Design and Analysis of Quality of Service on Distributed Fault-tolerant Communication Networks

by

Naixue XIONG

submitted to
Japan Advanced Institute of Science and Technology
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy

Supervisor: Associate Professor Xavier Défago

School of Information Science
Japan Advanced Institute of Science and Technology

March, 2008

Abstract

Fault-tolerant distributed systems are designed to provide reliable and continuous service despite the failure of (one or more faults within) some of its components. In such systems, failure detector is a basic block and also is at the core of many fault-tolerant algorithms and applications. It can be found in many systems, such as ISIS, Ensemble, Relacs, Transis, Air Traffic Control Systems. Fault-tolerant systems are designed to provide reliable and continuous services for distributed systems despite the failures of some of their components [4-8]. As an essential building block for fault-tolerant systems, failure detector (FD) plays a central role in the engineering of such dependable systems. Therefore, ensuring quality of service of failure detector is very significant for ensuring fault tolerance of distributed systems.

The goal of this thesis is to explore and present novel failure detectors and an Active Queue Management (AQM) scheme to improve quality of service (QoS) of communication networks.

On the one hand, I present three novel failure detectors: Tuning adaptive margin failure detector (TAM FD), Exponential distribution failure detector (ED FD) and Self-tuning failure detector (SFD) and analyze their implements. For the proposed TAM FD, it can effectively adjust its safety margin to achieve satisfactory quality of service in communication networks, especially, in unstable and frequently changeful networks. For ED FD, it is an optimization over existed methods. In ED FD, Exponential Distribution, instead of Normal Distribution in [18, 19], is used for estimation of the distribution for inter-arrival time. Experimental results demonstrated that ED FD over-performs the existed methods from the view of quality of service of failure detector. So far, a lot of failure detectors are designed to try to satisfy different QoS requirements. However, there are no any self-tuning scheme presented. That is, for a given QoS requirement, how do the parameters of failure detectors are tuned by itself to satisfy such requirement? Therefore, in this thesis we address this problem and present a self-tuned failure detector.

Furthermore, the κ failure detector [3] is an instance of accrual failure detector [19]. This allows for a clearer separation between the monitoring of the system and the interpretation of suspicion information by applications. Hayashibara in [3] gave the original idea and definitions about the κ FD. While the performance evaluation and analysis is not enough. Therefore, a question then arise: what is the performance characteristic of κ FD compared with the existed failure detectors? In this thesis, we analyze quality of service of κ failure detector based on a lot of experiments.

On the other hand, failure detection is generally based on distributed communication networks. Reversely, the performance of communication networks also affects quality of service of failure detector. Therefore, it becomes very necessary to improve the performance of communication network. The another goal of our research is to design and analyze new schemes for active queue management to support TCP flows. AQM is an effective method used in Internet routers for congestion avoidance, and to achieve a tradeoff between link utilization and delay. The de facto standard, the Random Early Detection

(RED) AQM scheme, and most of its variants use average queue length as a congestion indicator to trigger packet dropping. We proposes a novel packet dropping scheme, called Self-tuning Proportional and Integral RED (SPI-RED), as an extension of RED. SPI-RED is based on a Self-tuning Proportional and Integral feedback controller, which considers not only the average queue length at the current time point, but also the past queue lengths during a round-trip time to smooth the impact caused by short-lived traffic dynamics. Furthermore, we give theoretical analysis of the system stability and give guidelines for selection of feedback gains for the TCP/RED system to stabilize the average queue length at a desirable level. The proposed method can also be applied to the other variants of RED. The simulation results have demonstrated that the proposed SPI-RED algorithm outperforms the existing AQM schemes in terms of drop probability and stability. Thus, the presented active queue management schemes improved the communication performance of networks, so as to ensure good quality of service of failure detection.

In all, the contributions of this thesis are composed of two parts. First we presented several FD schemes to improve the performance of the existed failure detector. Then we design and analyze a new active queue management, called SPI-RED, to support TCP flows, thus it achieves smaller queuing delays and higher throughput by purposely dropping out packets.

keywords Failure detector, Fault tolerance, Distributed system, Computer networks, Communication system traffic, Communication networks, Feedback systems, Load flow analysis, Load flow control, Real time systems, Traffic control (communication).

Acknowledgments

During my time at Japan Advanced Institute of Science and Technology (JAIST), I have been lucky and honorable to be a member of Dependable Distributed Systems Laboratory (DDSL), Foundations of Software Laboratory. I appreciated Professor Xavier Défago very much for letting me join in JAIST in 2005. As my supervisor, Professor Xavier Défago is a great technical researcher and an erudite teacher. I would like express my sincere gratitude to my supervisor for advising me and guiding my research during my study at JAIST. He provided me a lot of encouragement and inspiration for my research. Also, he is an incredible source of knowledge, and provided excellent guidance on technical details and publication writing. I am especially thankful for all the time he provided me in improving this dissertation.

Professor Takuya Katayama gave me a lot of encouragement to pursue my own ideas and to manage my own research. Most notably, he provided a family- friendly environment that helped me balance my life with my studies.

Students in the DDSL made it a great place to work. Both supervisors and collegemates made my Ph.D. experience truly unique. In particular, Dr. Yan Yang deserve to be recognized for her friendship and great contributions during the most difficult and frustrating years of this process. Dr. Yan Yang has been both a great researcher and one of the most dependable people I know.

Furthermore, I want to thank Dr. Rami Yared and Dr. Samia Souissi for all their help and their encouragement. I also would like to thank Daiki Higashihara, my kind tutor, for his great help in my living and work.

I would like to thank Professor Hong Shen for his helpful discussions and suggestions, who is my supervisor of my sub-thesis (minor project). He is able to provide in-depth analysis of my work, and also give me strong spirit encouragement on how to do research.

I also wish to express my thanks to Chair Professor Xiaohua Jia in Department of Computer Science, City University of Hong Kong for his helpful suggestions, discussions, and continuous encouragements.

I devote my sincere thanks and appreciation to all of my colleagues for their potential care.

In particular, I sincerely appreciate the support from my family. I am grateful to my parents, who have always helped me out whenever asked. I often called my parents for communication, and their care gave me inspiration and happiness whenever times were difficult. Thus, they provided an incredible support net while growing up. I also grateful to elder brother and elder sister. they help me take care of my parents, this gives me free time and spirit for my pursuing research in science. They have encouraged me all along the way.

I would like to thank Japanese Government (Monbukagakusho: MEXT) Scholarship, and COE (Strategic Development of Science and Technology) foundation in Japan for supporting this research.

Contents

Abstract i

Acknowledgments iii

1 Introduction 1
- 1.1 Failure detector . . . 1
  - 1.1.1 Failure detector and motivation . . . 1
  - 1.1.2 Related work of failure detector . . . 3
- 1.2 Active queue management . . . 5
- 1.3 Relation between failure detector and active queue management . . . 7
- 1.4 Contributions . . . 8
- 1.5 Organization of the thesis . . . 9

2 Background, System Models and Definitions 10
- 2.1 System models . . . 10
  - 2.1.1 Synchrony . . . 10
  - 2.1.2 Process failure models . . . 11
  - 2.1.3 Interaction models . . . 11
- 2.2 Failure detectors . . . 12
- 2.3 Quality-of-service of failure detectors . . . 14
- 2.4 Our system model . . . 15
- 2.5 Adaptive failure detectors . . . 16
  - 2.5.1 Chen FD . . . 17
  - 2.5.2 Bertier FD . . . 17
  - 2.5.3 Accrual failure detectors . . . 18
  - 2.5.4 The ϕ FD . . . 18
  - 2.5.5 κ-Failure Detectors . . . 19

3 Tuning Adaptive Margin Failure Detection 23
- 3.1 Tuning adaptive margin failure detection . . . 23
- 3.2 Algorithm correctness . . . 24
  - 3.2.1 Strong completeness . . . 26
- 3.3 Performance evaluation . . . 30
  - 3.3.1 Experiment in a Cluster group . . . 31
  - 3.3.2 Experiment in a WiFi network . . . 33
  - 3.3.3 Experiment in a LAN . . . 35
  - 3.3.4 Experiment in a WAN . . . 36
  - 3.3.5 Comparative analysis for the four FDs . . . 38

3.3.6 Effect of history-track-record length on QoS of FDs . . . 38
3.4 Conclusion . . . 41

4 Exponential Distribution Failure Detector 44
4.1 Exponential distribution failure detector . . . 44
4.1.1 ED FD algorithm . . . 45
4.1.2 Why ED FD is an optimization over ϕ FD . . . 45
4.1.3 Implementation of ED FD . . . 47
4.2 Algorithm correctness . . . 48
4.3 Performance evaluation . . . 50
4.3.1 Experiment in a Cluster group . . . 51
4.3.2 Experiment in a Wireless network . . . 53
4.3.3 Experiment in a LAN . . . 54
4.3.4 Experiment in a WAN . . . 55
4.3.5 Comparative analysis of the four FDs . . . 58
4.4 Conclusion . . . 58

5 Performance Evaluation of Kappa Failure Detector 60
5.1 Experimental evaluation . . . 61
5.1.1 Experiment settings . . . 61
5.1.2 Experiments overview . . . 63
5.1.3 Experimental results and discussions . . . 63
5.2 Conclusion . . . 74

6 Self-tuning Failure Detection 75
6.1 The self-tuning failure detector scheme . . . 76
6.1.1 The theory of self-tuning failure detector . . . 76
6.1.2 The implementation of self-tuning failure detector . . . 77
6.2 Performance evaluation . . . 77
6.2.1 Experimental environment . . . 77
6.2.2 Experimental results . . . 78
6.3 Conclusion . . . 81

7 Active Queue Management 82
7.1 Background of AQM . . . 82
7.2 System model and definitions . . . 83
7.3 AQM schemes and probability functions . . . 85
7.4 The SPI-RED algorithm . . . 87
7.4.1 Packet drop probability . . . 87
7.4.2 Stability analysis and control gain selection . . . 88
7.4.3 The specific algorithm of SPI-RED . . . 94
7.5 Performance evaluation . . . 95
7.5.1 Simulation 1: stability under extreme conditions . . . 95
7.5.2 Simulation 2: response with a variable number of connections . . . 99
7.5.3 Simulation 3: comparisons with existing AQM schemes . . . 101
7.6 QoS of failure detectors influenced by AQM . . . 109
7.6.1 System structure . . . 109
7.6.2 Why improving QoS of network will improve QoS of failure detector 111

7.6.3 Theoretical network communication model for the wide-area failure detection service . . . 112
7.7 Conclusion . . . 113

8 Conclusion and Future Work **114**
8.1 Failure detector . . . 114
8.2 Active queue management . . . 116
8.3 Future work . . . 117

References **118**

Publications **126**

Chapter 1

Introduction

This part covers two related topics: one is about failure detector, the other one is active queue management scheme in network routers. In this part, we first talk about the related work and motivation about this two topic respectively. And then we present the main contributions of this thesis.

1.1 Failure detector

1.1.1 Failure detector and motivation

With the development of networks, distributed systems have gradually evolved to take a prominent and important position in our society, and they are playing an essential role in many activities. In particular, distributed systems on a very large-scale are with many participants and long distances.

Failure detector (FD) is a building block for fault tolerant computing in distributed systems [4-5]. The asynchronous (i.e., no bound on the process execution speed or message-passing delay) distributed systems make it impossible to determine precisely whether a remote process has failed or has just been very slow [6-9].

Fault-tolerance is particularly prominent to distributed systems in general, especially important in large-scale settings. Normally, users of these systems expect distributed systems to remain operational (continue working) in spite of technical failures, even if some of the participants of these systems have crashed. With a large number of participants and long running time, the probability that the hosts crash during the execution is inevitable, regardless of the physical reliability of each individual host. Thus, an effective system must be designed and executed in such a way that the system can tolerate seamlessly a reasonable number of host failures, and the occurrence of a reasonable number of host failures is accepted.

Failure detection and process monitoring are the basic components of most techniques for fault-tolerance (tolerating failures) in distributed systems, such as ISIS, Ensemble, Relacs, Transis, Air Traffic Control Systems. Now how to execute failure detectors over local networks is a rather well-known issue, but it is still far from being a solved problem with large-scale systems. Because large-scale distributed systems have a different character from the local networks: such as the potentially very large number of monitored processes, the higher probability of message loss, the ever-changing topology of the system, and the high unpredictability of message delays. The above prominent factors fail

to be addressed by the traditional solutions. To effective communication in large-scale distributed systems and because of its importance [6, 39, 48-51], it is highly desirable for failure detectors to be executed as a common generic service shared among distributed applications (similar to IP address lookup) (e.g., [6, 49, 50]) rather than as redundant ad hoc implementations (e.g., [1]). If such generic service can be got, it is very easy to apply failure detectors in any kinds of applications to ensure the requirement of fault tolerance. In spite of many ground-breaking advances made on failure detection, such a service still remains at a distant horizon [3].

The design of dependable FDs is a hard task, mainly because of the indefinable statistic behavior of communication delays. Furthermore, asynchronous (i.e., no bound on the process execution speed or message-passing delay) distributed systems make it impossible to determine precisely whether a remote process has failed or has just been very slow [9]. Failure detectors can be seen as one oracle per process. An oracle provides a list of processes that it currently suspects to have crashed. And the unreliable FD [9] can make mistakes by erroneously suspecting correct processes or trusting crashed processes. Many fault-tolerant algorithms have been proposed [9, 10, 11, 29] based on unreliable FDs. It is utmost important to ensure acceptable quality-of-service (QoS) of FD to properly tune its parameters for the most desirable QoS to be provided, because the QoS of FD greatly influences the QoS that upper layers may provide. However, there are few papers about comparing and implementing of these detectors [11].

A set of metrics have been proposed by Chen et al. in [30] to quantify the QoS of a FD: how fast it detects actual failures and how well it avoids false detections. However, so far it has still been a very difficult problem to ensure acceptable QoS due to the relative unpredictability of the networking environment. Furthermore, there are some other important problems need to address, such as finding tradeoff between metrics and satisfying variable application requirements.

The traditional failure detectors are based on a simple interaction model, where processes can only send heartbeat messages and use constant timeout to either trust or suspect the processes that are monitored. Several recent adaptive failure detectors, proposed in [30, 15, 16, 38, 1, 14], can adjust a proper timeout based on both network conditions and application requirements. One of the major difficulty to building such a service is: applications with completely different requirements and running simultaneously have to be effectively adjusted by the service to meet their needs. Moreover, many distributed applications can greatly benefit from providing different levels of failure detection to trigger different reactions (e.g., [32, 33, 34]). For instance, an application can take a precautionary measure when confidence in a suspicion reaches a given level (a certain level); while take more drastic action once the confidence rises above a second (much higher) level [18].

Large-scale distributed systems (e.g., grid), often have many users and running applications. Each has diverse requirements such as a quick reaction by the failure detector module even at the expense of low accuracy. A failure detection service has to address these requirements flexibly.

The long-term and final goal is to define and implement a generic failure detection service for large scale distributed systems, and to provide it as a generic network-service (e.g., Domain Name Service (DNS), Network Information System (NIS), Network File System (NFS), Sendmail, etc.). The service can be considered as consisting of two parts, failure detection and information propagation. The former monitors processes, nodes etc., and detects their failures. This part corresponds closely to traditional failure detectors.

The latter propagates information about failures of processes that spread over the system and need such information [18]. In this dissertation, the focus is mainly on improving the failure detection part.

1.1.2 Related work of failure detector

Besides the above related works, there have been some other alternate failure detection mechanisms. For example, [24] described a lot of experiments performed on Wide Area Network to assess and fairly compare QoS provided by a large family of FDs. The authors introduced choices for estimators and safety margins used to build several (30) FDs. Compared with [24], this paper considered comparing all kinds of adaptive FD schemes in different experiment environments.

Nunes et al. [12] evaluated the QoS of an FD based on timeout for different combinations of communication delay predictors and safety margins. As the results show, to improve the QoS, the authors suggested that one must consider the relation between the pair predictor/margin, instead of each one separately. But we think it is (maybe) not very easy to find such proper pairs.

Fabio et al. [13] adapt FDs to load fluctuations of communication network using Simple Network Management Protocol (SNMP) and artificial neural networks. The training patterns used to feed the neural network were obtained by using SNMP agents over Management Information Base variables. The output of such neural network is an estimation of the arrival time for the FD to receive the next heartbeat message from a remote process. This approach improves the QoS of the FD, while the training of neural network is a little more complex to achieve the same goal as in this paper.

Fetzer et al. [14] presented an adaptive failure detection protocol. This protocol enjoys the nice property of relying as much as possible on application messages to perform this monitoring. Differently from previous process crash detection protocols, it uses control messages only when no application message is sent by the monitoring process to the observed process. These measurements show that the number of wrong suspicions can be reduced by requiring each process to keep track of the maximum round trip delay between executions.

In the paper [18], a generic failure detection service outputs a value based on a continuous normalized scale. Roughly speaking, this value shows the degree of confidence in the judgement that the corresponding process has crashed. It is then left behind to each application process to set a suspicion threshold according to its own quality-of service requirements. In addition, even within the scope of a single distributed application, it is often desirable to trigger different reactions based on different degrees of suspicion. The main advantage of this approach [18] is that it decouples the failure detection service from running applications. This design allows it to scale well with respect to the number of simultaneously running applications and/or triggered actions within each application.

In order to improve the QoS of FD, a lot of adaptive FDs have been proposed [12-15], such as Chen FD [30], Bertier FD [16 - 17], and the ϕ FD [18]. In [30], Chen et al. proposed several implementations relying on clock synchronization and a probabilistic behavior of the system. The protocol uses arrival times sampled in the recent past to compute an estimation of the arrival time of the next heartbeat. The timeout is set according to this estimation and a constant safety margin, and it is recomputed for each interval. This technique provides a good estimation for the next arrival time. Furthermore, this

paper assumed that the communication history was driven by uncorrelated samples with an ergodic stationary behavior, and message delays followed some probabilistic distribution. However, it uses a constant safety margin because the authors estimate that the model presents a probabilistic behavior [16]. Therefore, Bertier FD [16 -17] provided an optimization of safety margin for Chen FD. It used a different estimation function, which combined Chen's estimation with Jacobson's estimation of the round-trip time (RTT). Bertier FD performed as a good aggressive FD [18], because this approach was primarily designed to be used over wired local area networks (LANs), that is, environments wherein messages are seldom lost. The ϕ FD [18-19] proposed an approach based on a probabilistic analysis of network traffic, it is similar as in Chen FD, and it assumes that the inter-arrival time follows a normal distribution. Furthermore, ϕ FD computes a value ϕ with a scale that changes dynamically to match recent network conditions. Differently from the others, this FD outputs a suspicion level on a continuous scale, instead of binary nature (suspect or trust). These above three FDs dynamically predict new timeout values based on observed communication delays to improve the performance of the protocols. The self-tuned FDs proposed in [20] and [21] use the statistics of the previously-observed communication delays to continuously adjust its timeout. In other words, they assume a weak past dependence on communication history.

Many papers have provided failure detection as an independent service (e.g., [9, 6, 49, 50]). While, several important issues should be addressed before an effectively generic service can be really executed.

(1) A failure detection service must adapt to changing network conditions, as well as to application requirements. Several solutions proposed recently address this issue specifically [16, 30, 14, 15].

(2) A failure detection service must adapt to diverse application requirements. A few schemes (e.g., [30]) have been made to adapt the parameters of a failure detector service to match the requirements, while they are designed to support a single class of requirements.

Cosquer et al. [38] identified the problem. Their scheme is wonderful, while it remains inflexible because they do not express the Boolean nature of failure detection.

Hayashibara et al. [2, 18] have pointed this out recently. They proposed a ϕ failure detector to deal with this point. While the QoS of failure detector is not enough for an effectively generic service.

(3) A failure detection service must propose a good QoS for users. However, so far as I know, many schemes try to express this point. While none of the solutions resolved it perfectly.

As mentioned above, lots of problems remain in implementing a generic failure detection service. Also, to the best of our knowledge, there is no work about self-tuning the parameters of failure detector to satisfy requirement of users. As we know, it is still an open problem in fault tolerant research area. Therefore, it is very necessary to address this question. In this thesis, we present a self-tuning failure detector for this objective, and it can adjust the parameters of failure detector to satisfy requirement of users by itself.

1.2 Active queue management

Internet congestion occurs when the aggregated demand for a resource (e.g., link bandwidth) exceeds the available capacity of the resource. Congestion typically results in long delays in data delivery, wasted resources due to dropping packets, and the possibility of a congestion collapse [52-55]. Congestion avoidance is an essential technology in the Internet. The Internet congestion avoidance can be usually done at two places: 1) by the end-to-end protocol, such as the TCP; and 2) by the active queue management (AQM) scheme, which is implemented in routers [56]. AQM [57] is a scheme employed by routers to control the traffic going through them. It can achieve smaller queuing delays and higher throughput by purposely dropping out packets. There are several AQM schemes that have been reported in the recent literature for congestion avoidance.

Random Early Detection (RED) [58-61], recommended for deployment by the Internet Engineering Task Force (IETF), is the most prominent and well studied AQM scheme [62-63]. It has been widely implemented in routers for congestion avoidance in the Internet. The main objective of RED is to keep the average queue length (average buffer occupancy) low. To do so, RED randomly drops out the incoming packets with a probability proportional to the average queue length, which makes the RED scheme adaptive to bursty traffics. One important metric in measuring the performance of a traffic controller is the stability, the stability of packet dropping rate and the stability of queue length. A major drawback of the RED method is that it is difficult to set the parameters of the RED traffic controller to stabilize the system under the diversity of the Internet traffics [64-65]. The problem becomes especially severe when the average queue length reaches a certain threshold, resulting in a sharp decrease of throughput and an increase of drop-rate [56].

There are several variants of RED that have been proposed to address the above problem, such as Adaptive-RED [66-68], Proportional Derivative RED controller (PD-RED) [56], Proportional Integral controller (PI-controller) [69-70], and so on. With the RED [58-61], the resulting average queue length is very sensitive to the level of congestion and initial parameter setting, which makes its behavior unpredictable [69]. Adaptive-RED attempts to stabilize router queue length at a level independent from the active connections, by using an additive-increase multiplicative-decrease (AIMD) policy [66-68]. Sun et al. proposed a new RED scheme based on the proportional derivative control theory, called PD-RED, to improve the performance of the AQM. Unfortunately, neither Adaptive-RED nor PD-RED provide any systematic method to configure the RED parameters. Moreover, the control gain selection in both methods is based only on empirical observation and simulation analysis. They often work in one situation, but fail in another. A theoretic model and analysis for control gain selection and parameter setting is required. Hollot et al. proposed a Proportional Integral controller, PI-controller, as a means to improve the responsiveness of the TCP/AQM dynamics and stabilize the router queue length around the target value in [69]. Similarly, Deng et al. proposed a Proportional Integral Derivative model, to improve system stability under dynamic traffic conditions in [71]. Both methods used feedback control theory to describe and analyze the TCP/RED dynamics. However, both of them used a simplified linear quadratic Gaussian controller for analysis [72], and they limited their discussion to the classical control elements. Consequently, their methods can only directly link traffic control parameters to one of the AQM objectives, which compromised the global performance.

Besides the work mentioned above, there have been some other alternate mechanisms on AQM. For example, the Stabilized Random Early Drop (S-RED) protocol [82] uses adaptive methods to adjust the max drop probability p_{max}, according to three events: buffer overflow, empty buffer and queue length increasing. However, this approach introduces additional parameters that need to be configured again [83].

BLUE [73] is another type of adaptive scheme. It adaptively calculates packet drop probability based on only two events: buffer overflows and empty buffer. When the buffer overflows (or empties), the protocol increases (or decreases) packet drop probability by δ_1 (or δ_2). However the BLUE protocol has trouble bringing the queue length to an expected value [83].

Adaptive Virtual Queue (AVQ) [84-85] uses only input rate $x(t)$ to control packet dropping and to achieve expected link utility γ, while keeping queue length small. Packet drop probability is basically proportional to the mismatch between input rate and expected link utility γ. Through maintaining a virtual queue, AVQ deterministically drops packets upon the arrival of a new packet, realizing the same effect of probabilistic packet dropping. AVQ can achieve low average queue length and high link utility [83], as is shown in [84]. However, as noted in [86], the rule for setting the AVQ control parameter is not scalable because the stability condition equation in [84] becomes unsolvable as the link capacity scales upwards. The reason for this is the coupling of all the parameters. We overcome the limitation of [84] and achieve scalability by decoupling the known parameters from the control parameters. Having explicitly formulated a tractable stability range given by (6.3.22), (6.3.24), (6.3.27) and (6.3.29), we can make sure that the admissible control parameters are within this range. This has been further clarified in the above Chapter 6.

Random Exponential Marking (REM) [52] also tries to bring the queue length to an expected value. It uses the linear combination of queue mismatch and input rate mismatch to calculate marking/drop probability. In REM, input rate mismatch is similarly simplified to queue variance between two continuous samplings. REM is stable for a more narrow variety of network environments than PI-controller [70] and LRED [83].

State Feedback Controller (SFC) [87] uses a more complete model and the TCP option of delay acknowledgment. It also uses queue mismatch and input rate mismatch as a congestion index. This way, it tries to stabilize the queue length in routers to the target value. Packet marking/drop probability in SFC is updated upon arrival of a new packet. These characterize the TCP dynamics more realistically and cause congestion window size (cwnd) decrease faster. While, SFC does not exploit internet traffic long range dependency to design AQM [88]. Neither does it enable the controller dynamically adapt (i.e., adapt online) to system parameter changes.

Loss Ratio based RED (LRED) [83] measures the latest packet loss ratio, and uses it together with queue length to dynamically adjust packet drop probability. However, when the network parameters are unknown a priori, LRED can only use conservative policy to guarantee stability, and often this causes large queue deviation and lower throughput.

Misra et at. in [57] discuss the difficulties in tuning RED parameters. They illustrate the benign oscillations in the instantaneous queue length, and say that they are currently investigating tuning RED parameters. Hollot et al. in [60] also focus on oscillations in the queue length, and use this starting point to recommend values for RED parameters. Firoiu et al. in [63] also considered problems with RED such as oscillations in the queue length, and made recommendations for configuring RED parameters. In particular, [63]

recommended that the ideal rate for sampling the average queue length is once per round-trip time [67].

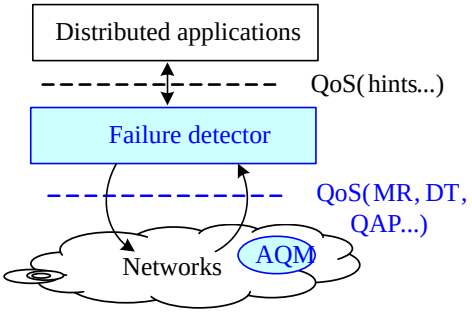


Figure 1.1: The relation structure model of failure detector and active queue management.

1.3 Relation between failure detector and active queue management

There are some relation between the failure detection and the active queue management scheme. Failure detection is generally based on distributed communication networks. Reversely, the performance (delay, throughput, rate of packets dropping, and so on) of communication networks also affects quality of service of failure detector. How AQM affects the performance of failure detector will be discussed in the following. Generally speaking, if the communication network is unreliable, i.e., heart messages have large delay and lots of heartbeat messages will be lost, then failure detector will suspect wrongly the processes with high probability. Thus, very high mistake rate and low accuracy probability will get for failure detector. More theory analysis are shown in Chapter 7.6. Therefore, it becomes very necessary to improve the performance of communication network.

In Figure 1.1, we find the relation in this figure. Based on the traditional scheme, Chandra and Toueg presented that failure detector is a distributed oracle that provides hints about the operational status of processes [9]. The basic metric for this scheme is focused on a hint, which is a parameter in the quality of service (QoS) between distributed applications (application processes) and failure detector. While hints may be incorrect. And failure detector may give different hints to different processes. Furthermore, failure detector may change its mind (over and over) about the operational status of a process. Thus, here this thesis focus on the QoS between failure detector and networks. For example, the mistake rate (MR), detection time (DT), and query accuracy probability (QAP). The DT means how fast it detects actual crashes, and MR and QAP mean how well it avoids mistakes (i.e., false detections).

Failure detector is still far from being a solved problem in large scale systems. Because there are many important factors in communication networks, which have great effect on

the output QoS of failure detector. Such as the high probability of message loss, the change of network topology, and the dynamic and unpredictable message delay, and so on. So the QoS in networks (such as, average queue length, stability, throughput, probability of packet loss, and so on) greatly affects the output QoS of failure detector. While the AQM could improve the QoS in networks. Therefore, it becomes very necessary to design an effective AQM scheme to improve the performance of communication network.

1.4 Contributions

Firstly, we explore the relative failure detection, which is an important issue for supporting dependability in distributed systems, and often is an important performance bottleneck in the event of node failure. In this field, we analyze the existing failure detections, and then develop four different schemes (Tuning adaptive margin failure detection; Exponential distribution failure detection; Kappa failure detection; Self-tuning failure detection) to ensure acceptable quality of service in unpredictable network environments.

Self-tuning failure detection: For the former failure detection scheme, all of them can not actively tune their parameters by themselves to satisfy the requirement of users. To the question, we present a self-tuning failure detection based on [30], called self-tuning failure detection. Finally, a lot of experimental results demonstrate that our scheme is effective. And we are sure that this idea also can apply into other failure detection to achieve self-tuning requirement.

Tuning adaptive margin failure detection: We introduce an improvement over existing methods, and evaluate their benefits. First, we propose an optimization to enhance the adaptation of [30], which significantly improves QoS, especially in the aggressive range and when the network is unstable. Second, we address the problem of most adaptive schemes, namely their need for a large window of samples. We study a scheme that is designed to use a fixed and very limited amount of memory for each monitored–monitoring link. The experimental results over several kinds of networks (Cluster, WiFi, wired LAN, WAN) show that the properties of the existing adaptive FDs, and that the optimization is reasonable and acceptable. Furthermore, the extensive experimental results show what is the effect of memory size on the overall QoS of each adaptive FD.

Exponential distribution failure detection: Observing from lots of experimental statistical results, we find it is not a good assumption that the Φ failure detection [18] uses the normal distribution to estimate the arrival time of the coming heartbeat, especially in large scale distributed network or unstable networks. Therefore, here we develop an optimization over Φ failure detection based on exponential distribution, called exponential distribution failure detection. This significantly improves quality of service, especially in the design of real systems. Extensive experiments have been carried out based on several kinds of networks (Cluster, WiFi, Wired local area network, and Wide area network). The experimental results have shown the properties of the existing adaptive failure detections, and demonstrated that the presented exponential distribution failure detection outperforms the existing failure detections in the aggressive range.

Analysis and evaluation of Kappa failure detection: Based on [3], a basic concept of κ failure detector is proposed. It has the formal properties of accrual failure detection. This scheme allows for gradual settings between an aggressive behavior and a conservative one. Here, we develop this idea and specially further discuss service

performance of κ failure detector in a variety of network environments.

Secondly, in order to make sure the network communication is effective, we explore active queue management schemes to support TCP flows in networks. In this part, we first present a self-tuning proportional and integral controller scheme based on average queue length of router. Then we prove the stability of the network system, and present an effective method for the control gain selection. After that, extensive simulations have been conducted with NS2. The simulation results have demonstrated that the proposed self-tuning proportional and integral controller algorithm outperforms the existing active queue management schemes in terms of drop probability and stability.

1.5 Organization of the thesis

The remainder of this thesis is organized as follows. In Chapter 2, system models are presented, together with the definitions and notations used in this dissertation. Furthermore, existing approaches to failure detectors are surveyed and classified in terms of large-scale distributed systems. In Chapter 3 proposes tuning adaptive margin failure detection. In Chapter 4, exponential distribution failure detection is developed as an adaptive failure detector for applications running on a large-scale distributed systems. In Chapter 5, κ failure detection is analyzed and developed, and a variety of experiments are carried out to evaluate the performance of κ failure detector. In Chapter 6, for the former failure detection scheme, all of them can not actively tune their parameters by themselves to satisfy the requirement of users in dynamic networks. To the question, we present a self-tuning failure detection based on [30], called self-tuning failure detection. In Chapter 7, in order to make sure the network communication is effective, we explore the related active queue management schemes to support TCP flows in networks. Finally, the conclusions to the dissertation are given in Chapter 8.

Chapter 2

Background, System Models and Definitions

2.1 System models

In this sub-chapter, we first discuss about the general system models, there are several different models. Then we discuss about the detailed system model for our failure detector.

2.1.1 Synchrony

System models are different due to their different level of timing assumptions. Mainly, these models are defined by (i) the relative speeds of the processors and (ii) message transmission delays. Thus, the definitions of system model are provided, and their characteristics are analyzed in this section.

Synchronous model In a synchronous system, there is a known upper bound on message delay, that is on the time it takes for a message to be delivered [9]; also there is a known upper bound on the time that elapses between consecutive steps of a process.

From the definition of synchronous model, if the time that one process waits for the message from another process is exceeded the upper bound, then the another process must be crashed. Therefore, in synchronous system, it is very easy to distinguish whether one process is crashed or just very slow. However, the problem in synchronous is that it is not easy to ensure all process synchronous all the time.

Asynchronous model There is no any timing assumption about processor speeds and message transmission delay, that is no bound on message delay, clock drift, or the time necessary to execute a step.

The paper [8] showed that consensus[1] cannot be solved in asynchronous system subject to crash failures. The fundamental reason why consensus cannot in completely asynchronous systems is the fact that, in such systems, it is impossible to reliably distinguish a process that has crashed from one that is merely very slow. Therefore, it is necessary to find a tradeoff model between two extremes: synchronism model and asynchronism model.

Partially synchronous model If we weaken any parameter in completely synchronous (such as message delay), then it will become a partial synchronous. For example,

[1] In the consensus problem, all correct processes propose a value and must reach a unanimous and irrevocable decision on some value that is related to the proposed values [9].

in synchronous, message delay is bounded and known, now if we assume message delay is exist but not known, then it will become partially synchronous.

Several types of model ranging from synchronous systems to asynchronous systems have been developed: (i) processors are completely synchronous and communication is partially synchronous, (ii) both processes and communication are partially synchronous, (iii) processes are partially synchronous and communication is synchronous [40]. Let us first consider the case in which the processes are completely synchronous and communication is partially synchronous. In this situation, the system has an upper bound U on message transmission delay but it is unknown, or it holds a known upper bound U after a global stabilization time (GST), unknown to the processors.

Then, an extension of this model in which both processors and communication, are partially synchronous can be considered. That is, the upper bound on the relative processor speeds U can exist but be unknown, or U can be known but actually hold only from some time GST onward. It is easy to define models where processors are partially synchronous and communication is synchronous (U exists and is known a priori).

2.1.2 Process failure models

In a distributed system, the two components of the system, both processes and channels, can fail. It is very important to define types of possible failure for further discussion about existing problems. Now, some failure models are introduced and the types of failure assumed are discussed.

