy Probability of the Impact FD. Furthermore, we observed that the Impact FD might weaken the rate of false suspicions when compared to Chen's

FD. 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, while the *detection time* does not increase in the same rate. 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 Impact FD presents better QoS than the Chen's FD if the application is interested in the degree of confidence in the system (trust level) as a whole.

Among many systems to which the Impact FD can be applied, we present some examples in Section 1.1: (1) healthcare monitoring with several sensors which have different levels of relevance, (2) wireless sensor networks (WSNs) that monitor environment conditions, (3) large-scale WSNs which group sensor nodes into clusters for scalability and resource saving reasons, and (4) replicated servers that offer some quality of service (QoS) such as bandwidth or response time. In addition, the concept of Impact FD can be incorporated to solutions that require failure detection with regard to a system (or set of processes) as a whole. In this light, we aim to integrate the Impact FD as the failure detection mechanism to the self-healing module, currently in development, of the Self-healing in Ubiquitous Environments project of the GPPD (Parallel and Distributed Processing Group). More details about this self-healing module are presented in the Appendix 2.

7.1 Future Work

We believe that the Impact FD opens interesting research directions for future work. We highlight some of them:

- *Extending the concept of confidence in the system*: The $trust\_level$ is defined as the sum of the impact factor of all trusted processes of S and the latter is trusted whenever the $trust\_level$ value is equal or greater than the $threshold$ value. As a near future work, we intend to generalize the trust level calculation as well as its comparison with the threshold. In view of such extension, the $Trust\_level(trusted, S^*)$ function could perform an operation over the impact factor of the trusted processes other than the sum (e.g., multiplication, average, etc.) and the threshold would not necessary be a lower bound (e.g., upper bound, equality, etc.). For instance, suppose that the impact factor of a node corre-

sponds 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;

- *Dynamic impact factor*: In the definition of the Impact FD, each node has an impact factor value which does not change. We propose a dynamic impact factor, i.e., the value of 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;
- *Processes recovery and other failure models*: The Impact FD considers only crash failures and the latter are permanent. Inspired by some existing works that propose FD for omission mode ((DELPORTE-GALLET; FAUCONNIER; FREILING, 2005), (CORTINAS et al., 2012), (FERNÁNDEZ-CAMPUSANO et al., 2016)), byzantine mode ((DOUDOU et al., 1999), (MALKHI; REITER, 1997), (KIHLSTROM; MOSER; MELLIAR-SMITH, 2003), (GREVE et al., 2012)), or processes recovery ((AGUILERA; CHEN; TOUEG, 2000), (FERNÁNDEZ-CAMPUSANO et al., 2016), (LARREA; MARTÍN; SORALUZE, 2011)), we could think of extending the Impact FD to other failure modes and/or consider that faulty processes can recover. We should point out that in Section 2.2.1, we have presented different process failure modes in distributed systems. There is an ordering relation between them: more severe failure modes cover less severe ones (Byzantine $\supset$ omission $\supset$ timing $\supset$ crash). Therefore, the extension of tradition crash failure detectors to more severe failures render their respective failure detector implementation more complex or even unfeasible;
- *New Impact FD implementations*: The Impact FD algorithms presented in this thesis considered that nodes are static, uniquely identified, only fail by crash, and that the membership of the system is known. By changing some of these assumptions(e.g., unknown membership, mobile processes, anonymous or homonymous processes, etc.) in order to broaden the application domain that can benefit from the Impact FD features and flexibility, new challenges will raise for the conception and prove of Impact FD algorithms suitable for these new environments. An implementation of the Impact FD for anonymous systems with un-

known membership is presented in the Appendix 1;

- *Performance evaluation on other type of networks*: We conducted experiments based on real traces files collected from nodes of PlanetLab, a wide area network. We intend to extend the evaluation of the QoS of the Impact FD to different networks such as MANET or LAN, comparing its performance with other well-known failure detectors;
- *Numerical analysis*: Considering some application scenarios, it would be interesting to carry out analytical evaluation of the Impact FD (and/or an algorithm that implements it), by varying different parameters, such as the number of nodes, impact factor, the maximum number of failures, the threshold, etc. For instance, let's consider a subset S where nodes have different impact factors that follow a given law. Some possible analysis would be: (a) If at most f nodes can fail, what is the probability of S to be trusted in relation to different threshold values ? (b) If the maximum number of failures varies, what are the possible distributions of failures, i.e., trust level values, which ensure a given threshold ? (c) what is the minimum number of nodes in relation to the number of failures and a given threshold that renders S always trusted ? (d) how does the Accuracy Probability (P_A) behave in a Query-Response implementation when α varies ?

7.2 Publications

During the PhD course, the following research results were published as first author:

Papers published in Conferences

- ROSSETTO, A. G. M.; Geyer, C.F.R.; ARANTES, L.; SENS, P. Implementing a Flexible Failure Detector that Expresses the Confidence in the System. In: 7th Latin-American Symposium on Dependable Computing (LADC), 2016, Cali. (B3)
- ROSSETTO, A. G. M.; Geyer, C.F.R.; ARANTES, L.; SENS, P. Failure Detector That Gives Information on the Degree of Confidence in the System. In: 20th IEEE Symposium on Computers and Communication (ISCC), 2015, Larnaca. (A2)
- ROSSETTO, A. G. M.; ARANTES, L.; SENS, P.; Geyer, C.F.R. Impact: Um detector de falhas baseado na relevância dos processos e no grau de confiança no sistema.

