Implementing a flexible failure detector that expresses the confidence in the system

Anubis Graciela de Moraes Rossetto, Claudio R. Geyer, Luciana Arantes, Pierre Sens

Implementing a flexible failure detector

Anubis G. de M. Rossetto
Federal Inst. Sul-rio-grandense, Passo Fundo, Brazil
Email: anubisrossetto@gmail.com

Cláudio F. R. Geyer
Institute of Informatics Federal University of Rio Grande do Sul (UFRGS) Porto Alegre, Brazil
Email: geyer@inf.ufrgs.br

Luciana Arantes and Pierre Sens
Sorbonne Universités, UPMC Univ. Paris 06, CNRS, Inria, LIP6, Paris, France
Email: {luciana.arantes, pierre.sens}@lip6.fr

Abstract*—Traditional** ***unreliable failure detectors **are per process oracles that provide a list of nodes suspected of having failed. In [1], we introduced the *Impact* failure detector that outputs a *trust level* value which is the degree of confidence in the system. An *impact factor* is assigned to each node and the *trust level* is equal to the sum of the impact factors of the nodes not suspected to have failed. An input *threshold* parameter defines an impact factor limit value, over which the confidence degree on the system is ensured. 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. We propose in this article two different algorithms, based on query-response message rounds, that implement the Impact FD whose conceptions were tailored to satisfy the Impact FD's flexibility. The first one exploits the time-free message pattern approach while the second one considers a set of bounded timely responses. We also introduced the concept that a process can be $PS{-}accessible$ (or $\Diamond PS{-}accessible$) which guarantees that the system S will always (or eventually always) be trusted to this process as well as two properties, $PR(IT)$ and $PR(\Diamond IT)$, that characterize the minimum necessary stability condition of S that ensures confidence (or eventual confidence) on it. In both implementations, if the process that monitors S is $PS{-}accessible$ or $\Diamond PS{-}accessible$, at every query round, it only waits (or eventually only waits) for a set of response that satisfy the *threshold*. A crucial facet of this set of processes is that it is not fixed, i.e., the set of processes can change at each round, which is in accordance with the flexibility capacity of the Impact FD.**

1. Introduction

There are several important works in the literature concerning *unreliable fault detectors* (FDs). A FD can informally be seen as a per process oracle which, when invoked, provide information about processes liveness [2].

In [1], we introduced the *Impact* failure detector. In contrast with traditional unreliable failure detectors [3] [4] that output the set of nodes suspected of having failed, the Impact FD outputs a *trust level* concerning a given system *S*. The output can be considered as the degree of confidence in the system. To this end, an *impact factor*, that is defined by the user, is assigned to each node and the *trust level* is equal to the sum of the impact factors of the trusted nodes, i.e., those nodes not suspected of failure. Furthermore, an input *threshold* parameter defines an impact factor limit value, over which the confidence degree on *S* is ensured. Hence, by comparing the trust level with the *threshold*, a process p that monitors S knows whether the system is trusted or not.

We consider "trusted" the expectation that a system will behave in a particular manner for a specific purpose even in the face of failures, i.e., the system is able to maintain the normal functionality. The impact factor indicates the relative importance of the process in the set *S*, while the threshold offers a degree of flexibility for failures and false suspicions, thus allowing a higher tolerance of instability in the system. For instance, in an unstable network, although there might be many false suspicions, depending on the value assigned to the threshold, the system can remain trustworthy. [5]

Having the above characteristics, the *Impact* FD can be applied for 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 systems that provide redundancy of software/hardware; (2) systems that organize nodes with some common characteristic into groups; (3) systems where the nodes can have different importance (relevance) or roles and, thus, their failures may have distinct impact on the system.

Among many, (1) wireless sensor networks (WSNs) that monitor environment conditions or (2) replicated servers that offer some quality of service (QoS) such as bandwidth or response time are two examples of systems where the Impact FD could be applied. In WSN used to collect environmental data, the monitored area can be divided into management zones in accordance with different characteristics. 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. In the second example, if the primary server that offers some quality of server fails, n backup servers could replace it, provided that this set of servers, together, offer the same (or higher) quality of service than the primary one. In such a scenario, the impact factor value of the main server or the sum of the impact factors of the backup servers must be greater or equal than the *threshold* value. If this holds, the system will always be trusted, offering the expected QoS.

We denote *flexibility property* the capacity of the Impact FD to tolerate a certain margin of failures or false suspicions, i.e., its capacity of accepting different set of responses that lead to a trusted state of S. Thus, this article presents two different implementations for the Impact FD tailored to satisfy the *flexibility property*. Both of them use query-response message rounds. The first one is based on the time-free message pattern approach [6] which does not assume bounds on process and communication delays but the relative speed among messages. A process p broadcasts a query message to the nodes of S that it monitors and then waits for responses from α processes or from a set Q of processes whose response satisfy the *trust level*. In the second approach, p can receive responses to its broadcast query from a set of processes within a bounded delay (timely responses).

