Stab-FD: a cooperative and adaptive failure detector for wide area networks

Pierre Sens[a], Luciana Arantes[a], Anubis Graciela De Moraes Rossetto[b], Olivier Marin[c]

[a]*Sorbonne University, CNRS, LIP6, 4 Place Jussieu, Paris, F-75005, France*
[b]*Federal Institute Sul-rio-grandense (IFSUL), Passo Fundo, Brazil*
[c]*Computer Science, NYU Shanghai, Shanghai, China*

Abstract

Failure detectors (FDs) are a fundamental abstraction that plays a central role in the design of distributed systems. FDs are distributed oracles that provide processes with unreliable information about process failures, often in the form of a list of trusted or suspected process identities. In this article, we propose a timer-based FD which assesses the quality of its input links, and exchanges its local estimations with other nodes. Nodes use this information to adjust their timers dynamically. Capturing the variations in the quality of each link reduces the number of false suspicions without degrading failure detection time. We present experiments on a dataset of real traces collected on PlanetLab, and compare our approach to well-known state-of-the-art algorithms. Our results show that our new algorithms yield a good trade-off in terms of failure detection speed and accuracy in real scenarios.

Keywords: Failure Detectors, Quality of Service, Fault Tolerance, Distributed Algorithms, Reliability

1. Introduction

Many distributed systems must handle failures in order to ensure reliable and continuous services. Some critical services, such as distributed storage

Email addresses: Pierre.Sens@lip6.fr (Pierre Sens), Luciana.Arantes@lip6.fr (Luciana Arantes), anubisrossetto@gmail.com (Anubis Graciela De Moraes Rossetto), ogm2@nyu.edu (Olivier Marin)

systems [1], [2], [3] replicate their state across geo-distributed nodes to remain available in the event of a region-wide system outage. These services often use State Machine Replication [4] based on consensus algorithms [5] to maintain the consistency between replicas despite failures. The performance of these services directly affect the user-perceived latency. More generally, in a cloud computing context, some of the servers spread on several datacenters may be available while others are heavy loaded or crashed and, in this case, users might have to wait for available servers to continue executing their respective application. In order to tolerate failures or temporal unavailability, distributed services must use a mechanism that detects node failures and a second one that guarantees the correct execution of the system/application despite of failures.Therefore, the first mechanism, failure detection, is crucial in the engineering of such services.

A failure detector (FD) is an oracle which provides information about process crashes [6]. It is unreliable because it can make mistakes, for instance by erroneously suspecting a non faulty node (false suspicion), or by not suspecting a process that has actually crashed. The FD can eventually correct detection mistakes.

The literature about FDs comprises several implementations [7, 8, 9, 10, 11, 12, 13, 14]. Most of these are timer-based FDs where nodes exchange heartbeat messages and wait for them within a time bound (timeout). For every node it monitors, the FD keeps track of the arrival time of the last n heartbeats in a sliding window; it then uses this log to estimate the arrival time of the next heartbeat. If the expected heartbeat does not arrive within the estimated arrival time, the FD suspects a failure of the monitored node. By adjusting the estimated arrival time of heartbeats, FDs try to reach a good trade-off between the speed of the detection (completeness property) and the avoidance of false suspicions (accuracy property).

Chen et. al's FD [7] was the first FD implementing a sliding window to estimate the arrival time of the node's next hearbeat. To mitigate the impact of sudden network latency slowdowns on false detections, the authors extend the arrival time estimation with a constant safety margin. Later FDs such as the ones presented in [8, 14, 15] also use heartbeats and sliding windows, but compute the safety margin dynamically. In [8], the computation of the safety margin exploits Jacobson's algorithm: the round-trip time estimation used in the TCP protocol. In [14], the authors propose two safety margin adaptation strategies: one relies on the prediction of error changes, and the other on the probability of correctness of the next arrival time estimate.

In [15] QoS parameters are self-configured, including the safety margin, by exploiting feedback control theory. All of these FDs provide good results in LAN environments where network conditions are relatively stable, yet they do not perform well over WANs where network conditions change fast due to traffic bursts. Furthermore, every FD relies on its own view to adapt the dynamic safety margin, even though another FD might have a more accurate view of the monitored node.

We propose a new cooperative failure detector, called Stab-FD, which observes the stability of communication links to adapt the safety margin. By detecting sudden delay variations in network links, Stab-FD dynamically calibrates the safety margin. This approach reduces the number of failure suspicions providing good accuracy, without degrading the failure detection time significantly.

A Stab-FD node (observer) assesses the stability of its input links from every monitored node. The observer uses these stability estimates to calibrate the timeout values. However, in a WAN context, the quality of links are not homogeneous and a well-connected node can have a better link quality to observe a node n relatively to some other nodes far from n. In order to improve the quality of detection, observer nodes also include their local view of suspected nodes and information about stability of their input links in their heartbeat messages to compare their observed relative quality of links. Based on this information, an observer node can better deduce the accuracy of its current safety margin with respect to a monitored node, and re-calibrate the safety margin when it receives a more precise view about the monitored node's liveness.

To evaluate the quality of service (QoS) of our FD, we conduct experiments over real traces collected on PlanetLab [16] using UDP heartbeat messages, and compare it with Chen [7] and Bertier [8] FDs, two well-known FDs in the literature. Our results show that our approach outperforms its predecessors according to the traditional evaluation metrics for FDs: false detection avoidance, query accuracy probability, and detection time.

In summary, aiming at reducing detection time and improving application stability, the main contribution of Stab-FD is to combine both dynamic adaptation of timers, thanks to a dynamic safety margin computation, and an opportunistic cooperation where nodes piggyback in heartbeat messages information about their local view of suspected nodes and stability of their input links.

The rest of this paper is organised as follows. Section 2 introduces our

model and provides some background about Chen's and Bertier's failure detectors. Section 3 describes Stab-FD and its algorithms. Section 4 presents our evaluation results. Section 5 discusses some related work, and we conclude in Section 6.

2. Model and Background

We consider a distributed system which consists of a finite set of N processes $\Pi = \{p_1, \ldots, p_N\}$, with one process per node. Processes can fail by crashing and can recover with the same *id*. We assume no bounds on message latencies [17] and that local clocks have bounded drift. In such systems, it is impossible to precisely determine whether a remote process has failed or whether it is just very slow [18]. Processes associate an increasing sequence number with each message emission to allow for the detection of message loss. For the current implementation, we consider that there exists a link $l_{p \to q}$ from every process p to every other process q.

Chen's FD relies on a heartbeat strategy where every non faulty process periodically sends an 'I am alive' message to other processes. The algorithm estimates the expected arrival time EA for the next heartbeat message, then uses EA to compute a *freshness point*. The freshness point time determines the moment the failure detector of a node p will start to suspect that a monitored node q has crashed if no message was received from q. The formula for the next freshness point $l+1$ is:

$$\tau_{l+1} = EA_{l+1} + \alpha \tag{1}$$

where α is a constant safety margin chosen by the user.

To compute EAs, Chen's FD maintains a sliding window of size n for every monitored node, with information regarding the n previous heartbeats. Let $s_1, \ldots, s_n$ be the sequence number of those heartbeats, and $A_1, \ldots, A_n$ their reception times at p. The estimation of the expected arrival time EA_{l+1} is:

$$EA_{l+1} \approx \frac{1}{n}\left(\sum_{i=1}^{n} A_i - \eta.s_i\right) + (l+1)\eta \tag{2}$$

where η is the heartbeat sending interval. This equation first normalises each A_i with a backwards shift of $\eta.s_i$ time units, then computes the average of the A'_is, and finally shifts it forward by $(l+1)\eta$.

In [8], Bertier et al. use the same mechanism as Chen for estimating expected arrival times (see Equation 1). The computation of EA also relies on a sliding window of the n previous heartbeat arrival dates. Their approach differs in that it computes freshness points in a dynamic way. It uses Jacobson's estimation [19] to adapt the safety margin upon every reception of a heartbeat. The adaptation of the safety margin α is a function of the *error* in the last estimation. Parameter γ represents the importance of a new measure with respect to the previous ones. The *delay* represents the estimate margin, and *var* the magnitude between errors. β and ϕ ponder the variance. Upon the reception of a heartbeat at time A, the estimation of α is computed as follows:

$$\begin{aligned} error &= A - EA - delay \\ delay &= delay + \gamma \cdot error \\ var &= var + \gamma \cdot (|error| - var) \\ \alpha &= \beta \cdot delay + \phi \cdot var \end{aligned}$$

3. Stable FD

In this section we introduce our *Stable Failure Detector* (*Stab*-FD) and its cooperative version $Stab^C$-FD.