In: 33º Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos, 2015, Vitória. (B2)

- ROSSETTO, A. G. M.; Rolim, C.O.; LEITHARDT, V. R. Q.; Borges, Guilherme; Geyer, C.F.R.; ARANTES, L. ; SENS, P. A new unreliable failure detector for self-healing in ubiquitous environments. In: IEEE International Conference on Advanced Information Networking and Applications (AINA), 2015, Gwangju. (A2)
- ROSSETTO, A. G. M.; GEYER, C. F. R.; ARANTES, L. ; Rolim, C.O.; Valderi, R.Q.L. An Architecture for Resilient Ubiquitous Systems. In: International Conference on Health Informatics, 2014, ESEO. (B4)
- ROSSETTO, A. G. M.; Geyer, C.F.R.; ARANTES, L. ; SENS, P. The Impact Failure Detector. In: Tenth European Dependable Computing Conference - EDCC 2014, Newcastle. (B2)
- ROSSETTO, A. G. M.; ARANTES, L.; SENS, P.; Geyer, C.F.R. Impact: A new unreliable failure detector. In: WSPPD 2014 - XII Workshop de Processamento Paralelo e Distribuído, 2014, Porto Alegre.
- ROSSETTO, A. G. M.; Geyer, C.F.R.; Rolim, Carlos Oberdan; Valderi, R.Q.L.; Silva, Jorge Sá. Monitoring System for Ubiquitous HealthCare. In: X Workshop de Processamento Paralelo e Distribuído, 2012, Porto Alegre.

Papers Published in Scientific Journals

- ROSSETTO, A. G. M.; Rolim, Carlos Oberdan ; LEITHARDT, V. R. Q. ; Dantas, M.A.R. ; GEYER, C. F. R. . An adaptive fault tolerance approach to enhance the execution of applications on multi-cluster grid configurations from mobile grid interfaces in wireless networks. International Journal of High Performance Systems Architecture (Print), v. 3, p. 202-215, 2011. (B2)

Book Chapter

- ROSSETTO, A. G. M.; ROLIM, C.O.; LEITHARDT, V. R. Q.; BORGES, Guilherme ; GEYER, C.F.R.; ARANTES, L.; SENS, P. A failure detector based on processes' relevance and the confidence degree in the system for self-healing in ubiquitous environments In: Pervasive Computing: Next Generation Platforms for Intelligent Data Collection. 1 ed. : Elsevier, 2016, v.1, p. 393-416. ISBN: 9780128036631

Technical Report

- ROSSETTO, A. G. M.; Geyer, C.F.R.; ARANTES, L.; SENS, P. Impact: an unreliable failure detector based on processes' relevance and the confidence degree in the system. [S.l.: s.n.], 2015. INRIA No hal- 01136595.

Submitted Paper to Scientific Journal

- Journal: Future Generation Computer Systems
- Title: Impact FD: An Unreliable Failure Detector Based on Processes Relevance and Confidence in the System
- Authors: Anubis Graciela de Moraes Rossetto, Claudio F. R. Geyer, Luciana Arantes and Pierre Sens.
- Qualis: A2

REFERENCES

ABBASI, A. Z. et al. A review of wireless sensors and networks' applications in agriculture. **Computer Standards & Interfaces**, [S.l.], v.36, n.2, p.263–270, 2014.

AGUILERA, M. K.; CHEN, W.; TOUEG, S. Failure Detection and Consensus in the Crash-recovery Model. **Distributed Computing**, [S.l.], v.13, n.2, p.99–125, Apr. 2000.

AGUILERA, M. K. et al. On Implementing Omega with Weak Reliability and Synchrony Assumptions. In: TWENTY-SECOND ANNUAL SYMPOSIUM ON PRINCIPLES OF DISTRIBUTED COMPUTING, 2003. **Proceedings...** [S.l.: s.n.], 2003. p.306–314.

AGUILERA, M. K. et al. Communication-efficient leader election and consensus with limited link synchrony. In: ACM SYMPOSIUM ON PRINCIPLES OF DISTRIBUTED COMPUTING, 2004. **Proceedings...** [S.l.: s.n.], 2004. p.328–337.

ARANTES, L. et al. Eventual leader election in evolving mobile networks. In: **Principles of Distributed Systems**. [S.l.]: Springer, 2013. p.23–37.

ARANTES, L. et al. Probabilistic Byzantine Tolerance for Cloud Computing. In: IEEE 34TH SYMPOSIUM ON RELIABLE DISTRIBUTED SYSTEMS (SRDS) 2015, 2015. **Proceedings...** [S.l.: s.n.], 2015. p.1–10.

CRUZ-CUNHA, M. M. (Ed.). **Handbook of Research on Mobility and Computing**: evolving technologies and ubiquitous impacts. [S.l.]: IGI Global, 2010. p.20.