We also defined two properties, $PR(IT)$ and $PR(\diamondsuit IT)$, that characterize the minimum necessary stability condition of S that ensures confidence (or eventual confidence) on it. In other words, if $PR(IT)$ (resp., $PR(\diamondsuit IT)$) holds, the system S is always (resp., eventually always) trusted for the monitor process p.

Inspired in [7], we have introduced the concept that a process can be $PS{-}accessible$ or $\diamondsuit PS{-}accessible$: 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, i.e., the trust level of S is greater or equal the $threshold^S$ value. In the case of the message pattern approach, this property implies that a query broadcast by p receives responses from a set of process Q which satisfy $threshold^S$ 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 that satisfy the $threshold^S$ 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. Interestingly that, in both implementations the set Q of processes is not fixed, i.e., can be different at each query, which is in accordance with the *flexibility property* of the Impact FD.

This paper is structured as follows. Section 2 outlines some basic concepts of unreliable failure detectors while Section 3 describes the system model. Section 4 describes the Impact Failure Detector and its properties. Section 5 presents two algorithms for the implementation of the Impact FD as well as their proofs of correctness. In Section 6 we discuss some existing related work. Finally, Section 7 concludes the paper and outlines some of our planned future research work.

2. Unreliable Failure Detectors

An important abstraction for the development of fault tolerant distributed systems is the unreliable failure detector [2]. The aim of the latter is to encapsulate the uncertainty of the communication delay between two distributed entities.

An unreliable FD can be seen as an oracle that gives (not always correct) information about process failures (either trusted or suspected). It usually provides a list of processes suspected of having crashed. 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 [8], unreliable FDs are so named because they can make mistakes (1) by erroneously suspecting a correct process (false suspicion), or (2) by not suspecting a process that has actually crashed. If the FD detects its mistake later, it corrects it. For instance, a FD can stop suspecting at time t + 1, a process that it suspected at time t. Although the unreliable FDs can not accurately determine the real state of processes, using them increases knowledge about the processes of the system [2]. This means that the aim of an unreliable FD is to encapsulate the uncertainty of the communication delay between two distributed entities.

Unreliable failure detectors are characterized by two properties, *completeness* and *accuracy*, as defined in [2]. Completeness characterizes the failure detector's capability of suspecting faulty processes, while accuracy characterizes the failure detector's capability of not suspecting correct processes, i.e., restricts the mistakes that the failure detector can make. FDs are then classified according to two completeness properties and four accuracy properties [2].

Notice that the type of accuracy depends on the synchronism or stability of the network. For instance, a strong accuracy requires a synchronous system while an eventual strong one relies on a partially synchronous system which eventually ensures a bound for message transmission delays and processes' speed.

2.1. Implementation of Failure Detectors

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

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

The *message-pattern* strategy does not use any mechanism of timeout. In [6], 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 messages and that one (or a set of nodes) always (or after a time) answers faster to the $QUERY$ than the other nodes.

3. System Model

In this work, we consider that there is one process by node (site) or sensor. Thus, the word process can mean either a node, a sensor, or a site. We consider a distributed system which consists of a finite set [1] of processes $S = \{q_1, \ldots, q_n\}$ with $|S| = n$.

Processes communicate by sending and receiving messages over bi-directional reliable links, i.e., they do not lose, duplicate or corrupt messages and they never generate spurious messages.The network is assumed to be fully connected, i.e., $\Lambda = (p, q)|p, q \in S$.

The system model is asynchronous. It is important to point out that an asynchronous system is characterized by the absence of bounds on process speed and message delay. As result, it is impossible for a process to distinguish a crashed process from a process that is slow or a process which communication is slow. Thus, for the two proposed implementations, the asynchronous system is enriched with new assumptions about synchrony of the links or the relative speed among.

Processes fail by crashing and do not recover. We define a *correct* process the one which never fails during the whole execution; otherwise it is *faulty*. We assume the existence of some global time denoted T. A failure pattern is a function $F : T \rightarrow 2^S$, where $F(t)$ is the set of processes that have failed before or at time t. The function $correct(F)$ denotes the set of correct processes, i.e., those that have never belonged to a failure pattern (F), while $faulty(F)$ denotes the set of faulty processes, i.e., the complement of $correct(F)$ with respect to S.

Besides S, we consider a correct process p ($p \in correct(F)$) that monitors S.

