MW-FD, A Failure Detector Algorithm with QoS, and an Analysis Towards Failure Detection as a Service
(extended abstract of the MSc dissertation)

Alejandro Zlatko Tomsic
Laboratoire d’ Informatique Paris 6 (LIP6)
Université Pierre et Marie Curie

Abstract—Distributed systems provide resources to a large number of applications today. They are designed to provide reliable and continuous services despite the failures of some of their components. Nevertheless, they are subject to a wide variety of failures and, therefore, failure detection plays a central role in the engineering of such dependable systems. Furthermore, many applications have timing constraints and require failure detectors that provide quality of service (QoS) with some quantitative timeliness guarantees. Moreover, applications usually implement their own ad-hoc failure detection modules. However, there is no systematic way of providing a failure detection service to a single host in the face of different application requirements.

In this work, we introduce the Multiple Windows Failure Detector (MW-FD). This failure detector presents an improved the QoS when compared to existing FD algorithms. We also analyse the idea of multiple applications or virtual machines, with different QoS requirements in terms of failure detection, running on a single host using a single FD as a shared service. The FD module is meant to give, to each application, the illusion of a dedicated FD that satisfies its particular needs on QoS while minimizing the number of messages exchanged in the network.

Our results show that there is always an improvement in terms of QoS when using our proposed algorithm, which varies in magnitude according to network conditions and test environments. Additionally, we have analysed how, when using a shared failure detection service, the required QoS of applications should be modified in order to coexist, showing that applications with weaker QoS requirements benefit from the ones with stronger ones by obtaining an improved QoS. Furthermore, the overall load imposed on the network is reduced when using the shared service, compared to the case of utilising one failure detector per application.

I. INTRODUCTION

Distributed systems are being increasingly important for providing resources to a large number of user applications. They are subject to a wide variety of failures and, therefore, it is important to provide knowledge information about service conditions and available resources.

Fault-tolerant distributed systems are designed to provide reliable and continuous services despite the failures of some of their components. Failure detection plays a central role in the engineering of such systems. A failure detector provides some information on which processes have crashed. i.e., it is currently being used in a wide variety of settings, such as network communication protocols, computer cluster management and group membership protocols. Effective

failure detection is essential to provide an acceptable Quality of Service (QoS) to applications and, therefore, it is necessary to find an optimized FD that can detect failures in a timely and accurate way before a generic FD service can be implemented for distributed applications.

Many applications have timing constraints. They require a FD that provides a quality of service (QoS) with some quantitative timeliness guarantees. To ensure acceptable QoS for a FD, parameters should be properly tuned to deliver a desirable QoS to the upper layers, as the QoS of FD greatly influences the QoS that upper layers are able to provide.

When detecting failures, there exists an inherent tradeoff between 1) conservative failure detection, i.e., reducing the risk of wrongly suspecting a correct process, and 2) aggressive failure detection, i.e., quickly detecting the occurrence of a real crash. There exists a continuum of valid choices between these two extremes, and an appropriate choice is strongly related to each application’s requirements on QoS.

Nowadays, applications usually implement their own ad-hoc failure detection modules. However, there is no systematic way of setting or adjusting the message sending rate or the length of the timeout in the face of different application requirements. Many people have been advocating that failure detection should be provided as a service [1], [2], similar to IP address lookup (DNS) or time synchronization (NTP). Unfortunately, in spite of important technical breakthroughs, this view has met little success so far. One of the major obstacles to building a failure detection service is that simultaneously running distributed applications with different quality-of-service requirements must be able to tune the service to meet their own needs without interfering with each other.

Finally, large scale distributed-applications have created the need for failure detectors which are efficient in terms of network load and therefore, failure detectors targeting this type of applications should address scalability and efficiency.

In this work, we propose and use a modified version of the FD algorithm developed by Chen et al. [3]. This framework provides a systematic way of choosing sending rates and timeouts to meet QoS requirements specified by applications according to network conditions. Our modified version of Chen FD algorithm improves the QoS of the original one, namely, by decreasing the number of mistakes (false detections) made per unit of time and the probability
of making mistakes for each given detection time. We also analyse the idea of multiple applications or virtual machines, with different QoS requirements in terms of failure detection, running on a single host using a single failure detector (FD) as a shared service. In such scenario, a crash of a remote host (or process) should be reported by the FD module to all applications monitoring the failed one. We analyse how a FD service with such characteristics should provide quality of service (QoS) guarantees to applications. Each application running on a host should be able to specify its own needs on speed and accuracy at which the FD detects crashes. The FD module should give, to each application, the illusion of a dedicated FD that satisfies their particular needs on QoS. Furthermore, each physical machine running the failure detection service should attempt to minimize the number of messages exchanged in the network.

We have evaluated and compared the QoS of our algorithm to the most important existing ones in terms of mistake rate and query accuracy probability. These algorithms and concepts will be introduced in section II. The obtained results show that there is always an improvement in terms of QoS when using our proposed algorithm, which varies in magnitude according to network conditions and test environments. Additionally, we have evaluated how, when using a shared failure detection service, the required QoS of applications should be modified in order to coexist. The results show that applications with weaker requirements benefit from the ones with stronger ones by getting an improved resulting QoS. Furthermore, the overall amount of messages sent through the network is reduced when using the shared service, compared to the case of utilising one failure detector per application.