Stab-FD and $Stab^C$-FD are heartbeat-based FDs that exploit both the heartbeat arrival estimation strategy of Chen's FD and the variation conditions of the network links. To this end, they observe the quality of each input link to obtain estimations about the stability of remote nodes. A node q is considered stable by p if q's heartbeat messages always arrive before the expiration of the timer associated with q. If such is not the case, for instance because of network delays or message losses, p will start viewing q as less stable. Thus, p should dynamically adapt its estimation of q's stability value in order to take into account network condition variations. Furthermore, as an attempt to better capture the current quality of input links, p includes q's stability value in the computation of the safety margin (Chen's FD) to calibrate the timer. Finally, since it is not possible to obtain information regarding arrival times of lost messages, we apply a uniform distribution of expected arrival times to fill in the gaps induced by adjacent message losses in Chen's FD sliding window.

Algorithms 1 and 2 actually correspond to $Stab^C$ FD, our extended cooperative version of *Stab* FD. To study the non cooperative *Stab* algorithm,

the reader may disregard all lines prefixed by $*$ and all underlined variables. We first describe *Stab* and the variables it uses, and then move on to the cooperative version $Stab^C$.

3.1. Stab-FD

p uses the following local variables and parameters:

- Δ_H: the frequency of heartbeat message emissions (an input parameter of the algorithm).
- *suspected*: the set of processes currently suspected of being faulty.
- *timer*: a vector where $timer[q]$ keeps the maximum delay that p will wait for the next heartbeat from q.
- *stab*: a vector where $stab[q]$ is p's estimation of the current stability value of $l_{q \to p}$, its input link from q.
- S_{init}: the initial stability value of all input links (an input parameter of the algorithm).
- *mist*: a vector where $mist[q]$ accumulates the number of mistakes that p has made about node q.

Algorithm 1 Stable Failure Detector algorithm for node p

1: **Initialization**
2: **for** each node $q \neq p$ in Π **do**
3: $stab[q] \leftarrow S_{init}$
4: $mist[q] \leftarrow 0$; $mist_{prev}[q] \leftarrow 0$
5: $count[q] \leftarrow 0$
6: $timer[q] \leftarrow \Delta_H$
7: start $timer[q]$
*8: $rmist[q] \leftarrow 0$; $rcount[q] \leftarrow 0$
*9: $\Delta_{RS}[q] \leftarrow \Delta_{init}$
10: **end for**
11: $suspected \leftarrow \emptyset$
12: $nmsg \leftarrow 0$
*13: **for** each node q in Π **do**
*14: **for** each node r in Π **do**
*15: $ldet[q][r] \leftarrow 0$
*16: **end for**
*17: **end for**

18: **Task** T1 every Δ_H [HeartBeat sending]
19: Send HB($nmsg, \underline{suspected}, \underline{stab}$) to $\Pi - \{p\}$
20: $nmsg \leftarrow nmsg + 1$
21: **End** of Task T1

22: **Task** T2 [timer expiration]
Upon expiration of timer[q]
23: **if** $q \notin suspected$ **then**
24: $suspected \leftarrow suspected \cup \{q\}$
*25: $detnode[q] \leftarrow \emptyset$
26: **end if**
27: **End** of Task T2

28: **Task** T3 [Heartbeat reception]
29: Upon reception of
30: HB($nh, \underline{suspected_q}, \underline{stab_q}$) from q at time t
31: stop $timer[q]$
32: **if** $q \in suspected$ **then**
▷ False suspicion of q
33: $suspected \leftarrow suspected - \{q\}$
34: $mist[q] \leftarrow mist[q] + 1$
*35: **for** each node in $r \in detnode[q]$ **do**
▷ False remote suspicion
*36: $rmist[r] \leftarrow rmist[r] + 1$
*37: **end for**
38: **end if**
39: $UpdateWindow(WA[q], t, nh)$
40: $EA[q] \leftarrow ComputeEA(WA[q])$
41: $count[q] \leftarrow count[q] + 1$
42: $timer[q] \leftarrow EA[q] + ComputeMargin(q)$

43: start $timer[q]$
▷ Remote detection of failure
*44: **for** each node $r \neq p \in suspected_q$ **do**
*45: **if** $stab_q[r] > (1 + \Delta_{RS}[q]) * stab[r]$ and $(ldet[q][r] = 0$ or $(ldet[q][r] \neq 0$ and $nh > ldet[q][r] + \Delta_{msg}))$ **then**
*46: **if** $q \notin detnode[r]$ **then**
*47: $ldet[q][r] \leftarrow nh$
*48: **end if**
*49: **if** $r \notin suspected$ **then**
*50: $suspected \leftarrow suspected \cup \{r\}$
*51: $detnode[r] \leftarrow \{q\}$
*52: **else**
*53: $detnode[r] \leftarrow detnode[r] \cup \{q\}$
*54: **end if**
*55: **end if**
*56: **end for**
57: **End** of Task T3

Algorithm 2 FD algorithm for node p: Update

58: **Task** T4 every Δ_U [Update]
59: **for** each node $r \neq p$ **do**
60: **if** $mist[r] = mist_{prev}[r]$ **then**
61: $\Delta_S \leftarrow 0.1$
62: **else**
63: $\Delta_S \leftarrow -(mist[r] - mist_{prev}[r])/count[r]$
64: **end if**
65: $stab[r] \leftarrow max(0, stab[r] + S_{init} * \Delta_S)$
66: $mist_{prev}[r] \leftarrow mist[r]$
67: $count[r] \leftarrow 0$
*68: **if** $rmist[r] \neq rmist_{prev}[r]$ **then**
*69: $\Delta_{RS}[r] \leftarrow \Delta_{RS}[r] + (rmist[r] - rmist_{prev}[r])/rcount[r]$
*70: **else**
*71: $\Delta_{RS}[r] \leftarrow Max(\Delta_{init}, \Delta_{init} + (\Delta_{RS}[r] - \Delta_{init}) * 0.95)$
*72: **end if**
*73: $rmist_{prev}[r] \leftarrow rmist[r]$
*74: $rcount[r] \leftarrow 0$
75: **end for**
76: **End** of Task T4

Algorithm 3 Safety margin estimation for remote node q

1: `Function` ComputeMargin(q)
2: $C_v \leftarrow stddev(stab)/mean(stab)$
3: **if** $stab[q] \leq Quartile(25, stab)$ **then**
4: $\Delta_{marg} \leftarrow 2 * (1 + C_v)$
5: **else if** $stab[q] < Quartile(50, stab)$ **then**
6: $\Delta_{marg} \leftarrow 1 + C_v$
7: **else if** $stab[q] = Quartile(50, stab)$ **then**
8: $\Delta_{marg} \leftarrow 0$
9: **else if** $stab[q] \leq Quartile(75, stab)$ **then**
10: $\Delta_{marg} \leftarrow -0.25 * (1 + C_v)$
11: **else**
12: $\Delta_{marg} \leftarrow -0.5 * (1 + C_v)$
13: **end if**
14: `return` $marginit * (1 + \Delta_{marg})$
15: `End` of Function

- Δ_U: the frequency with which the algorithm updates the *stab* vector entries (an input parameter of the algorithm).
- *count*: a vector where $count[q]$ corresponds to the number of messages received from q within Δ_U; count values are reset to 0 every Δ_U units

of time.

- *EA*: a vector where $EA[q]$ is the expected arrival date for the next heartbeat message from q.
- *WA*: a vector containing sliding windows of the last received messages. $WA[q]$ contains the arrival dates of the last n messages received from q (or expected dates in case of message losses). n is a parameter of the algorithm.
- *marginit*: initial value of the margin (an input parameter of the algorithm).

Algorithm 1 uses two procedures, *UpdateWindow()* and *ComputeEA()*; we decided against including their pseudo-code for the sake of clarity. *UpdateWindow(WA, t, number)* adds the timestamp *number* (i.e., the sequence number) of the message received at time t in the sliding window WA. Upon detecting a message loss (*number* is greater than the previous timestamp plus one), *UpdateWindow* inserts the missing timestamp in the window with a ghost arrival time corresponding to a uniform distribution. *ComputeEA()* applies Formula (2) to compute the next expected arrival date according to Chen's estimation.

Algorithm *Stab*-FD (1 and 2) consists of four tasks: T1, T2, T3, and T4.

In Task T1, each process sends a heartbeat message to all other processes every Δ_H units of time. In order to allow the detection of message losses, it associates a sequence number (timestamp) with each heartbeat.

Task T2 adds a node in the suspected set if its respective timer expires.

Upon receiving a heartbeat message from a node q (Task T3), p checks for the presence of q in its suspected set. If such is the case, it removes q from its suspected set, and increments the corresponding number of mistakes (lines 33 and 34). p then updates the sliding window of message receptions from q, and computes both the new arrival time estimation for q's next heartbeat and its safety margin value, assigning their sum to the timer associated with q (lines 39–42).

The `ComputeMargin` function (Algorithm 3) computes the safety margin of q. This function uses p's estimation for the current stability of the input link from q as input, and returns a safety margin value which may vary from the initial margin (*marginit*). If the link has a high stability value, i.e., the remote node has a low rate of false suspicions, its safety margin value

decreases, and consequently its timer too. Conversely, if a link from q has a low stability value, the algorithm increases q's link safety margin value. The function computes Δ_{marg}, the percentage of increase/decrease of the initial margin, in proportion to the coefficient of variation of stability values (C_v) which expresses the level of dispersion around the mean (line 2).