4. Impact Failure Detector

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 that monitors system 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 the set S.

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. 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 set S partitioned into m disjoint subsets where each S_i^* is a set composed of the tuple $\langle id, impact \rangle$, where id is a process identifier and $impact$ is the value of the impact factor of the process in question.

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

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

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

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

The $threshold^S$ is used by the application to check the confidence on the processes of S. If, there is s subsets of S^* such that the $trust\_level_i(t) \geq threshold_i$ $(1 \leq i \leq m)$, S is considered to be *trusted* at t by p, i.e., the confidence of p in S has not been compromised; otherwise S is considered *not trusted* by p at t.

Two points should be highlighted: (1) both the *impact factor* and $threshold^S$ render the estimation of the confidence in S flexible. For instance, it is possible that some processes in S might be faulty or suspected of being faulty but S can still be trusted; (2) the Impact FD can be easily configured to adapt to the needs of the environment. The $threshold^S$ can be tuned to provide a more restricted or softer monitoring.

In Table 1, we consider a set S, where S^* is 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 $threshold^S$ define that the subsets S_1, S_2 and S_3 must have at least two

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

correct processes. The table shows several possible outputs for FD (${I_p}^S$): the set S is considered trusted whenever, for each subset S_i^*, $trust\_level_i(t) \geq threshold_i$.

TABLE 1. EXAMPLE OF FD (${I_p}^S$) OUTPUT: S^* HAS THREE SUBSETS

t	F(t)	${trusted_p}^S(t)$	${trust\_level_p}^S$	Status
1	$\{q_2\}$	$\{\{\langle q_1,1\rangle,\langle q_3,1\rangle\},$ $\{\langle q_4,2\rangle,\langle q_5,2\rangle,\langle q_6,2\rangle\},$ $\{\langle q_7,3\rangle,\langle q_8,3\rangle,\langle q_9,3\rangle\}\}$	$\{2,6,9\}$	Trusted
2	$\{q_2,q_5\}$	$\{\{\langle q_1,1\rangle,\langle q_3,1\rangle\},$ $\{\langle q_4,2\rangle,\langle q_6,2\rangle\},$ $\{\langle q_7,3\rangle,\langle q_8,3\rangle,\langle q_9,3\rangle\}\}$	$\{2,4,9\}$	Trusted
3	$\{q_2,q_5,q_6\}$	$\{\{\langle q_1,1\rangle,\langle q_3,1\rangle\},$ $\{\langle q_4,2\rangle\},$ $\{\langle q_7,3\rangle,\langle q_8,3\rangle,\langle q_9,3\rangle\}\}$	$\{2,2,9\}$	Not Trusted

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

4.1. Flexibility of the Impact FD

The *flexibility* of the Impact FD characterizes its capacity of accepting different set of responses that lead to a trusted state of S. Let PS be the set that contains all possible subsets which satisfy a defined *threshold*:

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

Initially, the *TPowerSet* function generates the power set [2] for each subset (S_i) of S. Then, only the subsets of S_i whose sum of their parts is greater than, or equal to, $threshold_i$ are selected. Following this, the Cartesian product is applied to generate all possible combinations [3], i.e., all the subsets generated satisfy the $threshold^S$.

Let's consider the following example:

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

PowerSet(S_1^*, $threshold_1$) = $\{\{\langle q_1,1\rangle\},\{\langle q_2,1\rangle\},\{\langle q_1,1\rangle,\langle q_2,1\rangle\}\}$
PowerSet(S_2^*, $threshold_2$) = $\{\{\langle q_3,1\rangle\},\{\langle q_4,1\rangle\},\{\langle q_3,1\rangle,\langle q_4,1\rangle\}\}$
PowerSet(S_3^*, $threshold_3$) = $\{\{\langle q_5,1\rangle\},\{\langle q_6,1\rangle\},\{\langle q_5,1\rangle,\langle q_6,1\rangle\}\}$
PS = PowerSet(S_1^*, threshold$_1$) × PowerSet(S_2^*, threshold$_2$) × PowerSet(S_3^*, threshold$_3$)