Process failures Processes can fail for various reasons and they behave differently after failure. Process failures are classified three ways with respect to the behavior of a process after failure.

a) Fail-stop failure A faulty process stops permanently and does nothing from that point on but behaves correctly before stopping. All other processes, which do not crash, eventually detect the state with no erroneous detection. This failure is called fail-stop failure.

b) Crash failure A faulty process stops permanently and does nothing from that point on but behaves correctly before stopping. Some other processes, which do not crash, may not detect the state. This failure is called crash.

c) Omission failure A faulty process intermittently omits to send or receive messages.

d) Byzantine failure A faulty process can exhibit any behavior whatsoever, such as changing state arbitrarily. In this state, the process becomes crazy and may be any state that it wants.

All the algorithms and the systems in this dissertation assume only the crash-failure model in processes. Thus, we call processes that never crash correct and processes that have crashed faulty or incorrect. Note that correct/faulty are predicates over a whole execution: a process that crashes is faulty even before the crash occurs. Of course, a process cannot determine if it is faulty and some other components (i.e., failure detector modules) cannot make processes faulty.

2.1.3 Interaction models

The interaction models are discussed systematically in [6]. In a fault tolerant system, there are two kinds of processes, called monitored process and monitoring process. There

are three interfaces in a monitoring system: Monitor (failure detector), monitoring objects and notifiable objects. Among these, a failure detector in monitoring process is to collect the information about component failure; monitorable objects are objects that can be registered; and notifiable objects are asynchronously notified about object failures.

For service-oriented service, monitors are implemented by the service and do not need to be instantiated by the application. Basically, there are two kinds of forms of unidirectional flow, *push*(heartbeat) and *pull*(are-you-alive), plus several variants [91]. According to the network topology and the communication pattern of the application, the choice between a push or a pull monitoring model can have an important impact on the performance of the system.

In the push model, the direction of control flow matches the direction of information flow. With this model,monitorable objects are active. They periodically send heartbeat message to inform other objects that they are still alive. If a failure detector does not receive the heartbeat from a monitorable object within specific time bounds, it starts suspecting the object.

Differently from push model, the failure detector in the pull model is active. It will periodically send message "are you alive?" to its monitored objects, and if the monitored objects don't reply its question "Yes" with specific time bounds, the failure detector starts suspecting the monitored objects. Obviously, this model is less efficient than the push model since two-way messages are sent to monitored objects, but it is easier to use for the application developer since the monitorable objects are passive, and do not need to have any time knowledge [6].

So far, there have been many papers,like [92,93], which use the combination of the above two models, called push-pull model. Informally, the push-pull protocol works as follows. The protocol is split in two distinct phases. During the first phase, all the monitored objects are assumed to use the push model, and hence to send heartbeats. After some delay, the monitors switch to the second phase, in which they assume that all monitored objects that did not send a heartbeat during the first phase usee the pull model. In this phase, the monitors send a heartbeat to each monitored object, and expect a heartbeat from the latter. If the monitored object does not send this message within some specific time bounds, it gets suspected by the monitor.

2.2 Failure detectors

In distributed systems with failures, applications often need to determine which processes are up (operational) and which are down (crashed). This service is provided by failure detector. Failure detectors are at the core of many fault-tolerant algorithms and applications, such as group membership, group communication, atomic broadcast, atomic commitment, consensus and leader election, etc. Also failure detectors are found in many systems, such as ISIS, Ensemble, Relacs, Transis, Air Traffic Control Systems.

A failure detector can be viewed as a distributed oracle for giving a hint about the operational state of a process. In fact, a failure detector consists of failure detector modules that communicate with each other by exchanging messages. A process, called a monitoring process, can query its failure detector module about the status of some process, called a monitored process. The monitoring process thus obtains information about whether or not the monitored process is suspected to have crashed.

Table 2.1: Eight classes of failure detectors defined based on accuracy and completeness

Completeness	Accuracy			
	Strong	Weak	Eventual Strong	Eventual Weak
Strong	*Perfect P*	*Strong S*	*Eventually Perfect* $\diamondsuit P$	*Eventually Strong* $\diamondsuit S$
Weak	Q	*Weak W*	$\diamondsuit Q$	*Eventually Weak* $\diamondsuit W$

The definitions of failure detectors are first formally defined in [9]. The principle of unreliable failure detector in [9] are introduced as follows.

Unreliable failure detectors

Chandra and Toueg [9] define the notion of unreliable failure detectors, which are based on the following model. For every process p_i in the system, there is a module FD_i attached that provides p_i with potentially unreliable information on the status of other processes. At any time, p_i can query FD_i and obtain a set of processes containing those that are suspected of having crashed.

The impossibility result (i.e., several distributed agreement problems cannot be solved deterministically in asynchronous systems if even a single process might crash[8]) no longer holds if the system is augmented with some unreliable failure detector oracle. An unreliable failure detector is one that can, to a certain degree, make mistakes (over and over).

The failure detector is a distributed entity that consists of all modules and whose behavior must exhibit some well-defined properties. Depending on the properties that are satisfied, a failure detector can belong to one of several classes. Now, these properties are as follows:

Completeness properties

a) Strong completeness: Every faulty process is permanently suspected by *all* correct processes.

b) Weak completeness: Every faulty process is permanently suspected by *some* correct process.

Accuracy properties

a) Strong accuracy: *No* process is suspected before it crashes.

b) Weak accuracy: *Some* correct process is never suspected.

c) Eventual strong accuracy: There is a time after which *every* correct process is never suspected by any correct process.

d) Eventual weak accuracy: There is a time after which *some* correct process is never suspected by any correct process.

The classes of failure detectors defined by accuracy and completeness properties are shown in Table 2.1, which is divided from the aspect of quality. In Table 2.1, Perfect failure detector P satisfies strong completeness and strong accuracy. There are eight such pairs, obtained by selecting one of the two completeness properties and one of the four accuracy properties.

As an example, the class $\diamondsuit W$ is the weakest to solve consensus[2] Interestingly, any given failure detector that satisfies weak completeness can be transformed into a failure detector that satisfies strong completeness. There also exist transformation algorithms for failure detectors from strong completeness to weak completeness. This means that a

[2]In the consensus problem, all correct processes propose a value and must reach a unanimous and irrevocable decision on some value that is related to the proposed values [9].

failure detector with strong completeness and a failure detector with weak completeness are equivalent, thus, $\diamondsuit W$ is also the weakest failure detector for solving Consensus.

The problem is that, in asynchronous distributed systems, it is impossible to implement a failure detector of class $\diamondsuit S$ in a literal sense. The definition of $\diamondsuit S$ failure detector is nevertheless highly relevant in practice. Algorithms which assume the properties of a $\diamondsuit S$ are incredibly robust because they can tolerate an unbounded number of timing failures. In other words, and in a more pragmatic way, an application is guaranteed to make progress as long as the failure detector behaves well for long enough periods. Conversely, the application might stagnate during bad periods and resume only after the next period of stability.

In practice, the behavior of a failure detector largely depends on how well the failure detector is tuned to the behavior of the underlying network. In this thesis, we focused on failure detectors in the partially synchronous systems.

2.3 Quality-of-service of failure detectors

Chen et al. [30] proposed a set of metrics to evaluate the Quality-of-service (QoS) of failure detectors. In detail, considering two processes p and q where q monitors p, the QoS of the FD at q (called fd_q) can be determined from its transitions between the "trust" and "suspect" states with respect to p (see Figure 2.1). The metrics that are used in this dissertation are defined below, which is divided from the aspect of quantity.

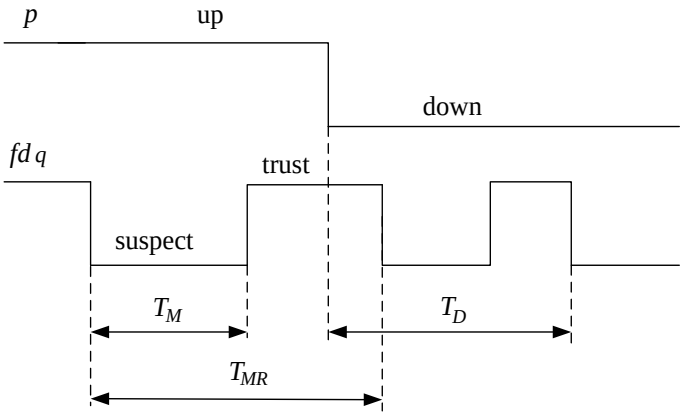


Figure 2.1: Basic Metrics for the QoS evaluation of an FD [30].

Definition 1 (Detection time T_D). The detection time is the time that elapses from the crash of q until p begins to suspect q permanently.

Definition 2 (Mistake recurrence time T_{MR}). The mistake recurrence time measures the time between two consecutive wrong suspicions. T_{MR} is a random variable representing the time that elapses from the beginning of a wrong suspicion to the next one. T^U_{MR} and T^L_{MR} also denote upper and a lower bounds respectively on the mistake recurrence time.

Definition 3 (Mistake duration T_M). The mistake duration measures the time that elapses from the beginning of a wrong suspicion until its end (i.e., until the mistake

is corrected). This is represented by the random variable T_M, the upper and lower bounds of which are denoted by T_M^U and T_M^L respectively.

Definition 4 (Good period duration T_G). This is a random variable T_G representing the duration from the time p begins to trusting q to the time p next begins of suspect q. It can be expressed as $T_G = T_{MR} - T_M$.

Definition 5 (Average mistake rate λ_M). This measures the average rate in unit time at which a failure detector generates wrong suspicions during the whole detection procedure. It can be expressed by $\lambda_M = 1 - E(T_{MR})$.

Definition 6 (Freshness point τ_i). The failure detector module computes a freshness point τ_i for the i^{th} heartbeat message. If the module does not receive a message from a process p until τ_i, it suspects p, otherwise it trusts p.

Notice that the first definition relates to completeness, whereas the other metrics relate to the accuracy of the failure detector.

2.4 Our system model

Model We consider a partially synchronous system, that means, after some unknown time (called GST for Global Stabilization Time), there are bounds on relative process speeds and on message transmission times. The inter-process communication model is based on message exchanges over the UDP communication protocol. Here we don't consider the relative speed of processes. However, we consider that processes have access to a local clock device that can be used to measure the time of heartbeat passage, and these clocks are synchronous or asynchronous. Furthermore, every process has access to a failure detection service.

Fault: We consider a distributed system consisting of a finite set of processes $\Pi = \{p_1, p_2, p_3, ..., p_n\}$. A process may fail by crashing, here a crashed process does not recover. A process behaves correctly (i.e., according to the specification) until it (possibly) crashes.

We assume the existence of some global time (unbeknownst to processes) denoted by GST, and that processes always make progress, furthermore, at least $\delta > 0$ time units elapse between consecutive steps (the purpose of the latter is to exclude the case where processes take an infinite number of steps in finite time) [19].

Communication channel Every pair of processes is assumed to be connected by one unreliable communication channel [22]. An unreliable channel is defined as a communication channel: there is no message creation, alteration or duplication, while it is possible to lose some messages.

It is a common approach to implementing failure detectors by using heartbeat messages. Consider a simple system model that consists of only two processes called p and q, which are arbitrarily taken from the large system Π, where process q monitors process p (see Figure 1). p may periodically send a message to q, or is subject to crash. Here the sending period is called the heartbeat interval Δt. Process q suspects process p if it does not receive any heartbeat message from p for a period of time determined by the fresh-point. In the sequel, we consider the same system model.

In Figure 2.2, d_i is the transmission delay of heartbeat m_i from p to q. For the incoming heartbeat m_j $(1 \leq j \leq i)$, q dynamically gives a response based on the new freshpoint FP_j, which is according to network conditions (e.g., the transmission delay d_j). This model describes three cases that may occur. The first one is that heartbeat message

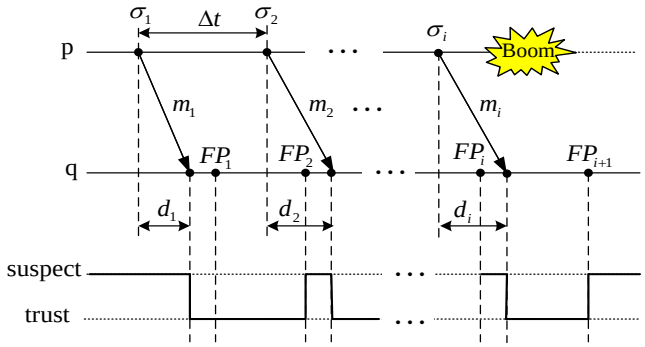


Figure 2.2: Basic heartbeat failure detection model.

m_1 from the sending time σ_1 of process p arrives at q before q's freshpoint FP_1, then q trusts p from the m_1 arrival time (here we assume that p is suspected in the initial case). The second case is that heartbeat m_2 from p arrives at q after q's freshpoint FP_2, then q suspects p from FP_2 until the m_2 arrival time. In the third case, after the sending time σ_i, p crashes, then q waits for that heartbeat m_{i+1} until its freshpoint FP_{i+1}, then q starts to suspect p.

In a common belief, the period Δt is a factor that contributes to the detection time. However, Müller [35] shows that, on several different networks, Δt is little determined by QoS requirements, but much by the characteristics of the underlying system, and [18] suggests that there exists, with every network, some nominal range for the parameter Δt with little or no impact on the accuracy of the FD.

In the conventional implementation of this model, the freshpoint is fixed. If the time between two next freshpoints is too short, the likelihood of wrong suspicions is high, though crashes are detected quickly. In contrast, if the time is too long, there is too much detection time, although there are fewer wrong suspicions.

An alternative implementation of this model sets the freshpoints based on the transmission delay of the heartbeat. The advantage is that the maximal detection time is bounded, but the disadvantage is that it relies on physical clocks with negligible drift[3] and a shared knowledge of the heartbeat interval Δt. The drawback is a serious problem in practice, when the regularity of the sending of heartbeats cannot be guaranteed, and the actual sending interval is different from the target one (e.g., timing inaccuracies due to irregular OS scheduling) [18].

The two methods have advantages and disadvantages, it is difficult to conclude which is better [18].

2.5 Adaptive failure detectors

Besides the general related work in Chapter 1.1.2, here we focused on several main recent failure detectors, which we compared our failure detectors in Chapter 3 6.

[3]A straightforward implementation of clocks requires synchronized clocks. Chen et al. [30] shows the method to do it with unsynchronized clocks, but this still requires the drift between clocks to be negligible.

The goal of adaptive FDs is to adapt to changing network conditions and application requirements. In general, adaptive FDs are based on a heartbeat strategy.

2.5.1 Chen FD

Chen et al. [30] proposed an approach based on a probabilistic analysis of network traffic. The protocol uses arrival times sampled in the recent past to compute an estimation of the arrival time of the next heartbeat. The timeout is set according to this estimation and a safety margin, and recomputed for each interval.

The algorithm is described as follows: The n most recent heartbeat messages, denoted by m_1, m_2, ..., m_n, are considered by each process q. A_1, A_2, ..., A_n are their actual receipt times according to q's local clock. When at least n messages have been received, the theoretical arrival time $EA_{(k+1)}$ can be estimated by:

$$EA_{(k+1)} = \frac{1}{n} \sum_{i=k-n}^{k} (A_i - \Delta_i * i) + (k+1)\Delta_i, \tag{2.1}$$

where Δ_i is the sending interval. The next timeout delay (which expires at the next freshness point $\tau_{(k+1)}$) is composed of $EA_{(k+1)}$ and the constant safety margin α. One has

$$\tau_{(k+1)} = \alpha + EA_{(k+1)}. \tag{2.2}$$

This technique provides an estimation for the next arrival time based on a constant safety margin.

2.5.2 Bertier FD

Bertier et al. [16-17] estimated the safety margin dynamically based on Jacobson's estimation of the round-trip time (RTT). [23]. It adapts the safety margin each time it receives a message. Simply speaking, the adaptation of the margin α is based on the variable *error* in the last estimation. Parameter γ represents the importance of the new measure with respect to the previous ones. The variable *delay* represents the estimate margin, and *var* estimates the magnitude between errors. β and ϕ is used to adjust the variance *var*. Typical values β, ϕ and γ are 1, 4 and 0.1, respectively. $EA(k)$ denotes the theoretical arrival time, as same as the above Chen FD. The recursive algorithm [16-17] is as follows:

$$error_k = A_k - EA_{(k)} - delay_{(k)}. \tag{2.3}$$

$$delay_{(k+1)} = delay_{(k)} + \gamma \cdot error_{(k)}. \tag{2.4}$$

$$var_{(k+1)} = var_{(k)} + \gamma \cdot (|error_{(k)}| - var_{(k)}). \tag{2.5}$$

$$\alpha_{(k+1)} = \beta \cdot delay_{(k+1)} + \phi \cdot var_{(k)}. \tag{2.6}$$

and

$$\tau_{(k+1)} = EA_{(k+1)} + \alpha_{(k+1)}. \tag{2.7}$$

Bertier's estimation provides a short detection time due to the effective prediction of communication delay.

2.5.3 Accrual failure detectors

Chandra and Toueg in [9] provide information of a boolean nature (i.e., trust vs. suspect) on unreliable failure detectors. On a continuous scale, accrual failure detectors express a level of suspicion [19].

The suspicion level of process q with respect to process p is defined as a function $sl_{qp}(t)$ of time to the positive real numbers and 0. It must satisfy the following two properties.

Property 2.1 (Accruement). If process p is faulty, then eventually, the suspicion level $sl_{qp}(t)$ is monotonously increasing at a positive rate[4].

Property 2.2 (Upper bound). If process p is correct, then the suspicion level $sl_{qp}(t)$ is bounded.

Depending whether the bound of Property 2.2 is known or not, Several classes of accrual failure detectors are defined, i.e., the properties hold for the whole system (i.e., all pairs of processes) or only part of it. Especially, the class $\diamondsuit P_{ac}$ of accrual failure detectors considers that the bound is unknown, and it requires that the two properties above hold between every pair of processes.

The two classes P and $\diamondsuit P_{ac}$ have been proved to be equivalent from a computational standpoint. This means that they can formally solve the same set of problems, i.e., one failure detector can be implemented, the other can as well. In particular, it is sufficient to solve agreement problems like Consensus [9] using a failure detector of class P, therefore so does one of class $\diamondsuit P_{ac}$.

When we discuss the implementations of failure detectors, one usually considers a somewhat weak model of partial synchrony [40], called M_3 [9], where there is an unknown time after which communication delays and process speed are bounded by an unknown bound. As we know, it is well-know that failure detectors of class P can be implemented in model M_3. Thus, it follows that an accrual failure detector of class $\diamondsuit P_{ac}$ can also be implemented in this model.

2.5.4 The ϕ FD

Although the research on FDs have important technical breakthroughs, they have obtained little success so far. So far as we know, there are two main reasons [18]: (1) an FD provides an information list of suspects about which processes have crashed. This information list is not always up-to-date or correct (e.g., an FD may falsely suspect a process that is alive). The reason, in practice, is due to the high unpredictability of message delays, the dynamic and changing topology of the system, and the high probability of message losses. (2) the conventional binary interaction (i.e., trust and suspect) makes it difficult to meet the requirements of several distributed applications running simultaneously. In practice, many classes of distributed applications require the use of different QoS of failure detection to trigger different reactions (e.g., [32-34]). For instance, an application can take precautionary measures when the confidence in a suspicion reaches a given low level, while it takes more drastic actions once the doubt rises above a higher level [6]. However, the traditional output of the FDs (Chen FD [30] and Bertier FD [16, 17]) is of binary nature[5].

[4]The definition allows for stationary periods provided that there is always some positive increase after some time [19].

[5]Bertier FD and Chen FD were aimed at other problems, which they both solved admirably well.

To resolve these questions, Hayashibara and Défago et al. [18-19] developed a ϕ FD, which outputs a suspicion level on a continuous scale, instead of providing information of a binary nature (trust or suspect). The characteristic and the principal merit of this approach are that it favors a nearly complete decoupling between application requirements and the monitoring of the environment. The ϕ FD can dynamically adapt current network conditions based on the suspicion level expressed.

The specific implementation is as follows: Let T_{last} denote the time when the most recent heartbeat was received; t_{now} is the current time; and $P_{later}(t)$ denotes the probability that a heartbeat will arrive more than t times units after the previous one. Then, the value of the suspicion level φ is calculated as follows:

$$\varphi(t_{now}) = -log_{10}(P_{later}(t_{now} - T_{last})). \tag{2.8}$$

Here,

$$P_{later}(t) = \frac{1}{\sigma\sqrt{2\pi}} \int_{t}^{+\infty} e^{-\frac{(x-\mu)^2}{2\sigma^2}} dx = 1 - F(t), \tag{2.9}$$

where $F(t)$ is the cumulative distribution function of a normal distribution with mean μ and variance σ^2 of the inter-arrival time. Then, the value of φ at time t_{now} is computed by applying the equation (2.8). Finally, comparing the value of φ and the threshold ϕ given by specific application, it can determine to suspect or trust. If the value of φ is larger than that of ϕ, then the application will suspect the process which communication with it; otherwise, it will trust it.

Now, many failure detectors can provided different QoS services to match many different QoS requirements of users. For a given QoS requirement, how can we set the parameters to provide corresponding QoS for a user? In Chen FD [30], it can give a list about the possible QoS services for users based on the different parameters of failure detector. If a user require a certain QoS service, ones can match the QoS requirement from the list of QoS services, and then choose the corresponding parameters of failure detector by hand. Obviously, it is not applicable for actual engineering applications. Thus, we presented a SFD, which optimizes the paper [30] and self-tunes its parameters to satisfy the requirement of different users.

2.5.5 κ-Failure Detectors

In this subchapter, we developed the κ-failure detector [3] (κ-FD) as a pragmatic implementation of accrual failure detectors. We first introduce the concept, which is followed by one possible implementation.

κ-FD is actually defined as a framework of failure detectors, which is not like the failure detectors in [16] and [30]. The design objective of κ-FD is to have a parametric failure detector that allows for a gradual transition between aggressive failure detection and conservative one.

Intuitively, κ-FD works as follows based on [3]. Each given expected heartbeat can contribute to the total suspicion level with a value ranging from zero to one. At some time t, the contribution of some heartbeat is zero if the heartbeat is not expected (too early or already received), then it gradually increases as the heartbeat is increasingly more expected, to finally reach one when it is considered too late. A contribution function defines how the contribution of a heartbeat increases with time.

In this subchapter, based on [3], we describe the κ-failure detector (a possible implementation is described in Chapter 2.6.4). The basic idea is that each missed heartbeat contributes to raise the level of suspicion of the failure detector. First, we introduce what the contribution of a heartbeat is. Then, we explain how the suspicion level is computed. Finally, we discuss some useful properties of the failure detector.

(1)Heartbeat contribution (definition)

An existence of a function of time is required in the κ-failure detector to represent the evolution of the confidence that a given heartbeat will not be received in the future (either because it was lost or because the sending process has crashed). This function, called contribution function, returns a value between 0 and 1, where the former means no confidence at all and the latter means total confidence. Initially, the value is zero and remains so until some time when the heartbeat begins to be expected. Then, the value increases and ultimately converges to one. We consider this function as a black box here, but we propose a concrete implementation for it in Chapter 5.

More precisely, the contribution function is defined as follows:

Definition 2.7 (Contribution function). The contribution function is a function of time which satisfies the properties below.

$$c : R \longrightarrow [0; 1]$$

- c is monotonic.
- $c(0) = 0$
- $\lim_{t \longrightarrow +\infty} c(t) = 1$

Each heartbeat uses the function to determine the evolution of the confidence with respect to that heartbeat. Notice that the function can be based on parameters that change dynamically, e.g., when a new heartbeat is received. We consider that there is a time, called the starting time, before which the heartbeat is not expected.

Definition 2.8 (Starting time). Let H^i denote the i-th heartbeat (with $i = 1, 2, \cdots$). Its starting time T^i_{st} has the following property.

- $\forall j (i < j \Leftrightarrow T^i_{st} < T^j_{st})$

It follows that the contribution of some heartbeat H_i can be computed simply by (2.10).

$$c^i(t) = c(t - T^i_{st}) \tag{2.10}$$

Notice that the nature of the contribution function is important for aggressive failure detectors but not so much for conservative ones. In practice, this is because the contribution function defines the meaning of the fractional part of the value output by κ.

(2)Computing the suspicion level

We can define the suspicion level by summing the contributions of all expected heartbeats with a rank higher than the most recent heartbeat received so far [3]. This is expressed by the following equation, where κ stands for the rank of the most recent heartbeat received so far.

$$sl_{qp}(t) = \kappa(t) = \Sigma_{i=k+1}^{\infty} c(t - T_{st}^{i}) \tag{2.11}$$

Notice that we also assume that, if process p is correct, then p sends infinitely heartbeat messages.

(3)Properties

The properties of the κ failure detector in the partially synchronous system model M_3 are discussed by Chandra and Toueg [9]. Based on the model M_3, there is some unknown time called GST (which stands for global stabilization time), after which communication and processing delays are bounded. Neither the bounds nor GST are known. There are also no guarantees whatsoever about the delivery of messages sent before GST (e.g., such messages can be dropped).

There are two arbitrary processes p and q, with q monitoring p, and we establish that, regardless of the actual contribution function, the suspicion level satisfies the properties of both accruement (Prop. 2.1) and upper bound (Prop. 2.2). The proofs of the two corresponding lemmas are in the appendix.

Lemma 1. Given a κ failure detector running between two processes in model M_3, its suspicion level satisfies the accruement property (Prop. 2.1).

Lemma 2. Given a κ failure detector running between two processes in model M_3, its suspicion level satisfies the upper bound property (Prop. 2.2).

These two lemmas lead directly to the following theorem.

Theorem 1 ($(\forall p \forall q,\ \kappa) \in \diamondsuit P_{ac}$). Given a partially synchronous distributed system with the properties of M_3, and in which all pairs of processes monitor each other according to κ, then the resulting failure detector belongs to class $\diamondsuit P_{ac}$.

(4)Implementation of κ FD

This subchapter describes a concrete implementation of the κ-failure detector, based on a failure detection strategy developed for the Phi FD [18]. We have used this implementation to run the experiments presented in Chapter 5.

In this implementation, the contribution function of heartbeats is computed from past heartbeat arrival times. We use the estimation made for the Phi failure detector [18], and consider that heartbeat arrivals are spread around their expected arrival time according to a normal distribution. This is chosen just as an approximation. The failure detector module is divided into two main tasks, namely, the sampling of heartbeat arrivals, and the computation of the current suspicion level. We now describe these two tasks.

Task 1: Sampling The sampling task is executed whenever a new heartbeat is received, and gathers information about recent heartbeat arrivals. In particular, the task maintains a sliding window of past arrivals with parameter WS as the window size. Upon receiving a new heartbeat, the task reads the process clock and stores the heartbeat

rank and arrival time into the sliding window (thus discarding the oldest heartbeat if necessary).

The information is used to compute the expected arrival times of future heartbeats, according to the method proposed by Chen et al. [30]. The idea is to consider that the interval between heartbeat arrivals are constant, with known interval Δ_i. The reference of expected arrivals is shifted, such that the error made retrospectively by the actual arrival times stored in the window is minimized.

Let A_j be the arrival time of heartbeat j according to the local clock of the receiving process, and EA_k be the time when heartbeat j is expected to arrive. It is estimate as follows:

$$EA_k := k\Delta_i + \frac{1}{n} \sum_{j=k-WS-1}^{k-1} (A_j - j\triangle_i)$$

The task keeps track of four values that are of particular importance for the estimation of the suspicion level: the mean μ and the variance σ^2 of arrival times, as well as the rank k and the arrival time A_k, where k is the highest rank among all received heartbeats. For the first two values, this is done by simply keeping track of the sum and the sum of squares of inter-arrival times.

Task 2: Computing κ This task is invoked when some application process queries the failure detector. The task reads the process clock and computes the suspicion level. This is done by approximating the contribution function of expected heartbeats and summing each of them.

Given the values obtained by the first task (i.e., μ, σ^2, k, A_k), the contribution function $c(t)$ is approximated from the cumulative normal distribution function.

$$c(t) = \begin{cases} \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{t} e^{-\frac{(x-\mu)^2}{2\sigma^2}} dx, & \text{if } t > 0; \\ 0, & \text{otherwise} \end{cases} \tag{2.12}$$

It follows that, for some heartbeat H^i, where $i > k$, the contribution is computed by the function $c^i(t)$ shown below. Also, in Eq. (2.12), the contribution of heartbeat H^i starts one heartbeat interval before its estimated arrival. Hence, the starting time T^i_{st} of heartbeat H^i is given by the following equation.

$$\begin{aligned} T^i_{st} &= EA_k + (i - k - 1)\Delta_i \\ c^i(t) &= c(t - T^i_{st}) \end{aligned} \tag{2.13}$$

Computing the current value of the function $\kappa(t)$ is done by summing the contribution of all heartbeats H^i for which the starting time is past (i.e., where $t > T^i_{st}$).

Chapter 3

Tuning Adaptive Margin Failure Detection

In order to improve the QoS of failure detection, a lot of adaptive FDs have been proposed [12-15], such as Chen FD [30], Bertier FD [16 - 17], and the ϕ FD [18]. In [30], Chen et al. proposed several implementations relying on clock synchronization and a probabilistic behavior of the system. However, it uses a constant safety margin because the authors estimate that the model presents a probabilistic behavior [16]. We know, the network environment is dynamic and unpredictable, heartbeats have quite different delay to arrive to receivers. So the constant safety margin is not applicable the actual network environments to obtain good QoS for users.

To answer these questions, this chapter compares the QoS of several adaptive FDs in terms of their respective QoS, and then introduces an optimization over Chen FD method, called tuning adaptive margin FD (TAM FD), and evaluate its benefits. The experiment results demonstrate that TAM FD significantly improves QoS, especially in the aggressive range and when the network is unstable.

Furthermore, We can find all of adaptive FDs (i.e., Chen FD, Bertier FD, and ϕ FD) use the memory[1] to predict the future. Therefore, questions then arise: *What is the effect of history-track-record length on the overall QoS of adaptive FDs?* we evaluate the impact of history-track-record length on the overall QoS of FDs from two aspects in the experiments. Firstly, we compare the performance of different FDs with a certain window size; And then, we explore the performance of FDs with different window sizes. Our experimental results over several kinds of networks (Cluster, WiFi, wired LAN, WAN) show that the properties of the existing adaptive FDs, and that the optimization is reasonable and acceptable. Furthermore, the extensive experimental results show what is the effect of memory size on the overall QoS of each adaptive FD.

3.1 Tuning adaptive margin failure detection

In order to dynamically adapt a safety margin to unpredictable network, we propose an optimization over Chen FD [30], called Tuning Adaptive Margin Failure Detector (TAM FD). According to [24], the implementations are expected to predict future delays

[1]It is very necessary to consider the effect of window size on FDs because larger history-track-record length requires more valuable memory and CPU resources.

by sampling the behavior of messages sent ovver a connection and by averaging those samples into a "smoothed" delay estimate $\hat{d}_{i+1}$.

When a heartbeat is sent to the receiver, the receipt time (how long it takes) is recorded. Here, we assume the drift rates of clocks are very small and can be ignored. And when many heartbeats are sent out, we can get a sequence (D) of delay samples: $D = d_1, d_2, d_3, ...$

With each sample d_i, the new predictive delay $\hat{d}_{i+1}$ is computed from the formula:

$$\hat{d}_{i+1} = \alpha \times \hat{d}_i + (1 - \alpha) \times d_i, \tag{3.1}$$

where α ($0 \leq \alpha \leq 1$) is a constant between 0 and 1, which controls how rapidly the $\hat{d}_{i+1}$ adapts to the delay change.

The fresh-point for heartbeat $(i + 1)$ can be computed by

$$\tau_{i+1} = EA_{i+1} + \beta \cdot (|\hat{d}_{i+1} - \bar{d}_i| + \epsilon), \tag{3.2}$$

where $EA_{i+1} = i \cdot \Delta(t) + \bar{d}_i$, $\Delta(t)$ is the average sending interval in a slide window size; $\bar{d}_i$ is the average value of actual sampled delay. β is a variable, chosen such that there is an acceptably small probability that the delay for the heartbeat will exceed timeout. Here, ϵ is a positive constant.

In network communication, the heartbeat messages can be lost or delayed, and we can't get the communication delay from the sender to the receiver when it is lost. In order to ensure the effectiveness of the proposed approach, and considering the influence of message loss, we adopt the time series theory to fill the gaps. In detail, we fill the gaps with a value computed by $d_i = (\Delta t \cdot \overline{nag}) + d_{i-1}$, where $\overline{nag}$ is the average number of observed adjacent gaps [25].

In order to describe the proposed approach clearly, we give the following specific algorithm (see Figure 3.1). The procedure works as follows: There are two processes p (sender) and q (receiver). Firstly, process p periodically sends heartbeat message to process q. Every heartbeat message includes a sequence number to distinguish the different messages and the sequence number is increased one every time.

And process q is to receive the message from process p. If p is in the suspecting list (*suspectlist*) of q, and q still didn't receive the expected message j during the last τ ticks of q's clock, then q starts suspecting p and adds p into its *suspectlist*. Upon receiving the expected heartbeat j from p, q will compare the sequence number j and the sequence number of the most recent heartbeat (ID_{msg}). If j larger than ID_{msg}, i.e., the received heartbeat is latest and effective, then process q will do the following work. Firstly, q checks if p is in the suspecting list. If $p \in$ *suspectlist*, then q will remove p from *suspectlist* and the value of β increase one; else the value of β is initialized 1. And then the current time is got, the new value of timeout is updated based on the equation (3.1) and (3.2). The sequence number of the most recent heartbeat is updated by j and the value of j is increased by 1. Therefore, the computation of timeout is updated based on the dynamic of network environments.

3.2 Algorithm correctness

Before the proof of algorithm, we would like to give the following symbols used in the proof of algorithm.

Initialization:
τ: initial value; WS: window size; Δt: sending interval;
/*slide window saves the recent sampled delay*/

$Win\_delay[\,] = \perp$; /*empty sliding window for delay*/

$suspectlist[\,] = \perp$; /*empty suspect list*/

$ID_{msg} = 0$; /*the sequence number of the most recent received heartbeat*/

Process p **(Sender)**:
For all $i \geq 1$, at time $(i \cdot \Delta t)$: Send heartbeat i to q;
Process q **(Receiver)**:
Task 1: If $p \notin suspectlist$, and q didn't receive the expected message j during the last τ ticks of q's clock

Add p to $suspectlist$; /*Suspect p*/;

Task 2: Upon receiving heartbeat j from p
If $j > ID_{msg}$,
{ **If** $p \in suspectlist$

{ Remove p from $suspectlist$;
$\beta = \beta + 1$; }

else
$\{\beta = 1;\}$

$t_{crt} = clock(\,)$; /*Get the current time*/

/*Compute the delay from sending to receiving*/

$d_j = t_{crt} - (j-1) \cdot \Delta t$; /*$j$: sequence number*/

$Win\_delay[\,] \leftarrow d_j$; /*save the sampled delay*/

$\bar{d}_j = \frac{1}{WS} \cdot \sum_{i=j-WS+1}^{j} Win\_delay[i]$; /*$\bar{d}_j$: average delay*/

$EA_{j+1} = j \cdot \Delta t + \bar{d}_j$; /*$EA_j$: next arrival time*/

$\hat{d}_{j+1} = \alpha \cdot \hat{d}_j + (1-\alpha) \cdot d_j$; /*$\hat{d}_{j+1}$: predictive delay*/

$\tau_{j+1} = EA_{j+1} + \beta \cdot (\mid \hat{d}_{j+1} - \bar{d}_j \mid + \epsilon)$; /*Compute timeout based on network variation*/

/*Update the sequence number of the most recent message ID_{msg} and the expected message

$ID_{msg} = j$;

$j = j + 1$;

}

Figure 3.1: Implementation of TAM FD

$correct(t)$: the set of correct processes at time instance t;

$crashed(t)$: the set of crashed processes at time instance t;

$suspect_p(t)$: the set of suspected processes by process p at time instance t;

t_{GST}: the global stabilization time;

Δ_{msg}: the maximum time after GST, between the sending of a message and the delivery and processing by its destination process;

t_{crash}: the time when a process crashes;

m_k: the message k, where k means the sequence number of message;

t_{rk}: the time when message k is received;

t_{sk}: the time when message k is sent;

τ_k: the value of timeout of the expected message k;

EA_k: the expected arrival time of message k;

α: a variable using for predicting the communication delay between sender and receiver;

β: a variable using for computing safety margin of timeout;

d_k: the time delay between the time that message k is started sending and the time that message k is received.