`ComputeMargin` sets Δ_{marg} according to the quartiles of the stability vector values. We have evaluated several formula for adapting the safety margin according to the current stability of node, using both linear and arctangent correlations, and finally the quartiles distribution of Algorithm 3. Our experiments showed that using the quartile distribution gives the best results. The idea is to quickly increase (resp., decrease) the safety margin such that the stability of the node is low (resp., high) relatively to the other nodes.

Function $Quartile(percent, set)$ returns the value that splits the sorted values of *set* into two subsets according to their percentile (parameter *percent*). The value of *percent* can be set to 25, 50, or 75; corresponding to the Q1, Q2, and Q3 quartiles respectively. For instance, if q's stability value is among the lowest ones, i.e., within the first quartile ($stab[q] \leq Quartile(25, stab)$), and if the coefficient of variation is high (e.g., $C_v = 1$), Δ_{marg} is set to 4 (see line 4) and, thus, its margin will be 5 times higher than the initial margin.

Task T4 updates the stability vector every Δ_U units of time (lines 58–67). The new stability value that p assigns to $l_{q \to p}$, its input link from q, directly depends on the number of mistakes (false suspicions) that p made on q during the last Δ_U. The stability value of $l_{q \to p}$ increases by 10 percent of the initial value S_{init} if no mistake occured during this period (line 61), otherwise it decreases proportionally to the number of mistakes (line 63).

3.2. $Stab^C$-FD

Aimed at improving the quality of failure detection even further, $Stab^C$-FD is a cooperative version of *Stab*-FD where nodes exchange their current view of the stability values and their list of suspected nodes. The idea is to opportunistically add stability information in heartbeat messages in order to reduce to detection time. The relative link stability are then compared: if a node q has a better vision on a suspected node s than a node p then p also suspected s. We point out that the cost to add this information is negligible as long as it can be included in the payload of a single UDP heartbeat message. We will show in Section 4.6 that a single heartbeat message can include such an information of more than 1100 nodes.

$Stab^C$-FD extends *Stab*-FD in the following way (lines prefixed by $*$ and underlined variables):

- Each node includes its stability vector and the list of nodes it suspects in every heartbeat message.
- Upon receiving a heartbeat from q, p checks the list of suspects it just received from q. If q suspects a node r, and if the detection of r by q is more reliable than its own, then p starts suspecting r too.

Note that $Stab^C$-FD does not need any extra message since additional information is piggybacked in heartbeat messages. $Stab^C$-FD introduces a minimum gap between the receiver's and the sender's stability values to decrease the rate of false suspicions, and computes this gap dynamically.

Our cooperative strategy implementation requires the following additional variables and parameters:

- $rmist$: a vector where $rmist[q]$ corresponds to the number of mistaken suspicions sent by node q (i.e., the number of times node q has wrongly informed p of a failure).
- $detnode$: a vector where $detnode[q]$ corresponds to the set of remote nodes which share the latest failure suspicion about node q.
- Δ_{RS}: a vector where $\Delta_{RS}[q]$ keeps the minimum gap between stability values used by p to decide if the information included in q's heartbeat should be taken into account or not.
- Δ_{Init}: initial value of Δ_{RS} entries.
- $ldet$: a matrix containing the timestamp of the node which detects a remote failure. $ldet[q][r]$ is the timestamp of the last heartbeat received from q which induced p to include r in its set of suspected nodes.
- Δ_{msg}: an input parameter corresponding to the minimum number of messages needed for considering new suspected information from a node.

The following paragraphs explain in which ways $Stab^C$-FD extends the different tasks to make them cooperative.

In Task T2, p resets its *detnote* set to $\emptyset$ (line 25) every time it suspects a new node q.

Task T3 allows p to adopt suspicions issued by other nodes, provided these suspicions are more reliable than the local detection by p. Upon receiving a heartbeat from q, p looks at the encapsulated list of suspects. For every node r suspected by q, p performs a double check: the stability value must be at least $\Delta_{RS}[q]$ greater for link $l_{r\to q}$ than for link $l_{r\to p}$, and q must have suspected r for at least the last Δ_{msg} messages q sent. This last condition avoids bursts of false information to ensure the stability of the algorithm. A greater stability value on the link $l_{r\to q}$ indicates that the detection of r by q is more reliable. When both conditions hold, p adopts the suspicion. If r does not belong to the local set of suspects yet, p adds it and resets $denote[r]$ to q (line 51). Otherwise it simply adds q to the detnode set of r (line 53). Note that p only saves the timestamp of the first of a continuous sequence of suspicions of r by q in $ldet$(line 47).

Upon detecting a false suspicion about a node q (line 35), p "punishes" all the nodes that issued the suspicion. p increases the remote mistake counter of every node in the $detnode[q]$ set (line 36). p will reset $detnote[q]$ the next time it suspects q (lines 25 and 51).

Task T4 updates Δ_{RS} entries periodically. The update is proportional to the corresponding remote mistake rate (lines 68–74). For a node r that has made no mistake since the last Δ_U (line 71), p decreases $\Delta_{RS}[r]$ by 5%. The rationale is that, if p considers that r has a more reliable view, p relaxes the minimum stability value gap restriction for accepting remote information received from r. Conversely, as r makes mistakes, p tightens the gap restriction by increasing $\Delta_{RS}[r]$ (line 69).

4. Evaluation

In this section, we present the results of the experiments we conducted to evaluate the performance of our algorithms. We compare the behaviour of *Stab*-FD and $Stab^C$-FD with that of Chen's and Bertier's. The latter, introduced in section 2, rely on a constant and on a dynamic safety margin respectively.

4.1. FD settings

Similarly to [12, 13, 11, 20], which have also evaluated FD in a WAN context, in our experiments, nodes send a heartbeat message every 100ms.

These papers, as well as Chen et al. paper [7], also suggest that the safety margin α should range from 0 to 1000 ms. We set it to 150 ms in order to stay close to the frequency of heartbeat emissions (100 ms). Several works aim to improve the QoS of failure detectors by tuning parameters such as the window size [21] [20] [22]. Their results show that Chen's FD performs better with smaller window sizes. Based both on these studies and on our experiments, we set the window size to 100. The FDs thus rely on the last 100 received heartbeat messages for computing the estimation of the next heartbeat arrival time. For Bertier's algorithm, we follow the suggestions of the authors for the parameters of Jacobson's algorithm to dynamically adjust the safety margin values: we set $\beta = 1$, $\Phi = 2$, and $\gamma = 0.1$.

In our algorithm, the update frequency Δ_U for vectors *stab* and Δ_{RS} impacts the performance. Therefore, we carried out a preliminary experiment to evaluate the impact of different values of Δ_U. We concluded that 10 seconds is a good choice which smooths the variation of stability values. We set the initial stability value of each link to 10 ($S_{init} = 10$).

4.2. Experimental environment

Traces. We performed all of our experiments on real traces we collected from ten nodes of PlanetLab [16], labeled nodes 0, 1, ..., 9, and summarized in Table 1. The trace collection on PlanetLab lasted one week (150 hours), with every node sending UDP heartbeat messages to every other node at a rate of one heartbeat every 100 ms (the sending interval Δ_H). Each node locally logged the timestamp of every heartbeat message and its arrival time. Traces and the monitoring tool are available at https://gitlab.lip6.fr/psens/latency-trace-planetlab/.

The logs are used later to replay the execution of each FD algorithm. We wish to point out that our PlanetLab traces cover a large amount of heartbeat emissions and receptions, including unstable periods of link failures and message losses which induce false suspicions. Thus, such traces characterize a distributed system that uses heartbeat-based FDs. Furthermore, our experimental scenarios and results are reproducible.

Trace analysis. Tables 2, and 3 give some information about the heartbeat messages received by each node. During the experiment, node *2* crashed and stopped sending messages after approximately 48 hours; overall, it sent 1,759,990 messages.