ARAÚJO MACÊDO, R. de; LIMA, F. R. L. e. Improving the quality of service of failure detectors with SNMP and artificial neural networks. In: SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES, 22., 2004. **Proceedings...** [S.l.: s.n.], 2004. p.583–586.

ARÉVALO, S. et al. Failure detectors in homonymous distributed systems (with an application to consensus). In: IEEE 32ND INTERNATIONAL CONFERENCE ON DISTRIBUTED COMPUTING SYSTEMS (ICDCS) 2012, 2012. **Proceedings...** [S.l.: s.n.], 2012. p.275–284.

AVIZIENIS, A. et al. **Fundamental concepts of dependability**. [S.l.]: University of Newcastle upon Tyne, Computing Science Newcastle upon Tyne, UK, 2001.

AVIZIENIS, A. et al. Basic concepts and taxonomy of dependable and secure computing. **Dependable and Secure Computing, IEEE Transactions on**, [S.l.], v.1, n.1, p.11–33, 2004.

BERTIER, M. et al. Performance Analysis of a Hierarchical Failure Detector. In: DEPENDABLE SYSTEMS AND NETWORKS, 2003. **Proceedings...** [S.l.: s.n.], 2003. v.3, p.635–644.

BONNET, F.; RAYNAL, M. On the road to the weakest failure detector for k-set agreement in message-passing systems. **Theor. Comput. Sci.**, [S.l.], v.412, n.33, p.4273–4284, 2011.

BONNET, F.; RAYNAL, M. Anonymous asynchronous systems: the case of failure detectors. **Distributed Computing**, [S.l.], v.26, n.3, p.141–158, 2013.

BONNET, F.; RAYNAL, M. Anonymous asynchronous systems: the case of failure detectors. **Distributed computing**, [S.l.], v.26, n.3, p.141–158, 2013.

BRUN, Y. et al. Smart redundancy for distributed computation. In: INTERNATIONAL CONFERENCE ON DISTRIBUTED COMPUTING SYSTEMS (ICDCS) 2011, 31., 2011. **Proceedings...** [S.l.: s.n.], 2011. p.665–676.

CHANDRA, T. D. et al. On the impossibility of group membership. In: ACM SYMPOSIUM ON PRINCIPLES OF DISTRIBUTED COMPUTING, 1996. **Proceedings...** [S.l.: s.n.], 1996. p.322–330.

CHANDRA, T. D.; HADZILACOS, V.; TOUEG, S. The weakest failure detector for solving consensus. **Journal of the ACM (JACM)**, [S.l.], v.43, n.4, p.685–722, 1996.

CHANDRA, T. D.; TOUEG, S. Unreliable failure detectors for reliable distributed systems. **Journal of the ACM (JACM)**, [S.l.], v.43, n.2, p.225–267, 1996.

CHARRON-BOST, B.; SCHIPER, A. Uniform consensus is harder than consensus. **J. Algorithms**, [S.l.], v.51, n.1, p.15–37, 2004.

CHEN, W.; TOUEG, S.; AGUILERA, M. K. On the quality of service of failure detectors. **Computers, IEEE Transactions on**, [S.l.], v.51, n.5, p.561–580, 2002.

CHU, F. Reducing Ω to $\Diamond W$. **Information Processing Letters**, [S.l.], v.67, n.6, p.289–293, 1998.

CORREIA, M.; NEVES, N. F.; VERÍSSIMO, P. From consensus to atomic broadcast: time-free byzantine-resistant protocols without signatures. **The Computer Journal**, [S.l.], v.49, n.1, p.82–96, 2006.

CORTINAS, R. et al. Secure failure detection and consensus in trustedpals. **IEEE Transactions on Dependable and Secure Computing**, [S.l.], v.9, n.4, p.610–625, 2012.

COSQUER, F. J.; RODRIGUES, L.; VERÍSSIMO, P. Using Tailored Failure Suspectors to Support Distributed Cooperative Applications. In: PARALLEL AND DISTRIBUTED COMPUTING AND SYSTEMS, 1995. **Proceedings...** [S.l.: s.n.], 1995. p.352–358.

COULOURIS, G. F.; DOLLIMORE, J.; KINDBERG, T. **Distributed systems**: concepts and design. [S.l.]: pearson education, 2005.

CRISTIAN, F.; FETZER, C. The timed asynchronous distributed system model. **IEEE Transactions on Parallel and Distributed systems**, [S.l.], v.10, n.6, p.642–657, 1999.

DÉFAGO, X. et al. Definition and specification of accrual failure detectors. In: INTERNATIONAL CONFERENCE ON DEPENDABLE SYSTEMS AND NETWORKS (DSN'05), 2005. **Proceedings...** [S.l.: s.n.], 2005. p.206–215.

DELPORTE-GALLET, C. et al. The weakest failure detectors to solve certain fundamental problems in distributed computing. In: ACM SYMPOSIUM ON PRINCIPLES OF DISTRIBUTED COMPUTING, 2004. **Proceedings...** [S.l.: s.n.], 2004. p.338–346.

DELPORTE-GALLET, C. et al. Mutual exclusion in asynchronous systems with failure detectors. **J. Parallel Distrib. Comput.**, [S.l.], v.65, n.4, p.492–505, 2005.