The rest of this document is organized as follows. Chapter II provides an introduction to the different technical areas related to our work. Chapter III introduces the Multiple Windows Failure Detector (MW-FD) and Chapter IV evaluates it, and compares it to other failure detection algorithms. Afterwards, in Chapter V we present our study on combining multiple application’s QoS requirements on a single failure detection service. Finally, in Chapter VI we conclude this document by summarizing its main points and proposing possible directions of future work.

II. RELATED WORK

In this section we introduce background concepts and existing solutions which are important to understand our work.

Throughout this work, we only consider heartbeat unreliable failure detectors that may suspect a process that has not failed (they may be inaccurate), and not suspect a process that has in fact failed (they may be incomplete).

A. Quality of Service (QoS) Specification for Failure Detectors

In this section, we present the notions introduced by Chen et al. [3] on QoS for failure detectors.

1) Model for QoS Specification: A system of two processes, p and q, is considered. The failure detector at q monitors p, and p never crashes. Real time is continuous and ranges from 0 to ∞.

At any given time t, the output of the failure detector in q can be either S, suspect, or T, trust. Whenever the output of the failure detector in q changes, we say that a transition occurs: an S-transition occurs when the output of q changes from T to S; and a T-transition happens when the output of the failure detector at q changes from S to T. Only a finite number of transitions can take place during a finite period of time.

2) QoS Metrics for Failure Detectors: QoS metrics measure how fast and accurate a failure detector is. These metrics are applicable to all failure detectors, regardless of how they are implemented. The most relevant metrics, as introduced by Chen et al. are described in this section. Note that the first one is related to a failure detector’s speed, while the remaining relate to its accuracy.

- Detection Time (TD): it is the time that elapses from the moment that process p crashes until the failure detector at q detects the failure and starts suspecting p for ever. More precisely, TD measures the time that elapses from the moment that the crash of p occurs to the moment when the final S-transition occurs (at q) and there are no further transitions (see Figure 1).

```
Figure 1: Detection Time TD
```

- Query Accuracy Probability (PA): this is the probability that the failure detector’s output is correct at a random time. This metric is useful for applications that interact with the failure detector by querying it at random times.

- Average Mistake Rate (TMR): this measures the rate at which a failure detector makes mistakes, i.e., it is the number of S-transitions per unit of time. This is an important metric for long-lived applications where a mistake results in a costly interrupt, such as group membership applications and cluster management (see Figure 2).

- Average Mistake Duration (TM): this measures the time a failure detector takes, on average, to correct a mistake. More precisely, TM measures the average time that elapses from an S-transition to the next T-transition (remember that in the considered model p never crashes and, therefore, an S-transition output by q represents a mistake, see Figure 2). The TM metric is useful for applications that operate in a degraded mode when a
process is incorrectly suspected.

**Figure 2:** Mistake Duration $T_M$ and Mistake Rate $T_{MR}$

![Diagram](image-url)

**B. Failure Detector Algorithms**

In this section, we present the most relevant failure detection algorithms to our work. This algorithms are later used in to compare our algorithm with existing solutions in section IV.

1) Chen’s Failure Detector: Chen et al. developed a failure detector that provides QoS. The detailed implementation of the algorithm works as follows [3]. The monitored process $p$ periodically sends heartbeat messages (tagged with its sequence number $i$) $m_1,m_2,m_3,...$ to $q$ every $\Delta_t$ time units, where $\Delta_t$ is a parameter of the algorithm. $\sigma_i$ denotes the sending time of message $m_i$. The monitoring process $q$ shifts the $\sigma_i$s forward by $\Delta_t$, the other parameter of the algorithm, to obtain the sequence of times $\tau_1 < \tau_2 < \tau_3 < ...$, where $\tau_i = \sigma_i + \Delta_t$. Process $q$ uses the $\tau_i$s and the times it receives heartbeat messages to determine whether to trust or suspect $p$, as follows: Consider time period $[\tau_i; \tau_{i+1})$, at time $\tau_i$, $q$ checks whether it has received some message $m_j$ with $j \geq i$. If so, $q$ trusts $p$ during the entire period $[\tau_i; \tau_{i+1})$ (see Figure 3.a). If not, $q$ starts suspecting $p$. In a suspect state, if at some time before $\tau_{i+1}$, $q$ receives some message $m_j$ with $j \geq i$, then $q$ starts trusting $p$ from that time until $\tau_{i+1}$ (see Figure 3.b). If, by time $\tau_{i+1}$, $q$ has not received any message $m_j$ with $j \geq i$, then $q$ suspects $p$ during the entire period (see Figure 3.c). Note that from time $\tau_i$ to $\tau_{i+1}$, only messages $m_j$ with $j \geq i$ can affect the output of the failure detector at $q$. For this reason, $\tau_i$ is called a freshness point. From time $\tau_i$ to $\tau_{i+1}$, messages $m_j$ with $j \geq i$ are still fresh (useful). Therefore, the algorithm is characterized by the following property: $q$ trusts $p$ at time $t$ if and only if $q$ received a message that is still fresh at time $t$.