$\hat{d}_k$: the predicted delay for message k;

$\bar{d}_{k-1}$: the average delay of the past messages in sliding window

WS: the sliding window size;

Δ_i: the sending interval of message;

ϵ: a constant using for computing safety margin;

$SM(t)$: the safety margin at time t for computing timeout.

3.2.1 Strong completeness

Theorem 3.1 *Strong Completeness. Eventually every process that crashes is permanently suspected by every correct process.*

$$\exists t_0, \forall t \geq t_0, \forall p \in correct(t), \forall q \in crashed(t),$$

$$q \in suspected_p(t).$$

That is there is a time t_{mute} after which no correct process q receives heartbeat messages from the crashed process p, and there is a time $t_{timeoutk}$ after which all correct processes q permanently suspect p.

Lemma 3.1. *If process p crashed at t_{crash}, then there is a time t_{mute} after which process q stops receiving messages from p.*

$$t_{mute} \leq t_{crash} + \Delta_{msg},$$

where Δ_{msg} means the maximum time after GST, between the sending of a message and the delivery and processing by its destination process, assuming that the destination process has not failed.

Proof: All the time instants considered in the rest of this section are assumed to be after GST. We also assume that, at these instants, all the messages sent before GST have already been delivered and processed. These assumptions allow us to consider in the rest of the section, that the unknown bounds on process speeds and on message transmission times hold.

$$\exists t_{GST} : \forall m_k | t_{sk} \geq t_{GST} : (t_{rk}) - t_{sk} < \Delta_{msg},$$

where t_{sk} is the time when q sends m_k and t_{rk} is the time when p receives m_k.

Suppose a process p crashed at t_{crash}. Then p stops sending heartbeat messages.

$$\nexists m_k \mid t_{sk} \geq t_{crash}.$$

The process q cannot receive message k from process p after $t_{rk} + \Delta_{msg}$. Hence process q cannot receive any message from process p after $t_{rk} + \Delta_{msg}$.

Lemma 3.2 *For any sequence of k messages received by process q from p, there is a time τ_k after which process q starts suspecting process p if it does not receive any message from p.*

From *Task 2*, when the process q receives a message m_{k-1} from process p, process q calculates a new τ_k after which process q starts suspecting process p. We must prove that the τ_k is always bounded. The τ_k is calculated as follows (Task 2):

$$\tau_k = EA_k + \beta \cdot (\mid \hat{d}_k - \bar{d}_{k-1} \mid + \epsilon).$$

For this algorithm, if some process (p) crashed at time t_{crash}, then the other processes (q) cannot receive any heartbeat from it. And p will be added into the *suspectlist* (Task 2 in Figure 3.1) and will never be removed from the list after t_{crash}. Therefore, p is permanently suspected by every correct process.

In Task 2, if $k > WS$, EA_k is equivalent at:

$$EA_k = \frac{1}{WS}(\sum_{i=k-WS-1}^{k} t_{ri} - \Delta_i * i) + k\Delta_i,$$

from our model

$$EA_k < \frac{1}{WS}(\sum_{i=k-WS-1}^{k} t_{si} + \Delta_{msg} - \Delta_i * i) + k\Delta_i,$$

as $t_{si} = t_{s(i-1)} + \Delta_{msg} + k\Delta_i$, then

$$EA_k < t_{s0} + \Delta_{msg} + k\Delta_i. \tag{3.3}$$

Partial result 1 *The expected arrival time of the m_k message is bounded by:*

$$t_{sk} < EA_k \leq t_{sk} + \Delta_{msg}.$$

In Task 2, the safety margin SM is obtained:

$$\hat{d}_{k+1} = \alpha \cdot \hat{d}_k + (1-\alpha) \cdot d_k,$$

$$SM = \beta \cdot (\mid \hat{d}_{k+1} - \bar{d}_k \mid + \epsilon),$$

i.e.,

$$SM = \beta(\mid \alpha \cdot \hat{d}_k + (1-\alpha) \cdot d_k - \bar{d}_k \mid + \epsilon). \tag{3.4}$$

From partially synchronous system, one has

$$0 < d_k = t_{rk} - t_{sk} < \Delta_{msg}. \tag{3.5}$$

$$0 \leq \bar{d}_k = \frac{1}{WS} \cdot \sum_{i=k-WS-1}^{k} d_i < \Delta_{msg}. \tag{3.6}$$

$$\hat{d}_{k+1} = \alpha \hat{d}_k + (1-\alpha) d_k.$$

By recursion property of equation (3.1), one gets

$$\begin{aligned} \hat{d}_{k+1} &= \alpha(\alpha \hat{d}_{k-1} + (1-\alpha) d_{k-1}) + (1-\alpha) d_k \\ &= \alpha^2 \hat{d}_{k-1} + \alpha(1-\alpha) d_{k-1} + (1-\alpha) d_k \\ &\quad ... \\ &= \alpha^{k+1} \hat{d}_0 + \alpha^k (1-\alpha) d_0 \\ &\quad + \alpha^{k-1}(1-\alpha) d_1 + ... + (1-\alpha) d_k. \end{aligned} \tag{3.7}$$

Because $d_0 = \hat{d}_0 = 0$, so

$$\begin{aligned} \hat{d}_{k+1} &= \alpha^{k-1}(1-\alpha) d_1 + \alpha^{k-2}(1-\alpha) d_2 + ... + (1-\alpha) d_k \\ &= (1-\alpha) \sum_{i=1}^{k} \alpha^{k-i} d_i. \end{aligned} \tag{3.8}$$

Therefore, using the same methods, we get

$$\hat{d}_k = (1-\alpha) \sum_{i=1}^{k-1} \alpha^{k-i} d_i. \tag{3.9}$$

Because $0 \leq d_i < \Delta_{msg}$, therefore

$$0 \leq \hat{d}_k < \alpha(1-\alpha^{k-2}). \tag{3.10}$$

From inequalities (3.14), (3.15) and (3.19), we get

$$0 \leq SM < \beta \cdot max(\Delta_{msg}, \alpha^2(1-\alpha^{k-2}) + (1-\alpha\Delta_{msg}) + \epsilon). \tag{3.11}$$

In partial result 1, we show that the expected arrival date for any message m_k is bounded, and in result 2, the safety margin SM is bounded. All components of τ_k are bounded, so we can deduce τ_k is bounded. If for each message m_{k-1} received from process p, process q activates a bound timeout, then there is a time after which q suspects p, if it receives no new message from p. Then strong completeness is proved.

Eventually strong accuracy

Theorem 3.2 *Eventually strong accuracy. There is a time after which a correct process is no longer suspected by any correct process.*

$$\exists t_{bound}, \forall t \geq t_{bound}, \forall p, q \in correct(t),$$

$$q \notin suspect_p(t).$$

Theorem 3.3 is verified if the τ_k of process q is large enough to avoid the situation where process q wrongly suspects process p.

Lemma 3.3. *Every time q times out and p is correct, then β is increased. There is a time t_{bound} where safety margin $SM = \beta(\mid \hat{d}_{k+1} - d_k \mid +\epsilon)$ is large enough to avoid false detection, and SM stops increasing. When SM becomes higher than Δ_{msg}, then no false detection can occur.*

$$\exists t_{bound}, \forall t \geq t_{bound}, SM(t) \geq \Delta_{msg}, and$$

$$SM(t) = SM(t+1).$$

Lemma 3.4. *There is a time after which τ_k is greater than $(t_{sk} + \Delta_{msg})$.*

$$\exists t_{bound}, \forall t \geq t_{bound}, \tau_k \geq t_{sk} + \Delta_{msg}.$$

Proof: From Lemma 3.1, 3.2 and 3.3, we can say that

$$\forall m_k, t_{sk} < EA_k.$$

With β increased, there is a time t_{bound}, after which $SM(t) \geq \Delta_{msg}$. Therefore, we can conclude that

$$\exists t_{bound}, \forall t > t_{bound}, \tau_k > t_{sk} + \Delta_{msg}.$$

Theorem 3.3 is verified because if τ_k is larger than $(t_{sk} + \Delta_{msg})$ then process p cannot be considered by process q as having failed.

Theorem 3.3 TAM FD implements a failure detector of class $\diamondsuit P$, on condition that the system is in accordance with the system model defined in Chapter 2.1 (see the proof of Theorem 3.1 in following).

Proof. From the result of Theorem 3.1 and 3.2, we know Theorem 3.3 holds.

3.3 Performance evaluation

In this section, we evaluate and comparatively analyze the performance of TAM FD, ϕ FD [18-19], Chen FD [30], and Bertier FD [16-17] in a Cluster, a WiFi network, a wired LAN and a wide area network (WAN). The experiments are carried out with two computers. One sends heartbeat message periodically (process p), and the other is used to record the arrival time of every heartbeat (process q). Using the *traceroute* and *ping* commands, we can observe that most of the traffic was actually routed without network breaking down. And we can use the *ping* command to check the RTT. All heartbeats are transmitted using UDP. In addition, we found that the average CPU load was nearly constant during the experiments, and load was also below the full capacity of the two computers.

All the FDs are compared with the same experiment condition: the same network model, the same heartbeat traffic, and the same experiment parameters (sending interval, slide window size, and communication delay, etc.).

Structure of the section The remainder of the section 3.4 is organized as follows: We first present the general experimental setup for the next experiments. Then in Section 3.4.1, we give the experiment in a Cluster group. Section 3.4.2 describes the experiment in a WiFi network. In Section 3.4.3, we conduct the experiment in LAN network. Section 3.4.4 describes the experiment in a WAN network. In Section 3.4.5, we give a simple conclusion for the comparative analysis for the above four FDs. Section 3.4.6 explores the impact of memory usage on the QoS of FDs.

Experimental setup In every experiment, we have logged heartbeat sending and arriving time into some log files. Then the transfer log files can be used to compute the statistics mentioned above. Furthermore, we replayed the receiving times logged for each different FD scheme and every different value of the parameters. Thus, the four FD schemes have the exactly same scenarios, so it is a fair comparison.

In our experiments, each FD scheme uses a slide window to save past samples to compute their estimations. Unless stated otherwise, the FDs use the same window size ($WS = 1000$). The small window size is useful to save memory and CPU resources. Furthermore, it is reasonable to analyze the data only after the slide window is full. The network is unstable and inefficient in warmup period, therefore, the data before the full window is excluded from our analysis.

The parameters used are as follows: in all the experiments, the basic parameters of TAM FD are set as $\alpha = 0.85$, and in order to find the best QoS and compare with the others, here $\beta \in [10^{-6}, 10^{6}]$. Here ϵ is assumed to be a very small constant, then we think $\epsilon \cdot \beta$ is close to 0, and $\epsilon \cdot \beta$ is ignored in our experiment. This is reasonable because one can always choose a small ϵ such that $\epsilon \cdot \beta$ is too small to be considered. For ϕ FD, the parameters are set the same as in [18-19]: $\Phi \in [0.5, 16]$. For Chen FD, the parameters are set the same as in [18]: $\alpha \in [0, 10000]$. For Bertier FD, the parameters are set the same as in [16-17]: $\beta = 1$, $\phi = 4$, $\gamma = 0.1$.

In these experiments, we focus on the following key performance metrics: mistake rate (MR), query accuracy probability (QAP) and detection time. In every experiment result, the different values of mistake rate, query accuracy probability and detection time are obtained with the respective parameters.

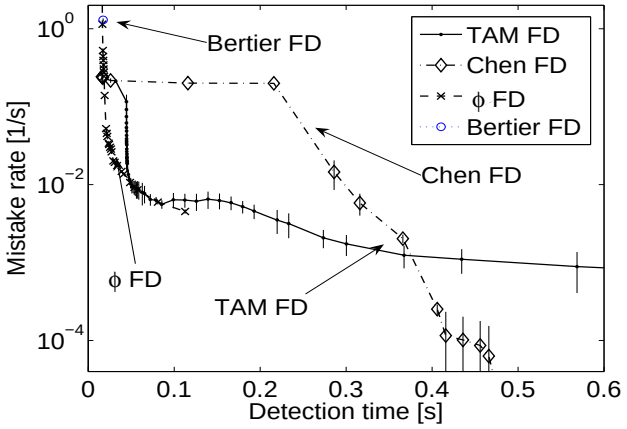


Figure 3.2: Mistake rate comparison of FDs with 95% confidence level in a Cluster case.

3.3.1 Experiment in a Cluster group

This experiment is performed with two cluster nodes, the sending node (monitored) and the receiving node (monitoring) were located in a Cluster Group, at Japan Advanced Institute of Science and Technology (JAIST), in Japan. The two nodes transfer messages through a normal network connection.

Experiment setting: hardware/software/ network

The two nodes were equipped with the same hardware and software: an Intel(R) Pentium(R) IV (CPU 2.80 GHz) and 512 Kb of cache size. The operating system was Fedora-Core 4 (Linux). The network is 1 Gb/s. The heartbeat messages were generated at a target rate of one heartbeat every 10 ms (the sending interval).

In order to make the experiment universal, we re-did the experiments 5 times with the same code, the same environment, and the same parameters (including the communication delay) for each FD scheme, but with different experiment time length. The experiment periods[2] are about 1 hour 1 minutes, 1 hour 4 minutes, 5 hours 3 minutes, 7 hours 5 minutes, and 9 hours 3 minutes, respectively.

For the five experiments, here we give the detailed experiment data for the one that has the longest experiment time.

Round-trip time In the experiment, we measured the RTT. The average RTT was 0.196 ms. The standard deviation was 0.233 ms with a minimum of 0.103 ms, and a maximum of 1.108 ms, so we get the relative standard deviation is $0.233/0.196 * 100\% = 118.88\%$.

Heartbeat sampling The average sending rate actually measured was of one heartbeat every 16.02 ms (standard deviation: 1.71 ms; min.: 0.005 ms; max.: 191.98 ms). We got about 2024972 samples (about 9 hours and 3 minutes), and no heartbeats were lost.

Experiment results

We got behaviors of detection time, mistake rate and query accuracy probability with a 95% confidence level in Figures 3.2-3.3. Figure 3.2 shows mistake rate comparison of FDs, the vertical axis is on a logarithmic scale. And best values are located toward the

[2]Here we perform experiments for five times with five different experiment time, then we could get general experimental results. Based on the results, we compute the more general results to compare our FD to other existing FDs with 95% confidence level in this Cluster case.

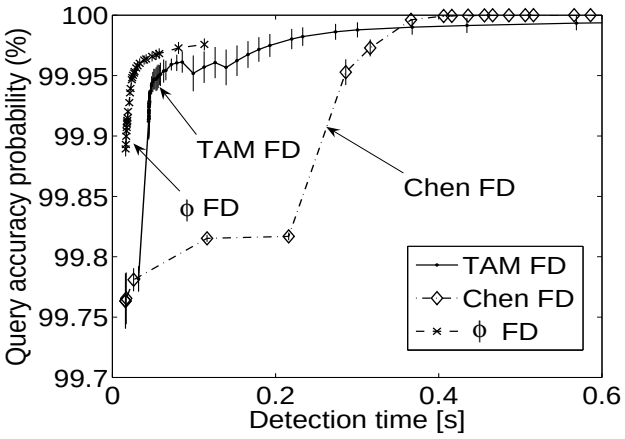


Figure 3.3: Query accuracy probability comparison of FDs with 95% confidence level in a Cluster case.

lower left corner, for it means that this FD provides a short detection time and has a low mistake rate. Figure 3.3 shows query accuracy probability comparison of FDs, the vertical axis is on a linear scale. And best values are located toward the higher and left corner, for it means that the FD provides a short detection time and has a high query accuracy probability.

In Figure 3.2, when $T_D < 0.04$ s, the ϕ FD can obtain the lowest mistake rates among the four schemes with the same detection time; And when $T_D > 0.38$ s, Chen FD has the fewest mistakes; While, when 0.04 s $< T_D <$ 0.38 s, TAM FD obtains the lowest mistake rates except some junctions of ϕ FD and TAM FD based on the confidence interval, where it means one can not say which one is better, both of them have good performance in this case. Here the behavior of Bertier FD is plotted as a single point, because it has no tuning parameters. And we found that Bertier FD doesn't perform very well compared with others. In Figure 3.3, we can obtain similar results about query accuracy probability and detection time as in Figure 3.2. In summary, the ϕ FD behaves a little better in the more aggressive range, Chen FD behaves a little better in the more conservative range, while, the TAM FD is a little better between the above two cases.

Here I analyze the possible reasons are as follows. The Bertier FD have no dynamic parameters to adjust, so it have only one point in Figure 3.2. It has very short detection time and high mistake rate. For the Chen FD, ϕ FD and TAM FD, they have dynamic parameters, and have different QoS based on the different parameters. So they are graphs. The ϕ FD is more aggressive than Chen FD, so the ϕ FD behaves a little better in the more aggressive range, while in the more conservative range, there are no data for ϕ FD. That is because for Chen FD, the safety margin can be any positive values, even very large safety margin hasn't been useful for actual applications. Because in actual application, large safety margin of failure detector brings large timeout, while the monitoring process can't wait for message from monitored process so long time. The reason why ϕ FD is so aggressive (the detection time is always very short), we can find the reason from the computing method of suspicion level Φ (equation (2.8)). Considering the equation (2.8), when a specific suspicion level Φ is given, the expected arrival time can be computed by equation $10^{-\Phi} = P_{later}(t_{now} - T_{last})$, where t_{now} is the expected arrival time of heartbeat and $\Phi \in (0,)$. When Φ becomes larger, $10^{-\Phi}$ becomes smaller quickly. Based on normal

distribution and very short range of $10^{-\Phi}$, the time is very limited, thus that leads to short detection time of Φ failure detector. The TAM FD is a development from Chen FD, and it is more aggressive than Chen FD, while it is conservative than ϕ FD. That is because the computing method of timeout in TAM FD is based on the network dynamic, and the delay of network communication is bounded, so the safety margin can't be any value like that in Chen FD. In all, in the more aggressive range, the ϕ FD behaves a little better than others; in the more conservative range, Chen FD behaves a little better than others; Between the two extreme range, the TAM FD is a little better than others.

3.3.2 Experiment in a WiFi network

These experiments involved two Mac computers and an AirMac Extreme base station. The AirMac extreme base station is used to build a private wireless LAN. The two Mac computers were located in our lab, in JAIST, Japan. And they were communicating through this normal WiFi (802.11g) network connection.

Experiment setting: hardware/software/ network

The two Macs were equipped with the same hardware and software: a PowerPC G4 (CPU 1.5 GHz) and 768 MB DDR SDRAM of Memory. The operating system was Mac OS X (Version 10.4.8). The heartbeat messages were generated at a target rate of one heartbeat every 100 ms (the sending interval), different from the first case, because the communication delay is longer in WiFi case than that in cluster group.

Similarly to the cluster case, we also re-did the experiment 5 times with the same experiment condition, but with different experiment time. The experiment periods[3] are about 1 hour 7 minutes, 2 hours 4 minutes, 6 hours 12 minutes, 9 hours 4 minutes, and 12 hours 11 minutes, respectively.

For the 5 experiments, here we give the detailed experiment data for the one (about 6 hours and 12 minutes).

Round-trip time By measurement, the average RTT is 1.829 ms. The standard deviation is 0.87 ms with a minimum of 1.176 ms, and a maximum of 21.941 ms, it is very easy to get the relative standard deviation $0.87/1.829 * 100\% = 4.76\%$.

Heartbeat sampling The average sending rate actually measured was of one heartbeat every 100.60 ms (standard deviation: 8.07 ms; min.: 1 μs; max.: 938.96 ms). We got about 221,574 samples, and no heartbeats were lost.

Experiment results

We got results for mistake rate and query accuracy probability with 95% confidence level in Figures 3.4-3.5. Figure 3.4 describes the relationship of mistake rate and detection time in these four FDs on a logarithmic scale. Figure 3.5 shows the change of the query accuracy probability with different detection time on a linear scale. We find that all of these FD have a similar trend.

In Figure 3.4, with $T_D < 0.34$ s, the ϕ FD has the lowest mistake rate, and except for that, TAM FD and ϕ FD have similar mistake rates considering the 95% confidence level, while the mistake rates of the TAM FD and ϕ FD are lower than Chen FD's. Furthermore, when $T_D < 0.24$ s, the TAM FD and Chen FD have the similar mistake rates, and except for that, TAM FD has a lower mistake rates than Chen FD. Here the behavior of Bertier FD is also plotted as a single point. In Figure 5, when $T_D < 0.28$ s, the ϕ FD has the

[3]Here we perform experiments for five times with five different experiment time, then we could compute the general results to compare our FD to other existing FDs with 95% confidence level in this WiFi case.

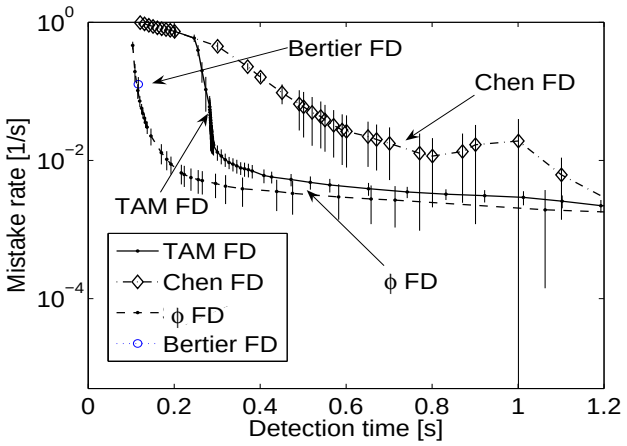


Figure 3.4: Mistake rate comparison of FDs with 95% confidence level in a Wireless case.

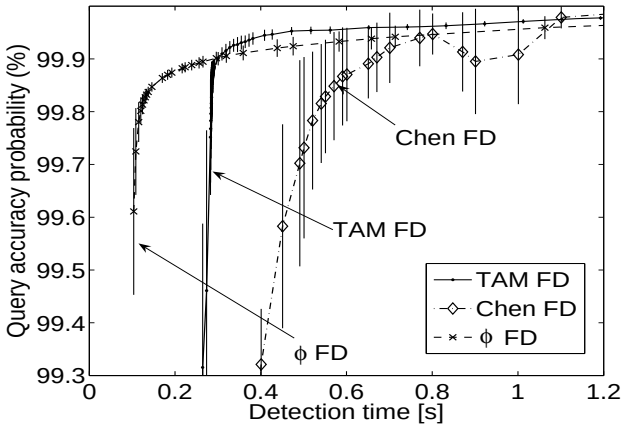


Figure 3.5: Query accuracy probability comparison of FDs with 95% confidence level in a Wireless case.

highest query accuracy probability. When T_D is between 0.28 s and 1.10 s, TAM FD has the highest query accuracy probability. When $T_D > 1.10$ s, all of TAM FD, ϕ FD and Chen FD achieve good query accuracy probability. In summary, in this WiFi case, ϕ FD has slightly better performance than the other three FDs in the aggressive range of FD. In the conservative range, TAM FD and ϕ FD have a similar result, and they behave a little better than Chen FD.

We analyze the reason for the above results in this WiFi case with the average RTT about 1.829 ms: there are no lost data, and the variation of delay is still small (RTT standard deviation is about 0.87 ms), so WiFi is a comparatively stable case. Thus, ϕ FD with normal distribution can obtain better performance, while TAM FD also obtains the similar performance to ϕ FD in the conservative range. Seen from the above results, we find in stable case, TAM FD and ϕ FD can achieve better performance than Chen FD and Bertier FD in terms of lower mistake rates, higher query accuracy probability and shorter detection time.

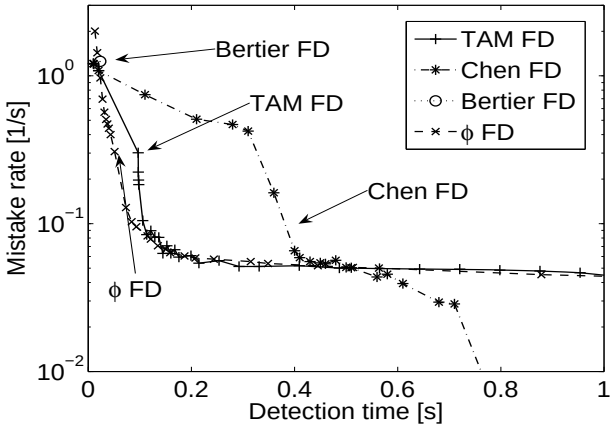


Figure 3.6: Mistake rate comparison of FDs in a LAN case.

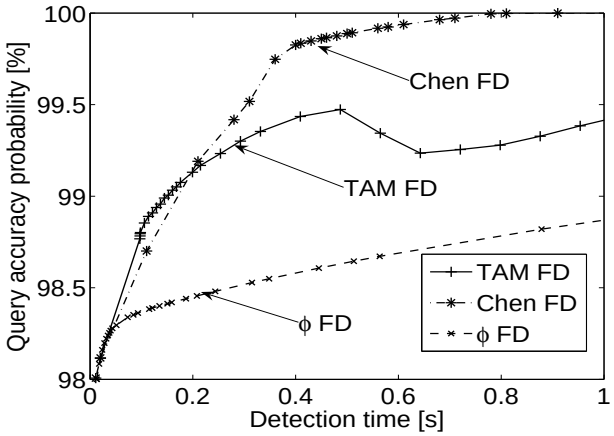


Figure 3.7: Query accuracy probability comparison of FDs in a LAN case.

3.3.3 Experiment in a LAN

This experiment also involved two Mac computers in a wired LAN. The sending host (monitored) and the receiving host (monitoring) were located in our lab, within about 30 meters, at JAIST, in Japan.

Experiment setting: hardware/software/ network

All the settings of the computers are the same as in the above WiFi experiment. The two computers are connected through a single 100 Mbps Ethernet hub, with no other systems attached.

Round-trip time The average RTT is 0.72 ms. The standard deviation is 0.70 ms with a minimum of 0.60 ms, and a maximum of 1.11 ms, so we get the relative standard deviation is $0.70/0.72 * 100\% = 97.22\%$.

Heartbeat sampling Considering the communication delay in LAN case, the heartbeat sending interval is 10 ms. The average sending rate actually measured was of one heartbeat every 10.19 ms (standard deviation: 8.48 ms; min.: 6 μs; max.: 887.47 ms). There were 367,662 samples received (1 hours 2 minutes and 26 seconds), and no heartbeats were lost.

Experiment results

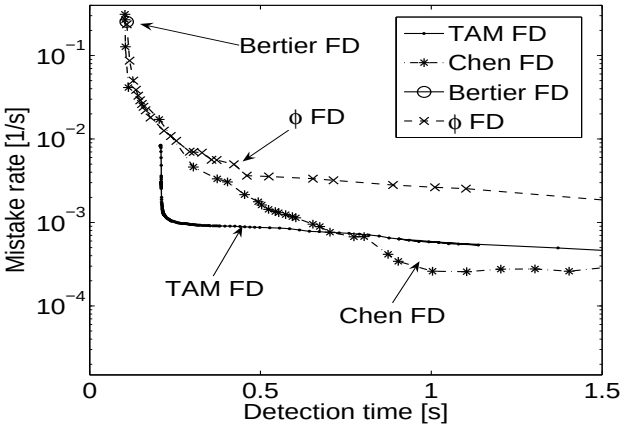


Figure 3.8: Mistake rate comparison of FDs with $WS = 1000$ in a WAN case.

The results of the experiment are depicted in Figures 3.6-3.7. In Figure 3.6, when $T_D < 0.109$ s, ϕ FD has the lowest mistake rate, and when $T_D > 0.109$ s, TAM FD and ϕ FD have similar mistake rates, while the mistake rates of the TAM FD and ϕ FD are lower than Chen FD's until $T_D = 0.503$ s. Eventually, when $T_D > 0.503$ s, Chen FD has the lowest mistake rates. Figure 3.7 describes the relationship between detection time and query accuracy probability for each FD. In Figure 3.7, when $T_D < 0.2$ s, TAM FD gets the highest query accuracy probability, after that period, Chen FD gets the highest query accuracy probability, and TAM FD obtains the higher query accuracy probability than ϕ FD. In summary, in the aggressive range: ϕ FD has a slightly lower mistake rates than the other three FDs, while TAM FD gets the highest query accuracy probability. In the conservative range ($T_D < 0.503$ s), Chen FD behaves a little better than the other three FDs. While, when 0.1 s $< T_D <$ 0.2 s, TAM FD gets a little better performance than the others.

The reasons of the above results in this wired LAN is: there are no lost data yet[4]. Compared with the previous cluster case, this wired LAN has longer average RTT (0.72 ms $>$ 0.196 ms), and the variation of delay for wired LAN is larger than that of cluster (0.70 ms $>$ 0.233 ms). Then we note that, when the network system becomes more unstable, compared with the other three FDs, TAM FD obtains a little better performance.

3.3.4 Experiment in a WAN

In order to further compare QoS of these four FDs, we carried out an experiment in WAN. In this experiment, we use the exactly same trace files from the paper ϕ FD [18-19], and these trace files are publicly available on our lab web [27]. So this gives a common ground for comparing the results of TAM FD, Chen FD [30] and Bertier FD [16-17] with the earlier ϕ FD results [18-19].

That experiment involves two computers: one was located at the Swiss Federal Institute of Technology in Lausanne (EPFL), in Switzerland. The other one was located in JAIST, Japan. The two computers communicate through a normal intercontinental Internet connection.

Experiment setting: hardware/software/network

[4]it is comparatively stable, while it have larger fluctuation than that of cluster.

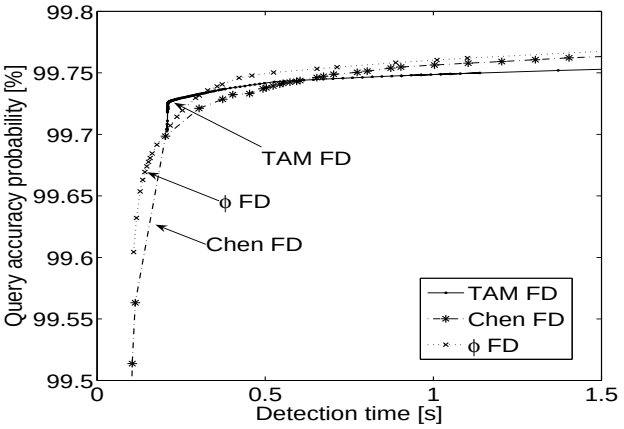


Figure 3.9: Query accuracy probability comparison of FDs with $WS = 1000$ in a WAN case.

In this experiment, the two computers have same equipments and same operating systems with those in [18-19]. By checking the network connection, one found that the two hosts and network link did not break down. Furthermore, the average CPU load for sending host and receiving host were 1/67 and 1/22, respectively.

Heartbeat sampling The experiment started on April 3, 2004 at 2:56 UTC, and finished on April 10, 2004 at 3:01 UTC. During the one week experiment period, the heartbeats were generated at a target rate of one heartbeat every 100 ms (the sending interval) due to the long communication delay in WAN. The average sending rate actually measured was one heartbeat every 103.501 ms (standard deviation: 0.189 ms; min.: 101.674 ms; max.: 234.341 ms). Furthermore, 5845713 heartbeat messages were sent out, while only 5822521 of them were received, so message loss rate was about 0.399 %.

By checking the traces files more closely, one found the messages losses were because of 814 different bursts. The majority of total bursts were the short length bursts. While the maximum burst-length was 1093 heartbeats (only one), it lasted about 2 minutes.

Round-trip time The average RTT is 283.338 ms, it is with a standard deviation of 27.342 ms, a minimum of 270.201 ms, and a maximum of 717.832 ms, the relative standard deviation is $27.342/283.338 * 100\% = 9.65\%$.

Experiment results

The results of the experiment are depicted in Figures 3.8-3.9. In Figure 3.8, when $T_D < 735.3$ ms, TAM FD obtains the lowest mistake rate among the four FDs. When $T_D > 735.3$ ms, Chen FD has the lowest mistake rate, compared with the other three FDs. It is clear that TAM FD has a slightly lower mistake rate than ϕ FD in all detection times. Here Bertier FD is still plotted as a single point. Query accuracy probability comparison of FDs in a WAN is shown in Figure 3.9. From Figure 3.9, we find: when $T_D < 394.3$ ms, TAM FD has the highest query accuracy probability; when $T_D > 394.3$ ms, ϕ FD has the highest query accuracy probability. With sufficient detection time, Chen FD has the highest query accuracy probability and eventually, and reaches the maximum value 100%. In summary, in the aggressive range of FD: TAM FD behaves a little better than the other three FDs in terms of the low mistake rate and high query accuracy probability. In the conservative range, Chen FD behaves a little better than the other three FDs.

We analyze the reasons of the above results in this WAN: there are some bursts or

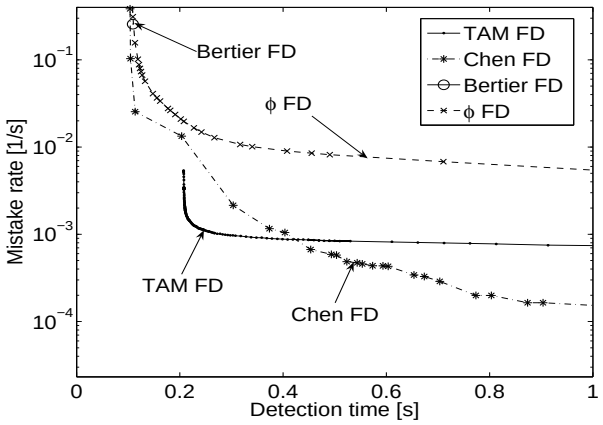


Figure 3.10: Mistake rate comparison of FDs with $WS = 100$ in a WAN case.

network congestion which lead to 0.399% lost data. Compared with the previous cluster case and wired LAN case, the WAN has larger RTT and more variability of transfer delay with more message loss, so the network is unstable. In this case, we found the TAM FD obtains a slightly better performance in the aggressive range of FD.

3.3.5 Comparative analysis for the four FDs

From all the above experimental results, the following remarks can be made: By predicting fresh point effectively with equations (3.10) and (3.11), all the above experiments demonstrate that TAM FD is an effective improvement over Chen FD and Bertier FD in terms of short detection time, low mistake rate and high query accuracy probability. In stable cases (no message loss and small variability of delay), such as in a cluster group, WiFi and wired LAN: In the aggressive case, TAM FD's QoS is better than Chen FD's and Bertier FD's; With more detection time, the performance of TAM FD is almost as good as that of ϕ FD.