PS = PowerSet(S_1^*, threshold$_1$) × PowerSet(S_2^*, threshold$_2$) × PowerSet(S_3^*, threshold$_3$)
PS = $\{\{\{\langle q_1,1\rangle\},\{\langle q_3,1\rangle\},\{\langle q_5,1\rangle\}\},\{\{\langle q_1,1\rangle\},\{\langle q_3,1\rangle\},\{\langle q_6,1\rangle\}\},$
$\{\{\langle q_1,1\rangle\},\{\langle q_3,1\rangle\},\{\langle q_5,1\rangle,\langle q_6,1\rangle\}\},\{\{\langle q_1,1\rangle\},\{\langle q_4,1\rangle\},\{\langle q_5,1\rangle\}\},$
$\{\{\langle q_1,1\rangle\},\{\langle q_4,1\rangle\},\{\langle q_6,1\rangle\}\},\{\{\langle q_1,1\rangle\},\{\langle q_4,1\rangle\},\{\langle q_5,1\rangle,\langle q_6,1\rangle\}\},$
$\{\{\langle q_1,1\rangle\},\{\langle q_3,1\rangle,\langle q_4,1\rangle\},\{\langle q_5,1\rangle\}\},$
$\{\{\langle q_1,1\rangle\},\{\langle q_3,1\rangle,\langle q_4,1\rangle\},\{\langle q_6,1\rangle\}\},$
$\{\{\langle q_1,1\rangle\},\{\langle q_3,1\rangle,\langle q_4,1\rangle\},\{\langle q_5,1\rangle,\langle q_6,1\rangle\}\},\ldots\}$

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

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

For instance, if ${trusted_p}^S(t_1) = \{\{\langle q_1,1\rangle\},\{\langle q_3,1\rangle\},\{\langle q_5,1\rangle\}\}$ and ${trusted_p}^S(t_2) = \{\{\langle q_1,1\rangle\},\{\langle q_3,1\rangle,\langle q_4,1\rangle\},\{\langle q_6,1\rangle\}\}$, p considers that the system S is trusted at both t_1 and t_2.

4.2. Some properties

Similarly to many existing works (see Section 6), we define some properties and process/link behaviors in order to introduce some synchrony on the asynchronous system.

Considering the set PS, which characterizes the flexibility of the Impact FD, we define the following properties:

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

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

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

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

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

Inspired by the concept that a process is $\Diamond f{-}accessible$ proposed in [7] (see Section 6), we also define the concept of a $PS{-}accessible$ and a $\Diamond PS{-}accessible$ process:

A process $p \in correct(F)$ is $PS{-}accessible$ (resp., $\Diamond PS{-}accessible$) 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., $\Diamond PS{-}accessible$), $PR(IT)$ (resp., $PR(\Diamond IT)$) 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 first $RESP$ messages received by p to this query are denoted *winning* responses [6], [9].

A process p is $PS{-}accessible$ (resp., $\Diamond PS{-}accessible$) if for $\tau_0 \geq 0$ (resp., $\exists \tau_0, \forall \tau \geq \tau_0$) there exists a set Q of processes such that $Q \in PS$ 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., $\Diamond PS{-}accessible$) if for $\tau_0 \geq 0$ (resp., $\exists \tau_0, \forall \tau \geq \tau_0$) there exists a set Q of processes such that $Q \in PS$ 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 $\diamondsuit PS{-}accessible$, the set $Q(\tau)$ is not fixed and can be different at distinct times which is in accordance with the *flexibility* property of the Impact FD.

5. Implementations of Impact FD

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.

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, $\forall q \in S$. At each new round, p broadcasts a *QUERY* message. The time interval between two consecutive rounds is finite and arbitrary (resp., bounded) for the message pattern (resp., timer-based) implementation. When process q receives a *QUERY* message from p, q confirms its reception with a *RESP* message.

The following local variables are used in both algorithms:

- r_p: round counter of process p;
- *trusted*: it is the set formed by $\{trusted_1, \ldots, trusted_m\}$ (m the number of subsets of S), where each $trusted_i$ ($1 \leq i \leq m$) contains the processes of S_i that are not considered faulty by p. Each $trusted_i$ is a set composed of the tuple $\langle id, impact \rangle$, where id is the process identifier and $impact$ is the value of the impact factor of the process that belongs to the subset $trusted_i$.
- *trust_level*: the set that contains the trust level of each subset of processes;
- *PS*: the set comprising all possible subsets formed by processes of S, where the sum of its elements is greater than, or equal to, the $threshold^S$.

We also defined the function $Add()$ which is used in both algorithms:

- *Add(set, subset,* $\langle q, impact \rangle$*)*: function that adds the process q, with impact factor $impact$ to the subset $subset$ of the set set provided that q is not already in $subset$.

5.1. Message-pattern Implementation

Algorithm 1 presents the message pattern approach implementation of the Impact FD of process p with respect to S.

Process p receives as input the set of process of the system S, the number of subsets of S^* (m), 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 threshold value for the number of messages to wait from the processes of subset S_i^*. For instance, if f_i denotes the maximum number of failures of processes of subset S_i^*, $\alpha_i \leq |S_i^*| - f_i$ (for $i = 1$ *to* m).