Table 1: Trace Collection Sites

ID	Node	Location
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, Mass.
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

Table 2: Standard deviation of heartbeat inter-arrival times (ms)

Sender / Receiver	0	1	2	3	4	5	6	7	8	9
0		6.3324	7.3067	6.5049	18.3887	6.6487	6.7250	16.0537	12.9870	6.4075
1	5.7801		6.6084	1.3246	16.8997	2.3359	2.2455	14.9136	11.3943	1.4797
3	5.9409	1.8698	6.6315		16.8307	2.7328	2.6436	14.8574	11.4066	2.0271
4	6.6272	3.3661	6.9757	3.4057		3.9096	3.8582	15.2150	11.8177	3.4732
5	6.4178	3.1791	6.7477	3.1786	16.9933		3.6801	15.0355	11.6597	3.2648
6	6.2730	2.7241	5.9071	2.7902	16.8667	3.4013		15.0761	11.6422	2.8571
7	16.3485	15.2939	9.2876	15.2943	22.6782	15.4887	15.4483		19.1355	15.3382
8	54.3717	54.1041	55.1194	54.0932	56.6306	54.1768	54.0585	56.1172		54.0673
9	5.9683	1.4544	5.6874	1.5402	16.8579	2.4972	2.4203	15.0134	11.4800	

We observe that the mean inter-arrival times of received heartbeats are very close to 100 ms on all links. However, we note some differences in standard deviation values in Table 2. We observe two phenomena. First: for some nodes such as node 8 and to a small extent node 7, the standard deviation is very high. These high values reflect an instability of the input links. Second: links are not symmetrical. Whereas the standard deviation values associated with the input links of node 8 sit around 54ms (row 8 in Table 2), the values associated with output links from 8 to the other nodes remain close to 11 ms and never reach above 20 ms (column 8 in Table 2).

Table 3 records the number of message losses detected by each node at

Table 3: Heartbeat losses

Sender / Receiver	0	1	2	3	4	5	6	7	8	9
0		1964	7599	3169	12484	15903	3195	4690	2087	6956
1	2607		7298	632	13356	13747	739	1732	667	5440
3	1854	28	8060		14298	12431	96	1292	219	1161
4	1724	93	8993	239		12032	331	1323	531	526
5	12541	9566	20186	13302	27725		9600	10677	11843	10129
6	2564	104	10434	108	14950	13098		1479	2139	4899
7	3052	1159	12092	1103	15823	14775	1168		1256	1923
8	10724	8882	15953	9815	22632	23114	9795	9867		13621
9	3372	1063	11877	364	14622	12612	1094	2179	1139	
Stddev	4277	3990	4298	5098	5331	3632	4087	3939	3844	4570
Mean	4805	2857	11388	3592	16986	14714	3252	4155	2485	5582
Total	38438	22859	102492	28732	135890	117712	26018	33239	19881	44655
Percent	0.078%	0.046%	0.208%	0.058%	0.275%	0.238%	0.053%	0.067%	0.040%	0.090%

the end of the execution. The results in this table confirm the instability of network links previously observed in Table 2. They also show that the distribution of heartbeat losses is not even. For instance, node 4 incurs 6.8 times more losses on its heartbeat emissions than node 8. Even if the global percentage of losses remains low (less than 0.28%), each loss can lead to an inaccurate estimation for the next heartbeat arrival date. We can also observe that links are highly asymmetrical. For instance, there are only 28 losses of heartbeats on link $l_{1\to3}$ from node 1 to node 3, whereas there are 3169 the other way round on link $l_{3\to1}$.

Figure 1 shows the high variation of heartbeat arrival times over link $l_{0\to5}$. The right axis gives the cumulative number of message losses. We observe several unstable periods for the inter-arrival times around the 20th, 50th, 120th, and 140th hour. During these periods (except for the last one at the 140th hour), the number of message losses also increases.

Figure 2 gives the evolution of message losses of the input links of node 5. The variation of losses highly differs from one input link to the other. A peak of message losses occurs on all links around the 17th hour. But afterwards, the input links $l_{3\to5}$ and $l_{6\to5}$ incur very few losses, whereas $l_{2\to5}$ and $l_{4\to5}$ incur a steadily high rate of message losses. We also observe sporadic sudden increases of losses on $l_{0\to5}$, $l_{7\to5}$, and $l_{9\to5}$.

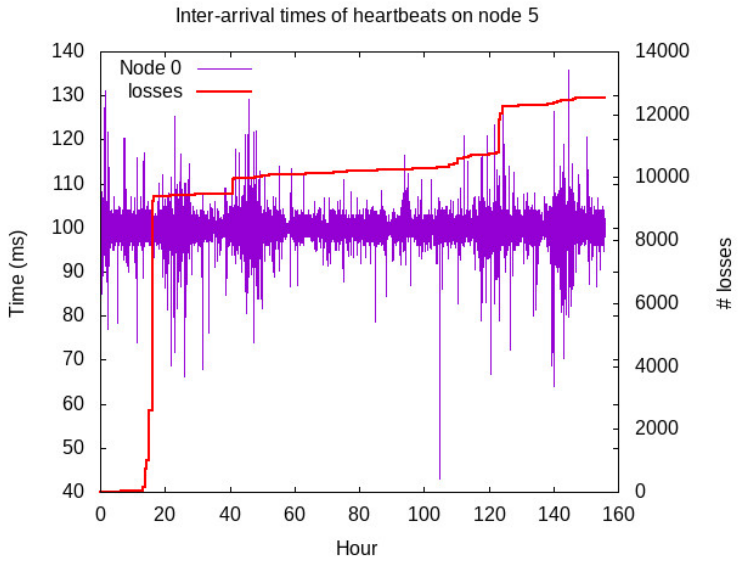


Figure 1: Inter-arrival times of heartbeats sent by node 0 (left axis). Cumulative number of message losses from node 0 (right axis)

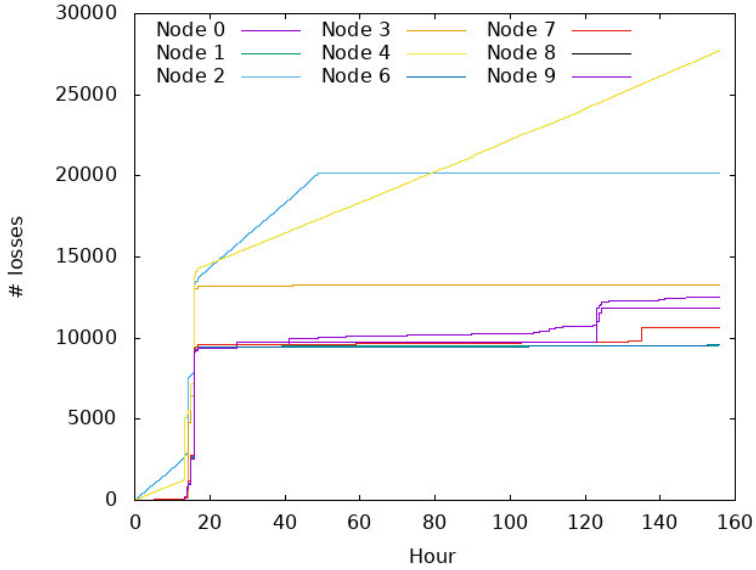


Figure 2: Message losses observed at node 5

Table 4: Stability vectors of nodes after 150 hours

Vector Nodes	0	1	2	3	4	5	6	7	8	9
0	0	1738	1778	1738	1736	1736	1738	1730	1592	1736
1	1738	0	1788	1750	1748	1750	1750	1744	1598	1750
3	1730	1758	1786	0	1756	1750	1758	1744	1590	1758
4	1746	1770	1794	1772	0	1764	1772	1756	1608	1772
5	1693	1717	1763	1717	1715	0	1717	1709	1549	1717
6	1728	1746	1790	1746	1744	1746	0	1738	1586	1746
7	1710	1732	1776	1736	1730	1728	1736	0	1570	1736
8	899	909	1489	909	909	909	909	909	0	909
9	1736	1754	1788	1754	1750	1752	1754	1742	1596	0
Stddev	293	317	105	317	338	318	317	309	68	341

4.3. Evaluation of the stability of links

Every *Stab*-FD node maintains a set of values to characterize the stability of its input links. Each row in Table 4 shows the stability vector of a node at the end of the experiments (after 150 hours). We observe a small variation of the stability values for node 8; and compared with all the other nodes, it is clearly the most unstable. Conversely, the low quality of all of its input links pushes node 8 to make many mistakes in its observations. As a consequence, node 8 considers all the other nodes (input links) as unstable; node 2 is an exception, since it fails around the 48th hour. Note that after that time, nodes do not generate new false suspicions about node 2 and its stability increases for all nodes, including node 8.

Figure 3 shows the evaluation over time of the stability value that node 8 associates with node 4. The right axis gives the cumulative number of mistakes by *Stab*-FD. Stability values follow the evolution of mistakes during the last Δ_U seconds. As follows from lines 63 and 65 of Algorithm 2, the stability value cannot decrease by more than ten (the value of S_{init}) upon every update. This prevents an undue inflation of stability values in case of a punctual burst of mistakes, as the figure shows at the 50th, 115th, and 130th hour.