Especially in unstable network environment, such as WAN, TAM FD obviously performs better than the others in aggressive case. However, we use the same fixed history-track-record length (1000) in the above experiments. Here, we would like further to explore how the history-track-record length affects on QoS of FDs in WAN case.

3.3.6 Effect of history-track-record length on QoS of FDs

In this section, we analyze the effect of history-track-record length on the FDs from two aspects: (1) Given a certain window size, we compare the performance of different FDs; (2) For each FD, we explore the performance change with different window sizes. Here we use the same experiment conditions and the same samples as in Section 3.6.4. Since the similar results were also got in other cases (cluster group, WiFi network, wired LAN), we didn't put them in this part due to the limited space. You can find them in the extended version of this chapter.

Performance comparison of different FDs

In Section 3.6.4, we have analyzed experiment results for different FDs with $WS = 1000$. Here we give results for two other cases $WS = 100$, and $WS = 10000$ in Fig-

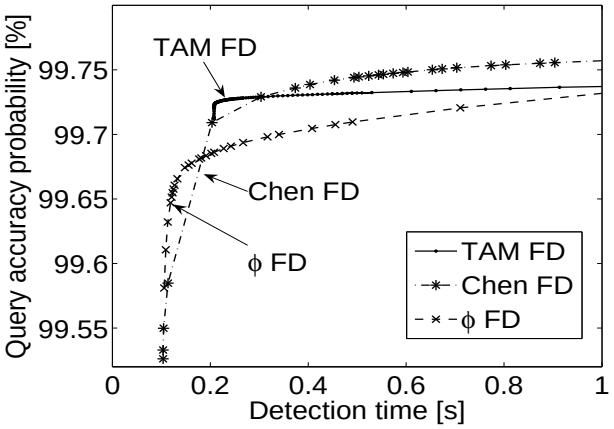


Figure 3.11: Query accuracy probability comparison of FDs with $WS = 100$ in a WAN case.

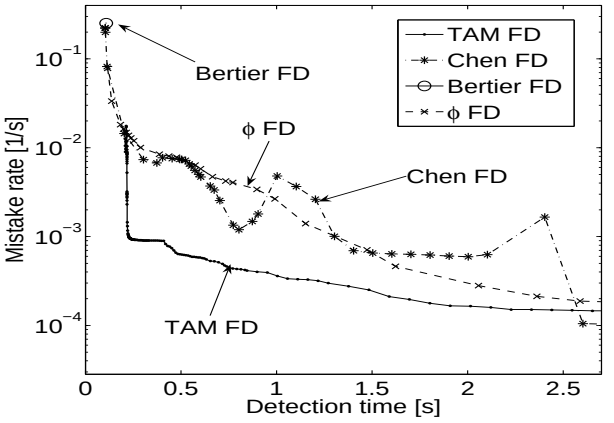


Figure 3.12: Mistake rate comparison of FDs with $WS = 10000$ in a WAN case.

ures 3.10-3.13. We find similar results to the $WS = 1000$ case.

In Figure 3.10, Chen FD has a slightly lower mistake rate than ϕ FD. When $T_D < 0.427$ s, TAM FD has the lowest mistake rate than the other FDs. Except for that, Chen FD has a little lower mistake rate than TAM FD. In Figure 3.12, when $T_D < 2.571$ s, TAM FD has the lowest mistake rate than the other FDs. Except for that, Chen FD has the lowest mistake rate. Furthermore, Chen FD and ϕ FD have many points of intersection. In Figure 3.11, TAM FD has the highest query accuracy probability than the others with $T_D < 0.309$ s, except for that, Chen FD has the highest value. In Figure 3.13, TAM FD also has the highest query accuracy probability than the others with $T_D < 0.562$ s, except for that, ϕ FD has the highest value. Figures 3.11 and 3.13 are respectively similar to Figures 3.10 and 3.12. In summary, in the aggressive range of FDs, TAM FD behaves a little better than the other three FDs in terms of the lowest mistake rate and the highest query accuracy probability with the shortest detection time.

As a whole, when window size changes from small to large (window size is set at 100, 1000, and 10000), TAM FD and ϕ FD become better and better compared with Chen FD and Bertier FD in the aggressive range; Furthermore, TAM FD outperforms ϕ FD, Chen FD and Bertier FD in the aggressive range.

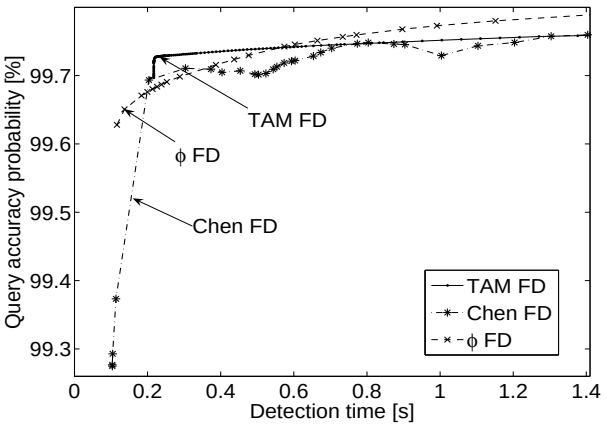


Figure 3.13: Query accuracy probability comparison of FDs with $WS = 10000$ in a WAN case.

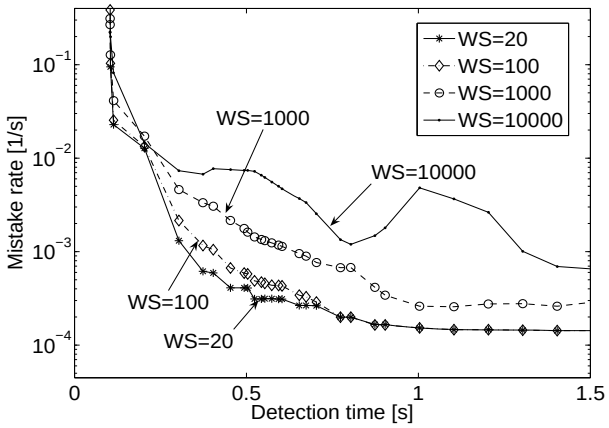


Figure 3.14: Mistake rate comparison of Chen FD with different window sizes in a WAN case.

Experiments with different window sizes

The extension experiment results are shown in Figures 3.14-3.15 (Chen FD), 3.16-3.17 (ϕ FD) and 3.18-3.19 (TAM FD) with different window sizes in a WAN. In Figures 3.14-3.15, it is clear that Chen FD with smaller window size has better performance in terms of short detection time, low mistake rate and high query accuracy probability.

In Figure 3.16, for 0.1414 s $< T_D <$ 0.21535 s, we can get lower mistake rate with larger window size. For other detection times, except for the $WS = 1000$ case, all the other four cases show that lower mistake rate is possible with larger window size. In Figure 3.17, except for the $WS = 1000$ case, larger window size provides better query accuracy probability. In Figures 3.18-3.19, we can find the change of window sizes has less effect on the performance change compared with Chen FD and ϕ FD. Therefore, TAM FD uses less window size to get acceptable performance and thus it can save valuable memory resources. For Bertier FD, the mistake rate $(1/s)$ and detection time (s) are $(0.10825, 0.255)$ for $WS = 20$, $(0.10886, 0.257)$ for $WS = 100$, $(0.10863, 0.254)$ for $WS = 1000$, and $(0.10859, 0.252)$ for $WS = 10000$. It is clear that the effect of window size on Bertier FD is smaller compared with Chen FD and ϕ FD.

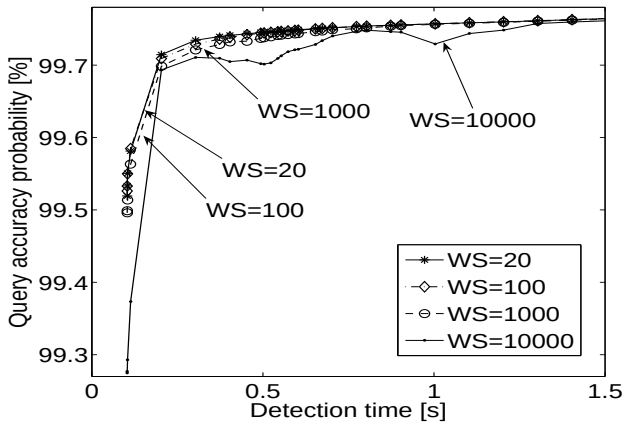


Figure 3.15: Query accuracy probability comparison of Chen FD with different window sizes in a WAN case.

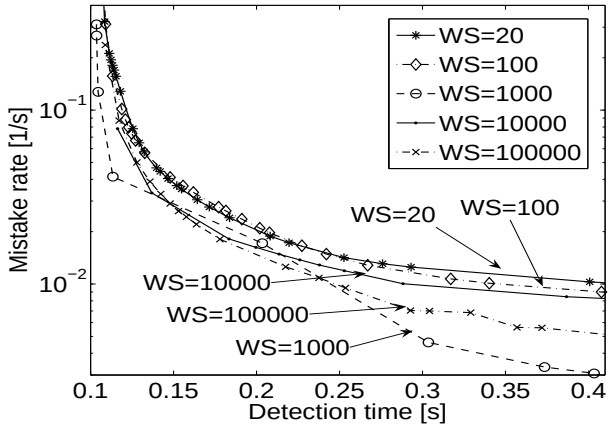


Figure 3.16: Mistake rate comparison of ϕ FD with different window sizes in a WAN case.

In summary, for the effect of history-track-record length on the QoS of FD, ϕ FD has a tendency for larger window size to achieve better performance, except for 1 case[5]; however, the larger window size for Chen FD leads to worse performance; for TAM FD and Bertier FD, the effect of window size on their QoS is negligible.

3.4 Conclusion

In this chapter, we compared QoS metrics of several adaptive FDs, discussed their properties and their relation, and then proposed one optimization over the existing methods.

From all the above experimental results, we use the same fixed history-track-record length (1000 samples). All the above experiments demonstrate that TAM FD is an effective improvement over Chen FD and Bertier FD in terms of short detection time, low mistake rate and high query accuracy probability. Here we compare TAM FD to ϕ FD from two aspects. (1)In stable cases (no message loss and small variability of delay),

[5]For ϕ FD, this conclusion in WAN is similar to the result in WiFi [28], where one needs a longer window size and a large threshold if one requires high accuracy.

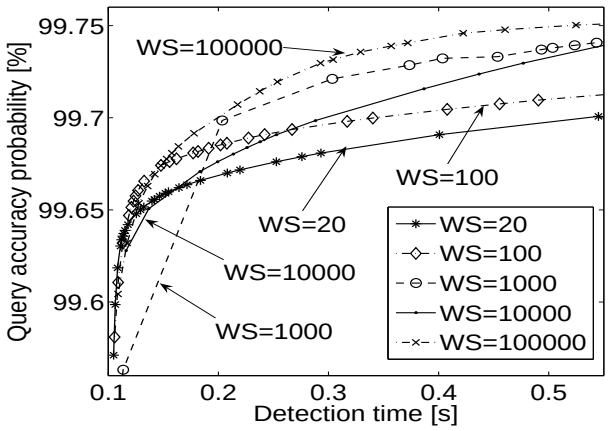


Figure 3.17: Query accuracy probability comparison of ϕ FD with different window sizes in a WAN case.

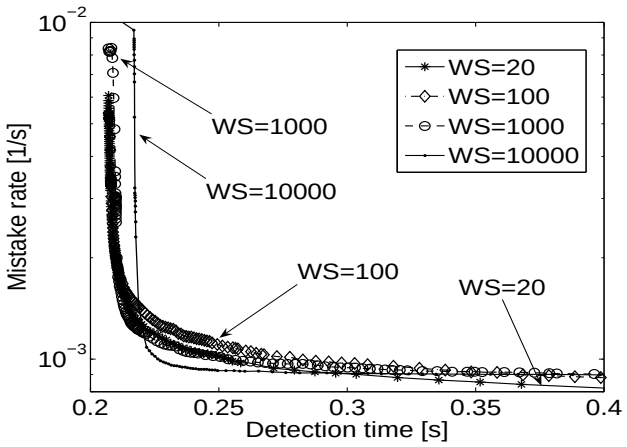


Figure 3.18: Mistake rate comparison of TAM FD with different window sizes in a WAN case.

such as in a cluster group, WiFi and wired LAN: In the aggressive case, TAM FD's QoS is better than Chen FD's and Bertier FD's; With more detection time, the performance of TAM FD is almost as good as that of ϕ FD. (2) In unstable network environment, such as WAN, TAM FD obviously performs better than the others in aggressive case.

Specially, the Bertier FD have no dynamic parameters to adjust, so it have only one point in Figure 3.2. It has very short detection time and high mistake rate. For the Chen FDϕ FD and TAM FD, they have dynamic parameters, and have different QoS based on the different parameters. The ϕ FD is more aggressive than Chen FD, so the ϕ FD behaves a little better in the more aggressive range, while in the more conservative range, there are no data for ϕ FD.

TAM FD is a development from Chen FD, and it could adjust his safety margin based on the network environments. TAM FD's QoS is better than Chen FD's in the aggressive case, while Chen FD is better in the conservative case.

The ϕ FD is very aggressive. In stable cases (such as in a cluster group, WiFi and wired LAN), ϕ FD behave better than Chen FD and TAM FD in more aggressive range; in the more conservative range, Chen FD behaves a little better than TAM FD and ϕ FD;

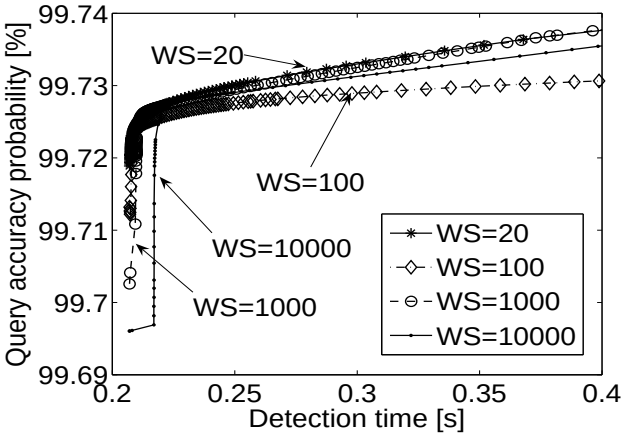


Figure 3.19: Query accuracy probability comparison of TAM FD with different window sizes in a WAN case.

Between the two extreme range, the TAM FD is a little better than others. In unstable cases, such as WAN, there are great change about the transfer delay, so at same detection time, ϕ FD and Chen FD get large mistakes. While our TAM FD could adjust his safety margin based on the network environments, and the less mistakes than ϕ FD and Chen FD. So TAM FD obviously performs better than the others in aggressive case.

Furthermore, we also analyzed the impact of history-track-record length on the performance of FDs by experiments. Experiment results demonstrate that the impact of memory usage on the overall QoS is different based on the different FDs. In summary, ϕ FD has a tendency for larger window size to achieve better performance, except for special cases. However, the larger window size for Chen FD leads to worse performance. For TAM FD and Bertier FD, the effect of window size on their QoS is very small and can be negligible.

By comparing the performance of the existed failure detectors by a lot of experiments, the users in actual application can choose a suitable failure detector. For example, for an application that has very limited memory (for example, there are very limited memory in some sensor networks), Bertier FD is suitable for such users. If the applications (for example, some applications on chemistry) are not sensitive to detection time, while needing as low as possible mistake rate, then Chen FD is more suitable for such requirements compared with other existed failure detectors. If an application (for example, the application on planes) more cares about detection time while allowing a certain mistake occurring, then TAM FD is a best choice. In all, this chapter first proposed an good failure detector and then provides useful information about performance of failure detectors to help application users selecting suitable failure detector based on their requirements.

Chapter 4

Exponential Distribution Failure Detector

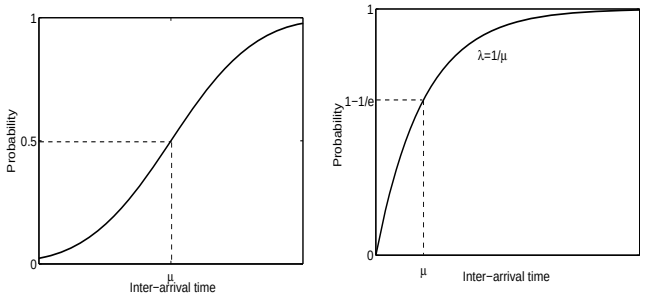
Hayashibara and Défago et al. [18] developed a ϕ FD, which assumes that the inter-arrival times follow a normal distribution, and computes a value ϕ with a scale that changes dynamically to match recent network conditions (i.e., this FD outputs suspicion level on a continuous scale, instead of traditional binary information. This is different from the other FDs). From the statistic analysis of the experimental results (see 4.1.2), we found the normal distribution is not a reasonable assumption for the approximation of the heartbeat inter-arrival time, especially in large scale distributed networks or unstable networks[1].

Therefore, in this chapter we propose a novel estimation of the distribution for the inter-arrival time, called the exponential distribution failure detector (ED FD), as an extension of ϕ FD [18]. Briefly speaking, the ED FD works as follows. The protocol uses a sliding window to maintain the most recent samples of the arrival time, similarly to conventional adaptive FDs [6, 30, 16]. The distribution of past samples in the sliding window is used as an approximation for the probabilistic distribution of future heartbeat messages. With this information, the suspicion level is computed using a scale that changes dynamically to match recent network conditions. By design, ED FD can adapt well to changing network conditions, and the requirements of any number of concurrently running applications. The experimental results demonstrated that ED FD provides the flexibility required for implementing a truly generic failure detection service. Furthermore, we comparatively evaluated our failure detection scheme with existing schemes (Chen FD [30], Bertier FD [16, 17], and ϕ FD [18]) by extensive experiments in four cases: a cluster group, LAN, and wide area network (WAN), and wireless network. The experimental results demonstrated that ED FD outperforms the existing FDs in terms of short detection time, low mistake rate and high query accuracy probability.

4.1 Exponential distribution failure detector

In this section, firstly an optimized ED FD over ϕ FD is presented, and then we explain why ED FD is an optimization over ϕ FD. Finally, we give a more precise description on

[1]Here the unstable networks means the networks have the high unpredictability of message delays, the great dynamic changing topology of system, and the high probability of message losses.



(a) (b)

Figure 4.1: Probability distribution vs. inter-arrival time: (a) for ϕ FD [18]; (b) for ED FD, and here $\mu = 1/\lambda$.

the implementation of ED FD.

The ϕ FD can output suspicion information on a continuous scale, and can adapt equally well to changing network conditions and the requirements of any number of concurrently running applications. But we found the normal distribution in ϕ FD (see Figure 4.1(a)) is not a reasonable assumption for the approximation of the heartbeat inter-arrival, especially, in large scale distributed networks or unstable networks. Thus, this paper develops the optimization over ϕ FD, called ED FD, to estimate the arrival time of the coming heartbeat.

4.1.1 ED FD algorithm

In this subsection, We assume that inter-arrival times follow an exponential distribution (see Figure 4.1(b)).

The ED FD implements the abstraction of an accrual FD in which the suspicion level is given by a value called e_d, expressed on a scale that is dynamically adjusted to reflect current network conditions. Then, the value of suspicion level e_d is calculated as follows.

$$e_d(t_{now}) \stackrel{\text{def}}{=} F(t_{now} - t_{last}), \tag{4.1}$$

where the $F(t)$ is an exponential distribution function, and one has

$$F(t) = 1 - e^{-\lambda t}, \tag{4.2}$$

where $t > 0$, and $\lambda = 1/\mu$ (μ is the average value of the past sampled inter-arrival times).

In this scheme, when $T_D = \mu$, it corresponds to a higher probability $(1 - 1/e)$ than 0.5. And when $T_D > \mu$, the sample data has larger probability than $T_D < \mu$. Then this scheme can catch the most sample data using a high probability, especially, the sample data that has the maximum probability of inter-arrival time. It is very reasonable and important for a good approximation.

4.1.2 Why ED FD is an optimization over ϕ FD

This section gives a comparative analysis of ED FD and ϕ FD based on the statistics of real sample data in several kinds of networks (a cluster group, wired LAN, WAN, and

wireless), and shows the probability distribution properties of arrival interval periods. Based on the properties, we analyze the normal distribution in ϕ FD [18], and then present the improved exponential distribution scheme over ϕ FD.

Experiment setting and method

For a wired LAN case, the sending host was located at Tsurugi, Ishikawa, Japan, and the receiving host was located at Japan Advanced Institute of Science and Technology (JAIST), Nomi, Ishikawa, Japan. There were 347, 940 samples received (about 1 hour and 6 minutes). We find the average inter-arrival time is 10, 782 μs (min.: 5 μs; max.: 488, 865 μs). Then from the minimum to the maximum, every 50 μs as a unit, we can get 9, 778 sample units, and the last one only has 10 μs time length. We can find the number is different in every statistic unit. So the probability for each unit is the value that the total sample data divided by the different number in every time unit. The other experimental setting for the wired LAN is shown in sub-Chapter 4.4.3.

For the cluster, wireless, and WAN cases, we used the same method to deal with the sample data, except that the WAN case used 100 μs as a statistic unit. Furthermore, the relevant experimental setting and sample data for cluster, wireless and WAN cases are shown in sub-Sections 4.4.1, 4.4.2, and 4.4.4, respectively.

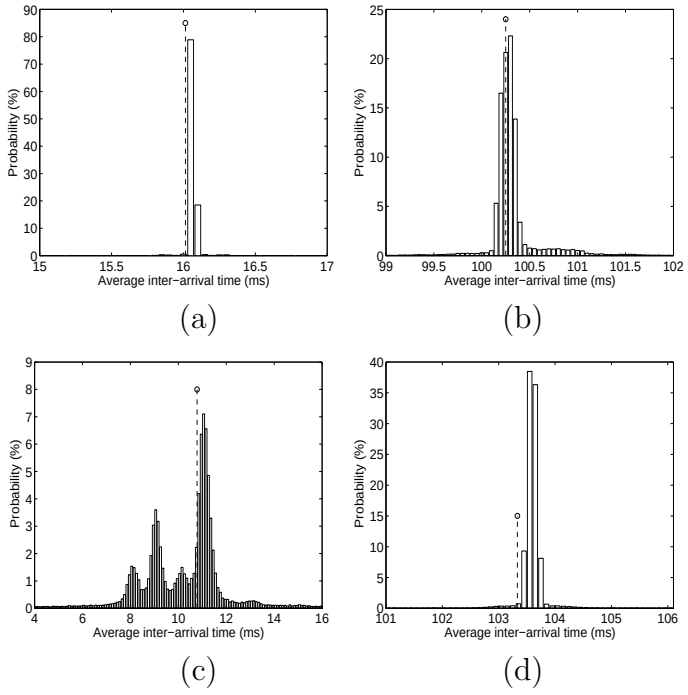


Figure 4.2: The experimental results of probability distribution vs. detection time: (a) for a cluster group, (b) for wireless, (c) for wired LAN, (d) for WAN.

Statistical experimental results

Based on the above experimental setting and statistic method, we got the experimental results of probability distribution and average detection time (see Figure 4.2). In each sub-figure of Figure 4.2, the circle point with a dashed line indicates the average value of all inter-arrival times. The average values are 16.05 ms for a Cluster case, 100.25 ms for a wireless case, 10.85 ms for a LAN case, and 103.3 ms for a WAN case, respectively. It is clear that, in general, probability near the average value of all inter-arrival times is higher than that far from the average value, and most samples have an inter-arrival time that is near the average value. Furthermore, the inter-arrival times with the maximum probability are 16.10 ms for a Cluster case, 100.30 ms for a wireless case, 11.15 ms for a LAN case, and 103.6 ms for a WAN case, respectively. Therefore it is clear that the inter-arrival time with the maximum probability in the each experiment is larger than the mean of all inter-arrival times, except that the inter-arrival time is similar to the mean of all inter-arrival times in wireless case.

The statistical analysis of sample data

There is an assumption that heartbeat inter-arrival times are influenced by a lot of independent unknown factors (central limit theorem) [18]. Chen el al. [30] and Hayashibara et al. [18] presented the estimation of the distribution for inter-arrival times: they follow a normal distribution. And the probability that a given heartbeat will arrive more than t time units later than the previous heartbeat is expressed in Equation (4.2).

Figure 4.1(a) shows the result of the probability distribution and inter-arrival time for ϕ FD [18], and μ is the mean of all inter-arrival time. It is clear that the inter-arrival time is μ with 0.5 probability. While, by doing a lot of statistic experiments, we found, in ED FD (see Figure 4.1(b)) inter-arrival time equals μ, the corresponding probability of inter-arrival time is $(1 - 1/e)$, it is much larger than 0.5.

Based on the Figure 4.2, it is obvious that the range near μ is a sensitive range, the Exponential distribution has a higher slope than that of Normal distribution. It means that in the sensitive range, Exponential distribution can depict the network heartbeat activity clearer than Normal distribution. Then it is a more aggressive scheme than ϕ FD, and ED FD estimation has a little less estimation errors than ϕ FD estimation.

Furthermore, we found local sample data in a window has similar statistical results as shown in Figure 4.2, and ϕ FD used the estimation Equations (2.8-2.9) to compute the value ϕ for the upper applications. Obviously, ED FD is more reasonable than ϕ FD.

4.1.3 Implementation of ED FD

This section first describes the architecture of ED FD, then presents the specific implementation algorithm of ED FD.

The architecture of ED FD

Conceptually, the implementation of ED FD on the a monitoring process q can be decomposed into three basic parts: Monitoring, Interpretation, and Action [18].

In traditional timeout-based FDs (Chen FD [30] and Bertier FD [16, 17]), the monitoring and the interpretation are combined within the FD, and the output is binary. While

ED FD, as an accrual FD, provides a lower-level abstraction that avoids the interpretation of monitoring information. Some value, the suspicion level associated with each process, is then left for the applications to interpret [18].

Application processes set a suspicion threshold according to their own QoS requirements: a low threshold generates many wrong suspicions but with a quick detection of an actual crash; Conversely, a high threshold is prone to generate fewer mistakes, but needs more time to detect actual crashes.

The implementation of ED FD

As an accrual FD, the implementation of ED FD is quite simple. After warm-up period, when a new heartbeat arrives, the inter-arrival time is put into a sampling slide window, and at the same time, the former oldest one is pushed out of the sampling window. Then the arrival time in the sampling window is used to compute the distribution of inter-arrival times, and get the average inter-arrival time μ in this slide window. After that, based on Equation (4.1) and Equation (4.2), we can compute the current value e_d. At last, applications compare the value e_d and its threshold, then they will carry out some actions, or start to suspect the process. The detail information for the implementation of ED FD is shown in Figure 4.3.

4.2 Algorithm correctness

From the view of theory, the ED FD belongs to the class $\diamondsuit P_{ac}$, that is sufficient to solve the Consensus problem. And it satisfied the property of accrual failure detector [19], accruement property and upper bound property.

Lemma 4.1 (Accruement) *If process p is faulty, then eventually, the suspicion level $sl_{qp}(t)$ is monotonously increasing at a positive rate.*

$$p \in faulty(F) \Longrightarrow \exists K \exists Q \forall k \geq K(sl_{qp}(t_q^{qry}(k)) \leq sl_{qp}(t_q^{qry}(k+1)) \wedge sl_{qp}(t_q^{qry}(k)) < sl_{qp}(t^{qry}(k+Q)))$$

proof: If process p is faulty, the most recent arrival time of heartbeat t_{last} is constant, at time slot $t_q^{qry}(k)$, the suspicion level e_d is

$$sl_{qp}(t_q^{qry}(k)) = e_d(t_q^{qry}(k)) = F(t_q^{qry}(k) - t_{last}) = 1 - e^{-\lambda(t_q^{qry}(k) - t_{last})}.$$

With time flying, in time slot $t_q^{qry}(k+1)$, the suspicion level is:

$$sl_{qp}(t_q^{qry}(k+1)) = e_d(t_q^{qry}(k+1)) = F(t_q^{qry}(k+1) - t_{last}) = 1 - e^{-\lambda(t_q^{qry}(k+1) - t_{last})}.$$

Since $t_q^{qry}(k) \leq t_q^{qry}(k+1)$, we get

$$-\lambda(t_q^{qry}(k) - t_{last}) \geq -\lambda(t_q^{qry}(k+1) - t_{last}),$$

and

$$e^{-\lambda(t_q^{qry}(k) - t_{last})} \geq e^{-\lambda(t_q^{qry}(k+1) - t_{last})}.$$

Therefore,

$$1 - e^{-\lambda(t_q^{qry}(k) - t_{last})} \leq 1 - e^{-\lambda(t_q^{qry}(k+1) - t_{last})},$$

Initialization:

WS: window size;

Δt: sending interval;

$t_{last} = 0$; /*Last arrival time of last heartbeat*/

$Win\_arr[\,] = \perp$; /*empty slide window for inter-arrival times*/

$ID_{msg} = 0$; /* the identifying number of most recent received heartbeats*/

Process p **(Sender)**:

For all $i \geq 1$, at time $(i \cdot \Delta t)$: Send heartbeat HT_i to q;

Process q **(Receiver)**:
Task 1: If q didn't receive message during a certain time period of q's clock

Increase e_d; /*Suspect p*/

Task 2: Upon receiving heartbeat HT_j from p
If $j > ID_{msg}$,
{

$t_{crt} = clock(\,)$; /*Get the current time*/

$Win\_arr[\,] = (t_{crt} - t_{last})$;

$\lambda = WS / \sum_{i=j-WS+1}^{i=j} Win\_arr[i]$;

$e_d = 1 - e^{-\lambda * (t_{crt} - t_{last})}$;

$t_{last} = t_{crt}$;

/*Update the sequence number of the most recent message ID_{msg} and the expected message

$ID_{msg} = j$;

$j = j + 1$;

}
Application k:

Compare e_d with the threshold E_d^k (from application requirement);

Carry out some actions or start to suspect p;

Figure 4.3: Implementation of ED FD.

i.e.,

$$sl_{qp}(t_q^{qry}(k)) \leq sl_{qp}(t_q^{qry}(k+1)).$$

At time slot $t_q^{qry}(k+Q)$, $Q > 0$, $t_q^{qry}(k+Q) > t_q^{qry}(k)$. Using the above same method and conclusion, we can get

$$sl_{qp}(t_q^{qry}(k)) < sl_{qp}(t_q^{qry}(k+Q)).$$

Therefore, ED failure detector satisfied the accruement property.

Lemma 4.2 (Upper bound) *If process p is correct, then the suspicion level $sl_{qp}(t)$ is bounded.*

$$p \in correct(F) \Longrightarrow \exists SL_{max} : \forall t(sl_{qp}(t) \leq SL_{max})$$

Proof: If process p is correct, based on the system model, the process p always make progress in finite step after some global time GST, that means, the q eventually receives the heartbeat message from p. That is, there exists t_{max}, when the heartbeat message from p arrives at q. At any arbitrary time t, where $t \leq t_{max}$.

$$sl_{qp}(t_{max}) = e_d(t_{max}) = F(t_{max} - t_{last}) = 1 - e^{-\lambda(t_{max} - t_{last})}.$$

$$sl_{qp}(t) = e_d(t) = F(t - t_{last}) = 1 - e^{-\lambda(t - t_{last})}.$$

Based on Property 1, we know $sl_{qp}(t) \leq sl_{qp}(t_{max}) = SL_{max}$.

Theorem 4.1 ED FD implements an FD of class $\diamondsuit P_{ac}$, on condition that the system is in accordance with the system model defined in Chapter 2.1.

Proof From Lemma 4.1 and 4.2, we can make the following remarks: The proposed ED FD satisfied the class $\diamondsuit P_{ac}$ of accrual failure detector.

4.3 Performance evaluation

In this section, we evaluate and comparatively analyze the performance of ED FD, ϕ FD [18], Chen FD [30], and Bertier FD [16, 17] in a cluster, a wireless network, a wired local area network (wired LAN) and a wide area network (WAN).

The experiments are carried out with two computers. One (process p) sends messages periodically using UDP, and the other one (process q) receives the messages from process p. In every experiment, the heartbeat sending and arrival times are logged into the log files. These log files are replayed for each FD scheme to ensure the fairness of comparison. That means all the FDs are compared with the same experimental conditions: the same network model, the same heartbeat traffic, and the same experiment parameters (sending interval, slide window size, and communication delay, etc.). Thus, it provides an exactly fair experimental platform for every FD.

Interestingly, using the *traceroute* and *ping* commands, we observed that most of the traffic was actually routed with no network breakdowns. And we also used the *ping* command to check the RTT between the sender and the receiver. In addition, we found

that the average CPU load was nearly constant during the experiments and was also below the full capacity of the two computers.

Structure of the section The remainder of the section 4.4 is organized as follows: We first present the general experimental setup for the next experiments. Then in Section 4.4.1, we give the experiment in a Cluster group. Section 4.4.2 describes the experiment in a WiFi network. In Section 4.4.3, we conduct the experiment in LAN network. Section 4.4.4 gives a simple conclusion for the comparative analysis for the above four FDs.

Experimental setup In the experiments, each FD scheme used a slide window to save past samples to compute their estimations for the future. All the experiments for the four FDs used the same fixed window size ($WS = 1,000$). Furthermore, it is reasonable to analyze the sampled data only after the slide window is full, because the network is unstable during warmup period.

The main parameters are as follows: In order to find the best QoS and compare with the others, here $E_d \in [10^{-4}, 10]$ for ED FD; For ϕ FD, the parameters are set the same as in [18]: $\Phi \in [0.5, 16]$; For Chen FD, the parameters are set the same as in [30]: $\alpha \in [0, 10000]$; For Bertier FD, the parameters are set the same as in [16, 17]: $\beta = 1$, $\phi = 4$, $\gamma = 0.1$. In each experiment, the other basic experimental parameters of FDs are the same.

In these experiments, we focus on the following key performance metrics: mistake rate, query accuracy probability and detection time. In every experiment result, different values of mistake rate, query accuracy probability and detection time were obtained with the respective parameters.

4.3.1 Experiment in a Cluster group

This experiment was performed with two cluster nodes, the sending node (monitored) and the receiving node (monitoring) were located in a Cluster Group, at Japan Advanced Institute of Science and Technology (JAIST), in Japan. The two nodes transfer messages through a normal network connection.

Experiment setting: hardware/software/network

The two nodes were equipped with the same hardware and software: an Intel(R) Pentium(R) IV (CPU 2.80 GHz) and 512 Kb of cache size. The operating system was Fedora-Core 4 (Linux). The network is 1 Gb/s. The heartbeat messages were generated at a target rate of one heartbeat every 10 ms (the sending interval).

In order to make the experiment general, we re-did the experiments 5 times with the same code, the same environment, and the same parameters for each FD scheme, but with different experiment times. The experiment periods[2] were about 1 hour 1 minute, 1 hour 4 minutes, 5 hours 3 minutes, 7 hours 5 minutes, and 9 hours 3 minutes.

For the 5 experiments, here we show the detailed experimental data for one typical example (about 1 hour 4 minutes).