DELPORTE-GALLET, C.; FAUCONNIER, H.; FREILING, F. C. Revisiting failure detection and consensus in omission failure environments. In: INTERNATIONAL COLLOQUIUM ON THEORETICAL ASPECTS OF COMPUTING, 2005. **Proceedings...** [S.l.: s.n.], 2005. p.394–408.

DELPORTE-GALLET, C.; FAUCONNIER, H.; GUERRAOUI, R. Shared memory vs message passing. **Technical Report**, [S.l.], v.77, 2003.

DOLEV, D.; DWORK, C.; STOCKMEYER, L. On the minimal synchronism needed for distributed consensus. **Journal of the ACM (JACM)**, [S.l.], v.34, n.1, p.77–97, 1987.

DOUDOU, A. et al. Muteness failure detectors: specification and implementation. In: EUROPEAN DEPENDABLE COMPUTING CONFERENCE, 1999. **Proceedings...** [S.l.: s.n.], 1999. p.71–87.

DWORK, C.; LYNCH, N.; STOCKMEYER, L. Consensus in the presence of partial synchrony. **Journal of the ACM (JACM)**, [S.l.], v.35, n.2, p.288–323, 1988.

FERNÁNDEZ-CAMPUSANO, C. et al. A communication-efficient leader election algorithm in partially synchronous systems prone to crash-recovery and omission failures. In: INTERNATIONAL CONFERENCE ON DISTRIBUTED COMPUTING AND NETWORKING, SINGAPORE, JANUARY 4-7, 2016, 17., 2016. **Proceedings...** [S.l.: s.n.], 2016. p.8:1–8:4.

FERNÁNDEZ-CAMPUSANO, C. et al. A communication-efficient leader election algorithm in partially synchronous systems prone to crash-recovery and omission failures. In: INTERNATIONAL CONFERENCE ON DISTRIBUTED COMPUTING AND NETWORKING, 17., 2016. **Proceedings...** [S.l.: s.n.], 2016. p.8.

FISCHER, M. J.; LYNCH, N. A.; PATERSON, M. S. Impossibility of distributed consensus with one faulty process. **Journal of the ACM (JACM)**, [S.l.], v.32, n.2, p.374–382, 1985.

GANEK, A. G.; CORBI, T. A. The dawning of the autonomic computing era. **IBM systems Journal**, [S.l.], v.42, n.1, p.5–18, 2003.

GEETA, D.; NALINI, N.; BIRADAR, R. C. Fault tolerance in wireless sensor network using hand-off and dynamic power adjustment approach. **Journal of Network and Computer Applications**, [S.l.], v.36, n.4, p.1174–1185, 2013.

GÓMEZ-CALZADO, C. et al. Fault-Tolerant Leader Election in Mobile Dynamic Distributed Systems. In: IEEE 19TH PACIFIC RIM INTERNATIONAL SYMPOSIUM ON DEPENDABLE COMPUTING PRDC, 2013. **Proceedings...** [S.l.: s.n.], 2013. p.78–87.

GREVE, F. et al. A time-free byzantine failure detector for dynamic networks. In: NINTH EUROPEAN DEPENDABLE COMPUTING CONFERENCE (EDCC) 2012, 2012. **Proceedings...** [S.l.: s.n.], 2012. p.191–202.

GREVE, F. et al. Eventually strong failure detector with unknown membership. **The Computer Journal**, [S.l.], v.55, n.12, p.1507–1524, 2012.

GUERRAOUI, R.; RODRIGUES, L. **Introduction to reliable distributed programming**. [S.l.]: Springer Science & Business Media, 2006.

GUPTA, T. et al. Improving availability in distributed systems with failure informers. In: USENIX SYMPOSIUM ON NETWORKED SYSTEMS DESIGN AND IMPLEMENTATION (NSDI 13), 10., 2013. **Proceedings...** [S.l.: s.n.], 2013. p.427–441.

HAYASHIBARA, N.; DÉFAGO, X.; KATAYAMA, T. Two-ways adaptive failure detection with the ϕ-failure detector. In: WORKSHOP ON ADAPTIVE DISTRIBUTED SYSTEMS (WADIS03), 2003. **Proceedings...** [S.l.: s.n.], 2003. p.22–27.

HAYASHIBARA, N. et al. The φ accrual failure detector. In: IEEE INTERNATIONAL SYMPOSIUM ON RELIABLE DISTRIBUTED SYSTEMS, 2004, 23., 2004. **Proceedings...** [S.l.: s.n.], 2004. p.66–78.

HUTLE, M. **Failure detection in sparse networks**. [S.l.: s.n.], 2005.

ISHIBASHI, K.; YANO, M. A Proposal of Forwarding Method for Urgent Messages on an Ubiquitous Wireless Sensor Network. In: ASIA-PACIFIC SYMPOSIUM ON INFORMATION AND TELECOMMUNICATION TECHNOLOGIES (APSITT) 2005, 6., 2005. **Proceedings...** [S.l.: s.n.], 2005. p.293–298.