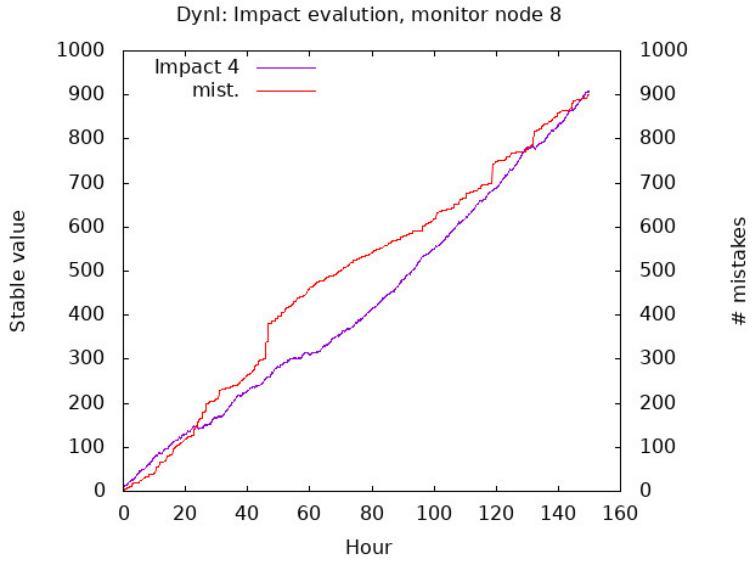


Figure 3: Evolution of the stability factor node 8 associates with node 4 (left axis). Cumulative number of mistakes (right axis)

4.4. Accuracy

We measure and compare the accuracy of the failure detectors by considering the number of false positives (mistakes). A mistake occurs when a node is falsely put into the suspected node list of one node. Table 5 gives the number of mistakes made by each node after the 150th hour. Compared to Chen's FD, both *Stab* FDs produce around 6.5 times fewer mistakes. We observe that Bertier's FD generates a huge number of false positives. We also note a small difference between *Stab* FD and its cooperative version $Stab^C$ FD. The latter generates only 0.7% more mistakes than the former because it includes false suspicion information in the heartbeats it sends. This small difference is a direct consequence of two mechanisms that prevent the propagation of false information: (1) the dynamic adaptation of Δ_{RS} retains only information issued from remote nodes with high stability values; (2) the algorithm discards Δ_{msg} consecutive mistakes issued from the same remote node. Without both of these mechanisms, the number of false positives is around 4.5 times higher.

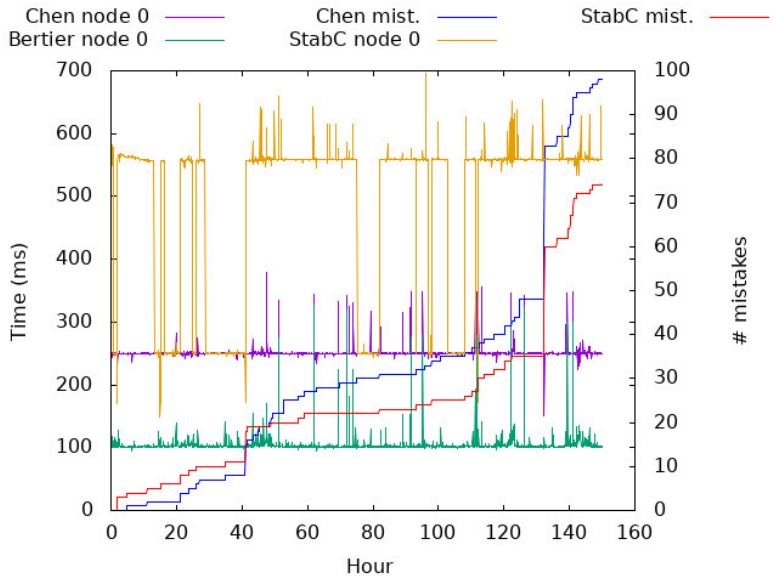
A short estimation of the next heartbeat message arrival time causes a false positive: the estimated time elapses before the actual arrival time of the heartbeat. As an example, Figures 4a and 4b compare the evolution

Table 5: Number of mistakes

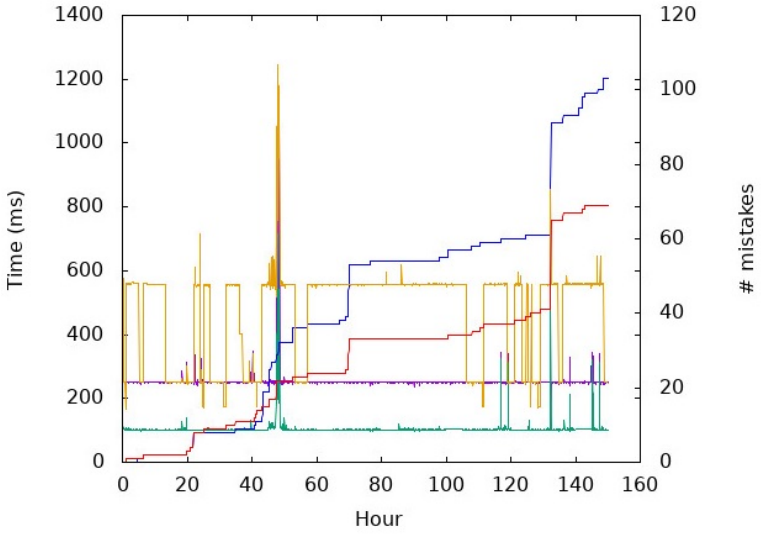
	Chen		Bertier		*Stab*-FD		$Stab^C$-FD	
Node	Number	Stddev	Number	Stddev	Number	Stddev	Number	Stddev
0	7980	2261	1163052	37027	878	47	864	47
1	8169	2441	1334539	56776	566	37	552	35
3	8811	2661	1434526	52200	1117	152	1160	164
4	781	2717	1588784	55725	574	39	576	36
5	9312	2717	1444727	55725	698	39	767	36
6	9340	2788	1797544	73405	982	49	982	49
7	9725	2816	1168183	46473	890	53	884	47
8	18191	2810	1531003	58906	6155	164	6155	164
9	9113	2742	1325668	47590	645	52	658	53
Total	81,422		1,2788,026		12,505		12,598	

over time of the timers set by each FD between nodes 0 and 9. We do not plot *Stab*, because the timer values of *Stab* and $Stab^C$ FDs are the same. The figures show the difference between the real arrival times of heartbeats and the timer values upon their reception. The right axis corresponds to the number of mistakes generated by Chen and by $Stab^C$. We also do not plot the number of mistakes produced by Bertier because it is too high. We observe that Chen's FD timer value is relatively stable over time: around 250 ms. This corresponds to the heartbeat interval (100 ms) added to the constant safety margin (150 ms). Bertier's FD is more aggressive with a short timer around 100 ms, whereas $Stab^C$ timer values vary with the number of earlier mistakes. We observe high instabilities on both nodes. Particularly, all three FDs incur a high increase of their timer value on node 0 at the 48th hour, thus inducing a significant increase in the number of mistakes. Since Bertier's FD algorithm is very sensitive to the variation of the last received heartbeat time, its timer quickly increases and decreases during an unstable period. On the other hand, the $Stab^C$ FD timer maintains a high value during the entire unstable period and, consequently, does not produce such an increase of false positives. We should point out that, besides the exception at the 130th hour, there is no correlation between the instabilities observed on links $l_{0\to 9}$ and $l_{9\to 0}$.

A high number of false positives can overload the application if some



(a) Heartbeats from node 0 at node 9



(b) Heartbeats from node 9 at node 0

Figure 4: Comparing FD accuracies: timers and heartbeat inter-arrival times (left axis), cumulative number of mistakes (right axis)

callback mechanism directly reports mistakes to the upper layer. However, a local FD oracle invoked via queries can mitigate the effect of false positives if the mistake duration is very short. Introduced in [7], the query accuracy probability (noted PA) reflects the probability that a failure detector's output is correct at any random time.

Table 6 summarizes the computed PA. As expected, *Stab* and $Stab^C$ FDs have the highest PA values, which are almost the same for both FDs (due to rounding, the values are the same in Table 6). The average difference between the two is less than 3.4×10^{-7}. We can also observe that the PA of Bertier's FD is quite high in regard with its large number of mistakes, which means that mistake durations are very short and Bertier's FD algorithm corrects its mistakes very fast.

Table 6: Query accuracy probability (PA)

Node	Chen Mean	Bertier Mean	*Stab*-FD Mean	$Stab^C$-FD Mean
0	0.99918	0.99638	0.99923	0.99923
1	0.99956	0.99673	0.99961	0.99961
3	0.99958	0.99664	0.99963	0.99963
4	0.99960	0.99795	0.99962	0.99962
5	0.99736	0.99461	0.99733	0.99733
6	0.99948	0.99650	0.99953	0.99953
7	0.99919	0.99643	0.99926	0.99926
8	0.99499	0.99177	0.99549	0.99549
9	0.99950	0.99658	0.99955	0.99955
Mean	0.99872	0.99596	0.99880	0.99880

4.5. Detection time

Failure injection. For our comparative assessment of detection times, we inject transient failures on a node pf. We do so by assigning several failure intervals to every simulation run: all monitoring nodes discard heartbeats sent by pf from the moment the heartbeat sequence number reaches the start value of a failure interval. Monitoring nodes start taking the heartbeats back into account when the heartbeat sequence number reaches the end value. In each simulation run, a single node pf periodically fails every

20 hours and recovers after 5 hours. Thus pf fails 7 times during each experiment: at the 20th, 40th, 60th, 80th, 100th, 120th, and 140th hour. We compute an *estimation* for the detection time by measuring the time elapsed on a node from the last heartbeat reception up until the failure detector reports its failure suspicion.