[2]Here we perform experiments for five times with five different experiment time, then we could get general experimental results. Based on the results, we compute the more general results to compare our FD to other existing FDs with 95% confidence level in this Cluster case.

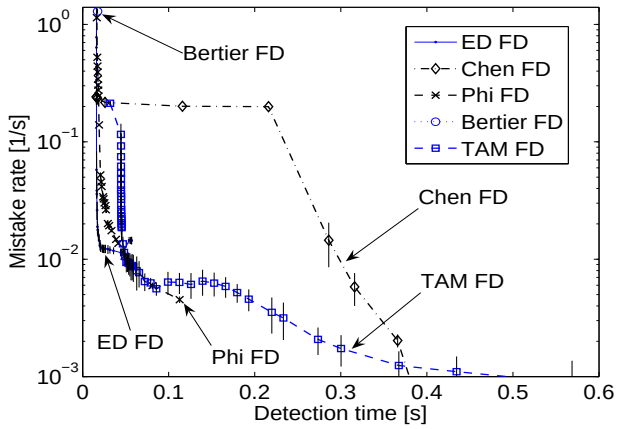


Figure 4.4: Mistake rate vs. detection time with 95% confidence level in a cluster.

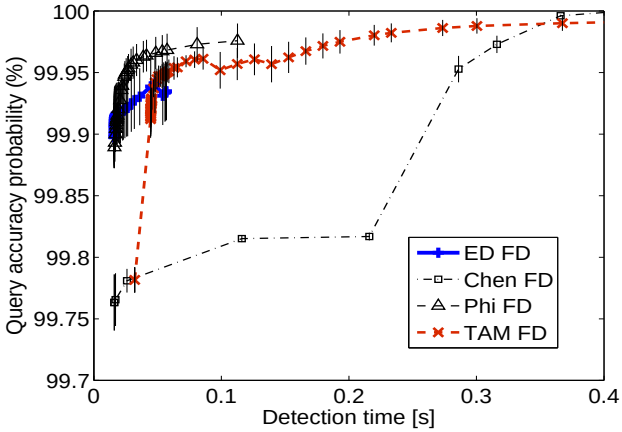


Figure 4.5: Query accuracy probability vs. detection time with 95% confidence level in a cluster.

Heartbeat sampling We got 229,453 samples, and no heartbeats were lost. The average sending rate actually measured was of one heartbeat every 16.015 ms (standard deviation: 1.709 ms; min.: 0.005 ms; max.: 191.977 ms).

Round-trip time In the experiment, we measured the RTT. The average RTT was 0.387 ms. The standard deviation was 0.756 ms with a minimum of 0.100 ms, and a maximum of 3.101 ms.

Experimental results

The experimental results for detection time, mistake rate and query accuracy probability with a 95% confidence level are shown in Figures 4.4-4.5. Figure 4.4 shows mistake rate comparison of FDs, where the vertical axis is on a logarithmic scale. We believe that the best values are located toward the lower left corner, for that means this FD provides short detection time and has a low mistake rate. Figure 4.5 shows query accuracy probability comparison of FDs, where the vertical axis is on a linear scale. And the best values are located toward the higher left corner, which means that the FD provides short detection time and has a high query accuracy probability.

In Figure 4.4, when $T_D < 48$ ms, the ED FD can obtain the lowest mistake rates among the five schemes with the same detection time; And when $T_D > 48$ ms, ϕ FD has the fewest mistakes. In Figure 4.5, ED FD, TAM FD and ϕ FD all obtain higher query accuracy probability than Chen FD with some junctions of ED FD and ϕ FD based on the confidence interval, all of ED FD, TAM FD and ϕ FD have good query accuracy probability.

In summary, the ED FD behaves a little better than the other four FDs in the more aggressive range (i.e., $T_D \leq 48$ ms). Chen FD behaves slightly better than the other four FDs in the more conservative range (i.e., $T_D \geq 327$ ms).

Here I analyze the possible reasons are as follows. Here the behavior of Bertier FD is plotted as a single point, because it has no tuning parameters. And obviously we found that Bertier FD doesn't perform very well compared with the others. The ϕ FD is more aggressive than Chen FD, and the ED FD is an improved detector from ϕ FD, it is more aggressive than ϕ detector.

4.3.2 Experiment in a Wireless network

These experiments involved two Mac computers and an AirMac Extreme base station. The two Mac computers were located in our lab, in JAIST, Japan. The AirMac extreme base station was used to build a private wireless LAN for two Mac computers.

Experiment setting: hardware/software/network

The two Mac computers were equipped with the same hardware and software: a PowerPC G4 (CPU 1.5 GHz) and 768 MB DDR SDRAM of Memory. The operating system was Mac OS X (Version 10.4.8). The heartbeat messages were generated at a target rate of one heartbeat every 100 ms (the sending interval).

Heartbeat sampling We got 435, 799 samples, (about 12 hours, 10 minutes and 34 seconds), and no heartbeats were lost. The average sending rate actually measured was of one heartbeat every 100.359 ms (standard deviation: 8.453 ms; min.: 0.0088 ms; max.: 792.418 ms).

Round-trip time By measurement, the average RTT is 2.829 ms. The standard deviation is 12.095 ms with a minimum of 1.598 ms, and a maximum of 231.678 ms.

Experimental results

We show results for detection time, mistake rate and query accuracy probability in Figures 4.6-4.7. Figure 4.6 describes the relationship of mistake rate and detection time among these four FDs on a logarithmic scale. Figure 4.7 shows the change of the query accuracy probability with different detection time on a linear scale.

In Figure 4.6, with $T_D < 273$ ms, the ED FD has the lowest mistake rate, and when $T_D > 505$ ms, Chen FD has the fewest mistakes; While, when 273 ms $< T_D <$ 505 ms, ϕ FD obtains the lowest mistake rates, except for some junctions of ϕ FD and ED FD.

In Figure 4.7, for the same detection time, the ED FD has the highest query accuracy probability of all. When $T_D > 470$ ms, Chen FD achieves higher query accuracy probability than ϕ FD. Except for that case, i.e., when $T_D < 470$ ms, ϕ FD achieves higher query accuracy probability than Chen FD.

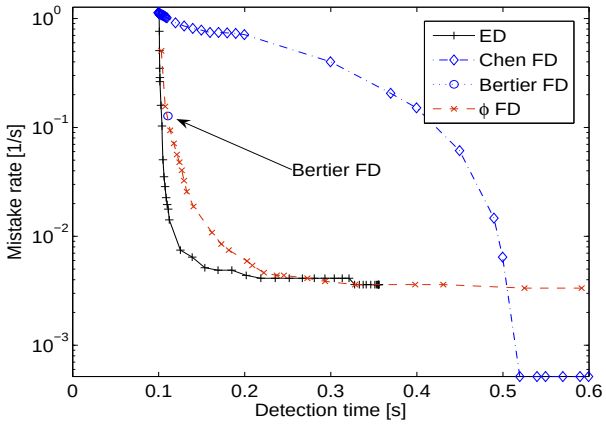


Figure 4.6: Mistake rate vs. detection time in a wireless network.

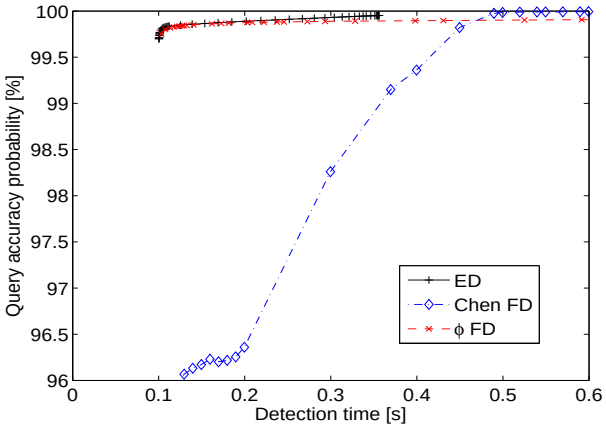


Figure 4.7: Query accuracy probability vs. detection time in a wireless network.

In summary, in this wireless case, ED FD has slightly better performance than the other three FDs in the aggressive range of FD. In the more conservative range (for examples, $T_D > 505$ ms), Chen FD behaves a little better than ϕ FD. While when 360 ms $< T_D < 470$ ms, ϕ FD has a little better performance than Chen FD.

We analyze the reason for the above results in this wireless case. It is obvious that the ϕ FD is more aggressive than Chen FD, and ED FD is an improved detector from ϕ FD. Furthermore, ED FD is more aggressive than ϕ detector.

4.3.3 Experiment in a LAN

This experiment also involved two Mac computers in a wired LAN. The sending host (monitored, on Floor 9) and the receiving host (monitoring, on Floor 6) were located at different labs in the same building, in JAIST, Japan.

Experiment setting: hardware/software/network

All the settings of the Mac computers are the same as those in the above wireless experiment. The two computers are connected through a single 100 Mbps Ethernet hub, with

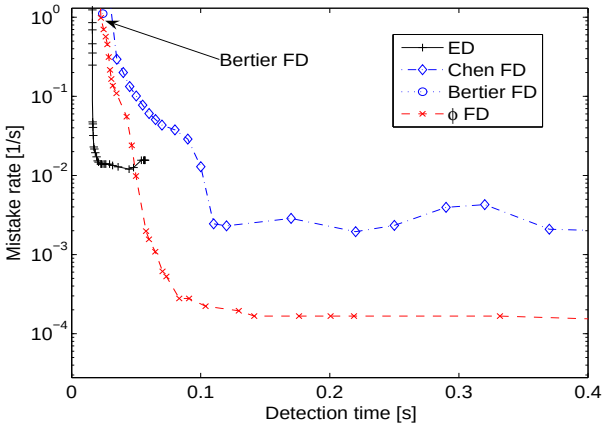


Figure 4.8: Mistake rate vs. detection time in a LAN.

no other systems attached.

Heartbeat sampling There were 1,797,026 samples received (10 hours, 15 minutes and 43 seconds), and no heartbeats were lost. The target of heartbeat sending interval is 20 ms. The average sending rate actually measured was of one heartbeat every 20.019 ms (standard deviation: 13.683 ms; min.: 3.099 μs; max.: 17, 950.169 ms).

Round-trip time The average RTT is 0.917 ms. The standard deviation is 0.146 ms with a minimum of 0.725 ms, and a maximum of 1.678 ms.

Experimental results

The results of the experiment are depicted in Figures 4.8-4.9. Figure 4.8 shows the relationship between detection time and mistake rate for the different FDs. In Figure 4.8, when $T_D < 49$ ms, ED FD has the lowest mistake rate, and when $T_D > 49$ ms, ϕ FD has lower mistake rate than other FDs. Figure 11 describes the relationship between detection time and query accuracy probability for each FD. In Figure 4.9, when $T_D < 54$ ms, ED FD has the highest query accuracy probability, after that period, ϕ FD first has the highest query accuracy probability with 54 ms $< T_D <$ 120 ms, and when $T_D > 120$ ms, Chen FD and ϕ FD have similar query accuracy probability, with many junctions of ϕ FD and Chen FD.

In summary for LAN, in the aggressive range: ED FD has a slightly better performance than the other three FDs. In the conservative range, ϕ FD behaves a little better than the other three FDs.

We analyze the reason for the above results in this LAN case. It is obvious that the ϕ FD is more aggressive than Chen FD. The experiment results demonstrate that ED FD is an improved detector from ϕ FD, and ED FD is more aggressive than ϕ detector.

4.3.4 Experiment in a WAN

All the above experiment environments are very stable, and there are no heartbeats lost. In order to further compare QoS of these four FDs, we carried out an experiment in WAN.

In this experiment, we use the exactly same trace files from the paper about ϕ FD [18], and these trace files are publicly available on our lab website [27]. Therefore, this provides

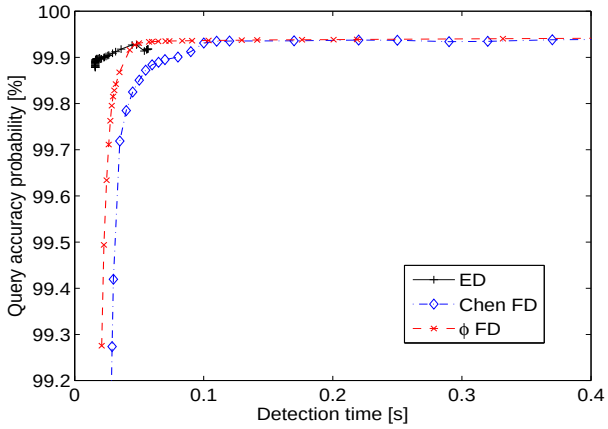


Figure 4.9: Query accuracy probability vs. detection time in a LAN.

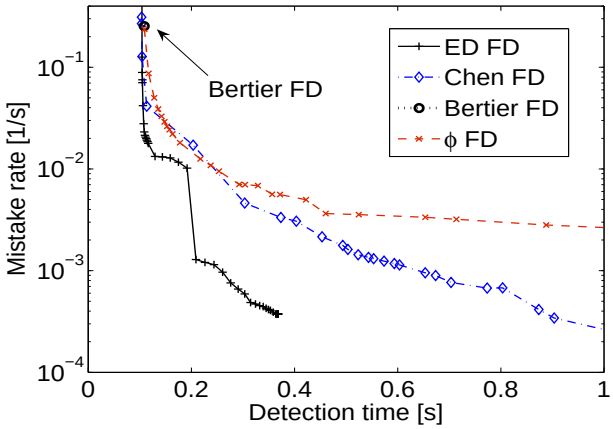


Figure 4.10: Mistake rate vs. detection time in a WAN.

a common ground for evaluating the performance of ED FD, Chen FD [30], Bertier FD [16, 17] and ϕ FD [18].

Experiment setting: hardware/software/network

In detail, the trace files and relevant data were gotten from the following experiment setting.

This experiment involves two computers: one was located at the Swiss Federal Institute of Technology in Lausanne (EPFL), in Switzerland. The other one was located in JAIST, Japan. The two computers communicate through a normal intercontinental Internet connection. The two computers have the same equipment and the same operating systems as those in [18]. By analyzing the trace files, we found the average CUP load for the sending host and the receiving host were 1/67 and 1/22, respectively. So they were below the full capacity of the computers.

Heartbeat sampling The experiment started on April 3, 2004 at 2:56 UTC, and finished on April 10, 2004 at 3:01 UTC. During the one week experiment period, the heartbeats were generated at a target rate of one heartbeat every 100 ms (the sending interval). The average sending rate actually measured was one heartbeat every 103.501

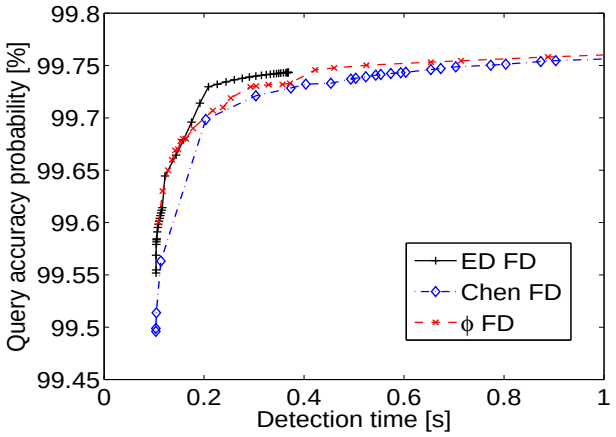


Figure 4.11: Query accuracy probability vs. detection time in a WAN.

ms (standard deviation: 0.189 ms; min.: 101.674 ms; max.: 234.341 ms). Furthermore, 5, 845, 713 heartbeat messages were sent out, while only 5, 822, 521 were received, so message loss rate was about 0.399 %.

By checking the traces files more closely, one found the messages losses were because of 814 different bursts. The majority of total bursts were short length bursts. While the maximum burst-length was 1, 093 heartbeats (only one), it lasted about 2 minutes. Furthermore, most of the heartbeats was not directly between Asia and Europe, but actually, routed through the United States.

Round-trip time The average RTT is 283.338 ms, with a standard deviation of 27.342 ms, a minimum of 270.201 ms, and a maximum of 717.832 ms.

Experimental results

The results of the experiment are depicted in Figures 4.10-4.11. Similar to the other experiments, we first give the relationship between detection time and mistake rate, as shown in Figure 4.10. In Figure 4.10, with the same detection time, ED FD obtains the lowest mistake rate among the four FDs, except for several initial junctions of ED FD, ϕ FD and Chen FD. When 148 ms $< T_D <$ 243 ms, ϕ FD obtains a lower mistake rate than Chen FD; except for that, i.e., when $T_D >$ 243 ms, Chen FD has the lower mistake rate compared with ϕ FD. The query accuracy probability comparison of FDs in a WAN is shown in Figure 4.11. From Figure 4.11, we find: when $T_D <$ 160 ms, ED FD and ϕ FD have the similar query accuracy probability with the same detection time. While, when $T_D >$ 160 ms, it is clear that ED FD has higher query accuracy probability than ϕ FD. Furthermore, Chen FD has a little lower query accuracy probability than ED FD and ϕ FD. In summary, in the aggressive range of FD: ED FD behaves a little better than the other three FDs in terms of short detection time, low mistake rate and high query accuracy probability.

We analyze the reason for the above results in this WAN case. It is obvious that the ϕ FD is more aggressive than Chen FD. The experiment results demonstrate that ED FD is an improved detector from ϕ FD, and ED FD is more aggressive than ϕ detector. Furthermore, in the aggressive range, ED FD have best performance to catch the dynamic network in the four detectors.

4.3.5 Comparative analysis of the four FDs

From all the above experimental results, the following remarks can be made: The four kinds of experiments demonstrate that ED FD is an effective improvement over ϕ FD in terms of short detection time, low mistake rate and high query accuracy probability. In stable cases (no message loss and small variability of delay), such as in a cluster group, wireless system, and wired LAN, ED FD has slightly better performance in the aggressive range of FD. Especially, in unstable network environment (larger variability of heartbeat delay and some message loss), such as WAN, ED FD obviously performs better than the others in aggressive case.

In all, for the applications that need failure detection to be timely and highly accurate, ED FD is an efficient choice.

4.4 Conclusion

Failure detection is a fundamental issue for supporting dependability in distributed systems. For providing corresponding QoS for a user, Hayashibara and Defago et al. [18] developed a ϕ FD, which assumes that the inter-arrival times follow a normal distribution, and computes a value ϕ with a scale that changes dynamically to match recent network conditions (i.e., this FD outputs suspicion level on a continuous scale, instead of traditional binary information. This is different from the other FDs). From the statistic analysis of the experimental results (see 4.1.2), we found the normal distribution is not a reasonable assumption for the approximation of the heartbeat inter-arrival time, especially in large scale distributed networks or unstable networks.

Therefore, this paper proposes a novel estimation of the distribution for the inter-arrival time, called the exponential distribution failure detector (ED FD), as an extension of ϕ FD [18]. Extensive experimental results have demonstrated that the proposed ED FD outperforms the existing adaptive failure detectors.

The contribution of this paper is as follows. Firstly, an optimized accural FD, called ED FD, is proposed. Secondly, we have comparatively evaluated our failure detection scheme with existing schemes (Chen FD [11], Bertier FD [12, 16], and ϕ FD [13]) by extensive experiments in four cases: a cluster group, LAN, and wide area network (WAN), and wireless network.

From all the above experimental results, the following remarks can be made: The four kinds of experiments demonstrate that ED FD is an effective improvement over ϕ FD in terms of short detection time, low mistake rate and high query accuracy probability. (1) In stable cases (no message loss and small variability of delay), such as in a cluster group, wireless system, and wired LAN, ED FD has slightly better performance in the aggressive range of FD. (2) Especially, in unstable network environment (larger variability of heartbeat delay and some message loss), such as WAN, ED FD obviously performs better than the others in aggressive case. The experimental results have shown the properties of the different adaptive FDs, and demonstrated that the proposed ED FD outperforms the existing FDs in terms of short detection time, low mistake rate and high query accuracy probability. In all, for the applications that need failure detection to be timely and highly accurate, ED FD is an effective choice.

The ED FD is well-designed for failure detection in the aggressive range (for example, some applications on chemistry) by observing the experiment results. While for the

conservative range (longer detection time and lower mistake rate, for example, the application on planes), it does not work well. It will be perfect if ED FD can work well both in aggressive range and in conservative range. Therefore, we are working on an extension of the ED FD by analyzing the effects of message losses and other possible factors on mistake rate of failure detector in future.

Chapter 5

Performance Evaluation of Kappa Failure Detector

Hayashibara et al. [18] proposed an earlier implementation of an accrual failure detector, called the Phi failure detector (ϕ FD). Experiments showed that the failure detector performed well in an aggressive range of failure detector in most environments considered. While, when trying to deploy this failure detector, it unfortunately turned out that rounding errors prevent it to be tuned for the very conservative case. This is not like simpler or elegant failure detection implementations proposed in the literature, such as the failure detector of Chen et al. [30]. The latter behaves better as a conservative failure detection than as an aggressive one.

Depending simply on the threshold of failure detector, κ failure detector presented in [3] develops an accrual failure detector that can be gradually tuned from a very aggressive to a very elegant one (has good performance in both aggressive and conservative range), it actually succeeds in addressing the shortcomings of the above FDs. On the one hand, κ FD allows for a spectrum of settings from a very aggressive behavior (i.e., short latency at the expense of accuracy) to a very conservative one (i.e., good accuracy at the expense of latency); On the other hand, it retains the 2 architectural benefits of accrual failure detection, thus supporting various usage patterns for distributed applications, and the formal properties of accrual failure detectors are also inherited naturally.

The κ failure detector [3] is an instance of accrual failure detector [19], outputting suspicion information on a continuous scale rather than the classical *trust or suspect* model. This allows for a clearer separation between the monitoring of the system and the interpretation of suspicion information by applications. In particular, since different recovery actions incur widely different performance costs, it is often difficult (if not impossible) for the failure detector to decide when a suspicion must be generated, without knowing considerable details about the application that takes the recovery action. Accrual failure detectors, including the κ failure detector, were designed to address this issue in particular. Hayashibara in [3] gave the original idea and definitions about the κ FD. While the performance evaluation and analysis is not enough. Therefore, a question then arise: what is the performance characteristic of κ FD compared with the existed failure detectors?

To answer this question, this chapter evaluates the performance of the κ failure detector and compares the performance of κ failure detector with that of the other state-of-the-art failure detectors. We conducted experiments in a number of typical settings, ranging from a dedicated Cluster group, Wired local area networks (LAN), Wireless LAN (WiFi),

to wide area networks (WAN). Our results demonstrate that the κ failure detector can adequately support both an aggressive and conservative range of failure detection.

The remainder of the chapter is organized as follows. Section 5.2 presents the experiment settings, experiment overview, and experimental results. In Section 5.3, we give the concludes of the κ-failure detector.

5.1 Experimental evaluation

We have analyzed the behavior of the implementation of the κ-failure detector over a large collection of environments, of which we present the most relevant ones. We begin by describing the various environments, then the methodology used for comparison, and finally we present and discuss the results obtained.

The main goal of our experiments was to observe the performance of the κ failure detector to be configured well for aggressive failure detection as well as conservative one. To do so, we compare κ-FD against Chen-FD, Bertier-FD, and ϕ-FD.

5.1.1 Experiment settings

In each environment, our experiment involves two computers, with one periodically sending heartbeat to the other one for an arbitrarily long period. The arrival and sending time of the heartbeats is logged at both ends and stored to a file. The logged arrival time is used to replay the execution. The logged sending time is used only for statistics. A low-frequency ping process ran in parallel with the experiment as a means to obtain a rough estimation of the round-trip time.

The experiment environments are described below, and the corresponding statistics are summarized in Table 5.2. All heartbeat messages used the UDP/IP protocol.

Environment 1 (Cluster). This set was conducted on a small cluster in our laboratory, with a private network dedicated to the experiment. The experiment occurred on September 23, 2006, and was set to last for one hour, with a target sending period of 10ms. The effective period was 16.015 ms, mostly due to clock granularity and imprecise OS scheduling. The network was made of one Gigabit Ethernet switch, and the nodes were two identical machines with Intel Pentium IV 2.8 GHz with 1.5GB of RAM, and running Linux kernel 2.6.15 (Fedora core 4).

Environment 2 (LAN). This set used a subnet of our university network, with two computers connected on a 100Mbps Ethernet with moderate activity. The experiment took place on November 11, 2006 and lasted for 10 hours, with a target sending interval of 20ms. The two machines used were two Apple Macintosh PowerBook G4 at 1.5GHz with 768MB of RAM, and running Mac OS X version 10.4.8.

Environment 3 (WiFi). This set used the same two machines as the LAN set, but connected through a WiFi (802.11g) network dedicated to the experiment. The WiFi network used a base station and shared the air with at five other WiFi networks. The experiment took place on July 6, 2006 which is still an active period in the year at our institution. The experiment lasted for one hour.

Wide-area network (WAN) experiments were conducted on the PlanetLab (http://www.planet-lab.org/), using nodes located in USA, Europe (Germany), Japan, and China (Hong Kong). The locations and hostnames are summarized in Table 5.1. Each WAN

Table 5.1: Summary of the WAN experiments

WAN case	Sender	Sender-hostname	Receiver	Receiver-hostname
WAN-1	USA	planet1.scs.stanford.edu	Japan	planetlab-03.naist.ac.jp
WAN-2	Germany	planetlab-2.fokus.fraunhofer.de	USA	planet1.scs.stanford.edu
WAN-3	Japan	planetlab-03.naist.ac.jp	Germany	planetlab-2.fokus.fraunhofer.de
WAN-4	China	planetlab2.ie.cuhk.edu.hk	USA	planet1.scs.stanford.edu
WAN-5	China	planetlab2.ie.cuhk.edu.hk	Germany	planetlab-2.fokus.fraunhofer.de

Table 5.2: Summary of the experiments: statistics

Experiment	Heartbeats		Heartbeats period			RTT
case	total (#msg)	loss rate	send (Avg.)	receive (Avg.)	receive (stddev)	(Avg.)
Cluster	229, 453	0%	16.015 ms	16.015 ms	1.709 ms	387 s
LAN	1, 797, 026	0%	20.012 ms	20.019 ms	13.683 ms	917 s
WiFi	39, 676	0%	100.353 ms	100.359 ms	8.453 ms	2.829 ms
WAN-1	6, 737, 054	0%	12.825 ms	12.83 ms	14.892 ms	193.909 ms
WAN-2	7, 477, 304	5%	12.176 ms	12.206 ms	19.547 ms	194.959 ms
WAN-3	7, 104, 446	2%	12.21 ms	12.235 ms	4.768 ms	189.44 ms
WAN-4	7, 028, 178	0%	12.337 ms	12.346 ms	22.918 ms	172.863 ms
WAN-5	7, 008, 170	4%	12.367 ms	12.94 ms	16.557 ms	362.423 ms

experiment was set to last for about 24 hours, with a target heartbeat interval set to 10 ms.

Environment 4 (WAN-1). This set was from USA to Japan, starting from March 12, 2007. The effective heartbeat interval was 12.825 ms. A total of 6, 737, 054 heartbeats were sent, with a loss rate of 0%. The mean heartbeat arrival time was 12.83 ms (thus showing a slight clock drift) with a standard deviation of 14.892 ms. The average round-trip time was 193.909 ms.

Environment 5 (WAN-2). This set was from Germany to USA, starting from March 8, 2007.

Environment 6 (WAN-3). This set was from Japan to Germany, starting from March 6, 2007.

Environment 7 (WAN-4). This set was from Hong Kong (China) to USA, starting from March 10, 2007.

Environment 8 (WAN-5). This set was from Hong Kong (China) to Germany, starting from March 11, 2007.

As a final note, we have monitored the CPU load average on the machines during the whole period of each experiment. We observed that the load was nearly constant throughout, and that it was always well below the capacity of the machines.

5.1.2 Experiments overview

The experiment was done in two phases. First, in each of the environment described above, we have logged the heartbeat arrivals on the monitoring computer. Second, in each environment, we have replayed the entire sequence of heartbeat to each of the failure detectors, and for many values of their configuration parameters (i.e., safety margin for Chen-FD, or threshold for Phi-FD and κ-FD). For each execution, after discarding some initial period, we have measured the three QoS metrics for the entire execution: detection time, mistake rate, and query accuracy probability.

Replaying the exact same sequence of heartbeat arrivals to each of the failure detectors allows for a fair comparison since, for each experimental setting, they use exactly the same input.

5.1.3 Experimental results and discussions

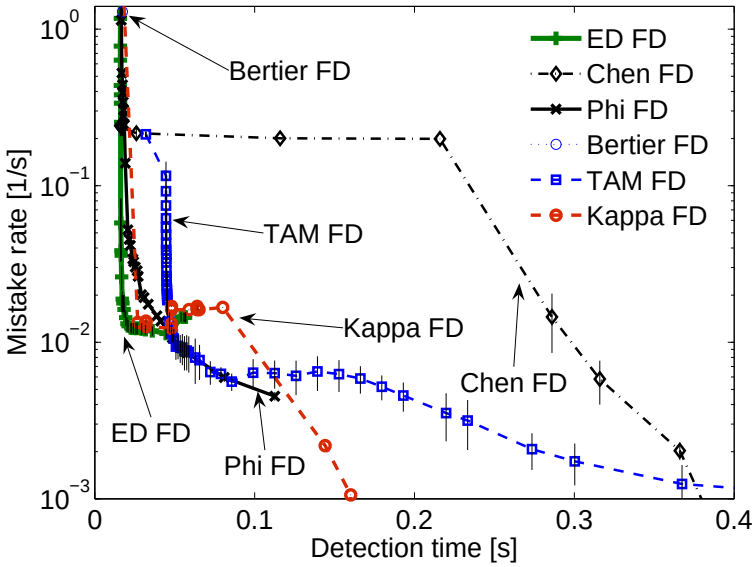
It is not easy to compare parametric failure detectors. Because, depending on the value set for their parameter, their behavior can be completely different. A common mistake is to set some arbitrary values for the parameters and then compare two parametric failure detectors based on the measured detection time and accuracy. This almost always leads to the erroneous conclusion that one is better for detection time while the other provides higher accuracy.

In contrast, we use the approach developed when conducting experiments for the ϕ FD [18]. The idea of the approach is based on the following question: given a set of QoS requirements, can the failure detector be parameterized to match these requirements? To answer this question, we consider a space of QoS defined by the detection time on one axis and an accuracy metric (e.g., mistake rate) on the other axis. Then, we measure the area covered by the failure detector when we vary its parameter from a highly aggressive behavior to a very conservative one. The area covered by a failure detector is the area that corresponds to a set of QoS requirements that can possibly be matched by that failure detector (i.e., on the top-right of each point of the curve), provided that it is tuned appropriately.

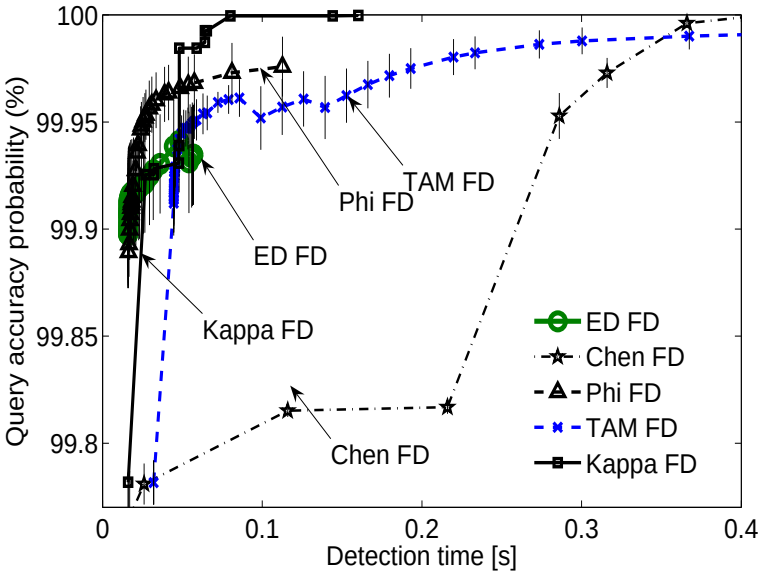
To compare two parametric failure detectors FD_A and FD_B, one must compare the area that they respectively cover. Indeed, any area covered by FD_A and not by FD_B corresponds to some QoS requirements that can possibly be satisfied by FD_A, but not by FD_B, regardless of how well one tries to set its parameters.

For each environment, we represent the characteristics of each of the failure detectors (Chen-FD, Bertier-FD, Phi-FD, and κ-FD) on two graphics. The first one relates the detection time to the mistake rate (on a log scale), while the second one relates the detection time to the query accuracy probability. In either case, points toward the bottom-left corner of the graphics are best, while those located toward the top-right corner are worse. Points located near the lower-right corner of the graphic show the respective failure detector in conservative mode (good accuracy but longer detection time), while those near the top-left corner correspond to an aggressive mode (short detection time but poorer accuracy).

The discussion of the results focuses on the mistake rate because that is where the differences between failure detectors are most obvious. Nevertheless, for the sake of completeness, we also show the graphics for the query accuracy probability and discuss them when appropriate.

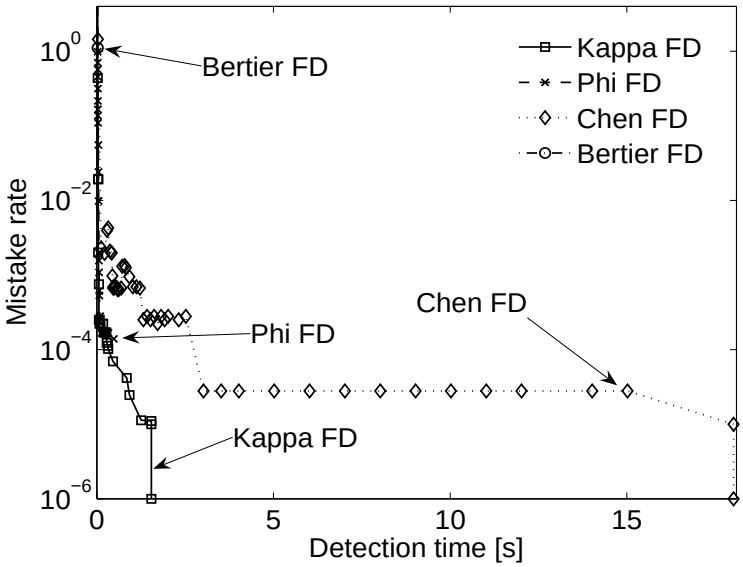


(a)

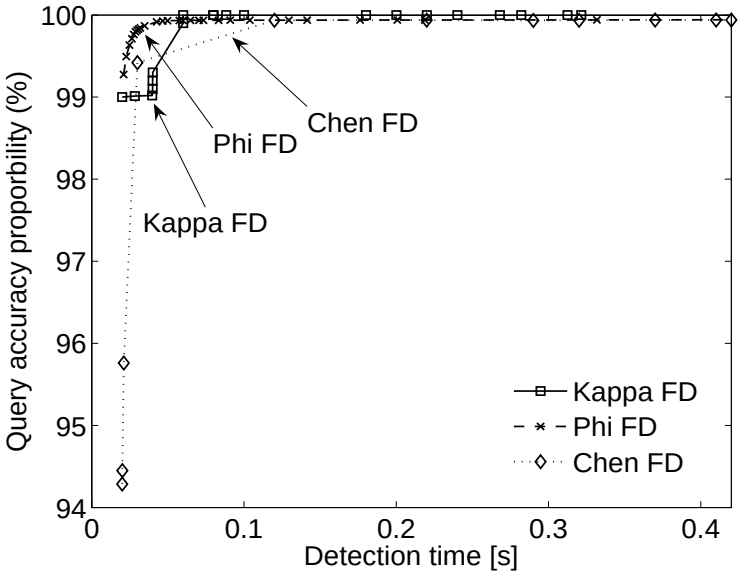


(b)

Figure 5.1: Cluster: (a) Mistake rate vs. detection time, (b) Query accuracy vs. detection time.