JACOBSON, V. Congestion avoidance and control. In: ACM SIGCOMM COMPUTER COMMUNICATION REVIEW, 1988. **Proceedings...** [S.l.: s.n.], 1988. v.18, n.4, p.314–329.

JALOTE, P. **Fault tolerance in distributed systems**. [S.l.]: Prentice-Hall, Inc., 1994.

JIMÉNEZ, E.; ARÉVALO, S.; FERNáNDEZ, A. Implementing Unreliable Failure Detectors with Unknown Membership. **Inf. Process. Lett.**, [S.l.], v.100, n.2, p.60–63, Oct. 2006.

JUNQUEIRA, F. P. et al. Threshold protocols in survivor set systems. **Distributed Computing**, [S.l.], v.23, n.2, p.135–149, 2010.

KIHLSTROM, K. P.; MOSER, L. E.; MELLIAR-SMITH, P. M. Byzantine Fault Detectors for Solving Consensus. **Comput. J.**, [S.l.], v.46, n.1, p.16–35, 2003.

LAMPORT, L. Time, clocks, and the ordering of events in a distributed system. **Communications of the ACM**, [S.l.], v.21, n.7, p.558–565, 1978.

LAMPORT, L.; LYNCH, N. **Distributed computing**: models and methods. [S.l.: s.n.], 1991. 1157–1199p.

LARREA, M.; ANTA, A. F.; ARÉVALO, S. Implementing the weakest failure detector for solving the consensus problem. **IJPEDS**, [S.l.], v.28, n.6, p.537–555, 2013.

LARREA, M.; FERNÁNDEZ, A.; ARÉVALO, S. On the implementation of unreliable failure detectors in partially synchronous systems. **Computers, IEEE Transactions on**, [S.l.], v.53, n.7, p.815–828, 2004.

LARREA, M.; MARTÍN, C.; SORALUZE, I. Communication-efficient leader election in crash–recovery systems. **Journal of Systems and Software**, [S.l.], v.84, n.12, p.2186–2195, 2011.

MALKHI, D.; OPREA, F.; ZHOU, L. Ω meets paxos: leader election and stability without eventual timely links. In: **Distributed Computing**. [S.l.]: Springer, 2005. p.199–213.

MALKHI, D.; REITER, M. Unreliable Intrusion Detection in Distributed Computations. In: IEEE WORKSHOP ON COMPUTER SECURITY FOUNDATIONS, 10., 1997. **Proceedings...** [S.l.: s.n.], 1997. p.116–. (CSFW '97).

MOSTÉFAOUI, A. et al. From W to Omega: A simple bounded quiescent reliable broadcast-based transformation. **J. Parallel Distrib. Comput.**, [S.l.], v.67, n.1, p.125–129, 2007.

MOSTÉFAOUI, A.; MOURGAYA, E.; RAYNAL, M. Asynchronous Implementation of Failure Detectors. In: INTERNATIONAL CONFERENCE ON DEPENDABLE SYSTEMS AND NETWORKS (DSN 2003), 22-25 JUNE 2003, SAN FRANCISCO, CA, USA, PROCEEDINGS, 2003., 2003. **Proceedings...** [S.l.: s.n.], 2003. p.351–360.

MOSTEFAOUI, A.; MOURGAYA, E.; RAYNAL, M. Asynchronous implementation of failure detectors. In: ANNUAL IEEE/IFIP INTERNATIONAL CONFERENCE ON DEPENDABLE SYSTEMS AND NETWORKS (DSN), 43., 2003. **Proceedings...** [S.l.: s.n.], 2003. p.351–351.

NUNES, R. C.; JANSCH-PORTO, I. Qos of timeout-based self-tuned failure detectors: the effects of the communication delay predictor and the safety margin. In: INTERNATIONAL CONFERENCE ON DEPENDABLE SYSTEMS AND NETWORKS, 2004, 2004. **Proceedings...** [S.l.: s.n.], 2004. p.753–761.

PLANETLAB. **Planetlab**. "Online. Access date: September 16, 2015"„ `http://www.planet-lab.org`.

RAYNAL, M. Eventual Leader Service in Unreliable Asynchronous Systems: why? how? In: SIXTH IEEE INTERNATIONAL SYMPOSIUM ON NETWORK COMPUTING AND APPLICATIONS, 2007 (NCA) 2007, 2007. **Proceedings...** [S.l.: s.n.], 2007. p.11–24.

SÁ, A. S. de; ARAÚJO MACÊDO, R. J. de. QoS self-configuring failure detectors for distributed systems. In: IFIP INTERNATIONAL CONFERENCE ON DISTRIBUTED APPLICATIONS AND INTEROPERABLE SYSTEMS, 2010. **Proceedings...** [S.l.: s.n.], 2010. p.126–140.

SATZGER, B. et al. A new adaptive accrual failure detector for dependable distributed systems. In: ACM SYMPOSIUM ON APPLIED COMPUTING, 2007., 2007. **Proceedings...** [S.l.: s.n.], 2007. p.551–555.

SCHIPER, N.; TOUEG, S. A robust and lightweight stable leader election service for dynamic systems. In: IEEE INTERNATIONAL CONFERENCE ON DEPENDABLE SYSTEMS AND NETWORKS WITH FTCS AND DCC (DSN), 2008. **Proceedings...** [S.l.: s.n.], 2008. p.207–216.