When clocks are not synchronized, it is not possible to use the sending times in $p$ to compute the values of arrival times $\tau$s at $q$. In this scenario, a different mechanism has to be used in order to compute the expected arrival times $EAs$ of heartbeats at $q$. This expected arrival times are then used to compute the freshness points $\tau_s$:

$$\tau_{s+1} = EA_{l+1} + \Delta_{to}$$  \hspace{1cm} (1)

where $\Delta_{to}$ is a constant safety margin chosen by the user based on her needs on detection time $T_d$.

Chen’s method to compute expected arrival times $EAs$ considers the $n$ previous messages (for some $n$), denoted $m_1',m_2',...m_n'$. Let $s_1,s_2,...,s_n$ be the sequence number of those messages and $A_{1},A_{2},...,A_{n}$ their receipt times at $q$. Then, $EA_{l+1}$ (where $l$ is the largest sequence number of heartbeats received so far) is estimated by:

$$EA_{l+1} \approx \frac{1}{n} \left( \sum_{i=1}^{n} A'_i - \Delta_s s_i \right) + (l + 1) \Delta_i$$  \hspace{1cm} (2)

This equation first normalizes each $A'_i$ by shifting it backwards $\Delta_s s_i$ time units. Then, an average of the $A'_i$s is computed and, finally, this computed average is shifted forward by $(l + 1) \Delta_i$.

2) Bertier Failure Detector: Bertier et al. introduced a failure detector principally intended for LAN environments [1]. Their algorithm uses the same mechanism as Chen for estimating expected arrival times $EAs$ (see Equation 2), but a dynamic way of computing freshness points based on Jacobson’s estimation [4], which is used in TCP to estimate the delay after which a transceiver retransmits a message. As in Chen’s FD, the arrival times of the $n$ previous messages are kept in order to compute $EAs$. Jacobson’s estimation supposes that the behavior of the system is not constant, and it is used in this algorithm to adapt the safety margin $\Delta_{to}$ each time a heartbeat is received. The adaptation of the safety margin $\Delta_{to}$ is done as a function of the error in the last estimation. Parameter $\gamma$ represents the importance of a new measure with respect to the previous ones. The delay represents the estimate margin, and $\varphi$ the magnitude between errors. $\beta$ and $\phi$ are used to ponder the variance and typical values are $\beta$ and $\phi = 4$. Upon the reception of message $m_l$, the estimation of $\Delta_{to,l+1}$ is computed as follows:

$$error_l = A_l - EA_l - delay_l$$  \hspace{1cm} (3)

$$delay_{l+1} = delay_l + \gamma.error_l$$  \hspace{1cm} (4)

$$var_{l+1} = var_l + \gamma. (|error_l| - var_l)$$  \hspace{1cm} (5)

$$\Delta_{to,l+1} = \beta delay_{l+1} + \phi.var_{l+1}$$  \hspace{1cm} (6)

Whenever a message $m_l$ is received (where $l$ is the largest sequence number seen by the failure detector at $q$ at a given moment), $\Delta_{to,l+1}$ is computed using Equations 3-6 and $EA_{l+1}$ is calculated using Equation 2. With this two values computed, the next freshness point $\tau_{s+1}$ is computed exactly as in Equation 1. The timeout for message $m_{l+1}$, activated by $q$ when it receives $m_l$, expires at the freshness point $\tau_{s+1}$. The complete algorithm is presented in the following section.
3) The \( \phi \) Accrual Failure Detector: In \( \phi \) FD, the suspicion level is given by a value called \( \phi \), expressed on a scale that is dynamically adjusted to reflect current network conditions. Let \( T_{last} \) denote the time when the most recent heartbeat was received, \( T_{now} \) the current time, and \( P_{later}(t) \) the probability of a heartbeat arriving more than \( t \) time units after the previously received one. Then, the value of \( \phi \) is calculated as follows:

\[
\phi(T_{now}) = -\log_{10}(P_{later}(T_{now} - T_{last}))
\]  

(7)

In this context, \( \phi \) has the following meaning. Given a threshold \( \Phi \), if the failure detector suspects \( p \) when \( \phi \geq \Phi \), then the probability that the \( \phi \) failure detector makes a mistake is about \( \frac{1}{10^\Phi} \).

The estimation of \( \phi \) is done as follows. When heartbeats arrive, their arrival times are stored in a sampling window (as Chen’s and Bertier’s FD algorithm, described in sections II-B1 and II-B2 respectively). These past samples are used to determine the distribution of interarrival times. Finally, the distribution is used to compute the current value of \( \phi \). The estimation of the distribution of interarrival times assumes that they follow a normal distribution. This is an approximation based on the assumption that heartbeat inter-arrival times are influenced by a very large number of independent unknown factors (central limit theorem). The parameters of the distribution are estimated by determining the mean \( \mu \) and variance \( \sigma^2 \) of the samples. Then, the probability \( P_{later}(t) \) that a given heartbeat will arrive more than \( t \) time units later than the previous heartbeat is given by the following equation:

\[
P_{later}(t) = \frac{1}{\sigma\sqrt{2\pi}} \int_t^{\infty} e^{-\frac{(x-\mu)^2}{2\sigma^2}} dx
\]  

(8)

\[
= 1 - F(t)
\]  

(9)

where \( F(t) \) is the cumulative distribution function of a normal distribution with mean \( \mu \) and variance \( \sigma^2 \). Finally, the value of \( \phi \) at time \( T_{now} \) is computed by applying Equation 7. This process is repeated by \( q \) for every new heartbeat received.