Table 7: Detection times (ms)

Fail	Chen		Bertier		*Stab*-FD		$Stab^C$-FD	
node	Mean	Stddev	Mean	Stddev	Mean	Stddev	Mean	Stddev
0	251.0	7.0	114.1	88.0	463.7	143.3	449.5	145.0
1	249.7	2.6	102.2	3.1	361.1	158.4	341.2	146.3
3	250.2	0.8	112.4	76.9	364.3	171.5	293.2	128.6
4	292.3	38.2	172.2	107.2	557.1	91.8	533.6	92.1
5	246.7	26.7	104.4	18.3	511.4	101.3	495.2	104.5
6	250.3	1.0	113.4	79.9	387.8	161.3	355.4	135.3
7	250.4	3.7	101.8	4.1	506.4	122.2	468.1	127.9
8	250.2	0.5	101.4	1.3	556.3	1.3	556.0	1.3
9	251.5	9.3	103.8	9.7	432.0	161.3	416.6	156.2
Mean	254.70		113.95		460.03		434.32	

Table 7 gives the detection times exhibited by each FD upon simulating periodic failure/recovery on each node except node 2. We observe a relatively small variation of the detection times for Chen's FD, with the exception of node 4. Chen's estimation is less sensitive to punctual variations in heartbeat arrival dates because it uses a constant safety margin: the mean detection time tends to converge towards the sum of the heartbeat interval with the constant safety margin. The variation in heartbeat arrival times directly impacts the safety margin of Bertier's FD, thereby decreasing its value. As mentioned earlier, Bertier speeds up detection time at the cost of a huge number of false positives.

Our strategy, implemented into the *Stab* and $Stab^C$ FDs, is more conservative about reducing timer values. On the one hand it improves the avoidance of false positives, on the other hand it impacts detection times negatively. The mean detection times of *Stab* exceed that of Chen's by 17% in the best case (detection of the failure of node 3), and by 122% in the worst case (node 8).

The cooperative approach of $Stab^C$ mitigates the above impact: nodes with stable links compute shorter detection times, propagate their suspicion, speeding up detection on other nodes. Tables 8 and 9 show the detection times computed on node 3 by *Stab* FD and $Stab^C$ respectively. In Table 9, the shaded cells highlight the detection times improved by the cooperative approach. In Table 8, we observe several values around 175, 250, and 550ms. The detection time depends on the average inter-arrival time of the last 100 received heartbeats, usualy around 100ms, plus the safety margin. The initial value of this margin is set to 150 ms and is updated by ComputeMargin function (Algorithm 3) each time a heartbeat is received. The margin evolves proportionally to the coefficient of variation (C_v) of stability values and to the relative stability of the observed link. A detection time around 175ms, is mainly due to a division by two of the safety margin. Such a result indicates that the observed link has a significantly higher stability than the others (i.e., those included in the upper 2 quartiles). For example, for a coefficient of variation of 1, if the link is in the highest third quartile, the margin will be equal to $150 * (1 - 0.25(1 + 1)) = 75ms$. A detection time around 250ms mainly means that the stability of the observed link is equal to the median value whereas a value around 550ms could result from a strong variation of stability values in the stability vector ($C_v = 1$) and a stability value of the link in the second quartile. In this case, the safety margin is equal to $150 * (1 + 2) = 450ms$.

With $Stab^C$ FD, a stable node that quickly detects a failure (for instance node 4 detects the failures injected at the 20th, 40th, 80th, and 100th hour in less than 174 ms) spreads this information over the network. Note that the receiver node does not take a remote detection into account unless it considers the issuer as stable enough. For instance, since node 7 maintains a high stability value on the other nodes, it only adopts suspected information sent by node 4 once (20th hour). This mechanism contribute significantly to the prevention of false positive rate increasing.

In our experiments, *Stab* produces 12,237 false positives; this number marginally rises to 12,557 for $Stab^C$, an increase of only 2.7%. In comparison, Chen generates 81,095 false positives. The cooperative strategy of $Stab^C$ converges towards an average detection time that is 19.5% lower than that of *Stab*. Our cooperative approach reduces the detection time of 18 failure detections out of the 56 that took place. If we focus on these 18 failure detections, the decrease of the average detection time sharply improves. The lowest improvement is faster by 40.2%, while the highest reaches up to 60.6%.

Table 8: Stab FD: Detection times (ms) of node 3

nodes Times(h)	0	1	4	5	6	7	8	9
20	251.0	250.1	172.9	174.0	250.1	250.1	556.1	250.0
40	250.1	555.3	173.6	557.4	556.2	173.4	557.1	557.1
60	250.7	557.7	249.3	557.2	559.1	577.3	558.7	558.7
80	173.7	558.5	173.0	250.4	250.3	600.1	250.2	250.2
100	250.0	249.9	173.5	558.5	172.9	604.3	559.4	559.4
120	559.3	250.1	250.3	558.8	249.9	606.0	559.0	559.0
140	253.4	172.8	173.3	251.1	172.7	250.5	250.0	250.0

Table 9: $Stab^C$ FD: Detection times (ms) of node 3 (shaded cells highlight improvement on the non cooperative approach)

nodes Times(h)	0	1	4	5	6	7	8	9
20	251.0	236.0	173.2	174.4	239.3	240.3	219.0	250.0
40	250.1	232.2	173.8	322.1	236.0	173.4	244.3	244.3
60	250.7	332.1	249.3	556.9	333.8	577.3	343.4	343.4
80	173.7	230.0	173.1	250.4	250.3	600.1	250.2	250.2
100	250.0	236.5	173.5	248.5	172.9	604.3	275.1	275.1
120	559.3	250.1	250.3	558.6	249.9	606.0	558.9	558.9
140	253.4	172.8	173.3	251.1	172.7	250.5	250.0	250.0

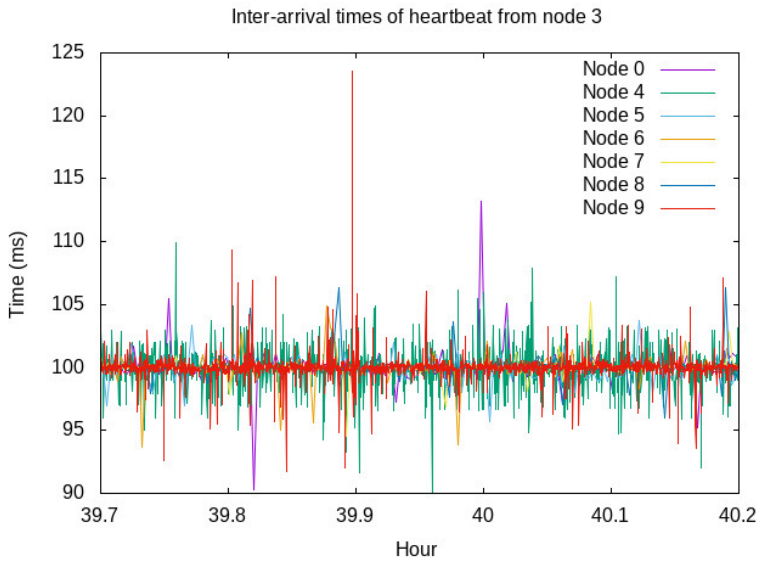


Figure 5: Stability of links from node 3 around the 40th hour

We should point out that our cooperative approach is effective when the outgoing links from a monitored node exhibit large variations in terms of stability and at least one of the links is stable. For instance, column 3 of Table 2 shows that the stability of the input links of node 3 is strongly heterogeneous. Conversely, in terms of standard variation, the quality of the outgoing links from node 8 is quite stable. As a result, the stability values associated with these links are close, and hence so are the safety margin values. In this case, the cooperative approach is much less likely to improve the detection time.

The variations of heartbeat arrival dates from node 3 at times of failure, observed around the 40th hour in Figure 5 and around the 120th hour in Figure 6, show the influence of stable links on cooperative detection. Heartbeat arrival dates are very heterogeneous around the 40th hour and, in particular, the link connectivity from node 3 to node 4 is much better than to other nodes. In such a configuration, cooperation is quite efficient, as observed in row 2 (40th hour) of Table 9. Conversely, the homogeneity of heartbeat arrival dates around the 120th hour inhibits the efficiency of the cooperation approach.