TOMSIC, A. et al. 2w-fd: a failure detector algorithm with qos. In: IEEE INTERNATIONAL PARALLEL AND DISTRIBUTED PROCESSING SYMPOSIUM (IPDPS) 2015, 2015. **Proceedings...** [S.l.: s.n.], 2015. p.885–893.

TURCHETTI, R. C. et al. Um Serviço de Detecção de Falhas com QoS Auto-ajustável para Múltiplas Aplicações Simultâneas. In: SIMPóSIO BRASILEIRO DE REDES DE COMPUTADORES E SISTEMAS DISTRIBUíDOS (SBRC 2016), 2016. **Proceedings...** [S.l.: s.n.], 2016. p.179–192.

VERISSIMO, P.; RODRIGUES, L. **Distributed systems for system architects**. [S.l.]: Springer Science & Business Media, 2012. v.1.

VÉRON, M. et al. RepFD-Using reputation systems to detect failures in large dynamic networks. In: INTERNATIONAL CONFERENCE ON PARALLEL PROCESSING (ICPP-2015), 44., 2015. **Proceedings...** [S.l.: s.n.], 2015.

XIONG, N. et al. A self-tuning failure detection scheme for cloud computing service. In: IEEE 26TH INTERNATIONAL PARALLEL & DISTRIBUTED PROCESSING SYMPOSIUM (IPDPS), 2012. **Proceedings...** [S.l.: s.n.], 2012. p.668–679.

YANG, Z. et al. An Exponential Smoothing Adaptive Failure Detector in the Dual Model of Heartbeat and Interaction. **Journal of Computing Science and Engineering**, [S.l.], v.8, n.1, p.17–24, 2014.

YIM, S.-J.; CHOI, Y.-H. An adaptive fault-tolerant event detection scheme for wireless sensor networks. **Sensors**, [S.l.], v.10, n.3, p.2332–2347, 2010.

Appendix 1 Impact FD on an Anonymous System

Considering that processes are anonymous (i.e., non identifiable) and both the membership and the cardinality of the system are unknown, the timer-based Algorithm 14 implements a Impact FD that runs on a R-S-δ-Π system. The latter is defined as:

R-S-δ-Π: a system such that p in S, $S = \Pi$, and all processes of Π are synchronous and execute the Impact FD algorithm. Every pair of processes is connected by reliable bidirectional links and there exists a known upper bound δ for the delay of messages.

As Algorithm 14 implements a perfect failure detector, we denoted it IAP (*anonymous perfect impact* failure detector). We assume that all processes start executing the algorithm at the same time. Moreover, the local clocks of processes are synchronized and they do not present drifts.

We denote :

- $S^*_{corr}(t)$: the set of tuples $\langle \_, I \rangle$ of all non faulty processes at t.
- $sum(S^*, t) = \sum_{\langle \_, I \rangle \in S^*(t)} I$

When invoked in p at t, the failure detector of the class IAP returns the trust level of p in relation to Π. It satisfies the following properties:

- *Safety*: $\forall p \in correct(F), \forall t \in T \geq 0, trust\_level_p(t) \geq sum(S^*_{corr}, t)$
- *Liveness*: $\exists t \in T, \forall p \in correct(F), \forall t' \in T \geq t, trust\_level_p(t') = trust\_level_p(t)$.

The *safety* property states that at t, the $trust\_level_p$ is always greater or equal than the sum of impact factor values of no faulty processes at t, while the *liveness* property ensures that there is a time after which $trust\_level_p$ does not change, i.e., the system is either always $trusted$ or always $untrusted$ for p.

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

At the initialization, the set S^*_1 is reset (line 3) and the impact factor of p ($\langle \_, I_p \rangle$) is added to $tmp\_S_1{}^*$. Note that all tuples $\langle id, I \rangle$ are included in the subset 1 of S^* (S^*_1 and $tmp\_S_1{}^*$), without the identifier id, since processes are anonymous. The timer

Algorithm 14 Timer-based implementation

1: **Begin**

Input
2: I_p, Δ, δ_t
Init
3: $S_1^* = \emptyset$
4: $tmp\_S_1{}^* = \{\langle \_, I_p \rangle\}$
5: $timeout = \delta_t$

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

Task T2
8: **Upon reception of** $(ALIVE, I)$ **do**
9: $tmp\_S_1{}^* = tmp\_S_1{}^* \cup \{\langle \_, I \rangle\}$
10: **end**

Task T3 - When timeout expires
11: **Upon** $expiration\ of\ timer$ **do**
12: $S_1{}^* = tmp\_S_1{}^*$
13: $tmp\_S_1{}^* = \{\langle \_, I_p \rangle\}$
14: **end**

Task T4
15: **Upon invocation of** $IAP()$ **do**
16: **if** $S_1^* \neq \emptyset$ **then**
17: **return** $Trust\_level(\emptyset, S_1{}^*)$
18: **else** ▷ before the first timeout
19: **return** ∞
20: **end if**
21: **end**

22: **End**

$timeout$ is then initialized to δ_t.