4) Exponential Distribution Failure Detector (ED FD): The Exponential Distribution failure detector (ED FD) [5] is based on the same principle as the \( \phi \) accrual failure detector. The difference lies in the distribution considered for message delays by the ED FD is exponential. In ED FD, the suspicion level is given by a value called \( e_d \), which is calculated as follows:

\[
e_d = F(T_{now} - T_{last})
\]  

(10)

\[
F(t) = 1 - e^{-\frac{t}{\mu}}
\]  

(11)

where \( T_{now}, T_{last} \) and \( \mu \) have the same meaning as in the \( \phi \) accrual failure detector (section II-B3).

Summary

In this chapter we have introduced related work in the area of failure detection for distributed systems. We started by introducing some important concepts to understand this work. Then, we briefly surveyed some of the most relevant practical systems that are related to our work.

The algorithms presented throughout this section are able to provide QoS guarantees to applications when the network behaves according to some probability distribution. Nevertheless, in the presence of bursts of lost messages, this algorithms output a large number of false positives until the network stabilizes. This occurs because the estimation of expected arrival times and freshness points by using these probabilistic approaches are dependent on the past history of observed arrival times of messages. This dependence on a big number of past samples prevents these algorithms from quickly reacting to sudden changes. In the next chapter, we further explain this problem and introduce our algorithm, which addresses unstable network behaviors.

III. MULTIPLE WINDOWS FAILURE DETECTOR

In this section we introduce the Multiple Windows Failure Detector (MW FD) and the idea behind it.

A. Dealing with Bursty Traffic

In some scenarios, the probabilistic behavior of the network (message delay and message loss) can change. The algorithms presented in section II can adapt their behavior to changing network conditions. Chen’s failure detector (see section II-B1) can be made adaptive by recomputing the heartbeat interval \( \Delta_i \) and timeout \( \Delta_{to} \), every certain period of time (the procedure for this computation will be introduced in section V-A). This way, Chen’s failure detector can adapt to gradually changing network conditions.

There are cases where network conditions change very frequently due to bursty traffic. This is more likely to happen in WAN scenarios, where message delays and losses are affected by a bigger amount of factors. Chen’s algorithm can be adapted with the above mentioned mechanism in the cases that the following conditions hold [3]:

1) the occurrences of bursts are independent of each other and follow some slowly changing probabilistic distribution.
2) the duration of each burst is short (smaller than the heartbeat interval \( \Delta_i \)).

In this case, the situation is no different from the previous one, as heartbeats behave independently of each other, according to some new slowly changing probability distribution that takes into account the occurrence of bursts.

When 1) or 2) do not hold, some mechanism to estimate the current behavior of the network and adapt to it is needed. In the following section, we present our algorithm. It introduces a mechanism to tackle this problem.
B. Rationale

In order to adapt to bursty-traffic conditions, we propose the use of two components for the estimation of expected arrival times EA$s$ and freshness points $\tau$s. Namely:

1) a short-term component that considers only the most recent messages, which is used to quickly react to sudden changes in network conditions, maybe due to bursty traffic, and

2) a long-term component that considers a bigger amount of recently received messages, that is not sensitive to momentary fluctuations, which is used to make more conservative estimations when the recent messages have been fast.

Whenever a message $m_i$ is received by $q$, both components are used to estimate the freshness point $\tau_{i+1}$ for $m_{i+1}$, as we will explain in the following section.

C. The Algorithm

The Multiple Windows Failure Detector (MW FD) algorithm is a variation of Chen’s failure detector (introduced in section II-B1). The main difference lays in the fact that the MW FD keeps two arrays, $A(n_1)$ and $A(n_2)$ (of sizes $n_1$ and $n_2$ respectively), of recently received heartbeat arrival times instead of keeping only one. Whenever a message $m_i$ sent by $p$ is received by $q$, $q$ adds the arrival time of message $m_i$ to $A(n_1)$ and to $A(n_2)$. The following step is to compute, using the values stored in $A(n_1)$ and to $A(n_2)$, the expected arrival times $EA_{t+1}(n_1)$ and $EA_{t+1}(n_2)$. The key of the algorithm is that, from the EA's computed, it uses the maximum of these estimations for the computation of the freshness point $\tau_{t+1}$:

$$\tau_{t+1} = \max(EA_{t+1}(n_1), EA_{t+1}(n_2)) + \Delta_{to} \quad (12)$$

where $\Delta_{to}$ is a constant safety margin as specified by the QoS requirements on $TD$ (see section II-A). Finally, if message $m_{i+1}$ is not received before time $t = \tau_{i+1}$, $q$ starts suspecting $p$. The rest of the process occurs exactly as in Chen’s failure detector (section II-B1).

Intuitively, given two window sizes $WS_1$ and $WS_2$, our algorithm should be able to make less mistakes than Chen’s algorithm when using any of $WS_1$ or $WS_2$, as for each analysed sample, the MW-FD computes the maximum of the expected arrival times that would be computed by Chen’s algorithm for each window size. This implies that this algorithm will only make the mistakes that Chen’s FD would make when using both window sizes $WS_1$ and $WS_2$. This means:

$$\text{Mistakes(MW}_{WS_1, WS_2} = \text{Mistakes(Chen}_{WS_1} \cap \text{Mistakes(Chen}_{WS_2} \quad (13)$$

We will analyse this with an example in our evaluation chapter (see section IV-C4).