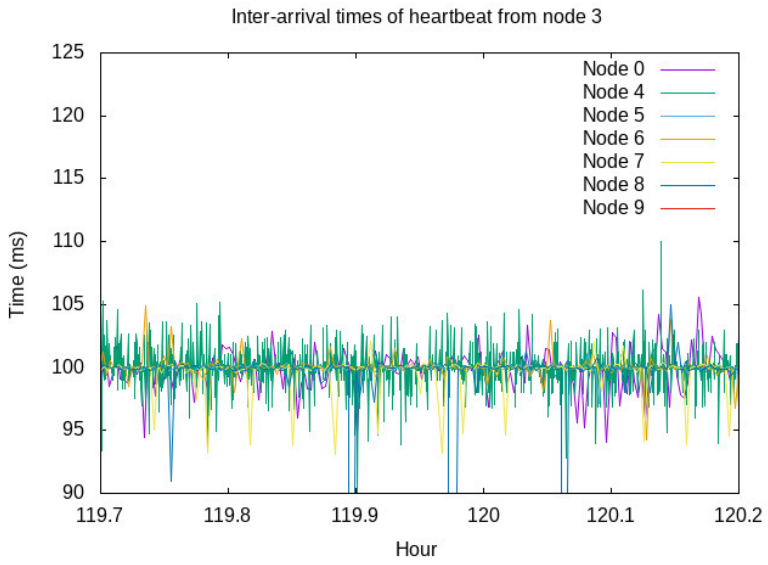


Figure 6: Stability of links from node 3 around the 120th hour

4.6. Scalability issues

In order to study the scalability of our approach, we compute the size of the messages for different network configurations. Figure 7 shows the size of data carried on by heartbeat messages when the number of nodes increases. Heartbeat messages include the suspected set and the set of impacts of all nodes. We overestimated the size of the suspected set assuming 5 percent of suspected nodes. The impact set is an array of integers with one entry per node of the system. We have generated impact values using a normal distribution considering two configurations: (1) 'low var.' curve which corresponds to a homogeneous network with a low standard deviation of the impact values (stddev = 5%); (2) 'high var.' curve which corresponds to a more heterogeneous network with a higher standard deviation of the impact values (stddev = 30%). We have applied a lzma compression to reduce the size of the two sets. The horizontal line represents the size of the MTU for a UDP packet, above this size the heartbeat message is split by the IP layer. We observe a single UDP heartbeat packet is sufficient to carry on information of more than 1100 nodes even in a heterogeneous configuration where the compression algorithm is less effective.

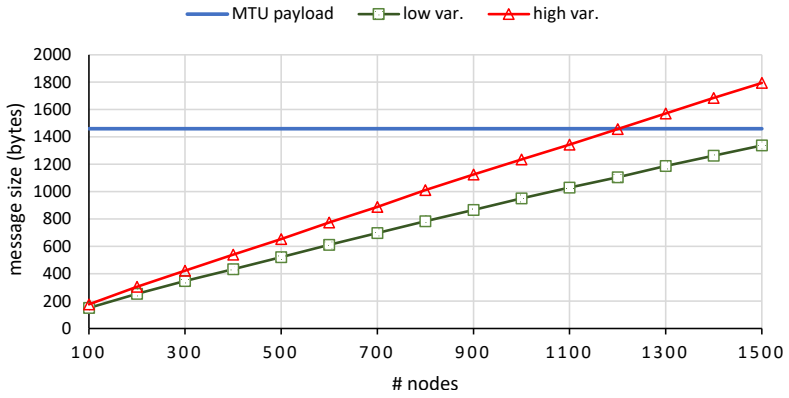


Figure 7: Heartbeat size (5% of suspected nodes and mean impact = 1700)

4.7. Experiments conclusion

We can draw the following conclusions from the results of our experiments:

- Unstable periods are very common in WAN traces such as the one we have collected.
- The value of the safety margin is a key point for the performance of FDs that rely on sliding windows. A constant safety margin (eg. Chen) prevents an accurate adaptation of the timer value when the network becomes unstable.
- An adaptation of the safety margin that relies only on the last variations of heartbeat arrival time (eg. Bertier), shortens detection times at the expense of a higher number of false positives.
- Monitoring link stability to calibrate the safety margin improves FD accuracy. When a link becomes unstable, its safety margin value increases, thus reducing the number of false positives.
- A cooperative approach can improve detection time when the quality of the links is heterogeneous: nodes with a good connection to faulty nodes can quickly detect failures and spread the information to other nodes.
- Assessing the quality of the source of information prevents the dissemination of wrong information. In our case, information is more reliable

when it comes from nodes with a better view of the system in terms of stability.

5. Related work

Tomsic et al. [20] propose a Two Windows Failure Detector (2W-FD) to extend the single sliding window estimation. 2W-FD uses two sliding windows of different sizes to store information about recent heartbeat history. The goal of the two windows is to react to sudden network changes using the best of the two estimations provided by each window. However, the algorithm assumes a constant safety margin.

The authors of [14] propose two variable safety margin strategies: a prediction error-based margin (*peb*), and a confidence interval based margin (*cib*). Similarly to Bertier's FD and $Stab^C$-FD, *peb* adapts the safety margin value whenever the FD receives a heartbeat and detects variations in the network load. *cib* applies probability on the heartbeat arrival estimation distribution in order to adapt the safety margin. Contrarily to our cooperative approach, both *peb* and *cib* rely on information that remains local to every FD in order to adapt the safety margin.

Some authors [15, 23] have proposed adaptive failure detectors also based on local observations.The autonomic failure detector (AFD) [15] defines a new metric, the *Failure Detector Availability* (AV), that combines both feedback control theory and traditional FD metrics such as *Mistake Duration* (T_M) and *Mistake Recurrence Time* (T_{MR}). AFD computes the Failure Detector Availability with $AV = (T_{MR} - T_M)/T_{MR}$, and uses it to suggest safety margin values that decrease the rate of false detections and converge towards the desired detection availability. AFD increases the safety margin to improve detection accuracy when the detection service is inaccurate (i.e., AV is low); decreases the safety margin value to improve the detection speed when AV is high. The authors of [23] propose Adaptare-FD that uses both Jacobson RTT and *cib* estimators and rely on a coverage parameter to configure the FDs. In [24], the authors analyze the QoS of a given pull-based (pinging) FD which uses multiple probing messages strategy to increase accuracy. It proposes a model to dynamically compute the delay before starting the resend of probing message and the number of messages to resend.

Another approach to adapt failure detection is to use prediction algorithms. In [25], the authors propose a method that uses a back propagation neural network based on particle swarm optimization (PSO) to predict the

arrival time of heartbeat information, focusing on remote disaster recovery. In order to improve the accuracy of FD detection, they propose a Pull checking when heartbeat messages are not receive in time which slow down the detection time. On the other hand, such an approach needs to tune many parameters to configure both the neural network and the PSO algorithm.

Several works have proposed collaborative failure detector [26, 27, 28, 29].Veron et al. propose RepFD [26], a collaborative failure detector which exploits information about the behavior of nodes to increase its detection quality. RepFD relies on a reputation service where nodes periodically exchange heartbeat messages to classify the behavior of nodes. The reputation of a node dynamically increases if it sends its heartbeats on time, and decreases if some heartbeats get lost or arrive after the expected dates. The detector considers a node as correct if its reputation value exceeds a given threshold T. The principle of this approach is close to our proposal, however the main limitation of RepFD lies in the static nature of the threshold T and how to compute its initial value. In [28], the authors reuse Bertier safety margin estimation approach to build a FD service shared by several applications running on the same nodes. Therefore, only FDs located on these nodes and associated to different applications cooperate by sharing heartbeat messages.

Swim FD [27] and Medley [29], its recent extension to IoT networks, both relying on a collaborative pinging. These FDs exchange information remotely to increase the accuracy of the detection. Each node p periodically pings another node q at random (direct pinging phase). If q does not reply in time according to a predefined timeout, p suspects this node of being faulty and asks k other nodes to check the potentially faulty node (indirect pinging phase). The two phases, direct and indirect pinging, reduce the mistake rate but can slowdown the detection time in case of failure. Furthermore, the randomization used by these protocols makes the definition of timeout values difficult, since the monitoring sets may frequently change over time.

All the above failure detectors, as well as $Stab^C$-FD, output a set of processes suspected of being either correct or faulty. The Accrual failure detector [30] follows another approach: it outputs a suspicion level on a continuous scale, which represents the risk that the process is indeed faulty. It uses this suspicion level to adapt to dynamic network conditions. Similarly to many failure detector arrival estimations, including Stab-FD, 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. In [31], the authors extend the Accrual FD by exploiting the histogram density estimation. Taking into account a sampled inter-arrival time and the time of the last heartbeat reception, the algorithm estimates the probability that no further heartbeat messages will arrive from a given process. Exponential Distribution FD [32] also extends the accrual failure detector, but considers an exponential distribution of the message delays. In [22], an automatic self-tuning failure detector (SFD) optimizes accrual FDs.

Synthesis. Table 10 summarizes the main characteristics of FDs presented in this section. The main originality of $Stab^C$-FD is to combine both dynamic adaptation of timers and an opportunistic cooperation by piggybacking global information inside heartbeats.