Algorithm 1 Message pattern implementation

```
1:  Begin
    Input
2:      S, m, threshold^{S*}, α

    Init
3:      r_p ← 0
4:      for i = 1 to m do                          ▷ for each subset
5:          trusted_i ← S_i
6:          tmp_trusted_i ← ∅
7:          trust_level_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| ≥ α_i ∀i ∈ [1,m]) or (tmp_trusted ⊆ PS)
13:         trusted ← tmp_trusted
14:         for i = 1 to m do                      ▷ for each subset
15:             tmp_trusted_i ← ∅
16:         end for
17:         r_p ← r_p + 1
18:     end loop

    Task T2
19:     Upon reception of (RESP, r, I_q, subset_q) from q do
20:         if r = r_p then
21:             Add(tmp_trusted, subset_q, ⟨q, I_q⟩)
22:         end if
23:     end

    Task T3
24:     Upon invocation of Impact() do
25:         for i = 1 to m do                      ▷ for each subset
26:             trust_level_i ← sum(trusted_i)
27:         end for
28:         return trust_level
29:     end

    Task T4                                        ▷ executed by q ∈ S
30:     Upon reception of (QUERY, r_p) from p do
31:         send(RESP, r_p, I_q, subset_q) to p
32:     end
33: End
```

The algorithm has four tasks. Tasks T1, T2, and T3 are executed by p, while T4 is executes by $q \in S$.

At the initialization, $trusted$ is initialized with the nodes 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 $threshold^{S*}$.

The variable $tmp\_trusted$, at every query round, gathers the identification of processes that answered to the current query.

Task T1 of p has an infinite loop. Firstly it sends the 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 that satisfy the $threshold^{S*}$) (line 12). Finally, the round counter (r_p) is incremented (line 17).

Task T2 handles the reception of messages *(RESP, r, I_q $subset_q$)* sent by q. The message contains the round, the impact factor and the subset to which q belongs. If round r is equal to r_p, then q is added to the $trusted_q$ set.

Task T3 handles the invocation of the *Impact()* function (line 24), which computes the *trust_level* of each subset and returns the *trust_level* of the trusted processes (line 28).

Task T4 is responsible for the reception of messages *QUERY* by process $q \in S$. When q receives a *(QUERY, r)* message from the monitor process p (line 30), q must respond with a *RESP* message containing the round, its impact factor, and the number of its subset (line 31).

Sketch of Proof

Lemma 1. *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 of nodes included in $tmp\_trusted$ by Task T2 within round r_p are 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 which will render the first condition true, and p will be unblocked. In other words, since α_i is bounded by n and f_i and no query or response messages are lost, such a condition always ensures the progress of the failure detector. □

Lemma 2. *At every query r_p issued by p, $trusted_p$ is updated.*

Proof. From Lemma 1, Task 1 never blocks and, thus, line 13 is always executed. This line is the only point where $trusted$ is updated after initialization (line **??**). Hence, at every query round, $trusted_p$ is updated. □

Lemma 3. *If p is $PS{-}accessible$ (resp., $\Diamond PS{-}accessible$), Algorithm 1 ensures that $PR(IT)$ (resp., $PR(\Diamond IT)$) holds for p.*

Proof. For $t = 0$, $trusted_p = S$. Let Q(r) be the set of winning responses and r_0 be the first round where $Q(r_0) \in PS$. Thus, for r_0, $trusted = tmp\_trusted \subseteq PS$. Let then consider that $\forall r \geq r_0$, $Q(r) \in PS$ which characterize the $PS - accessibility$ of p. In this case, since from Lemma 2, $trusted$ is updated at every query round, $trusted = tmp\_trusted \subseteq PS$. Let t be the time when $trusted$ is updated in round r_0 (line **??** or line 13). Therefore, $\forall t' \geq t$, when the impact FD is invoked, $PR(IT)$ (resp., $PR(\Diamond IT)$) holds.

□

Theorem 1. *If p is $PS{-}accessible$ (resp., $\Diamond PS{-}accessible$), $\forall t \geq 0$ (resp. $\exists t' \in T, \forall t \geq t$), $trust\_level(t)_p \geq threshold$.*

Proof. The proof follows directly from Lemma 3. □

5.2. Timer-based Implementation

Algorithm 2 shows a timer-based implementation of the Impact FD of process p with respect to S.

For this implementation we added 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. If $p \in correct(F)$ is $PS{-}accessible$ (resp., $\Diamond PS{-}accessible$) it always (resp. eventually always) receives $RESP$ messages from a set Q of processes $Q \in PS$, in a delay of time smaller than δ.