Algorithm 1 Multiple Windows Failure Detector Algorithm

```
Process $p$: \triangleright Using $p$’s local clock
1: for all $i \geq 1$ do
2: at time $i \cdot \Delta_i$ send heartbeat $m_i$ to $q$
3: end for

Process $q$: \triangleright Using $q$’s local clock
4: Initialization:
5: $\tau_0 = 0$;
6: $l = -1$; \triangleright keeps the largest sequence number of
7: $A(n_1) = \{0\}$: \triangleright contains the last $n_1$ arrival
8: $A(n_2) = \{0\}$: \triangleright contains the last $n_2$ arrival
9: $EA_0 = EA_{t+1}(n_1) = EA_{t+1}(n_2) = 0$;

10: upon $\tau_{i+1}$ = the current time:
11: output $\leftarrow S$; \triangleright suspect $p$ since no message
received is still fresh at this time
12: upon receive message $m_j$ at time $t$:
13: if $j > l$ then \triangleright Received a message with a higher
sequence number
14: $l \leftarrow j$;
15: $A \leftarrow A(n_1) \cup \{t\}$
16: $A \leftarrow A(n_2) \cup \{t\}$
17: Compute $EA(n_1)_{t+1}$ and $EA(n_2)_{t+1}$ \triangleright using
Equation 2
18: $EA_{t+1} \leftarrow \max(EA_{t+1}(n_1), EA_{t+1}(n_2))$
19: $\tau_{i+1} \leftarrow EA_{t+1} + \Delta_{to}$ \triangleright set the next freshness
point using Equation 1
20: if $t < \tau_{i+1}$ then
21: output $\leftarrow T$;
22: end if
23: end if
```

D. Benefits of Using Two Windows

Intuitively, our algorithm is meant to, and should, work better than Chen’s FD mainly in the presence bursty traffic and rapid changes in network conditions, because of the reasons explained in section III-C.

In our implementation it is easy to see that, by picking the maximum of the estimations on expected arrival times for expected heartbeats, the failure detector at $q$ becomes more tolerant, i.e., more conservative. This occurs because for each heartbeat, our failure detector will wait for the maximum of the times estimated by each window, a fact that directly reduces the probability of making mistakes. At first sight, this would suggest that this algorithm would not be able to work in very aggressive ranges of detection, where the required $TD$ is very small. Our measurements show that this is not the case, and that this algorithm performs better
than all the ones we compared it to in terms of amount of mistakes ($T_{MR}$) and accuracy ($P_A$), even in the aggressive ranges. We will provide empirical evidence of this fact in our evaluation chapter (section IV).

IV. EVALUATION

In this section, we present evaluation results of the tests we performed to study the performance of our algorithm and the algorithms presented in section II-B. Namely, we evaluate and comparatively analyze the performances of MW FD, the $\phi$ accrual FD, ED FD, Chen’s FD, and Bertier’s FD when ran over traces taken from two experimental environments, a WAN and a LAN.

A. About the experiments

All experiments were performed on traces of experiments ran on two computers. They worked as follows: one process $p$ periodically sends heartbeat messages to another process $q$ for an arbitrarily long period of time. When heartbeats are received, their arrival times are logged by the monitoring computer $q$. Then, these logged arrival times are used to replay the execution for each FD algorithm. Therefore, all failure detectors were compared in the same experimental conditions. Heartbeat messages were sent using the UDP/IP protocol. During the experiments the average CPU load was nearly constant and below the computers’ full capacity.

B. Tests Scenarios

For our experiments, we utilised two traces. One taken from a WAN scenario, and the other one, from a LAN scenario.

1) WAN Scenario: This experiments involved two computers: one located in Switzerland, and the other in Japan. They communicated through a normal intercontinental Internet connection. We used exactly the same trace files as in [6], which are publicly available [7]. Neither machine failed during the experiment. There is a period (April 6 and 7) where more messages were lost. According to the authors, this was likely caused by an outbreak of the W22/Netsky:T@mm Internet worm (dates coincided).

2) LAN Scenario: This scenario is practically the same, with the difference that computers are interconnected through a LAN cable. The experiment used two identical computers located at JAIST and connected through a single unshared 100 Mbps Ethernet hub. The heartbeat interval $\Delta_i$ was set to 20ms and the experiment run for a little more than a day. 7,104,446 samples were collected. Not a single heartbeat was lost. The largest interval between the reception of two heartbeats was about 1.5 seconds. Nevertheless, the variance was very small. The average transmission delay was around 100 $\mu$s.

C. Experiments

1) MW-FD - Window Sizes: This experiment measures the effect of window sizes on the performance of our developed algorithm, the Multiple Windows failure detector. We set both windows from a very small size (one sample) to a very large one (10,000 samples) and measured the accuracy obtained with our failure detector when run over the WAN trace.

Figure 4 shows the results on mistake rate $T_{MR}$ vs. detection time $T_D$ in the WAN scenario. $T_{MR}$ is represented on the vertical axes, expressed in logarithmic scale, and $T_D$ in the horizontal axes. Figure 5 shows the results on query accuracy probability $P_A$ vs. detection time $T_D$ in the same scenario.