Table 10: Failure detectors comparison

Protocol	Output	Polling strategy	Detection	Safety margin	Cooperation
[20]	node	periodic push	distribution of HB in 2 windows	constant	-
[14]	node	periodic push	sample of HB	prediction error-based (*peb*), confidence interval based (*cib*)	-
[15]	node	dynamic pull	feedback control theory	dynamic	-
[23]	node	periodic push	distribution of n last HB: mean, low pass filter	Jacobson, *cib*	-
[24]	node	periodic pull + push	last pull round	-	-
[25]	node	periodic push + pull	neural networks	-	-
[28]	node	periodic push	mean of n last HB	-	local sharing of HB
[26]	node	periodic push	reputation based	-	periodic exchange of reputations
[27, 29]	node	periodic pull	Fix timeout + indirect detection	-	multicast of failure information
[30, 32, 31]	susp. level	periodic push	distribution of n last HB: mean + variance [30], exponential [32], histogram density [31]	-	-
$Stab^C$	node	periodic push	mean of n last HB	impact based quartile distribution	periodic exchange of impact

6. Conclusion

Links in wide area networks (WANs) are much more likely to incur unstable periods, with higher network delays and number of message losses, than local area networks (LANs). Therefore, link quality variations represent crucial information for timer-based FDs in large scale systems. In this article, we propose Stab-FD, a detector that assesses the stability of its input links. We show that an adaptive calibration of safety margin values based on input link stability can improve the quality of service of failure detectors. We extend this strategy with a cooperative approach: nodes exchange their local assessment of the stability they associate with input links, filter reliable information, and re-calibrate their safety margins dynamically.

By conducting extensive experiments on Planetlab traces, we characterize the variations of link stability over time, and compare Stab-FD with Bertier's FD and Chen's FD. Our results show that our FD captures better network variations: it produces fewer failure suspicions and a higher accuracy, without degrading the failure detection time significantly.

References

[1] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: Amazon's highly available key-value store," *SIGOPS Oper. Syst. Rev.*, vol. 41, no. 6, p. 205–220, oct 2007. [Online]. Available: https://doi.org/10.1145/1323293.1294281

[2] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: A distributed storage system for structured data," *ACM Trans. Comput. Syst.*, vol. 26, no. 2, jun 2008. [Online]. Available: https://doi.org/10.1145/1365815.1365816

[3] A. Lakshman and P. Malik, "Cassandra: A decentralized structured storage system," *SIGOPS Oper. Syst. Rev.*, vol. 44, no. 2, p. 35–40, apr 2010.

[4] X. Yan, L. Yang, and B. Wong, "Domino: Using network measurements to reduce state machine replication latency in wans," in *Proceedings of the 16th International Conference on Emerging Networking EXperiments*

and Technologies, ser. CoNEXT '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 351–363. [Online]. Available: https://doi.org/10.1145/3386367.3431291

[5] L. Lamport, "Paxos made simple," *ACM SIGACT News (Distributed Computing Column) 32, 4 (Whole Number 121, December 2001)*, pp. 51–58, December 2001.

[6] T. D. Chandra and S. Toueg, "Unreliable failure detectors for reliable distributed systems," *J. ACM*, vol. 43, no. 2, pp. 225–267, Mar. 1996. [Online]. Available: http://doi.acm.org/10.1145/226643.226647

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

[8] M. Bertier, O. Marin, and P. Sens, "Implementation and performance evaluation of an adaptable failure detector," in *Proceedings of the 2002 International Conference on Dependable Systems and Networks*, ser. DSN '02, 2002, pp. 354–363.

[9] ——, "Performance analysis of a hierarchical failure detector," in *In Proceedings of the International Conference on Dependable Systems and Networks*, 2003, pp. 635–644.

[10] B. Deianov and S. Toueg, "Failure detector service for dependable computing," *Proc, 2000 Int'l Conf. Dependable Systems and Neteworks*, pp. B14–B15, June 2000.

[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*, 2004, pp. 66–78.

[12] N. Xiong, A. V. Vasilakos, L. T. Yang, L. Song, Y. Pan, R. Kannan, and Y. Li, "Comparative analysis of quality of service and memory usage for adaptive failure detectors in healthcare systems," *IEEE J.Sel. A. Commun.*, vol. 27, no. 4, pp. 495–509, May 2009. [Online]. Available: http://dx.doi.org/10.1109/JSAC.2009.090512

[13] N. Xiong, A. V. Vasilakos, Y. R. Yang, S. Wei, C. Qiao, and J. Wu, "General traffic-feature analysis for an effective failure detector in fault-tolerant wired and wireless networks," Tech. Rep., 2011.

[14] R. C. Nunes and I. Jansch-Pôrto, "Qos of timeout-based self-tuned failure detectors: The effects of the communication delay predictor and the safety margin," in *2004 International Conference on Dependable Systems and Networks (DSN 2004), Proceedings*, 2004, pp. 753–761.

[15] A. S. de Sá and R. J. A. Macêdo, "Qos self-configuring failure detectors for distributed systems," in *IFIP International Conference on Distributed Applications and Interoperable Systems*. Amsterdam, The Netherlands: Springer Berlin Heidelberg, 7-9 June 2010, pp. 126–140.

[16] PlanetLab, "http://www.planet-lab.eu/."

[17] R. Guerraoui and L. Rodrigues, *Introduction to Reliable Distributed Programming*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[18] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of distributed consensus with one faulty process," *J. ACM*, vol. 32, no. 2, pp. 374–382, Apr. 1985. [Online]. Available: http://doi.acm.org/10.1145/3149.214121

[19] V. Paxson and M. Allman, "Computing tcp's retransmission timer," United States, 2000.

[20] A. Tomsic, P. Sens, J. Garcia, L. Arantes, and J. Sopena, "2w-fd: A failure detector algorithm with qos," in *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. Hyderabad, India: IEEE, 25-29 May 2015, pp. 885–893.

[21] N. Hayashibara, X. Defago, R. Yared, and T. Katayama, "The φ accrual failure detector," in *23rd International Symposium on Reliable Distributed Systems SRDS*. Florianopolis, Brazil: IEEE Computer Society, 18-20 October 2004, pp. 66–78.

[22] N. Xiong, A. V. Vasilakos, J. Wu, Y. R. Yang, A. Rindos, Y. Zhou, W.-Z. Song, and Y. Pan, "A self-tuning failure detection scheme for cloud computing service," in *26th International Parallel & Distributed*

Processing Symposium (IPDPS). Shanghai, China: IEEE, 21-25 May 2012, pp. 668–679.

[23] M. Dixit and A. Casimiro, "Adaptare-fd: A dependability-oriented adaptive failure detector," in *Proceedings of the 2010 29th IEEE Symposium on Reliable Distributed Systems*, ser. SRDS '10. USA: IEEE Computer Society, 2010, p. 141–147. [Online]. Available: https://doi.org/10.1109/SRDS.2010.24

[24] J. Dong, X. Ren, D. Zuo, and H. Liu, "An adaptive failure detector based on quality of service in peer-to-peer networks," *Sensors*, vol. 14, no. 9, pp. 16 617–16 629, 2014. [Online]. Available: https://www.mdpi.com/1424-8220/14/9/16617

[25] Liang, Min, Gao, Fei, and Shi, Min, "A failure detection method of remote disaster recovery and backup system," *ITM Web Conf.*, vol. 47, p. 01002, 2022. [Online]. Available: https://doi.org/10.1051/itmconf/20224701002

[26] 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*. Beijing, China: IEEE Computer Society, 1-4 September 2015, pp. 91–100.

[27] A. Das, I. Gupta, and A. Motivala, "SWIM: scalable weakly-consistent infection-style process group membership protocol," in *2002 International Conference on Dependable Systems and Networks (DSN 2002), 23-26 June 2002, Bethesda, MD, USA, Proceedings*, 2002, pp. 303–312.

[28] R. C. Turchetti, E. P. Duarte Jr., L. Arantes, and P. Sens, "A qos-configurable failure detection service for internet applications," *J. Internet Serv. Appl.*, vol. 7, no. 1, pp. 9:1–9:14, 2016. [Online]. Available: https://doi.org/10.1186/s13174-016-0051-y

[29] R. Yang, S. Zhu, Y. Li, and I. Gupta, "Medley: A novel distributed failure detector for iot networks," in *Proceedings of the 20th International Middleware Conference, Middleware 2019, Davis, CA, USA, December 9-13, 2019*, 2019, pp. 319–331.

[30] X. Défago, P. Urbán, N. Hayashibara, and T. Katayama, "Definition and specification of accrual failure detectors," in *International Conference on Dependable Systems and Networks, DSN*, 2005, pp. 206–215.

[31] 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.

[32] N. Xiong, A. V. Vasilakos, J. Wu, Y. R. Yang, A. Rindos, Y. Zhou, W.-Z. Song, and Y. Pan, "General traffic-feature analysis for an effective failure detector in fault-tolerant wired and wireless networks," Tech. Rep., 2011.