Process p monitors the processes of S and receives as input the set S, the number of subsets m, the $threshold^{S*}$, the interval time ($\Delta > \delta$) to send the broadcast message and the upper bound on the round-trip delay of messages (δ) for timely links.

The algorithm has five tasks. Tasks T1, T2, T3, and T5 are executed by p, while T4 is executes by $q \in S$.

At the initialization, each subset of $trusted$ and $trust\_level$ is initialized as empty (lines 3 - 4). 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 10-13). Periodically (interval of Δ time units), process p sends to the processes in S a $QUERY$ message (line 14) and starts the timer $timeout$ (line 15).

In Task T2, when process p receives a $RESP$ message sent by q, if round r is equal to r_p and q is not in $tmp\_trusted$, q is added to $tmp\_trusted$ (lines 17-18). At this point, if $tmp\_trusted$ is a subset of PS (i.e., contains processes that satisfy the $threshold^{S*}$) (line 20), then the variable $tmp\_trusted$ is assigned to $trusted$ and the $timeout$ is stopped.

Task T3 and T4 are the same of Algorithm 1 : task T3 handles the invocation of the *Impact()* function by p, which returns the sum of impact factor of the trusted processes (line 25); in Task T4, process $q \in S$ receives a *(QUERY, r)* message from p (line 31), q answers with a *RESP* message containing the the round, its impact factor, and the number of its subset (line 32).

Algorithm 2 Timer-based implementation

1: **Begin**

Input
2: S, m, $threshold^{S*}$, Δ, δ
Init
3: **for** $i = 1$ *to* m **do** ▷ for each subset
4: $trusted_i \leftarrow \emptyset$
5: $trust\_level_i \leftarrow \emptyset$
6: **end for**
7: $PS \leftarrow TPowerSet(S, threshold^{S*})$
8: $r_p \leftarrow -1$
9: $timeout \leftarrow \delta$

Task T1 - Repeat forever every Δ time unit
10: **for** $i = 1$ *to* m **do** ▷ for each subset
11: $tmp\_trusted_i \leftarrow \emptyset$
12: **end for**
13: $r_p \leftarrow r_p + 1$
14: $broadcast(QUERY, r_p)$
15: $start\ timeout$

Task T2
16: **Upon reception of** $(RESP, r, I_q, subset_q)$ from q **do**
17: **if** $r = r_p$ **then**
18: $Add(tmp\_trusted, subset_q, \langle q, I_q \rangle)$
19: **end if**
20: **if** $tmp\_trusted \in PS$ **then**
21: $Stop\ timeout$
22: $trusted \leftarrow tmp\_trusted$
23: **end if**
24: **end**

Task T3
25: **Upon invocation of** $Impact()$ **do**
26: **for** $i = 1$ *to* m **do** ▷ for each subset
27: $trust\_level_i \leftarrow sum(trusted_i)$
28: **end for**
29: $return\ trust\_level$
30: **end**

Task T4 ▷ executed by $q \in S$
31: **Upon reception of** $(QUERY, r_p)$ from p **do**
32: $send(RESP, r_p, I_q, subset_q)$ to p
33: **end**

Task T5 - When timeout expires
34: **Upon** $expiration\ of\ timer$ **do**
35: $trusted \leftarrow tmp\_trusted$
36: **end**
37: **End**

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

Sketch of Proof

Lemma 4. *At every query r_p issued by p, $trusted_p$ is updated.*

Proof. At every new query rp in task T1, p starts a timer (line 15). As $\Delta > \delta$, a new query will not be issued before the timer expires or is stopped. If the set of Q messages received for this query are such that $Q \in PS$, the timer is stopped and $trusted_p = Q$ by Task T2 (lines 21 - 22). Otherwise, the timer will expires (Task T5) and $trusted_p$ will be updated with the set of processes that sent $RESP$ messages to p (line 35) within a delay of δ . □

Lemma 5. *If p is $PS{-}accessible$ (resp., $\Diamond PS{-}accessible$), Algorithm 2 ensures that $PR(IT)$ (resp., $PR(\Diamond IT)$) holds for p.*

Proof. From Lemma 4, $trusted_p$ is updated at every query. Let $t_0 \in T$ be the time where for every query issued by p at $t \geq t_0$, a $QUERY$ message broadcast by p at t receives a $RESP$ message from processes of a $Q(t) \in PS$ within $t+\delta$. In this case, the timer started in every query at line 15 will never expire and, therefore, line 21 will be never executed $\forall t' \geq t$. On the other hand, since $Q(t) \in PS$, the test of line 20 is always true. Hence $\forall t' \geq t$,, when the impact FD is invoked $PR(IT)$ (resp., $PR(\Diamond IT)$) holds for p. □

Theorem 2. *If p is $PS{-}accessible$ (resp., $\Diamond PS{-}accessible$), $\forall t \geq 0$ (resp. $\exists t' \in T, \forall t \geq t$), $trust\_level(t)_p \geq threshold$.*

Proof. The proof follows directly from Lemma 5. □

6. Related Work

We can divide related studies of the literature into two groups: (1) unreliable failures detectors and (2) works which consider additional assumptions for asynchronous systems.