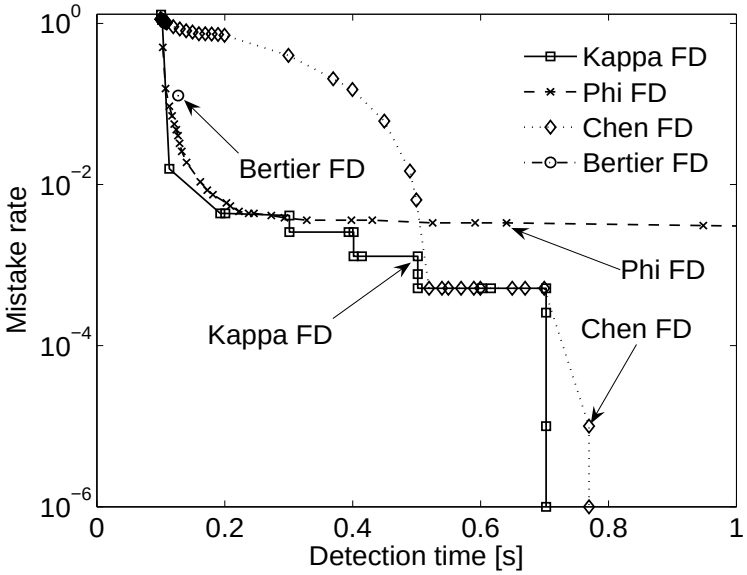


(a) Mistake rate vs. detection time

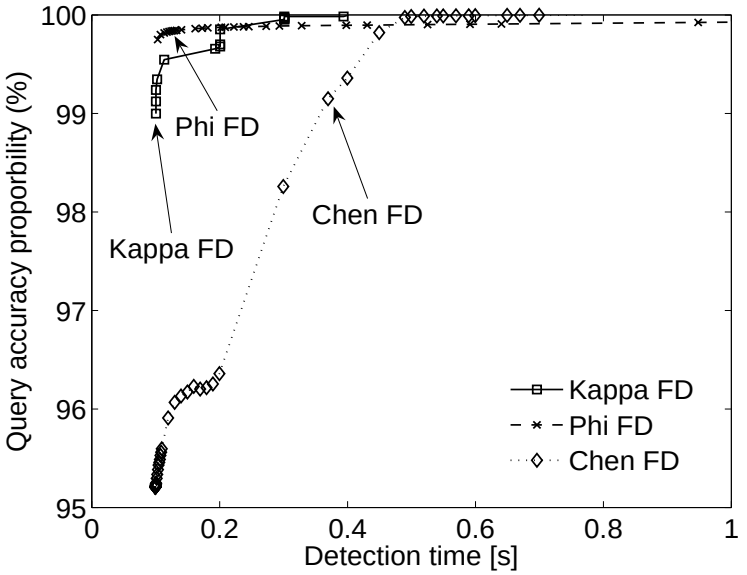


(b) Query accuracy vs. detection time

Figure 5.2: LAN: (a) Mistake rate vs. detection time, (b) Query accuracy vs. detection time.



(a) Mistake rate vs. detection time



(b) Query accuracy vs. detection time

Figure 5.3: WiFi: (a) Mistake rate vs. detection time, (b) Query accuracy vs. detection time.

Cluster The results for the cluster environment are show in Figure 5.1. This is the set for which the arrivals time are the most predictable, due to the almost complete absence of external influence on network delays. This is also the environment in which the choice of the failure detector makes the biggest difference.

Figure 5.1(a) shows that Bertier-FD (which is not parametric) behaves aggressively, with about one suspicion generated per second. We find this behavior consistently in every environment.

For ϕ-FD, the curve stops at a detection time of 100ms and mistake rate a little below $10^{-2}/s$. This is because rounding errors made it impossible to compute the suspicion level.

For Chen-FD and κ-FD, in the very aggressive range, both start with characteristics similar to that of Bertier-FD. However, this quickly changes and, for the same detection time, κ-FD generates almost an order of magnitude less suspicions than Chen-FD. Then, κ-FD no longer generates any suspicion (in this run) with a detection time of about 160 ms. In comparison, Chen-FD reaches zero suspicion with a detection time of about 410 ms.

We can find that ED FD gets best performance in the six schemes in the aggressive range (here about the detection time is below 0.03 s). While in the conservative range (here $T_D > 0.4$ s), the Kappa FD and Chen FD get better performance than other schemes, while Kappa FD is not too bad for the aggressive range. So it is ubiquitous and suitable for the general applications, which are not clear about the application environment.

The results are similar for the query accuracy probability. To sum up, the κ-FD provides good performance in a cluster-like environment, both for aggressive and conservative failure detection. At the same time, it overcomes the problem of rounding errors that plagues the Phi-FD.

LAN The results obtained in the LAN are depicted in Figure 5.2. Results are similar to those found for the cluster. The main difference is that it takes considerably longer to the Chen-FD to reach zero suspicions (with a detection time of about 13 s). This is explained by the fact that a single very late heartbeat makes it necessary to set the safety margin to a very high value. In contrast, κ-FD does adapt to some extent to changing network conditions and thus can keep the average detection time low.

WiFi The results obtained in the wireless environment are depicted on Figure 5.3.

For Phi-FD, the curve decreases sharply at first, but then remains flat with a mistake rate slightly under $10^{-2}s^{-1}$. Again, this is almost surely due to rounding errors making it impossible to compute the suspicion level accurately enough.

Starting in the aggressive range, Chen-FD and κ-FD behave very differently initially. While κ-FD quickly decreases to a mistake rate lower than $10^{-2}s^{-1}$, Chen-FD keeps a high mistake rate. In a more conservative range, both failure detectors behave similarly.

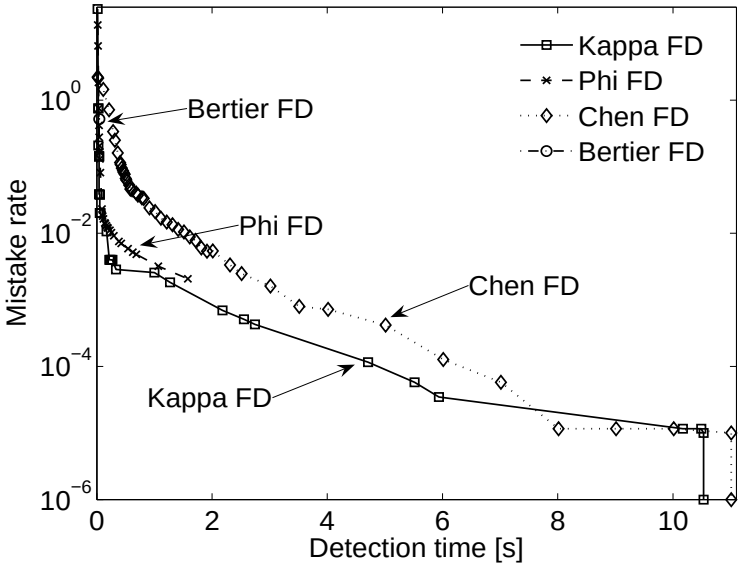
The difference between the failure detectors is even more obvious when considering the query accuracy probability, as depicted on Figure 5.3(b).

WAN-x The results obtained on the PlanetLab are depicted in Figures 5.4-5.8.

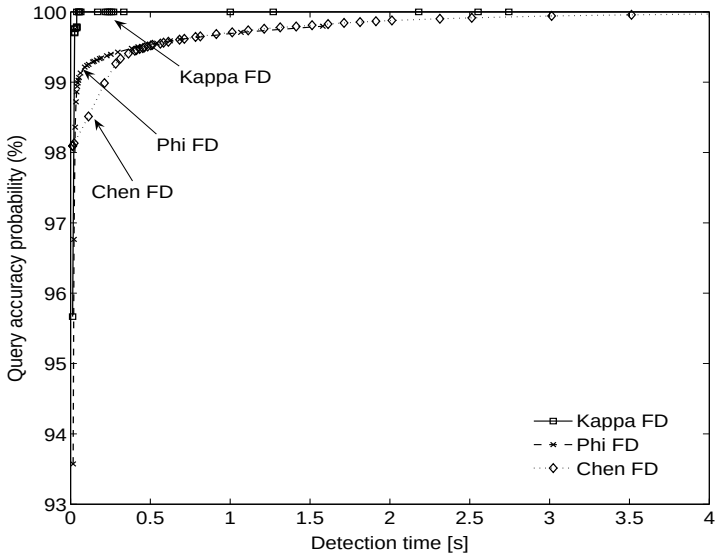
We first observe the results obtained in the set WAN-1, between USA and Japan. Bertier-FD behaves as an aggressive failure detector also in this setting. For Phi-FD, the rounding errors prevent to compute points in the conservative range, and the curve stops with a detection time of 2 s and a mistake rate near 10^{-3}.

The difference between Chen-FD and κ-FD is much less clear than with the previous experiments. This is mostly due to the fact that different strategies can only do little to

overcome the inherently high uncertainty of the environment. Nevertheless, after starting with similar characteristics in the very aggressive range, κ-FD generates less suspicions than Chen-FD for some range. Then, both failure detectors behave comparably, or better in the conservative range. A similar behavior can be observed in the different experimental settings. It is most obvious in the experiment WAN-3 (between Japan and Germany). The difference is least obvious in experiment WAN-2 (between Germany and USA) for which the two failure detectors exhibit nearly identical characteristics.

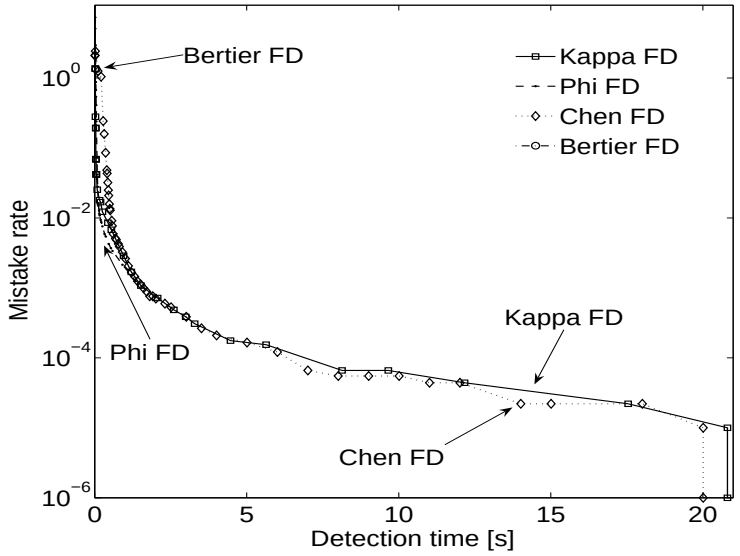


(a) Mistake rate vs. detection time

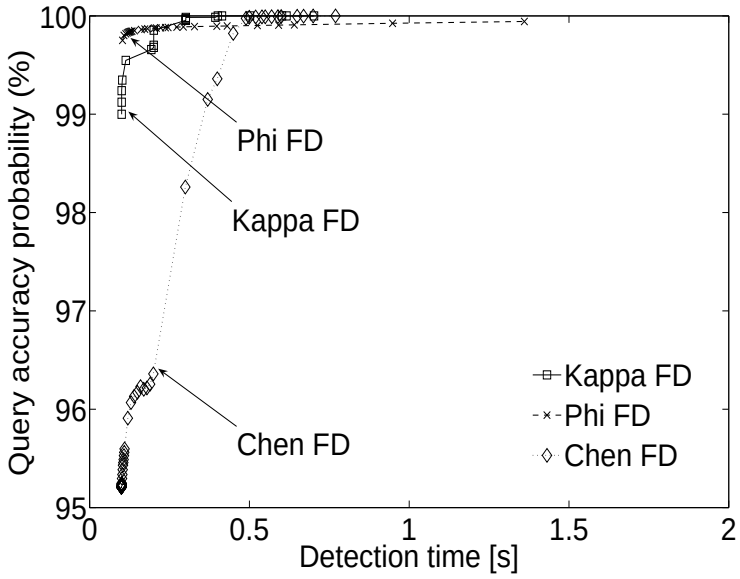


(b) Query accuracy vs. detection time

Figure 5.4: WAN-1, USA⟶Japan: (a) Mistake rate vs. detection time, (b) Query accuracy vs. detection time.

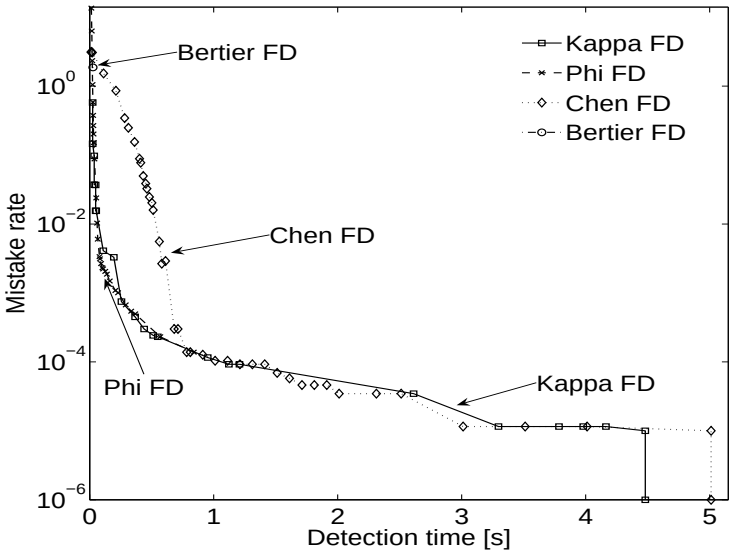


(a) Mistake rate vs. detection time

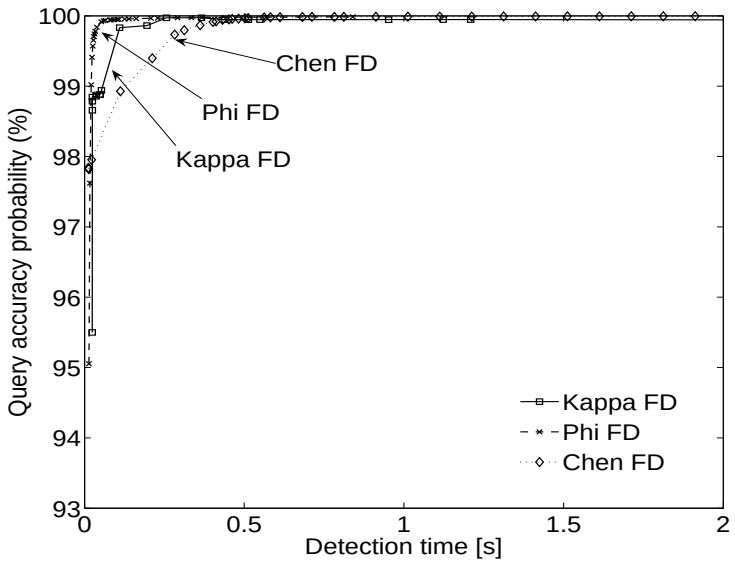


(b) Query accuracy vs. detection time

Figure 5.5: WAN-2, Germany⟶USA: (a) Mistake rate vs. detection time, (b) Query accuracy vs. detection time.

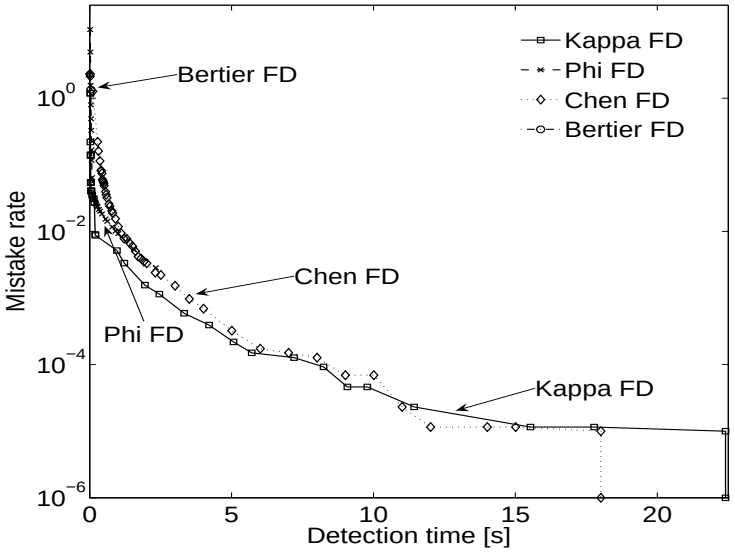


(a) Mistake rate vs. detection time

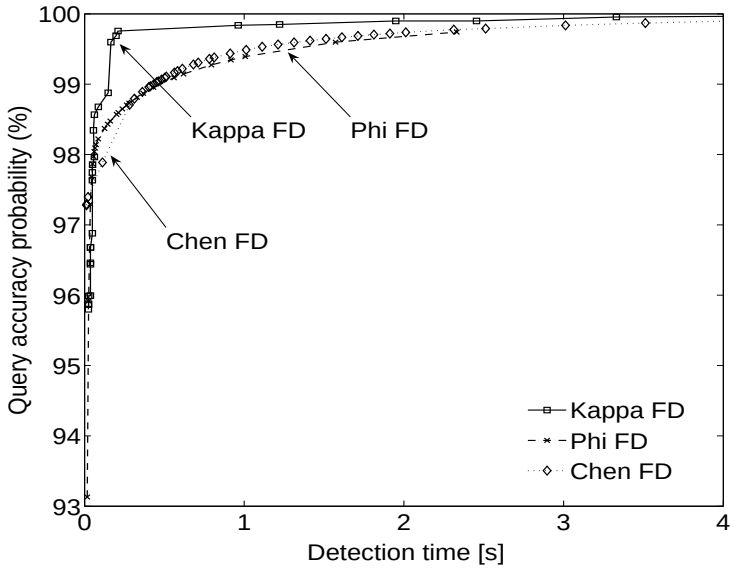


(b) Query accuracy vs. detection time

Figure 5.6: WAN-3, Japan⟶Germany: (a) Mistake rate vs. detection time, (b) Query accuracy vs. detection time.

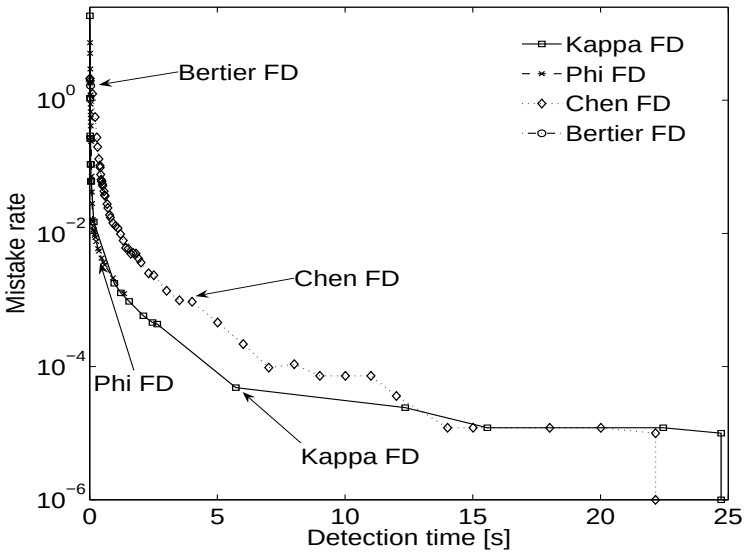


(a) Mistake rate vs. detection time

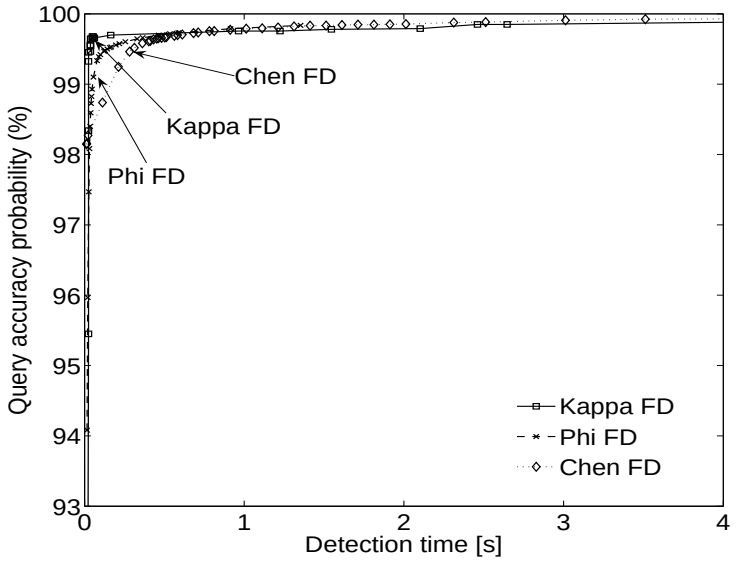


(b) Query accuracy vs. detection time

Figure 5.7: WAN-4, China⟶USA: (a) Mistake rate vs. detection time, (b) Query accuracy vs. detection time.



(a) Mistake rate vs. detection time



(b) Query accuracy vs. detection time

Figure 5.8: WAN-5, China⟶Germany: (a) Mistake rate vs. detection time, (b) Query accuracy vs. detection time.

5.2 Conclusion

We evaluate the performance of κ failure detector [3] in a variety of network environments. A lot of experimental results have shown that the κ failure detector can indeed provide good performance when set to an aggressive failure detector, generating nearly one order of magnitude less wrong suspicions than other state-of-the-art failure detector implementations. At the same time, it can be set to provide good conservative failure detection, when compared to other failure detectors. The experiments cover most typical execution environments found at present, ranging from a dedicated cluster to wireless and wide area networks.

The strongpoint of κ failure detector is that: as an instance of accrual failure detector, it can provide different "hint" information for different application users, that is, κ failure detector can satisfy different users' requirements using the same FD module. While, the following weakness exists in κ failure detector: the thresholds provided by application users are related closely with the different contribution functions. The application users should know the inner contribution function in failure detector and provide a suitable threshold; otherwise, the desired requirement can not be satisfied. That is, the parameter used by failure detector should be given a suitable one by hand. Therefore, it is necessary to find an effective method to self-tune corresponding parameters to satisfy the requirements of users not by hand.

Except for Self-tuning FD, this is true for all FD implementations. Namely, they can be tuned, but the application would need to know details about the implementation because the parameters are related to measure that are interpretable by the applications. In contrast, QoS is understandable by applications, but applications would need a way to map desired QoS to some parameter values. We note that self-tuning could be applied in principle to accrual FD by having a self-tunable threshold function.

Chapter 6

Self-tuning Failure Detection

There are three statements that failure detection should need some adaptive self-tuning. (1) Accrual failure detection delegates the tuning to the applications by matching QoS requirements with parameters. Thus, the question of tuning is not expressed. (2) If the QoS can be matched by parametric failure detection, then the QoS strongly relies on proper tuning. (3) Chen FD [30] can provided different QoS services to satisfy many different QoS requirements of users by hand. The QoS tuning depends on stable network environments. Therefore, adaptive self-tuning is very important for an effective failure detection.

How can we set the parameters to provide corresponding QoS for a user? In Chen FD [30], it can give a list about the possible QoS services for users based on the different parameters of failure detector. If a user require a certain QoS service, ones can match the QoS requirement from the list of QoS services, and then choose the corresponding parameters of failure detector by hand. Obviously, it is not applicable for actual engineering applications, and there are no any self-tuning scheme presented. That is, for a given QoS requirement, how to tune by itself the parameters of failure detectors to satisfy such requirement? Therefore, in this chapter, we address this problem and present a self-tuned failure detector (SFD). SFD optimizes the paper [30] and self-tunes its parameters to satisfy the requirement of different users.

This chapter compares QoS of several adaptive failure detection schemes (Chen FD [30], Beriter FD [16, 17], and Phi FD [18]). Also, we introduce an optimization over the existing methods, called SFD, which significantly improves QoS, especially in the design of real-time pragmatic systems. Experiment has been carried out over a wide area network (WAN) case. The experimental results have shown the properties of the adaptive FDs, and demonstrated that the proposed SFD is a pragmatic detector. SFD can adjust its parameters by itself based on the dynamic network environment to let the output QoS satisfy many different QoS requirements of users.

The remainder of the chapter is organized as follows: In Section 6.2, the system model and self-tuning failure detection are introduced, including the theory analysis and the implementation of SFD. In Section 6.3, we carry out the experiment in WAN, and analyze the performance evaluation of SFD. Finally, we conclude our work in Section 6.4.

6.1 The self-tuning failure detector scheme

System model and definitions are shown in Chapter 2.1. Here the user p hopes Failure Detector (FD) in process q detects p with a certain Quality of Service (QoS) requirement. And the FD in q can adjust its parameters by itself to satisfy the QoS (see Figure 6.1).

6.1.1 The theory of self-tuning failure detector

We develop the implementation of SFD [3] in the view of performance evaluation. It is an accrual failure detector in that it associates a real value representing a suspicion level to each process, rather than the traditional binary information (trust vs. suspect).

In actual application of failure detector, the users always hope that the used failure detector can detect its components or other users with a certain quality of service requirement. In other words, there are some bounds for quality of service requirement (see Figure 6.2), for example, detection time is not more than the target value $\overline{T}$ ($T_D \leq \overline{T}$) and mistake rate is not more than the target value $\overline{MR}$ ($MR \leq \overline{MR}$). On the one hand, conservative failure detection ensures that wrong suspicions are rare, while only at the expense of the detection time; On the other hand, aggressive failure detection ensures a short detection time, while results in a higher probability of generating wrong suspicions. The user doesn't expect to wait for the information of failure detector too long, and neither expect high probably wrong information from failure detector. Therefore, the used failure detector should adjust its parameters to satisfy the quality of service of the user.

There are several parameters to describe the output QoS of FD. Here we focus on the three main parameters, i.e., detection time T_D, mistake rate MR, and query accuracy probability QAP. Furthermore, we set the target QoS as $\overline{QoS}$. And if the output QoS of FD satisfies the $\overline{QoS}$, we have $QoS < \overline{QoS}$.

In Figs. 6.1 and 6.2, $\overline{QoS}$ is the target QoS for the heartbeats. If the output QoS of the FD does not satisfy the $\overline{QoS}$, i.e., $QoS > \overline{QoS}$. Then the feedback information ($QoS - \overline{QoS}$) is returned to FD. Based on the feedback information, FD adjusts its parameters (for example, timeout T_o for the timeout-based schemes). Then eventually FD can satisfy the $\overline{QoS}$, if there is a certain field for FD, where FD can satisfy the $\overline{QoS}$. Otherwise, FD give a response: This FD can not satisfy the $\overline{QoS}$ for upper application.

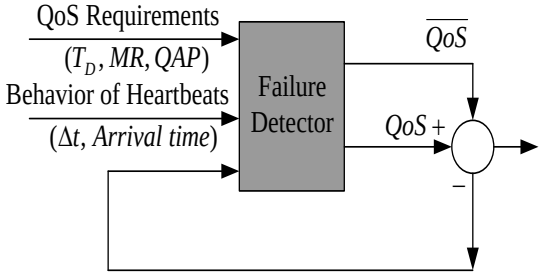


Figure 6.1: Basic theory 1 of selftuing failure detection, $\overline{QoS}$ is the target QoS for the heartbeats.

T_D, MR, and QAP in QoS are the performance parameters for a period experiment, not for a time slot. Then the output QoS of FD is based on all the former time period. Actually, we can not adjust the parameters of FD based on multiple feedbacks to let the improved output QoS of FD satisfy the $\overline{QoS}$ in a time slot. While we can adjust the parameters of FD based on multiple feedbacks to get the improved output QoS of FD in a time slot. Thus, we improve output QoS of FD gradually. For the former time is limited, we assume the experiment time is long enough to let output QoS of FD satisfy the $\overline{QoS}$ for upper application.

6.1.2 The implementation of self-tuning failure detector

Here we combine the Chen-FD [30] scheme, and adjust the predictive next freshness point $\tau_{(k+1)}$ based on the feedback information. So we have

$$\tau_{(k+1)} = SM + EA_{(k+1)}, \tag{6.1}$$

where $EA_{(k+1)}$ is same as the parameter in Chen-FD, while SM is the dynamic safety margin, and can be adjusted to satisfy the predefined $\overline{QoS}$. Here we have

$$SM_{(k+1)} = SM_k + Sat_k\{QoS, \overline{QoS}\} \cdot \alpha, \tag{6.2}$$

where the α is same as the constant safety margin in Chen-FD, and we set $SM_1 = \alpha$ in the first time slot of the experiment. Furthermore,

$$Sat_k\{QoS, \overline{QoS}\} = \begin{cases} \pm k, QoS > \overline{QoS}; \\ 0, QoS \leq \overline{QoS}. \end{cases} \tag{6.3}$$

where k is a constant value, and $k \in (0, 1)$.

Here we will discuss about how to choose the k, and we focus on the two parameters T_D and MR (see Fig. 6.2), for QAP is relative to MR.

If $T_D > \overline{T_D}$, and $MR < \overline{MR}$: $Sat_k\{QoS, \overline{QoS}\} = k$;
If $T_D > \overline{T_D}$, and $MR > \overline{MR}$: Give a response;
If $T_D < \overline{T_D}$, and $MR < \overline{MR}$: $Sat_k\{QoS, \overline{QoS}\} = 0$;
If $T_D < \overline{T_D}$, and $MR > \overline{MR}$: $Sat_k\{QoS, \overline{QoS}\} = -k$;
This scheme can be applied to the other adaptive timeout-based FD schemes.

6.2 Performance evaluation

6.2.1 Experimental environment

In order to compare QoS of our SPD to that of the existing four FDs (Chen FD [30], Beriter FD [16, 17], and Phi FD [18]), we carried out an experiment in WAN. In this experiment, we use the exactly same trace files from the paper ϕ FD [18-19], and these trace files are publicly available on our lab web [27]. So this gives a common ground for comparing the results of TAM FD, Chen FD [30] and Bertier FD [16-17] with the earlier ϕ FD results [18-19].

That experiment involves two computers: one was located at the Swiss Federal Institute of Technology in Lausanne (EPFL), in Switzerland. The other one was located

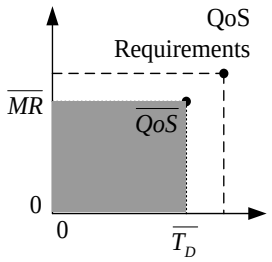


Figure 6.2: Basic theory 2 of selftuing failure detection, $\overline{QoS}$ is the target QoS for the heartbeats.

in JAIST, Japan. The two computers communicate through a normal intercontinental Internet connection.

Experiment setting: hardware/software/ network

In this experiment, the two computers have same equipments and same operating systems with those in [18-19]. By checking the network connection, one found that the two hosts and network link did not break down. Furthermore, the average CPU load for sending host and receiving host were 1/67 and 1/22, respectively.

Heartbeat sampling The experiment started on April 3, 2004 at 2:56 UTC, and finished on April 10, 2004 at 3:01 UTC. During the one week experiment period, the heartbeats were generated at a target rate of one heartbeat every 100 ms (the sending interval) due to the long communication delay in WAN. The average sending rate actually measured was one heartbeat every 103.501 ms (standard deviation: 0.189 ms; min.: 101.674 ms; max.: 234.341 ms). Furthermore, 5845713 heartbeat messages were sent out, while only 5822521 of them were received, so message loss rate was about 0.399 %.

By checking the traces files more closely, one found the messages losses were because of 814 different bursts. The majority of total bursts were the short length bursts. While the maximum burst-length was 1093 heartbeats (only one), it lasted about 2 minutes.

Round-trip time The average RTT is 283.338 ms, it is with a standard deviation of 27.342 ms, a minimum of 270.201 ms, and a maximum of 717.832 ms, the relative standard deviation is $27.342/283.338 * 100\% = 9.65\%$.

6.2.2 Experimental results

When it is a larger α value, it will lead to larger T_D and shorter MR in Chen-FD. Just because larger α value provided larger safety margin in Chen-FD. To this point, our scheme is same to Chen-FD.

In Figs. 6.3-6.4, we compared our scheme SFD with existing Chen FD [30], Beriter FD [16, 17], and Phi FD [18].

At the beginning, SM was very small value, then output QoS in FD had a short detection time T_D ($T_D < \overline{T_D}$) and a large mistake rate MR ($MR > \overline{MR}$). It means that the output QoS is not satisfied for $\overline{QoS}$. Then based on Equations (6.2) and (6.3), we

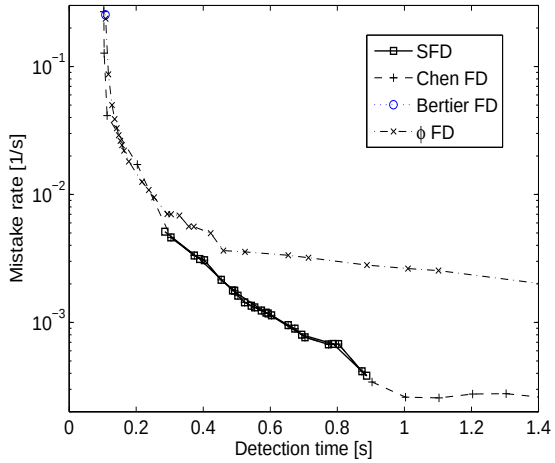


Figure 6.3: Mistake rate vs. detection time in a WAN.

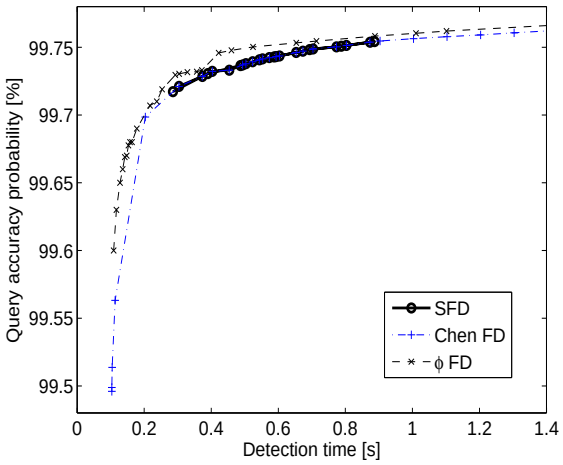


Figure 6.4: Query accuracy probability vs. detection time in a WAN.

have

$$Sat_k\{QoS, \overline{QoS}\} = k, (QoS > \overline{QoS}),$$

So

$$SM_{(k+1)} = SM_k + k \cdot \alpha.$$

The above equation means

$$SM_{(k+1)} > SM_k.$$

Thus, our scheme increased T_D in next freshness point τ to improve the MR of output QoS than the former one (which was still larger than $\overline{MR}$) i.e., got smaller MR at the expense of the detection time.