The results clearly show that, in terms of both $T_{MR}$ and $P_A$, our algorithm behaves better when using:

1) one small window (the smaller, the better) and
2) one big window (the bigger, the better).

The results also show that the performance to accuracy ratio of the algorithm decreases as the size of the small window increases, and as the size of the big window decreases. Besides, it’s also noticeable from the figures that curves for tests which share the same size for the small window tend to behave similarly. We also conclude from this analysis that the increase in accuracy obtained by our algorithm is negligible for each value of $T_D$ for sizes of the long term (big) window bigger than 1,000. For the small window, the experiments suggest that the best size is one
This captures the idea introduced in section III-B that one component should be reactive to very recent behavior.

2) Comparison to Other Algorithms: In this experiment, we compare the behavior of the MW-FD with four well-known failure detectors. Namely, the failure detectors of Chen et al., Bertier et al., and the two accrual failure detectors; $\phi$ and ED FD. In this experiment we intend to show that the MW-FD presents the best detection time to accuracy ratio. Chen and the MW failure detectors share a common tuning parameter, the safety margin $\Delta_{to}$, which we vary in our experiments to get the different values of detection time. The tuning parameter for the accrual failure detectors was the threshold $\Phi$. Unlike the rest of the failure detectors, Bertier’s has no tuning parameter. For this reason, its behavior is plotted as a single point on the figures. Finally, the values of window sizes for the rest of the failure detectors were set to:

- 1 and 1.000 for Chen’s failure detector because we use these values for our failure detector. Besides, 1.000 is the commonly used value used in related work experiments.
- 1000 for the $\phi$ and the ED failure detectors. These failure detectors benefit from using large window sizes. However, we have tested and discovered that for window sizes beyond 1.000 samples, the improvement these algorithms obtain is negligible. Furthermore, this is the value used by their authors in their respective works [6], [5].
- 1000 for Bertier’s FD, as that is the value their authors use in their work [1].
- One window of size 1 and one window of size 1000 for our algorithm, the MW-FD. These values were chosen as the algorithm presents the best behavior under such configuration.

Figure 6 shows the results on mistake rate $T_{MR}$ vs. detection time $T_D$ in the WAN scenario. $T_{MR}$ is represented on the vertical axes, expressed in logarithmic scale, and $T_D$ in the horizontal axes. Figure 7 shows the results on query accuracy probability $P_A$ vs. detection time $T_D$ in the same scenario. Results in our tested LAN scenario present the same behavior and we do not plot them as they do not add relevant information.

The results indicate that all algorithms follow the same tendency. Our algorithm seems to outperform the rest in both scenarios, and in both the aggressive and conservative ranges. It presents the lowest mistake rate and the best query accuracy probability for all measured detection times. Note that in both graphs, the curve of the accrual failure detector with normal distribution ($\phi$) is stopped early. This is due to the rounding error preventing the graphs to the very conservative case [5].

3) Analysis for a Single Detection Time: $T_D = 215ms$: This experiment measures the number of mistakes each failure detector makes in the WAN scenario as time passes. We evaluate this for a unique detection time $T_D=215ms$. We chose this value because it makes failure detectors work in an aggressive range and we are able to obtain results for most of the failure detectors. The only failure detector that can not be parametrized to obtain this $T_D$ is Bertier’s. The goal of this experiment is to further analyse the behavior of the MW-FD and to determine which conditions make this FD obtain an overall better accuracy.

We split the total sample space into four subsamples, as Table I shows. First, there is one stable period that we call Stable 1, which starts from the beginning of the sample and lasts until the moment before the big mistake burst in the middle of the graph (on the horizontal axes) starts. After this period, there is the Burst period and then, the Worm period that coincides with the W32/Netsky.T@mm Internet worm (see section IV-B1). The last Stable 2 period starts after the Worm period, and lasts until the end of the sample.

<table>
<thead>
<tr>
<th>Name</th>
<th>From Sample</th>
<th>To Sample</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stable</td>
<td>1</td>
<td>2900000</td>
</tr>
<tr>
<td>Burst</td>
<td>2900001</td>
<td>2930000</td>
</tr>
<tr>
<td>Worm Period</td>
<td>2930001</td>
<td>4860000</td>
</tr>
<tr>
<td>Stable 2</td>
<td>4860001</td>
<td>5845712</td>
</tr>
</tbody>
</table>

Table I: Division of the WAN sample into smaller subsamples

Figure 8 shows the total number of mistakes each failure
detector made on each sub-period and The results show that the MW-FD performs better in all scenarios, but particularly better during the Burst period. This is consequent with the algorithm’s goal.

**Figure 8:** WAN: Total Mistakes made during each subsample for fixed $T_D=215$ ms

4) MW-FD vs. Chen’s FD:  
**Mistake Reduction:** As introduced in Equation 13 (see section III-C), given two window sizes $W_1$ and $W_2$, the MW-FD makes less mistakes than Chen’s algorithm when using any of $W_1$ or $W_2$. In this section we illustrate this idea with an example in the WAN scenario, where $T_D=215$ms, $W_1 = 1$ and $W_2 = 1000$. We compare MW-FD($W_1$, $W_2$) to Chen-FD($W_1$) and Chen-FD($W_2$). Figure 9 shows which mistakes each failure detector make. One can note the effect of the use of two windows as the picture clearly shows that MW-FD($W_1$, $W_2$) only makes the mistakes that both Chen-FD($W_1$) and Chen-FD($W_2$) make.