Unreliable failure detectors: Most of the unreliable fault detectors in the literature are based on a binary model and provide as output a set of process identifiers, which usually informs the set of processes currently suspected of having failed ([2] [4]). However, in some detectors, such as class Σ (resp., Ω) [10], the output is the set of processes (resp., one process) which are (resp., is) not suspected of being faulty, i.e., $trusted$.

The ϕ Accrual failure detector [11] proposes an approach where the output is a suspicion level on a continuous scale, rather than providing information of a binary nature (trusted or suspected). It is based on an estimation of inter-arrival times which assuming that the latter follow a normal distribution. 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. In [12], the authors extended the Accrual FD by exploiting histogram density estimation. Taking into account a sampled inter-arrival time and the time of the last received heartbeat, the algorithm estimates the probability that no further heartbeat messages arrive from a given process, i.e., it has failed. The aim of Accrual failure detectors is to decouple monitoring from interpretation.

Starting from the premise that applications should have information about failures to take specific and suitable recovery actions, the work in [13] proposes a service to report faults to applications. The latter also encapsulates uncertainty which allows applications to proceed safely in the presence of doubt. The service provides status reports related to fault detection with an abstraction that describes the degree of uncertainty.

Considering that each node has a probability of being byzantine, a voting node redundancy approach is presented in [14] in order to improve reliability of distributed systems. Based on such probability values, the authors estimate the minimum number of machines that the system should have in order to provide a degree of reliability which is equal to or greater than a threshold value.

In [15], 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 to increase its quality in terms of detection. When classifying the behavior of the nodes, it includes a reputation service where the nodes periodically exchange heartbeat messages.

Additional assumptions for asynchronous systems: Several works have tried to circumvent the impossibility of the consensus in pure asynchronous systems in the presence of failures [16]. Therefore, the challenge is to identify properties that can be satisfied "nearly always" by the underlying asynchronous system enriched by some assumptions which allow an algorithm, for instance Ω (which can then be used to solve consensus in a system with a majority of correct processes) to be implemented during the "periods" in which the properties are satisfied [17].

The Message Pattern approach does not assume eventual bounds on process and communication delays. In [6], 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.

Aguilera et al. introduced the $\diamondsuit f - source$ assumption in [5] aiming at providing communication-efficient leader and consensus protocol implementations. 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.

In [7], the authors introduce the notion of eventual $\diamondsuit f{-}accessible$. A process p is eventual $f{-}accessible$ if there is a time τ_0 such that, at any time $\tau \geq \tau_0$, there is a set $Q(\tau)$ of f processes such that $p \notin Q(\tau)$ and a message broadcast by p at τ receives a response from each process of $Q(\tau)$ by time $\tau + \delta$ (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.

In [18] and [19], 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.

7. Conclusion and Future Work

In this paper we have shown the flexibility capacity of our Impact FD provided by the *impact factor* and the *threshold* which enables the user to define the importance (e.g., degree of reliability) of each node and an acceptable margin of failures respectively.

We have defined two properties, $PR(IT)$ and $PR(\diamondsuit IT)$, which denote the capacity of the Impact FD for accepting different set of responses that lead to a trusted state of the system S as well as the concept of a $PS{-}accessible$ (resp., $\diamondsuit PS{-}accessible$) process such that every $QUERY$ message sent by this process obtains from the beginning (resp., eventually) a set Q of responses that satisfy the degree of confidence in S ($threshold_S$). Interestingly, that in both definitions of $PS{-}accessible$ and $\diamondsuit PS{-}accessible$, the set Q is not fixed and can be different at distinct times which is in accordance with the *flexibility* property of the Impact FD.

Then, we have presented two algorithms that implement the Impact FD tailored to exploit its flexibility feature. The first algorithm is 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 $threshold^S$. The second algorithm assumes that for every query issued by p, the latter can receive timely responses. We have proved that, for both algorithms, if the monitoring

process p is $PS{-}accessible$ (resp., $\Diamond PS{-}accessible$) the system S is always (eventually always) trusted by p.