In Task T1, process p periodically sends to the other processes an ALIVE message with its impact factor value I_p (line 6) and starts the $timeout$ (line 7).

When process p receives an ALIVE message from an anonymous processes (Task T2), it includes the information about the impact value of this process in its $tmp\_S_1{}^*$ set.

Upon expiration of the timer $timeout$ (Task T3), p assigns its current knowledge about the received impact factor values (i.e., $tmp\_S_1{}^*$) to $S_1{}^*$ and re-initialize the $tmp\_S_1{}^*$ set.

Task T4 handles the invocation of the *IAP()* function by p, which returns the sum of impact factor values of the processes in S_1^* (line 17), provided that p has a view (even if partial) of the set of impact factor values of the system, i.e., at least one timeout has expired; otherwise, it returns ∞.

The followings points should be highlighted:

- The function $Trut\_level(trusted, S^*)$ needs to be redefined since the processes do not have identifier (anonymous) and thus, the id of the processes can not be assigned to $trusted$. Hence, we define:

$$Trust\_level(trusted, S^*) = \{trust\_level \mid trust\_level = \sum_{\langle \_, I_j \rangle \in S^*} I_j\}$$

- As defined in Section 5, we assume that local processing time is negligible with respect to message communication delays. Furthermore, by assumption, all processes start executing at the same time and send $ALIVE$ messages at each round (i.e., within the expiration of two consecutive timeouts). Let δ_h be the maximum difference (delay) between the sending time of the heartbeats by all correct processes in a single round and δ_{lat}, be the maximum time for the transmission of a heartbeat message. The value of the $timeout$ δ_t must, thus, be greater than $\delta_t = \delta_h + \delta_{lat}$ while Δ, interval between the sending of two heartbeats from a node, must be greater than the δ_t;
- Since the system is synchronous (no false suspicions), the trust level value returned by Impact FD at invocation i is always equal or smaller than the value returned by the previous invocation $i-1$;
- If f is the maximum number of failures, when all of the f faults take place, the

value returned by the Impact FD will always be the same;

- The algorithm can also applied to both homonymous (several processes may have the same identifier (ARÉVALO et al., 2012)) and identifiable systems;
- If we consider the impact value equals to 1, a similar approach is proposed by the $\overline{AP}$ perfect failure detector for anonymous networks (BONNET; RAYNAL, 2013b), which returns an estimation of the current number of processes in anonymous synchronous systems.

Sketch of Proof

Theorem 15. *Algorithm 14 ensures the* **safety** *property.*

Proof. Let denote $F^*(t)$ (resp., ${df_p}^*(t)$) the set of faulty (resp., detected faulty) processes and their respective impact factors until t;

Since the channels are reliable, the system is synchronous, and $\delta_t > \delta$, every timeout expiration (Task T3) at t, $S^*_{corr}(t) \subseteq {S_1}^*(t)$, i.e., every ALIVE message sent by a non faulty process at t is received by p before the expiration of the timeout at t.

Let's consider p correct and two cases: (1) the first timeout has not expired yet; (2) at least one timeout has expired; In case (1), as a timeout has never expired, task T3 has never executed and, consequently, neither line 12. S^*_1 keeps then its initial value $S^*_1 = \emptyset$ and, thus, when $IAP()$ is invoked, $trust\_level_p = \infty$. Hence, the safety property is ensured. For case (2), let's denote $t_1 \in T$ the time of the last timeout expiration (Task T3). In this case, S^*_1 has been updated at line 12 and we denote its value $S^*_1(t_1)$. Then, $\forall\, t_2 \in T \geq t_1$, if $F^*(t_2) = {df_p}^*(t_1)$, i.e., all failures were detected by p at the last timeout expiration at t_1, $S^*_1(t_1) = S^*_{corr}(t_2)$, which implies that $trust\_level_p(t_2) = sum(S^*_1, t_1) = sum(S^*_{corr}, t_2)$. Otherwise, if ${df_p}^*(t_1) \subset F^*(t_2)$, i.e., not all failures were detected by p, $S^*_1(t_1) = S^*_{corr}(t_2) \cup (F^*(t_2)/{df_p}^*(t_1)) \supset S^*_{corr}(t_2)$ which implies that $trust\_level_p(t_2) = sum(S^*_1, t_1) > sum(S^*_{corr}, t_2)$. Therefore, the safety property is ensured.

□

Corollary 1. *Algorithm 14 ensures that:*

$\forall p \in correct(F), \forall t_1, t_2 \in T >= 0, t_2 > t_1, trust\_level_p(t_2) \leq trust\_level_p(t_1)$.