**Figure 9:** Mistakes made by Chen and MW-FD, WAN scenario

V. TOWARDS FAILURE DETECTION AS A SERVICE

In this section, we study the possibility of providing a single failure detection service to multiple applications or virtual machines running on a single (same) physical computer. Each of these applications (or VMs) present their own, and probably different, requirements on failure detection QoS. In particular, we study how to combine these requirements in order to send a single heartbeat per physical machine. The goal of this analysis is to provide each application with the illusion of a dedicated failure detector that fulfills its particular requirements in terms of QoS while minimizing the load imposed on the network.

A. Configuring a FD to Satisfy a QoS Specification

Chen et al. [3] introduced a mechanism for applications to express their QoS requirements and obtain, for specific network conditions, the rate $\Delta_i$ at which a monitored machine ($p$ in our model) should send heartbeats to a monitoring machine ($q$) and the timeout time $\Delta_{io}$ this machine should wait for a new heartbeat before it starts suspecting $p$, that make the algorithm satisfy the required QoS.

The QoS requirements are assumed to be expressed as a tuple $(T_D, T_{MR}, T_M)$, where $T_D$ is an upper bound on the detection time, $T_{MR}$ is an upper bound on average mistake rate (or, equivalently, a lower bound on the average mistake recurrence time), and $T_M$ is an upper bound on the average mistake duration. In other words, the failure detector should provide to the application, a tuple $(T_D, T_{MR}, T_M)$ that satisfies:

- $T_D \leq T_D^U$
- $T_{MR} \leq T_{MR}^U$
- $T_M \leq T_M^U$

Chen et al. introduced a configuration mechanism based on a programing problem that can be solved by using a numerical method. This mechanism is used to configure their algorithm, and ours, in order to meet QoS requirements. The explanation of how this problem was formulated can be found in [3]. The configuration procedure takes as inputs:

1) the QoS requirements. Namely, the tuple $(T_D^U, T_{MR}^U, T_M^U)$, and
2) the probabilistic behavior of the heartbeat messages. Namely $p_L$, the probability of a message being dropped by the network; and $V(D)$, the variance of message delays.

Then, it outputs the failure detector parameters $\Delta_i$ and $\Delta_{io}$, so that the failure detector satisfies the application’s specific QoS requirements. Furthermore, to minimize the network bandwidth used, the configuration procedure finds the largest inter-sending interval $\Delta_{i,max}$ that satisfies these QoS requirements. It works as follows [3]:

- **Step 1:** Compute
  $$\gamma' = \frac{(1-p_L)(D_T^U)^2}{V(D) + (D_T^U)^2}$$
  (14)
  and let
  $$\Delta_{i,max} = min(\gamma' \dot{D}_T, T_M^U).$$
  (15)

  If $\Delta_{i,max} = 0$, then the QoS cannot be achieved and stop; else continue.

- **Step 2:** Let
  $$f(\Delta_i) = \frac{[T_D^U/\Delta_i]^{-1}}{V(D) + (T_D^U - j \Delta_i)^2}$$
  $$\prod_{j=1}^{[T_D^U/\Delta_i]} V(D) + p_L(T_D^U - j \Delta_i)^2$$
  (16)
Find the largest $\Delta_i \leq \Delta_{i,max}$ such that $f(\Delta_i) \leq T^{U}_{MR}$.

Such an $\Delta_i$ always exist and can be computed using a numerical method.

- **Step 3**: Set $\Delta_{to} = T^{U}_{D} - \Delta_i$ and output $\Delta_i$ and $\Delta_{to}$.

The information output by this process, plus the expected arrival times $EAs$ are used by the MW-FD (or Chen’s FD) for estimating expected arrival times. Finally, note that it is possible to run the configuration procedure periodically in order to make the algorithm adaptive to changes in the probabilistic behavior of the network.

1) Estimating $p_L$ and $V(D)$: It is easy to estimate $p_L$ and $V(D)$ using heartbeat messages; $p_L$ can be estimated by using the sequence numbers of the heartbeat messages to count the number of missing heartbeats and then dividing this count by the highest sequence number received so far. To estimate $V(D)$, the mechanism is as follows; when $p$ sends a heartbeat $m$, it timestamps $m$ with the sending time $S$ and, when $q$ receives $m$, $q$ records the receipt time $A$. In this way, $A - S$ is the delay of $m$. Then the average and variance of $A - S$ are computed for multiple past heartbeat messages, obtaining accurate estimates for $V(D)$. This process works even though the clocks are not synchronized as the procedure estimates $V(D)$ by computing the variance of $A - S$ of multiple heartbeat messages, where $A$ is the time (with respect to $q$’s local clock) when $q$ receives $m$ and $S$ is the time (with respect to $p$’s local clock) when $p$ sends $m$. When clocks are not synchronized, $A - S$ is the delay of $m$ plus the skew between the clocks of $p$ and $q$. Thus, the variance of $A - S$ is the same as the variance $V(D)$ of message delays.

**B. Determining $\Delta_i$ and $\Delta_{to}$ from Multiple QoS Requirements**

In this section, we explain how to combine the QoS requirements of multiple applications running on a single physical computer in order to determine the $\Delta_i$ and $\Delta_{to}$ values for the heartbeats of a physical machine running a failure detection service for multiple applications.