With the time increasing, SFD gradually ensures a smaller MR of output QoS, while results in longer detection time.

If the detection time T_D increases, and until $T_D > \overline{T_D}$, we have $MR > \overline{MR}$, then we conclude that this SFD can not provide the target service $\overline{QoS}$.

If this SFD can provide the target service $\overline{QoS}$ for the upper application, then with the real time increasing, we can find the detection time T_D is increasing and the MR is reducing. As far as a certain time, we can obtain both $T_D < \overline{T_D}$ and $MR < \overline{MR}$. Thus, this is a case that SFD can let output QoS satisfy the $\overline{QoS}$, i.e., $QoS < \overline{QoS}$, and we satisfy the $\overline{QoS}$ for the upper application.

Interestingly, there are some bursts in this WAN experiment. So for every α value, after the output QoS of FD has been adjusted to satisfy the $\overline{QoS}$, due to the busts, there were some fluctuations for the output QoS of FD. Thus, it may lead to a longer T_D ($T_D > \overline{T_D}$) and a smaller MR ($MR < \overline{MR}$).

For those α values with $T_D > 0.9$ in Chen-FD, our scheme can reduce the SM in next freshness point τ to get shorter T_D gradually, while at the same time, it led to a little larger MR. Because based on Equations (6.2) and (6.3), we have

$$Sat_k\{QoS, \overline{QoS}\} = -k, (QoS > \overline{QoS}),$$

So

$$SM_{(k+1)} = SM_k + (-k) \cdot \alpha.$$

The above equation means

$$SM_{(k+1)} < SM_k.$$

Thus, our scheme increased MR in next freshness point τ to improve the T_D of output QoS than the former one (which was still larger than $\overline{T_D}$) i.e., got smaller T_D at the expense of the MR.

If T_D is farther away the target detection time 0.9 in Chen-FD, we should spend more steps to reduce the T_D and let $T_D < \overline{T_D}$ eventually. With the time increasing, SFD gradually ensures a shorter T_D of output QoS, while results in larger MR.

Here in our experiment, this SFD can provide the target service $\overline{QoS}$ for the upper application. With the real time increasing, we can find the detection time T_D is reducing and the MR is increasing. As far as a certain time, we can obtain both $T_D < \overline{T_D}$ and $MR < \overline{MR}$. Thus, this is a case that SFD can let output QoS satisfy the $\overline{QoS}$, i.e., $QoS < \overline{QoS}$, and we satisfy the $\overline{QoS}$ for the upper application.

Here, we introduce the original idea and relevant concepts of SFD. In future work, we need to further explore how to apply SFD into the other existed failure detection scheme, even to our living.

6.3 Conclusion

This chapter compares QoS of several adaptive failure detection schemes. Also, we introduce an optimization over the existing methods, called SFD, which significantly improves QoS, especially in the design of real-time pragmatic systems. The SFD addresses important practical shortcomings of previous state-of-the-art implementations. On the one hand, it allows for gradual settings between an aggressive behavior (i.e., short latency at the expense of accuracy) and a conservative one. On the other hand, it retains the structural benefits and the formal properties of accrual failure detection. Experiment has been carried out over a wide area network (WAN) case. The experimental results have shown the properties of the adaptive FDs, and demonstrated that the proposed SFD is a pragmatic detector. SFD can adjust its parameters by itself based on the dynamic network environment to let the output QoS satisfy many different QoS requirements of users.

SFD can adjust its parameters by itself to satisfy the users' requirements based on dynamic change of network topology. Therefore, SFD can also be applied into mobile network environments as a good failure detection method, like in ad hoc network. Thus, for all the proposed failure detectors so far, the common point of them is that they all can detect the directly connected processes. While, for some processes that are not directly connected, i.e., they only can communicate each other by some middle processes, all failure detectors are not applicable. Therefore, an open question arises: how to design an indirect failure detector to make sure any two processes, even they are not connected directly, can detect each other effectively.

All previous FDs discussed here can somehow adapt themselves to changes. What they cannot do is to relate to actual QoS values. In other words, self-tuning FD is parameterized by the QoS rather than some undependable parameters.

Chapter 7

Active Queue Management

In the previous chapters, we introduced several failure detectors to improve quality of service (QoS) of the existed ones. There are some relation between the failure detection and the active queue management scheme. Failure detection is generally based on distributed communication networks. Reversely, the performance (delay, throughput, rate of packets dropping, and so on) of communication networks also affects quality of service of failure detector. Generally speaking, if the communication network is unreliable, i.e., heart messages have large delay and lots of heartbeat messages will be lost, then failure detector will suspect wrongly the processes with high probability. Thus, very high mistake rate and low accuracy probability will get for failure detector. More theory analysis of the relationship between AQM and FDs is shown in Chapter 7.6. Therefore, it becomes very necessary to improve the performance of communication network.

7.1 Background of AQM

Internet congestion occurs when the aggregated demand for a resource (e.g., link bandwidth) exceeds the available capacity of the resource. Congestion typically results in long delays in data delivery, wasted resources due to dropping packets, and the possibility of a congestion collapse [52, 53, 54, 55]. Congestion avoidance is an essential technology in the Internet. The Internet congestion avoidance can be usually done at two places: 1) by the end-to-end protocol, such as the TCP; and 2) by the active queue management (AQM) scheme, which is implemented in routers [56]. AQM [57] is a scheme employed by routers to control the traffic going through them. It can achieve smaller queuing delays and higher throughput by purposely dropping out packets. There are several AQM schemes that have been reported in the recent literature for congestion avoidance.

Random Early Detection (RED) [58, 59, 60, 61], recommended for deployment by the Internet Engineering Task Force (IETF), is the most prominent and well studied AQM scheme [62, 63]. It has been widely implemented in routers for congestion avoidance in the Internet. The main objective of RED is to keep the average queue length (average buffer occupancy) low. To do so, RED randomly drops out the incoming packets with a probability proportional to the average queue length, which makes the RED scheme adaptive to bursty traffics. One important metric in measuring the performance of a traffic controller is the stability, the stability of packet dropping rate and the stability of queue length. A major drawback of the RED method is that it is difficult to set the parameters of the RED traffic controller to stabilize the system under the diversity of the

Internet traffics [64, 65]. The problem becomes especially severe when the average queue length reaches a certain threshold, resulting in a sharp decrease of throughput and an increase of drop-rate [56].

There are several variants of RED that have been proposed to address the above problem, such as Adaptive-RED [66, 67, 68], Proportional Derivative RED controller (PD-RED) [56], Proportional Integral controller (PI-controller) [69, 70], and so on. With the RED [58, 59, 60, 61], the resulting average queue length is very sensitive to the level of congestion and initial parameter setting, which makes its behavior unpredictable [69]. Adaptive-RED attempts to stabilize router queue length at a level independent from the active connections, by using an additive-increase multiplicative-decrease (AIMD) policy [66, 67, 68]. Sun et al. [56] proposed a new RED scheme based on the proportional derivative control theory, called PD-RED, to improve the performance of the AQM. Unfortunately, neither Adaptive-RED nor PD-RED provide any systematic method to configure the RED parameters. Moreover, the control gain selection in both methods is based only on empirical observation and simulation analysis. They often work in one situation, but fail in another. A theoretic model and analysis for control gain selection and parameter setting is required. Hollot et al. proposed a Proportional Integral controller, PI-controller, as a means to improve the responsiveness of the TCP/AQM dynamics and stabilize the router queue length around the target value in [69]. Similarly, Deng et al. proposed a Proportional Integral Derivative model, to improve system stability under dynamic traffic conditions in [71]. Both methods used feedback control theory to describe and analyze the TCP/RED dynamics. However, both of them used a simplified linear quadratic Gaussian controller for analysis [72], and they limited their discussion to the classical control elements. Consequently, their methods can only directly link traffic control parameters to one of the AQM objectives, which compromised the global performance.

The main contributions of this paper are as follows. Firstly, we propose a novel packet dropping scheme, called Self-tuning Proportional and Integral RED (SPI-RED), for the TCP/RED dynamic model of [57]. The core idea of this scheme is a new probability function for packet dropping. Secondly, we give a theoretical analysis of the stability of the proposed probability function and give a theoretical guidance in determining the parameters for the proposed SPI-RED method. Our theoretical analysis uses optimal control methodologies. This makes the system analysis more realistic. Finally, we conduct extensive simulations to compare the performance of our proposed methods with the existing schemes. The simulation results have shown that our scheme outperforms the existing schemes in terms of stability and throughput.

7.2 System model and definitions

In this study, our objective is to develop an active queue management system to improve the stability of bottleneck queue in a TCP network.

In [57], a dynamic model of TCP behavior was developed using fluid-flow and stochastic differential equation analysis. Simulation results demonstrated that this model accurately captured the dynamics of TCP. Following this work, a packet dropping scheme was proposed for active queue management for Internet routers. We follow the model introduced in [57] (see Figure 7.1). The differential equations of the model are as the following:

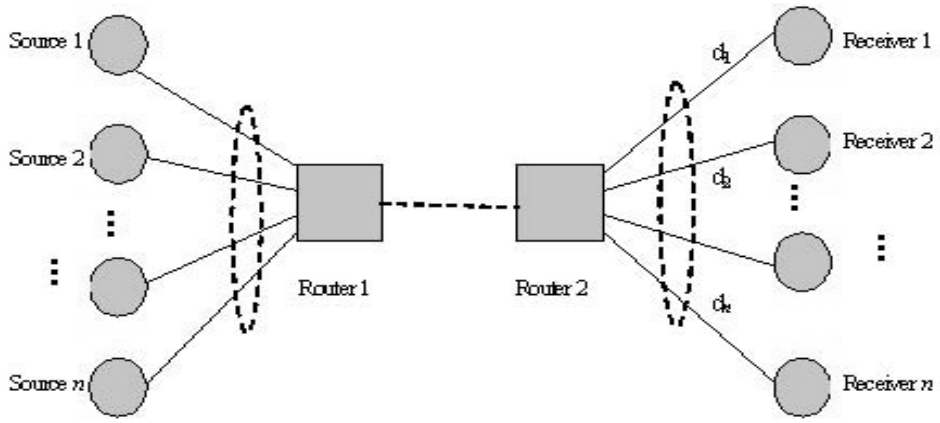


Figure 7.1: The system network model

$$\begin{cases} \delta\dot{W}(t) = -\dfrac{2N\delta W(t)}{R_0^2 C} - \dfrac{R_0 C^2}{2N^2}\delta p(t - R_0), \\ \delta\dot{q}(t) = \dfrac{N}{R_0}\delta W(t) - \dfrac{1}{R_0}\delta q(t). \end{cases} \tag{7.1}$$

In system (7.1), we denote $\delta W(t) = W(t) - W_0$, $\delta\, q(t) = q(t) - q_0$ and $\delta p(t) = p(t) - p_0$, where (W_0, q_0, p_0) is the equilibrium point of the system. Here, $\dot{W}(t)$ and $\dot{q}(t)$ are the time-derivatives of $W(t)$ and $q(t)$ respectively. The parameters of Equation (7.1) are shown in Table 7.1.

The first equation models the AIMD behavior of TCP. The component $-2N\delta W(t)/(R_0^2 C)$ corresponds to the additive increase behavior of TCP, which increases the window size by one packet every round-trip time. The component $[-R_0 C^2 \delta p(t - R_0)/(2N^2)]$ corresponds to the multiplicative decrease behavior of TCP, which halves the window size whenever a timeout occurs. In the second equation, the component $[-\delta q(t)/R_0]$ models the decrease in queue length due to the leaving service of packets. The component $[N\delta W(t)/R_0]$ corresponds to the increase in queue length due to the arrival of packets from the N TCP flows.

In [73], the average queue length $q_{avg}(n)$ was used as a measure of congestion of the network. With the RED scheme, the relationship between the average queue length and the queue length at time interval n is the following:

$$q_{avg}(n) = (1 - w_q)q_{avg}(n-1) + w_q q(n), \tag{7.2}$$

where $q_{avg}(n)$ and $q(n)$ denote the average and the instantaneous queue length, respectively, at the n^{th} interval, and w_q is a weight parameter, $0 < w_q < 1$. Let the sampling interval equal to 1, and then we can obtain the continuous-time form of Equation (7.2) as follows:

$$q(t) = \frac{1}{w_q} q_{avg}(t) + (1 - \frac{1}{w_q}) q_{avg}(t-1). \tag{7.3}$$

By differentiating both sides of Equation (7.3), we have:

$$\dot{q}(t) = \frac{1}{w_q} \dot{q}_{avg}(t) + (1 - \frac{1}{w_q}) \dot{q}_{avg}(t-1). \tag{7.4}$$

Table 7.1: PARAMETERS OF MODEL

Parameter	Description
W	Expected TCP window size (packets)
q	Current queue length (packets)
q_{avg}	Average queue length (packets)
p	Probability of packet dropping
R_0	Round-trip time (second)
q_{ref}	Desired queue length (packets)
B	The buffer size of the congested router
C	Link capacity (packets/second)
N	Load factor (number of TCP connections)

Later, Equations (7.3) and (7.4) will be used together with Equation (7.1) for calculating the average queue length.

7.3 AQM schemes and probability functions

The main idea of the RED scheme [58-61] is to constantly adjust a packet drop probability such that the average queue length is maintained at a target value. The scheme calculates the average queue length, denoted by q_{avg}, using a low-pass filter with an exponential weighted moving average, and compares with two thresholds: the minimum threshold min_{th}, and the maximum threshold max_{th}. When q_{avg} falls below min_{th}, no packets are dropped. When q_{avg} is greater than max_{th}, then every arriving packet is dropped. Dropping the coming packets thus ensures that the average queue length does not exceed the maximum threshold. When q_{avg} is between min_{th} and max_{th}, each arriving packet is dropped by a probability p, given by the following function [58]:

$$p_0(t) = \frac{max_p(q_{avg}(t) - min_{th})}{(max_{th} - min_{th})},$$

$$p(t) = \frac{p_0(t)}{1 - count * p_0(t)},$$

where max_p is the maximum value of $p(t)$ and *count* denotes the number of arriving packets since the last dropped one. Thus, the probability of dropping an arriving packet from a particular connection is roughly proportional to the share of the queue length of the connection. The RED scheme keeps the average queue length low while allowing occasional bursts of packets in the queue and is designed to accompany a transport-layer congestion avoidance protocol such as TCP. The RED scheme has two major drawbacks: a) it is difficult to optimally configure the parameters, i.e., max_p, q_{avg}, max_{th} and min_{th};

and b) average queue length is very sensitive to the level of congestions, making its behavior unpredictable [69].

Floyd et al. proposed an adaptive RED algorithm to increase the robustness of RED's active queue management in [67]. It uses an AIMD policy as follows. It periodically (e.g., every 500 ms) compares the average queue length q_{avg} with the desired queue length q_{ref}, and adjusts the probability of packet dropping p as the following:

$$p = \begin{cases} p + \alpha, & if(q_{avg} > q_{ref}) \ \ and \ \ p \leq 0.5; \\ p \times \beta, & if(q_{avg} < q_{ref}) \ \ and \ \ p \geq 0.01; \end{cases}$$

where $\alpha = min(0.01, max_p/4)$ and $\beta = 0.9$ are the increment and decrement factors, respectively. This scheme requires constant tuning of the parameters α and β based on the current traffic conditions, in order to achieve a desirable average network delay. It introduces extra complexity for configuring additional parameters and it is hard to be applied to variable traffic conditions.

Hollot et al. proposed two simple designs for RED, namely, a Proportional Control and a Proportional Integral controller (PI-controller) as means to improve the responsiveness of the TCP/RED dynamics and stabilize the router queue length around the target in [69-70]. The packet drop probability of PI-controller at sampling interval t is given by the following:

$$p(t) = a * (q(t) - q_{ref}) - b * (q(t-1) - q_{ref}) + p(t-1),$$

where a, b are the PI-controller gains. The PI-controller is largely insensitive to variations in the load level, so it has better robustness. However, this function considers only the probability and instantaneous queue length in the last time interval, while ignoring the history signals in a round-trip time. Another disadvantage of this scheme is the "windup" phenomenon [74] of the integral controller that may cause performance degradation due to the inappropriate drop probability function. Lim et al. proposed a PI AQM with an anti-windup compensator to resolve this problem in [74]. They adopted a static compensation method and designed an anti-windup compensator for nominal system. But, they did not develop the dynamic compensation method. They also did not consider the delay in the control input.

Sun et al. proposed a Proportional Derivative controller (PD-RED) in [56]. The probability function for packet dropping can be described as follows:

$$p(t) = p(t-1) + k_p \frac{(q_{avg}(t) - q_{ref})}{B} + k_d \frac{(q_{avg}(t) - q_{avg}(t-1))}{B},$$

where k_p is the proportional gain and k_d is the derivative gain. By using the above drop probability function, the average queue length is maintained within a target range. In this scheme, there was no integral part, and it did not take into consideration of the past history information (it only considered the probability and average queue length in the last time interval). So it is sensitive to the network change caused by some short-lived or non-TCP traffics. Due to the lack of theoretical analysis and theoretical guidence for choosing the proper control gains, this method is often ad hoc in nature, and it may only be effective under very specific conditions.

7.4 The SPI-RED algorithm

In this section, we present a novel packet dropping algorithm called SPI-RED and give a theoretic analysis and algorithm for choosing the proportional and integral parameters to achieve the system stability.

7.4.1 Packet drop probability

In order to make the queue length stabilize near the target value in AQM, we should choose an appropriate drop probability based on dynamic networks. Proportional Integral Derivative (PID) controllers are popular technologies used in most of the industrial processes, because of its simplicity and robustness [53-54]. Based on our extensive experimental results and observations, we apply the classical control system techniques and come up with the design of a SPI feedback controller for AQM. The new drop probability function is as follows.

$$\begin{aligned} p(t) = p_0 + \frac{K_P(q_{avg}(t) - q_{ref})}{B} \\ -\frac{K_I}{B}\int_{(t-1)-R_0}^{t-1}(q_{avg}(v) - q_{ref})dv, \end{aligned} \tag{7.5}$$

where $q_{avg}(t)$ denotes the average queue length at time t. The component $(q_{avg}(t)-q_{ref})/B$ is the ratio of the current error signal to the buffer size, and $\int_{(t-1)-R_0}^{t-1}(q_{avg}(v) - q_{ref})dv$ is the sum of the history error signals during a round-trip time. This drop probability function considers the buffer size, the current error signal, and the history error signals. We use both the history values and the current value of average queue lengths to calculate the drop probability. By doing so, we avoid the sharp fluctuations of queue length caused by some short-lived flows or non-TCP flows.

The two parameters K_P and K_I need to be set in the above probability function. K_P is the proportional control gain, which is the coefficient of current error signal. K_I is the integral control gain, which is the coefficient of accumulated error signals during a round-trip time. We expect both K_P and K_I greater than zero, i.e., $K_P > 0$ and $K_I > 0$ in the original design. As we discussed before, selecting the right control gains in the stability areas is crucial to ensure the system stability. Next, we will analyze system stability and give theoretic guidelines for choosing proper control gains to optimize network performance.

In this paper, we focus our discussions on stabilizing the queue length as a target value. Stabilizing the queue length is a key to improve the network performance over several metrics. Firstly, the network throughput is improved. Without properly stabilizing the queue, the queue length could be too short, which means the network bandwidth is under utilized, or too long, which would result in network congestion. Secondly, it smooths out burst traffic and avoid bandwidth-waste on repeated retransmissions in TCP flows. Without proper control of queue length, it would be forced to drop any incoming packets in the presence of burst traffic, which triggers TCP to retransmit the lost packets and it causes more congestion. It would eventually block the network. Thirdly, with properly smoothing out the traffic, QoS (e.g., bandwidth and delay requirements) can be better served.

7.4.2 Stability analysis and control gain selection

In this section, we use the Routh-Hurwitz theorem, a common technique in control theory to analyse the stability of our proposed model and to determine the ranges of the control gains K_P and K_I. The stability of the system is measured by the fluctuation of the queue length. The lower the fluctuation amplitude of the average queue length, the better the stability of the network system. The stability of the system effectively ensures that the average queue length converges to a certain desirable value. The stability of the congested queue length is an important performance metric for queue management, because large fluctuation in queue length would lead to high packet dropping rate and poor system throughput. By substituting Equations (7.3) and (7.4) into Equation (7.1), we have:

$$\begin{cases} \delta\dot{W}(t) = -\dfrac{2N}{R_0^2 C}\delta W(t) - \dfrac{R_0 C^2}{2N^2}\delta p(t - R_0), \\ \dfrac{1}{w_q}\dot{q}_{avg}(t) + (1 - \dfrac{1}{w_q})\dot{q}_{avg}(t-1) + \dfrac{1}{R_0} \\ \quad \cdot[\dfrac{1}{w_q}q_{avg}(t) + (1 - \dfrac{1}{w_q})q_{avg}(t-1)] \\ \quad = \dfrac{N}{R_0}\delta W(t) + \dfrac{q_0}{R_0}. \end{cases} \tag{7.6}$$

In order to analyze the stability of the system and determine the values of the control parameters, we must obtain the characteristic equation so that we can determine the stable area. We first linearize the above network system by performing a Laplace Transform of Equations (7.5) and (7.6), and obtain:

$$\begin{cases} (s + \dfrac{2N}{R_0^2 C})\delta W(s) = -\dfrac{R_0 C^2}{2N^2}e^{-R_0 s}\delta P(s), \\ [\dfrac{1}{w_q} + (1 - \dfrac{1}{w_q})e^{-s}](s + \dfrac{1}{R_0})Q_{Avg}(s) \\ \qquad = \dfrac{N}{R_0}\delta W(s) + \dfrac{q_0}{R_0 s}, \\ \delta P(s) = \dfrac{K_P}{B}(Q_{Avg}(s) - \dfrac{q_{ref}}{s}) \\ \qquad -\dfrac{K_I}{Bs}[e^{-s} - e^{-(R_0+1)s}]Q_{Avg}(s), \end{cases} \tag{7.7}$$

where $\delta W(s)$, $Q_{Avg}(s)$, $\delta P(s)$ denote the Laplace Transform of $\delta W(t)$, $q_{avg}(t)$, and $\delta p(t)$ respectively. By rewriting the Equations in (7.7), we have:

$$\begin{cases} \delta W(s) = -[\dfrac{R_0 C^2}{2N^2}/(s + \dfrac{2N}{{R_0}^2 C})]e^{-R_0 s}\delta P(s), \\ Q_{Avg}(s) = [\dfrac{N}{R_0}\delta W(s) + \dfrac{q_0}{sR_0}] \\ \qquad /[(\dfrac{1}{w_q} + (1 - \dfrac{1}{w_q})e^{-s})(s + \dfrac{1}{R_0})], \\ \delta P(s) = Q_{Avq}(s)[\dfrac{sK_P - K_I(e^{-s} - e^{-(R_0+1)s})}{Bs}] \\ \qquad -\dfrac{K_P \cdot q_{ref}}{Bs}. \end{cases} \tag{7.8}$$

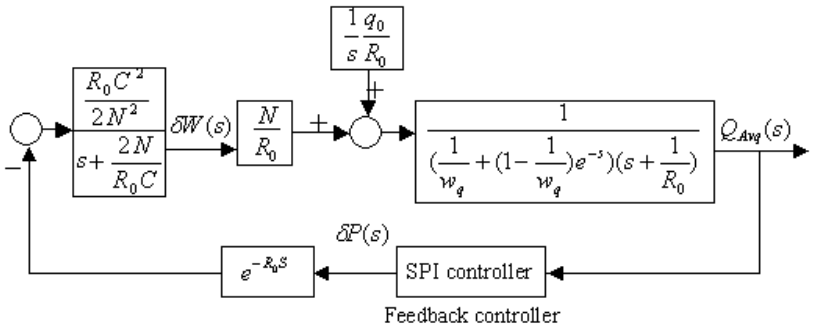


Figure 7.2: The closed-loop system network model

Table 7.2: THE ROUTH ARRAY

s^N	∂_N	∂_{N-2}	∂_{N-4}	...
s^{N-1}	∂_{N-1}	∂_{N-3}	∂_{N-5}	...
s^{N-3}	b_{N-2}	b_{N-4}	b_{N-6}	...
...	...	...	...	...
s^2	d_2	d_0	0	...
s^1	e_1	0	0	...
s^0	f_0	0	0	...

Figure 7.2 illustrates the relationships among $\delta W(s)$, $\delta P(s)$ and $Q_{Avg}(s)$ in the Equations in (7.8). It basically shows how $\delta P(s)$ transits to $\delta W(s)$, which further transits to $Q_{Avg}(s)$, which is eventually transits back to $\delta P(s)$. The transitions are based on the Equations in (7.8). From the diagram in Figure 7.2, we can see the system does not need any external input. It can adjust and stabilize itself purely based on internal feedback. This is a self-tuning controller [75].

Taking $\delta W(s)$, $\delta P(s)$ and $Q_{Avg}(s)$ as three variables in the three Equations in (7.7), we solve $Q_{Avg}(s)$ and obtain the following characteristic function of $Q_{Avg}(s)$:

$$Q_{Avg}(s) = [-\frac{R_0C^2e^{-R_0s}K_Pq_{ref}}{2Bs} - (s+\frac{2N}{R_0^2C})\frac{Nq_0}{s}]/A(s), \tag{7.9}$$

where $A(s)$ denotes:

$$\begin{aligned} A(s) = (s+\frac{2N}{R_0^2C})[\frac{1}{w_q}+(1-\frac{1}{w_q})e^{-s}](NR_0s+R_0s+1) \\ +\frac{R_0C^2}{2N}e^{-R_0s}[\frac{K_P}{B}-\frac{K_I}{Bs}(e^{-s}-e^{-(R_0+1)s})]. \end{aligned} \tag{7.10}$$

There are several ways to test the stability of $Q_{Avg}(s)$ in function (7.9). We employ the Routh-Hurwitz stability test [76] to determine the stability conditions of $Q_{Avg}(s)$. According to the control theory, the system of $Q_{Avg}(s)$ is stable if and only if all the zeros of $A(s)$ are in the open left half-plane (OLHP) [76]. $A(s)$ is the *characteristic polynomial*

of the network system in Equation (7.9) given by the original network system (7.1) with the controller (7.5). The conditions that make all the zeros of $A(s)$ in the OLHP are called stability criteria. We use the Routh-Hurwitz stability test to formulate the *stability conditions*.

Considering a general polynomial function:

$$A_N(s) = \sum_{n=0}^{N} \partial_n s^n, \partial_n > 0. \tag{7.11}$$

The system is stable if and only if all the solutions of s that make $A_N(s) = 0$ are inside the OLHP. The Routh-Hurwitz stability below will give the necessary and sufficient conditions for this system stability.

Given the polynomial function $A_N(s)$, we first construct the Routh array shown in Table 7.2. As seen from Table 7.2, the first two rows of the Routh array are filled by the coefficients of $A_N(s)$, starting with the leading coefficient ∂_N. The elements in the third row are given by

$$b_{N-2} = \frac{\partial_{N-1}\partial_{N-2} - \partial_N\partial_{N-3}}{\partial_{N-1}} = \partial_{N-2} - \frac{\partial_N\partial_{N-3}}{\partial_{N-1}},$$

$$b_{N-4} = \frac{\partial_{N-1}\partial_{N-4} - \partial_N\partial_{N-5}}{\partial_{N-1}} = \partial_{N-4} - \frac{\partial_N\partial_{N-5}}{\partial_{N-1}},$$

$$\cdots$$

The elements in the fourth row are given by

$$c_{N-3} = \frac{b_{N-2}\partial_{N-3} - \partial_{N-1}b_{N-4}}{b_{N-2}} = \partial_{N-3} - \frac{\partial_{N-1}b_{N-4}}{b_{N-2}},$$

$$c_{N-5} = \frac{b_{N-2}\partial_{N-5} - \partial_{N-1}b_{N-6}}{b_{N-2}} = \partial_{N-5} - \frac{\partial_{N-1}b_{N-6}}{b_{N-2}},$$

$$\cdots$$

The other rows are computed in a similar fashion.

The Routh-Hurwitz stability test states that the system is stable (i.e., all the zeros of $A_N(s)$ are located in OLHP) if and only if all the elements in the second column of the Routh array are all strictly positive (> 0). The Routh-Hurwitz test can be used to derive simple conditions for stability, expressed directly in terms of the coefficients of $A_N(s)$.

In order to compute the characteristic polynomial $A(s)$ in (7.10), we use the approximation $e^{-R_0 s} \approx 1 - R_0 s + R_0 s^2/2$. Based on Equation (7.10), we have:

$$
\begin{aligned}
A(s)\cdot N= & s^4\left[\tfrac{1}{2}R_0^3(1-\tfrac{1}{w_q})\right] \\
& +s^3\left[(1-\frac{1}{w_q})(\frac{NR_0}{C}-\frac{R_0^2}{2})+\frac{R_0^5C^2K_I}{8NB}+\frac{R_0^4C^2K_I}{4NB}\right] \\
& +s^2\left[\frac{N}{Cw_q}+\frac{R_0}{w_q}-\frac{N}{C}+\frac{R_0^3C^2K_P}{4NB}-\frac{R_0^4C^2K_I}{2NB}\right. \\
& \quad \left.-\frac{R_0^3C^2K_I}{2NB}\right] \\
& +s\left[1+\frac{2N}{R_0Cw_q}-\frac{R_0^2C^2K_P}{2NB}+\frac{3R_0^3C^2K_I}{4NB}\right. \\
& \quad \left.+\frac{R_0^2C^2K_I}{2NB}\right] \\
& +s^0(\frac{R_0C^2K_P}{2NB}+\frac{2N}{R_0^2C}-\frac{R_0^2C^2K_I}{2NB}).
\end{aligned}
\tag{7.12}
$$

Based on the former RED theory, $0 < w_q < 1$ and $R_0 > 0$, so we have $0.5(R_0)^3(1-\frac{1}{w_q}) < 0$. For the ease of using the Routh-Hurwitz stability test theory, we let $-a_4$, $-a_3$, $-a_2$, $-a_1$ and $-a_0$ denote the coefficients in the above equation of s^4, s^3, s^2, s^1 and s^0, respectively. That is:

$$a_4 = -\frac{1}{2}R_0^3(1-\frac{1}{w_q}), \tag{7.13}$$

$$a_3 = -(1-\frac{1}{w_q})(\frac{NR_0}{C}-\frac{R_0^2}{2})-\frac{R_0^5C^2K_I}{8NB}-\frac{R_0^4C^2K_I}{4NB}, \tag{7.14}$$

$$a_2 = -\frac{N}{Cw_q}-\frac{R_0}{w_q}+\frac{N}{C}-\frac{R_0^3C^2K_P}{4NB}+\frac{R_0^4C^2K_I}{2NB}+\frac{R_0^3C^2K_I}{2NB}, \tag{7.15}$$

$$a_1 = -1-\frac{2N}{R_0Cw_q}+\frac{R_0^2C^2K_P}{2NB}-\frac{3R_0^3C^2K_I}{4NB}-\frac{R_0^2C^2K_I}{2NB}, \tag{7.16}$$

$$a_0 = -\frac{R_0C^2K_P}{2NB}-\frac{2N}{R_0^2C}+\frac{R_0^2C^2K_I}{2NB}. \tag{7.17}$$

Based on Equation (7.10), we have:

$$
\begin{aligned}
A'(s) &= -A(s)\cdot N \\
&= a_4s^4+a_3s^3+a_2s^2+a_1s^1+a_0s^0,
\end{aligned}
\tag{7.18}
$$

Based on the above Routh-Hurwitz stability test, we get the Routh Table (Table 7.3), where:

$$r_{31} = \frac{a_3a_2-a_4a_1}{a_3}, \tag{7.19}$$

$$r_{41} = \frac{r_{31}a_1-a_3a_0}{r_{31}}. \tag{7.20}$$

Table 7.3: THE ROUTH TABLE

s^4	a_4	a_2	a_0
s^3	a_3	a_1	0
s^2	r_{31}	a_0	0
s^1	r_{41}	0	0
s_0	a_0	0	0

The network system of $Q_{Avg}(s)$ in (7.9) is stable if and only if the values in the second column of Table 7.3 are all greater than zero, i.e.:

$$a_4 > 0, a_3 > 0, r_{31} > 0, a_0 > 0, \text{and } r_{41} > 0. \tag{7.21}$$

We will analyse the stability conditions given in (7.21) one by one.
For a_4 in (7.13), since $0 < w_q < 1$ and $R_0 > 0$, then we have:

$$a_4 > 0.$$

For a_3 in (7.14), to make $a_3 > 0$, we need:

$$-(1 - \frac{1}{w_q})(\frac{NR_0}{C} - \frac{R_0^2}{2}) - \frac{R_0^5 C^2 K_I}{8NB} - \frac{R_0^4 C^2 K_I}{4NB} > 0.$$

Solving the above inequation, we have:

$$K_I < \frac{(w_q - 1)(CR_0 - 2N)}{2w_q R_0^3 C^3 (R_0 + 2)}. \tag{7.22}$$

For r_{31} in (7.19), to make:

$$\frac{a_3 a_2 - a_4 a_1}{a_3} > 0,$$

since $a_3 > 0$, we have:

$$a_3 a_2 - a_4 a_1 > 0. \tag{7.23}$$

By substituting a_1, a_2, a_3 and a_4 into the above inequation and solving K_P, we have:

$$\begin{aligned} K_P < \{&R_0^3(1 - \frac{1}{w_q})(\frac{1}{2} + \frac{N}{R_0 C w_q} + \frac{R_0^2 C^2 K_I}{4NB} \\ &+ \frac{3R_0^3 C^2 K_I}{8NB}) + [\frac{N}{C} + \frac{R_0^3 C^2 K_I (1 + R_0)}{2NB} \\ &- \frac{(CR_0 + N)}{Cw_q}] \cdot [\frac{(2NR_0 - CR_0^2)}{2C}(1 - \frac{1}{w_q}) \\ &+ \frac{C^2 R_0^4 K_I (2 + R_0)}{8NB}]\}/(\frac{R_0^4 C}{4B} + \frac{R_0^5 C^2}{8NB} \\ &- \frac{R_0^4 C}{4Bw_q} - \frac{R_0^5 C^2}{8NBw_q} + \frac{R_0^7 C^4 K_I}{16N^2 B^2} + \frac{R_0^8 C^4 K_I}{32N^2 B^2}). \end{aligned} \tag{7.24}$$

For r_{41} in (7.20), by substituting r_{31} in (7.19) into Equation (7.20), we have:

$$r_{41} = \frac{a_1 a_2 a_3 - (a_1)^2 a_4 - a_0 (a_3)^2}{a_2 a_3 - a_1 a_4}.$$

Since in (7.23), we already made:

$$a_2 a_3 - a_1 a_4 > 0.$$

To make $r_{41} > 0$, we need to have:

$$f(K_P) = a_1 a_2 a_3 - a_1^2 a_4 - a_0 a_3^2 > 0. \tag{7.25}$$

We express inequation (7.25) as a function of K_P, because we now focus on finding the range of K_P. Since a_3 and a_4 are irrelevant to K_P, we substitute a_0, a_1, and a_2 into (7.25), obtaining:

$$\begin{aligned}
&[-1 - \frac{2N}{R_0 C w_q} + \frac{R_0^2 C^2 K_P}{2NB} - \frac{3R_0^3 C^2 K_I}{4NB} - \frac{R_0^2 C^2 K_I}{2NB}] \\
&\cdot a_3 \cdot [-\frac{N}{C w_q} - \frac{R_0}{w_q} + \frac{N}{C} - \frac{R_0^3 C^2 K_P}{4NB} + \frac{R_0^4 C^2 K_I}{2NB} \\
&+ \frac{R_0^3 C^2 K_I}{2NB}] - a_4 \cdot [-1 - \frac{2N}{R_0 C w_q} + \frac{R_0^2 C^2 K_P}{2NB} \\
&- \frac{3R_0^3 C^2 K_I}{4NB} - \frac{R_0^2 C^2 K_I}{2NB}]^2 - (a_3)^2 \cdot (-\frac{R_0 C^2 K_P}{2NB} \\
&- \frac{2N}{R_0^2 C} + \frac{R_0^2 C^2 K_I}{2NB}) > 0.
\end{aligned} \tag{7.26}$$

Inequation (7.26) is a quadratic function of K_p. Furthermore, the coefficient of K_p^2 is

$$a_3 (\frac{R_0^2 C^2}{2NB})(\frac{-R_0^3 C^2}{4NB}) - a_4 (\frac{-R_0^2 C^2}{2NB})^2.$$

Since $a_3 > 0$, $a_4 > 0$, $C > 0$, $B > 0$, $R_0 > 0$ and $N > 0$, we know:

$$a_3 (\frac{R_0^2 C^2}{2NB})(\frac{-R_0^3 C^2}{4NB}) - a_4 (\frac{-R_0^2 C^2}{2NB})^2 < 0.$$

The coefficient of K_p^2 is negative. Function $f(K_P)$ in (7.26) is in a parabolic curve. Therefore, there are two points of K_P that make $f(K_P) = 0$ and any value of $f(K_P)$ between the two points of K_P is greater than zero. Let K_{P1} and K_{P2} be the two points of K_P that make function $f(K_P) = 0$, and $K_{P1} < K_{P2}$. In order to make inequation (7.26) true, the range of K_P must be:

$$K_{P1} < K_P < K_{P2}. \tag{7.27}$$

Finally, considering a_0, we have

$$-\frac{R_0 C^2 K_P}{2NB} - \frac{2N}{R_0^2 C} + \frac{R_0^2 C^2 K_I}{2NB} > 0.$$

a) Sample the current queue length $q(n)$ and compute $q_{avg}(n)$ by formula (2);

b) Obtain the network parameters, R_0, N, and C;

c) Compute the range of K_I by inequality (22);

d) Choose the proper value of K_I within its range;

e) Compute the range of K_P by inequalities (24), (27) and (29);

f) Choose the proper value of K_P within its range;

g) Compute the packet drop probability $p(n)$ by formula (5);

Figure 7.3: The SPI-RED algorithm

$$K_P < R_0 K_I - \frac{4N^2 B}{R_0^3 C^3}. \tag{7.28}$$

By substituting K_I in (7.22) into (7.28), we have:

$$K_P < \frac{(w_q - 1)(CR_0 - 2N)}{2w_q R_0^2 C^3 (R_0 + 2)} - \frac{4N^2 B}{R_0^3 C^3}. \tag{7.29}$$

We have now determined the stability condition of K_I in (7.22), and of K_P in (7.24), (7.27) and (7.29). For K_P, its ranges are given in (7.24), (7.27) and (7.29). We do not know which range is tighter from this analysis. The tighter range can only be obtained by computing all the inequations out in real system situations (or in simulations). There are methods for online estimation of network parameters such as R_0, N, C available in the literature. For example, the algorithm in [77] can accurately measure the round-trip time R_0 by accepting only good samples and using the retransmission back-off strategy. Link capacity C and TCP workload N can also be estimated according to the method proposed in [78-79].