Proof. The corollary directly holds from Theorem 15. □

Theorem 16. *Algorithm 14 ensures the* **liveness** *property.*

Proof. Let consider that p is correct. Since processes and links are synchronous and the timeout value δ_t is greater that message delay δ, there are no false suspicions. Let $t \in T$ be the time after which all faulty processes are crashed, i.e., $\forall t' \in T \geq t, S^*_{corr}(t) = S^*_{corr}(t')$ and all $ALIVE$ messages sent by these faulty processes before they crashed were delivered to p. Thus, after t, p will receive no more $ALIVE$ messages from faulty processes. Then, in the next expiration of the timeout after t, faulty processes are not in S^*_1 and p will always receive the same number of $ALIVE$ messages, i.e., the messages from the correct processes ($S^*_{corr}(t)$). Hence, $\forall t' \in T \geq t, S^*_1 = S^*_{corr}(t') = S^*_{corr}(t)$ and, therefore, $\forall t' \in T \geq t, trust\_level(t') = trust\_level(t)$. The liveness property is thus ensured. □

Corollary 2. *Algorithm 14 ensures that* $\exists t \in T$ *such that, if* $trust\_level_p(t) \geq threshold_p^{S^*}$ *(resp.,* $trust\_level_p(t) < threshold_p^{S^*}$*),* $\forall t' \in T \geq t$*, S is* $trusted$ *(resp.,* $untrusted$*) for p, i.e., it will be permanently* $trusted$ *(resp.,* $untrusted$*) for p after t.*

Proof. The corollary directly holds from Theorem 16. □

Appendix 2 The Self-healing Module

Ubiquitous systems have become a common technology in our everyday lives, increasing then our dependency on them. However, the occurrence of failures in this type of systems can reduce their applicability/usability which may induce some hard, or even dangerous, consequences. Such systems must, therefore, present self-healing capabilities in order to detect failures and make the necessary adjustments to prevent their impact on applications. Self-healing is a property of autonomic computing. A system designed with this feature automatically discovers, diagnoses, and reacts to disruptions (GANEK; CORBI, 2003). The system must then be able to detect failures and make the necessary adjustments to prevent them from having an undesirable impact on the application which should keep active and available. Hence, failure detection service which delivers monitory information about the aliveness of the system nodes is a crucial feature for self-healing systems. Moreover, ubiquitous systems have some inherent features that must be taken into account and which are not always addressed in the existing literature, in particular in regard to the failure detection mechanism.

In this context, a Self-healing Module is proposed with capabilities to detect failures and make the necessary adjustments to prevent their impact on applications. The module is shown in Figure B.1 and has two components: the Detection Service (DS) and Adaptation Manager (AM). The former is responsible for detecting failures of different entities (nodes, sensor, etc.) that have to be monitored in the system. The Adaptation Manager provides suitable adaptation strategies and is designed to reduce the impact of detected failures on the application.

The architecture also has a *Repository* that is used to keep all the previous settings required for the self-healing system to operate. This *Repository* allows the user (who is likely to be an administrator) to configure all the data that will be used by the Detection Service and Adaptation Manager components.

The structure of the Self-healing Module provides a mechanism for node failure detection, makes decisions about the action required to prevent damage to the system and reports them to the ubiquitous applications so that they can execute them. The proposed approach is a generic self-healing module that can be integrated with legacy systems, i.e. existing software systems. Considering these premises, the project of Self-

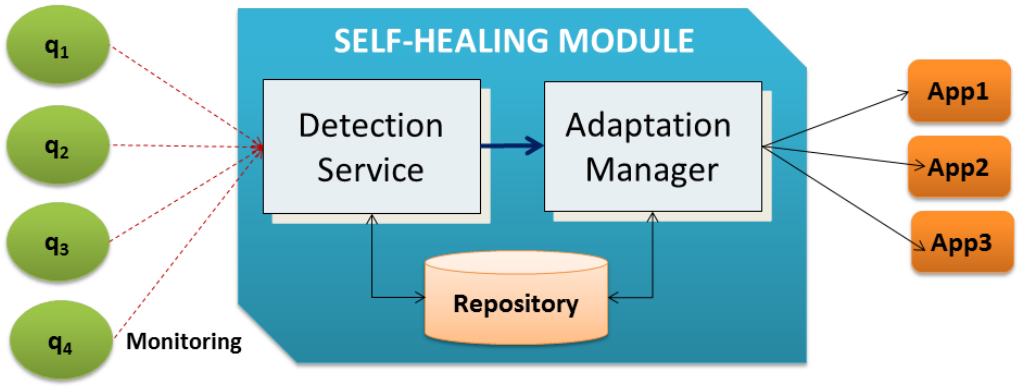

Figure B.1: Self-healing Module

healing Module intends to use the Impact FD as a sub-module of *Detection Service* for monitoring of nodes.

The Detection Service comprises the detecting stage which includes a monitoring mechanism of the nodes and the analysis that determines whether the system is in a degraded state or not. Through the monitoring of a set of nodes (S) and comparing the output of Impact FD ($trust\ level$) with the $threshold$, the detection service defines its *status* which can be either *"trusted"* or *"untrusted"*. When the *status* of set (S) is considered to be *"untrusted"*, the Detection Service notifies the Adaptation Manager so that suitable adaptation strategies can be adopted. In this case, the Adaptation Manager decides if some measures must be taken (their degree of urgency depending on the trust level output). For instance, when the *trust level* of a subset is smaller than its threshold, one or more action plans may be required. The plans are the reconfiguration strategies which have been predefined by the administrator that set up the Repository. After the definition of the plan, the Adaptation Manager communicates to the Application which will activate it .