1) Measuring the Impact of Single Parameters on $\Delta_i$ and $\Delta_{to}$: We propose that applications express their QoS requirements as a tuple ($T^{U}_{D}, T^{U}_{MR}, T^{U}_{M}$). In this section, we study how the variation of each of the parameters in the tuple affects the resulting $\Delta_i$ and $\Delta_{to}$.

**Varying Detection Time:** Figure 10 shows the impact of varying the detection time $T_D$ on the resulting $\Delta_i$ and $\Delta_{to}$, computed by the mechanism introduced in section V-A. The graph shows that, as $T_D$ grows, both $\Delta_i$ and $\Delta_{to}$ grow linearly. This is obvious as $T_D = \Delta_i + \Delta_{to}$. The ratio at which they grow is determined by the remaining QoS parameters.

**Varying Mistake Rate:** Figure 11 shows the impact of varying the detection time $T_{MR}$ on the resulting $\Delta_i$ and $\Delta_{to}$. The graph shows that, as $T_{MR}$ grows, $\Delta_i$ decreases and $\Delta_{to}$ grows. This happens as the requirement becomes more constrained when $T_{MR}$ grows (less mistakes are allowed). After a certain point (in this example, when $T_{MR}=56$s), $\Delta_i$ and $\Delta_{to}$ remain constant. The ratio at which $\Delta_i$ and $\Delta_{to}$ vary, as the point where they start remaining constant is determined by the remaining QoS parameters.

**C. $\Delta_i$ and $\Delta_{to}$ for multiple QoS**

In this section, we propose a solution for adapting $\Delta_i$ and $\Delta_{to}$ to multiple applications with different QoS requirements running on the same physical computer while minimizing the number of sent messages through the network. Considering $n$ applications or VMs running on a single physical machine, the procedure works as follows:
• **Step 1:** For each application $app_j$, where $j = 1, ..., n$, input the QoS requirements tuple and compute $\Delta_{i,j}$ and $\Delta_{to,j}$ using Chen's equation (introduced in section V-A).

• **Step 2:** From all computed $\Delta_{i,j}$, compute $\Delta_{i,min} = \min(\Delta_{1,2}, ..., \Delta_{i,n})$ and use it for the physical machine (and for every $app_j$).

• **Step 3:** Use, for each $app_j$, $\Delta_{to,j} = T_{D,j} - \Delta_{i,min}$, where $T_{D,j}$ is the detection time of $app_j$ and $\Delta_{i,min}$ is the minimum of the heartbeat intervals computed in step 1.

• **Step 4:** The FD service uses $\Delta_{i,min}$ for sending heartbeat and computes freshness points $\tau_{i,j}$ differently for each $app_j$ by using each $\Delta_{to,j}$ and timeout differently for each $app_j$.

1) **Consequences on the QoS of different applications:**

By using this mechanism, for each $app_j$:

- the detection time is maintained, exactly, as $T_{D} = \Delta_{i} + \Delta_{to}$,
- the applications that obtain a modified (adapted) QoS reduce their mistake rate. This happens because, when decreasing the $\Delta_{i}$ and increasing $\Delta_{to}$, the $T_{MFR}$ is reduced as a consequence of the illustrated in figure 11, and
- the applications that obtain a modified QoS also reduce their mistake duration. This happens because, when decreasing the $\Delta_{i}$ and increasing $\Delta_{to}$, the $T_{MFR}$ is reduced as a consequence of the illustrated in figure 12.

As a conclusion, using the shortest heartbeat interval and adapting the timeout to meet exactly the detection time required by each application improves the QoS of the adapted applications (namely, the applications which do not present $\Delta_{i,min}$) in terms of mistake duration and mistake rate and, therefore, this mechanism maintains of improves the overall failure detection QoS that a single service provides to all applications. Furthermore, network traffic is reduced from the case of using a single failure detector per application, because in that case, for each $app_j$ a heartbeat should be sent every $\Delta_{i,j}$.

**VI. CONCLUSIONS AND FUTURE WORK**

In this work, we introduced the Multiple Windows Failure Detector (MW-FD), an algorithm for failure detection in distributed systems which provides QoS guarantees. Our experiments in both WAN and LAN scenarios indicate that our failure detector presents a better QoS when comparing to existing FD algorithms. Namely, it reduces the number of mistakes (false detections) made per unit of time. The difference varies in magnitude according to network conditions and test environments.

We have also studied the idea of multiple applications or virtual machines, with different QoS requirements in terms of failure detection, running on a single host using a single FD as a shared service. We have proposed a mechanism for adapting heartbeat inter-sending rates and timeout values of applications coexisting in a single physical machine that use a single failure detection mechanism. Our proposed idea implies that when using a shared failure detection service, applications with weaker QoS requirements benefit from the ones with stronger ones by obtaining an improved QoS. Furthermore, the overall load imposed on the network is reduced when using the shared service, compared to the case of utilising a failure detector per application.

An empirical analysis on resulting QoS of applications using the service as well as a study on how network traffic is reduced by using this approach are possible directions of future work.

**ACKNOWLEDGMENTS**

Parts of this work have been performed in collaboration with other members of the Laboratoire d’Informatique Paris 6 (LIP6), namely, Luciana Arantes and Julien Sopena.

**REFERENCES**