As future work, we intend to extend the Impact FD so that it could be able to address dynamic impact factor values, i.e., they can vary during execution, which express the degree of reliability of the node (or reputation). We are also working on the reduction and equivalence of Impact FD in regard with other detectors (e.g., Sigma and Omega [13]), which will require some new assumptions and/or new definitions. In [1], we presented some performance evaluation results with a timer-based implementation of the Impact FD and traces collected from execution that we conducted on the PlanetLab platform [20] that include messages losses and failures of nodes. However, we still aim to conduct further performance experiments on different networks such as WiFi or LAN and, thus, comparing the Impact FD with other well-known failure detectors.

References

[1] A. G. Rossetto, C. F. Geyer, L. Arantes, and P. Sens, "A failure detector that gives information on the degree of confidence in the system," in *2015 IEEE Symposium on Computers and Communication*, 2015, pp. 532–537.

[2] T. D. Chandra and S. Toueg, "Unreliable failure detectors for reliable distributed systems," *Journal of the ACM (JACM)*, vol. 43, no. 2, pp. 225–267, 1996.

[3] W. Chen, S. Toueg, and M. K. Aguilera, "On the quality of service of failure detectors," *Computers, IEEE Transactions on*, vol. 51, no. 5, pp. 561–580, 2002.

[4] M. Bertier, O. Marin, P. Sens *et al.*, "Performance analysis of a hierarchical failure detector." in *DSN*, vol. 3, 2003, pp. 635–644.

[5] M. K. Aguilera, C. Delporte-Gallet, H. Fauconnier, and S. Toueg, "Communication-efficient leader election and consensus with limited link synchrony," in *Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing*. ACM, 2004, pp. 328–337.

[6] A. Mostefaoui, E. Mourgaya, and M. Raynal, "Asynchronous implementation of failure detectors," in *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE Computer Society, 2003, pp. 351–351.

[7] D. Malkhi, F. Oprea, and L. Zhou, " meets paxos: Leader election and stability without eventual timely links," in *Distributed Computing*. Springer, 2005, pp. 199–213.

[8] N. Hayashibara, X. Défago, and T. Katayama, "Two-ways adaptive failure detection with the ϕ-failure detector," in *Workshop on Adaptive Distributed Systems (WADiS03)*. Citeseer, 2003, pp. 22–27.

[9] M. Raynal, "Eventual leader service in unreliable asynchronous systems: Why? how?" in *Network Computing and Applications, 2007. NCA 2007. Sixth IEEE International Symposium on*. IEEE, 2007, pp. 11–24.

[10] C. Delporte-Gallet, H. Fauconnier, R. Guerraoui, V. Hadzilacos, P. Kouznetsov, and S. Toueg, "The weakest failure detectors to solve certain fundamental problems in distributed computing," in *Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing*. ACM, 2004, pp. 338–346.

[11] N. Hayashibara, X. Defago, R. Yared, and T. Katayama, "The φ accrual failure detector," in *Reliable Distributed Systems, 2004. Proceedings of the 23rd IEEE International Symposium on*. IEEE, 2004, pp. 66–78.

[12] B. Satzger, A. Pietzowski, W. Trumler, and T. Ungerer, "A new adaptive accrual failure detector for dependable distributed systems," in *Proceedings of the 2007 ACM symposium on Applied computing*. ACM, 2007, pp. 551–555.

[13] J. B. Leners, T. Gupta, M. K. Aguilera, and M. Walfish, "Improving availability in distributed systems with failure informers," in *Proc. of NSDI*, 2013.

[14] Y. Brun, G. Edwards, J. Y. Bang, and N. Medvidovic, "Smart redundancy for distributed computation," in *Distributed Computing Systems (ICDCS), 2011 31st International Conference on*. IEEE, 2011, pp. 665–676.

[15] M. Véron, O. Marin, S. Monnet, and P. Sens, "Repfd-using reputation systems to detect failures in large dynamic networks," in *44th International Conference on Parallel Processing (ICPP-2015)*, 2015.

[16] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of distributed consensus with one faulty process," *Journal of the ACM (JACM)*, vol. 32, no. 2, pp. 374–382, 1985.

[17] A. F. Anta and M. Raynal, "From an asynchronous intermittent rotating star to an eventual leader," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 21, no. 9, pp. 1290–1303, 2010.

[18] L. Arantes, F. Greve, P. Sens, and V. Simon, "Eventual leader election in evolving mobile networks," in *Principles of Distributed Systems*. Springer, 2013, pp. 23–37.

[19] F. Greve, P. Sens, L. Arantes, and V. Simon, "Eventually strong failure detector with unknown membership," *The Computer Journal*, vol. 55, no. 12, pp. 1507–1524, 2012.

[20] PlanetLab, "Planetlab," http://www.planet-lab.org, 2014, online. Acessado 16-setembro-2014.