Compared to the results reported by Hollot et al. [69] and by Low et al. [80], the stability conditions in our analysis give a clear relationship between the stability and the network parameters. The analytical results provide good guidelines for choosing the important parameters of SPI-RED, leading to the desired stability and satisfactory overall performance.

7.4.3 The specific algorithm of SPI-RED

Based on the above stability analysis, we can select the proper control gains that can ensure system stability and therefore improve network performance. The algorithm for computing $p(n)$ for time n (the n^{th} sampling interval) can be summarized as the Figure 7.3.

Notice that after we compute the ranges of K_I and K_P, we simply choose a value for each of them randomly within their ranges. At this moment, we do not have theoretical guidance for choosing a better value. According to our simulation results, any values within the ranges make the system stable.

7.5 Performance evaluation

In this section, we evaluate the performance of the proposed SPI-RED packet dropping algorithm by a number of simulations performed using *ns2* [81]. The performance of SPI-RED is compared with RED [60] and other RED variants such as PI-controller [70], PD-RED [56] and adaptive RED [67]. The network topology used in the simulation is a dumbbell topology with a single common bottleneck link of 45 Mb/s capacity (see Figure 7.1), which is the same as the one used in [64, 70]. This dumbbell topology is a good abstract of multiple bottleneck links for study of network congestions. By studying the behavior of two end-routers of a congested link, it can capture the effectiveness of the congestion control method. Along the bottleneck link, there are with many identical, long-lived and saturated TCP/Reno flows. In other words, the TCP connections are modeled as greedy FTP connections, which always have data to send as long as their congestion windows permit. According to the study in [89], long-lived TCP flows constitute about 95% of the Internet traffic, so they are the main factor that contributes to network congestion. The receiver's advertised window size is set sufficiently large so that the TCP connections are not constrained at the destination. The ack-every-packet strategy is used at the TCP receivers. For these AQM schemes, we maintain the same test conditions: the same topology (as described above), the same saturated traffic, and the same TCP parameters.

The parameters used are as follows: the mean packet size is 500 bytes, the round-trip propagation delay is $0.1s$, and the buffer size B is set to be 1125 packets (twice the bandwidth-delay product of the network). The basic parameters of RED (see notation in [56, 58, 60]) are set at $intervaltime = 0.5s$, $min_{th} = 15$ packets, $max_{th} = 785$ packets, $max_p = 0.01$ and $w_q = 0.002$, where the $intervaltime$, min_{th}, max_{th}, and max_p show the sampling interval time, minimum queue threshold, the maximum queue threshold and the maximum drop probability, respectively. For Adaptive RED, the parameters are set the same as in [56, 67]: $\alpha = 0.01$, $\beta = 0.9$. For PI-controller, PI coefficients a and b that are implemented are 1.822×10^{-5} and 1.816×10^{-5}, respectively [70]. For PD-RED, the parameters are set the same as in [56]: $\delta t = 0.01$, $k_p = 0.001$ and $k_d = 0.05$. For SPI-RED, we set $\delta t = 0.01$, $K_P = 12$ and $K_I = 0.01$.

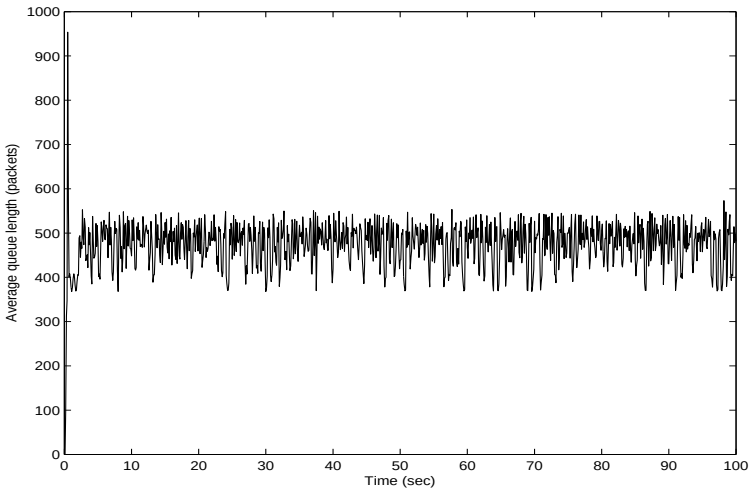
In this simulation, we focus on the following key performance metrics: throughput (excluding packet retransmissions), average queue length and its standard deviation, and packet drop probability. The average queue length is defined as the arithmetic mean value of instantaneous queue lengths.

7.5.1 Simulation 1: stability under extreme conditions

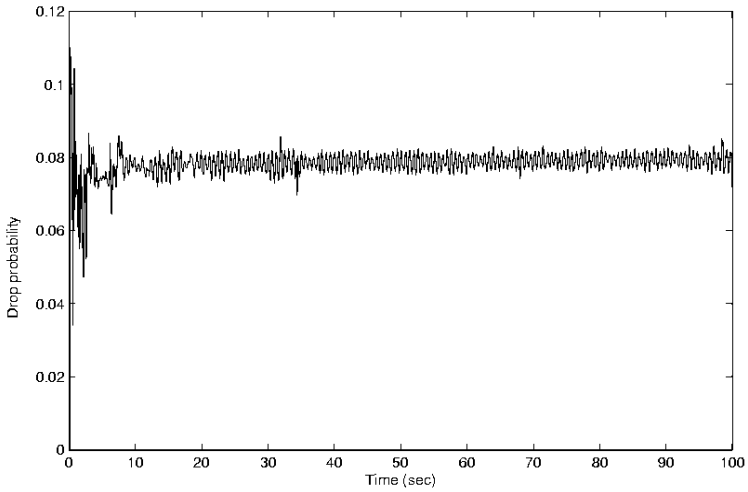
In this experiment, all TCP flows are persistent, and the stability of the AQM schemes are investigated under two extreme cases: 1) light congestion with a small number of TCP flows N ($N = 400$ connections), 2) heavy congestion with a large N ($N = 2000$ connections). We set the queue target at 500 packets. Other parameters are the same as those described above.

Figure 7.4 and Figure 7.5 demonstrate the dynamic change of the average queue length and the drop probability of the SPI-RED algorithm under light congestion ($N = 400$ connections) and heavy congestion ($N = 2000$ connections). The average queue lengths in Figures 7.4-7.5 stabilize after about 2.5 s and 6 s, respectively. It can be seen that,

although the average queue length and the drop probability fluctuate at first, they quickly stabilize near the queue targets of 500 packets for the average queue length and 0.078 for the drop probability. Both the fluctuation amplitude of SPI-RED's average queue length and the variance of its drop probability are small. In summary, SPI-RED shows good stability and quick response under both light and heavy congestion.

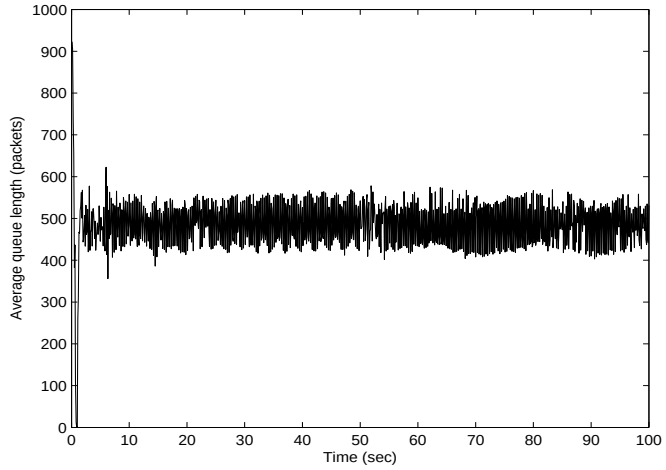


(a) Average queue length

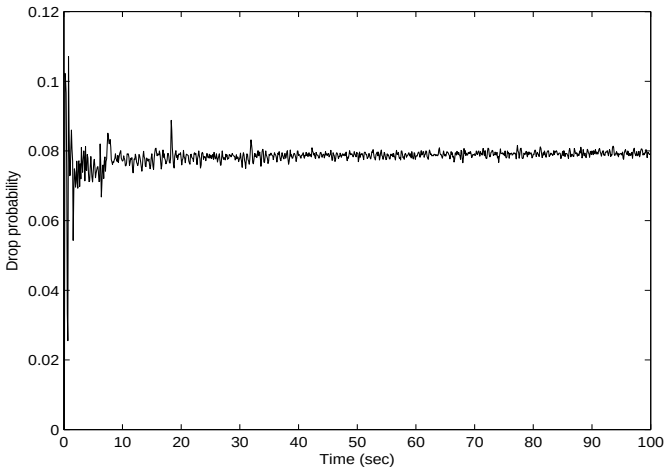


(b) Drop probability

Figure 7.4: Average queue length and drop probability for light congestion (for SPI-RED, $N = 400$). Although Average queue length and drop probability fluctuate at first, they quickly stabilize near the target values of 500 and 0.08, respectively. The fluctuation amplitudes are small. SPI-RED shows good stability and quick response under light congestion.



(a) Average queue length



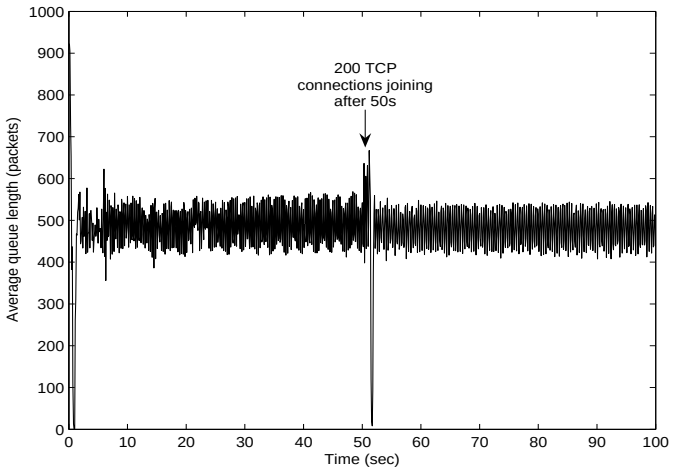
(b) Drop probability

Figure 7.5: Average queue length and drop probability for heavy congestion (for SPI-RED, $N = 2000$). SPI-RED shows good stability and quick response under heavy congestion.

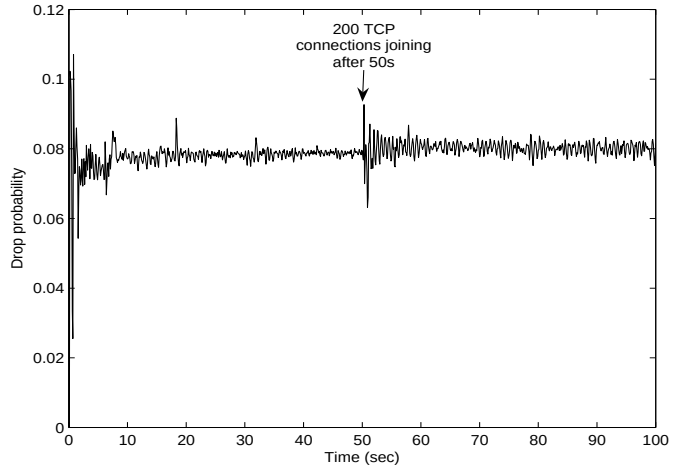
7.5.2 Simulation 2: response with a variable number of connections

In this section, the simulation is performed with a variable number of TCP connections. The goal is to show the impact that joining connections can have on the stability of the system. In the experiment, the number of connections is initially set to 2000. We leave time for the system to stabilize, and then, at time 50.01 s, 200 additional TCP connections suddenly join the link. Other parameters are unchanged from the values used in Simulation 1.

Figure 7.5.2 also shows the average queue length and the drop probability for the variable number of connections. We can find that, in the first half, after about 6 s, the average queue length becomes relatively stable near the queue target of 500 packets, as well as the drop probability, which stabilizes near 0.078. At 50.01 s, the new TCP connections joining the link make the average queue length and drop probability fluctuate briefly. But, we can see that these values stabilize again quickly. The stablization time is only about 2 s. From these figures, we can find that SPI-RED achieves a short response time, good stability and good robustness.



(a) Average queue length



(b) Drop probability

Figure 7.6: Average queue length and drop probability for variable number of connections (for SPI-RED). After 50 s, there are about 200 TCP connections joining, the lines fluctuate accordingly. But, they stabilize again quickly. From these figures, we can find that SPI-RED achieves short response time, good stability and good robustness.

Table 7.4: Simulation results comparison

AQM scheme	Average queue length (packets)	Std. dev. of average queue length (packets)
RED	953.11	6.48
PD-RED	400.48	1.92
Adaptive RED	433.57	2.76
PI controller	414.61	5.07
SPI-RED	398.91	1.83

7.5.3 Simulation 3: comparisons with existing AQM schemes

In this simulation, we compare the performance of the SPI algorithm with existing AQM schemes. For all AQM schemes mentioned in this part, the average queue length target is set at 400 packets; the initial number of connections is set to 500, and then 100 connections have their start-time uniformly distributed over a period of 100 s. Other parameters are the same as those in the above simulations (Simulation 1 and Simulation 2).

Table 7.4 summarizes the steady-state performance, by giving, for each scheme, the average and the standard deviation of the average queue length. In the table, it is clear that the SPI-RED and PD-RED have a little better performance than the other three schemes (i.e., RED, Adaptive RED, and PI-RED) in terms of the standard deviation of the average queue length and the average queue length. The SPI-RED has a little lower standard deviation of the average queue length than that of the PD-RED. While the average queue length for PD-RED is very fine (near the target 400), here our scheme is just a slight poor, while it is also good.

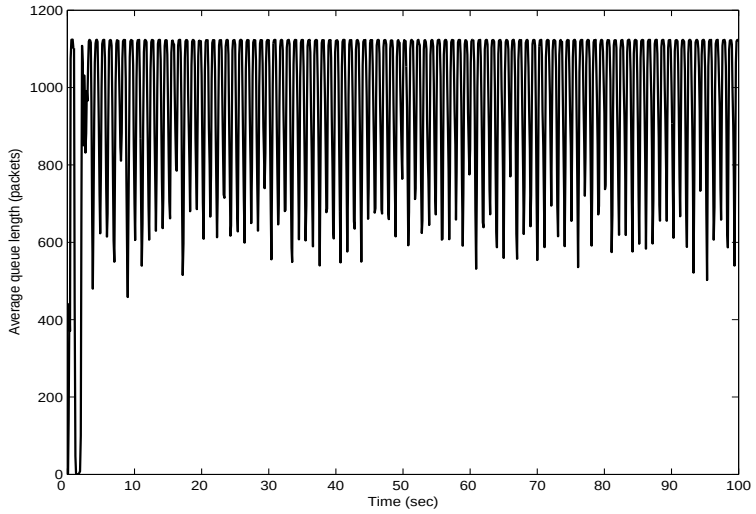
Figures 7.7-7.12 show the average queue length and the drop probability obtained with various AQM schemes, namely, RED [60], Adaptive RED [67], PD-RED [56], PI-controller [70] and SPI-RED, respectively. In Figure 7.7, the experiment shows that, with RED, both the average queue length and the drop probability oscillate and remain nearly out of control. So many RED variants use different packets dropping probability as indicator to trigger packets dropping. From Figure 7.8, with Adaptive RED, the queue stabilizes after about 10 s. Besides, the average queue length and the drop probability have smaller fluctuations with Adaptive RED than they do with RED. This shows that Adaptive RED behaves much better than RED in this experiment. In Figure 7.9, we see that, for PD-RED, the fluctuation amplitude of the average queue length is smaller than that of both RED and Adaptive RED; the variance of the drop probability is also much smaller than that of these two latter schemes; and the queue stablization time is also less than that of these two latter schemes, only about 7 s. These observations are consistent with the experiment of Sun et al. in [56]. Figure 7.10 shows the average queue length and drop probability for PI-controller (Sun et al. did not compare with this scheme in [56]). The figure shows that the average queue length fails to stabilize at the target value of 400 packets (it does at about 300 packets). This shows that PI-controller does a poor job in terms of control, leading to a waste of resources. Besides, PI-controller temporarily reaches longer average queue lengths than PD-RED during the beginning of

the experiment, its stabilization period is more erratic, and its drop probability stabilizes to a significantly higher value than with PD-RED or even Adaptive RED. To sum up the results comparing existing schemes; PD-RED clearly outperforms the other schemes.

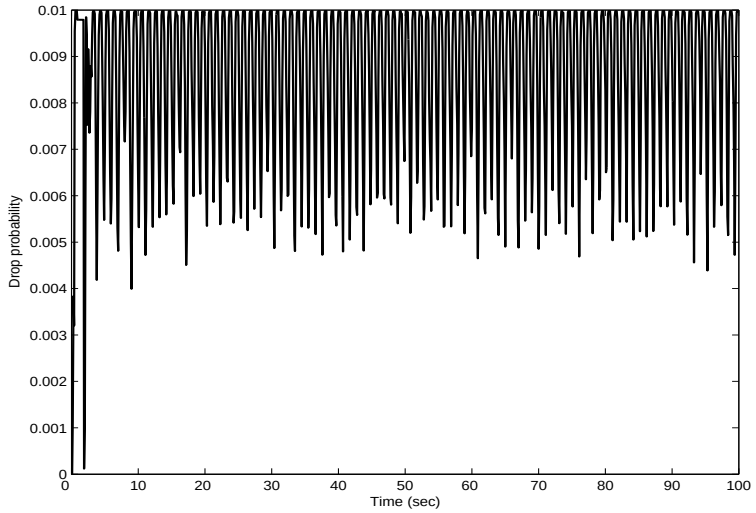
Finally, Figure 7.11 shows the simulation results for SPI-RED (i.e., our proposed scheme). The figure shows that the average queue length quickly stabilizes. It would reach the steady state within less than a second, if not for one short-lived small spike after about 5 s. After that, the average queue length remains stable, just at the target of 400 packets. The drop probability stabilizes after less than 10 s, around a steady-state value between 0.08 and 0.09. When comparing with PD-RED, we find that SPI-RED stabilizes at roughly the same values, although SPI-RED seems to take a little less time to stabilize and then fluctuate less after reaching the steady state. Since there are no other contenders, further experiments compare only SPI-RED with PD-RED.

In Figures 7.12 and 7.13, we compare the throughput obtained with SPI-RED (dashed line) and PD-RED (solid line). Initially, it is clear that SPI-RED is with a slightly better throughput than that of PD-RED during the transient period (first 10 s), which is clear shown in Figure 7.13: Part 1. From about 10 s to 20 s, as shown in Figure 7.13: Part 2, the throughput is similar to the both schemes. After that, the throughput of PD-RED degrades with the time passes, and reaches about 90% of the theoretical maximum at 100 s. While SPI-RED sustains a better throughput than PD-RED, with a slight but steady improvement as time passes. After only 100 s of simulation, the difference in throughput is already more than 5%. This is significant given that we are already very near the theoretical maximum. Thus, it is clear that SPI-RED has higher throughput than that of PD-RED.

From the above simulation results, we conclude that the proposed SPI-RED scheme exhibits better network performance than RED [60], adaptive RED [67], PD-RED [56] (in most cases), and PI-controller [70].

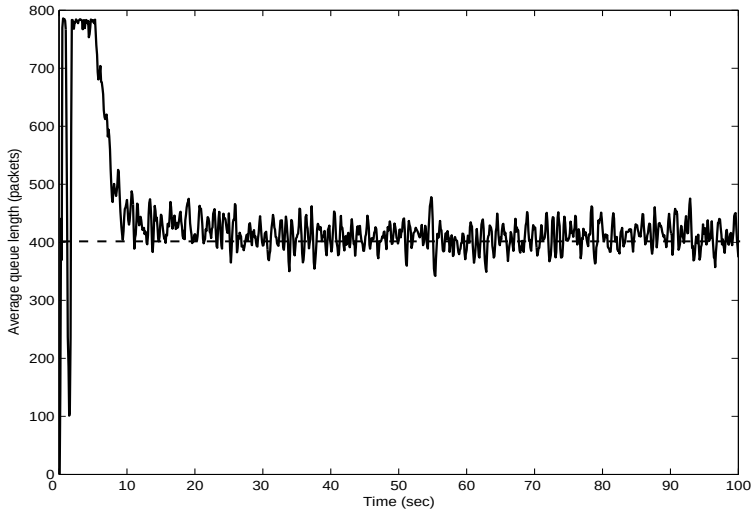


(a) Average queue length

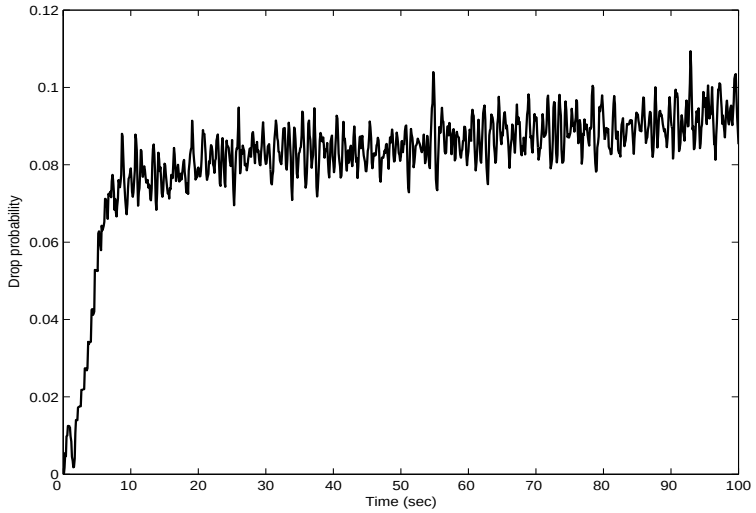


(b) Drop probability

Figure 7.7: Average queue length and drop probability for RED [60], the Average queue length of RED fails to stabilize to the target value 400.

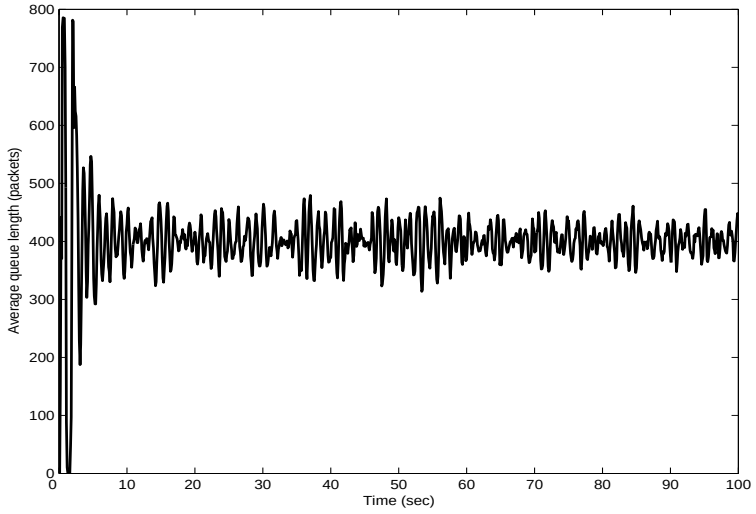


(a) Average queue length

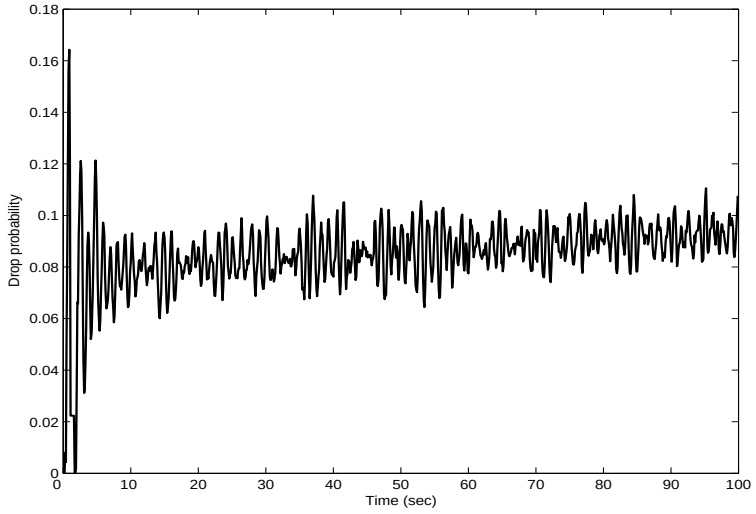


(b) Drop probability

Figure 7.8: Average queue length and drop probability for Adaptive-RED [67], comparing with Fig. 7.7, this shows that Adaptive RED behaves much better than RED in this experiment.

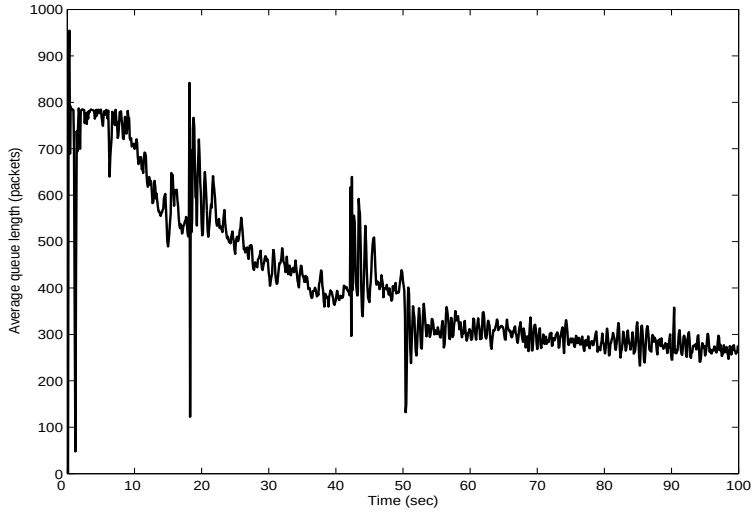


(a) Average queue length

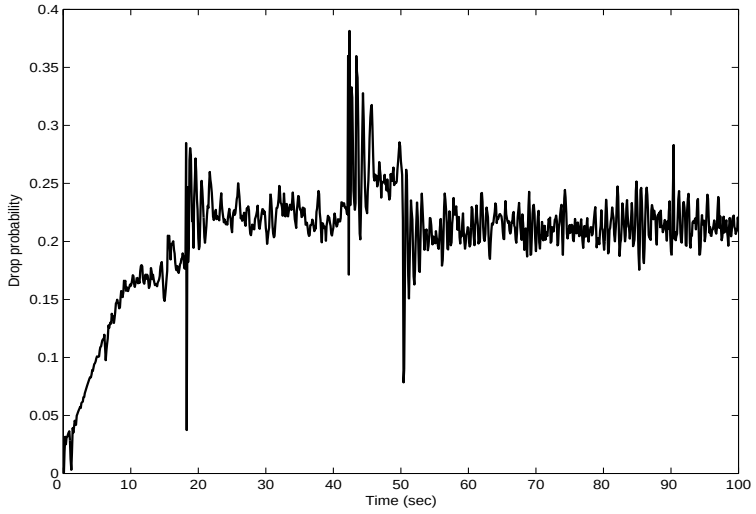


(b) Drop probability

Figure 7.9: Average queue length and drop probability for PD-RED [56], comparing with Figs. 7.7-7.8, this shows that PD-RED behaves much better than RED and Adaptive RED in this experiment.

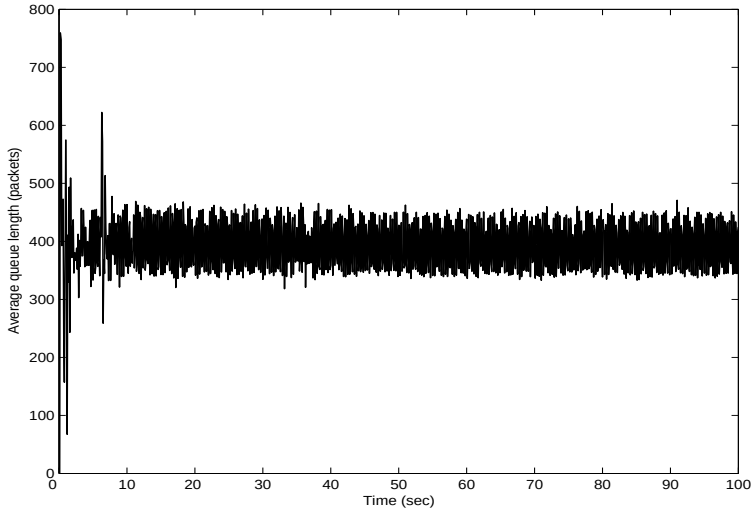


(a) Average queue length

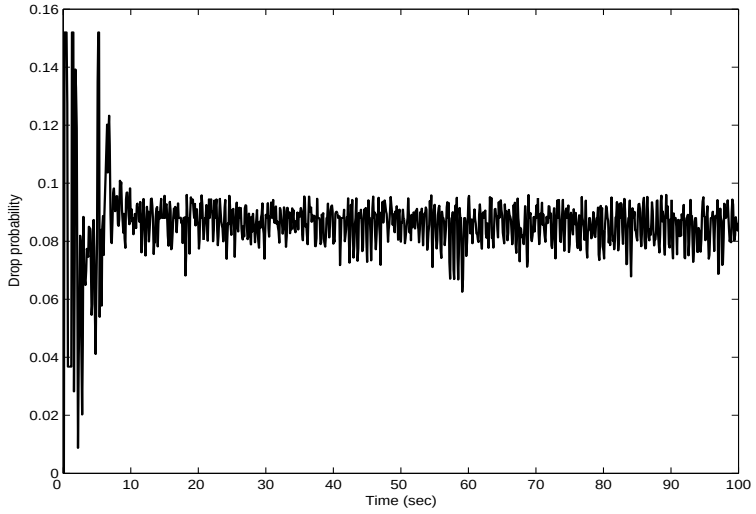


(b) Drop probability

Figure 7.10: Average queue length and drop probability for PI-RED [70], comparing with Figs. 7.9, PI-RED is poor than PD-RED.



(a) Average queue length



(b) Drop probability

Figure 7.11: Average queue length and drop probability for SPI-RED, comparing with Fig. 7.9, the proposed SPI-RED scheme exhibits better network performance than PD-RED (in most cases).

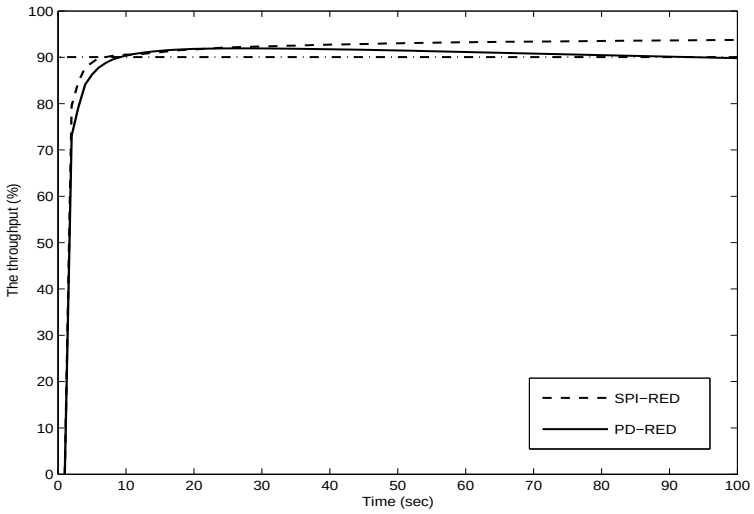
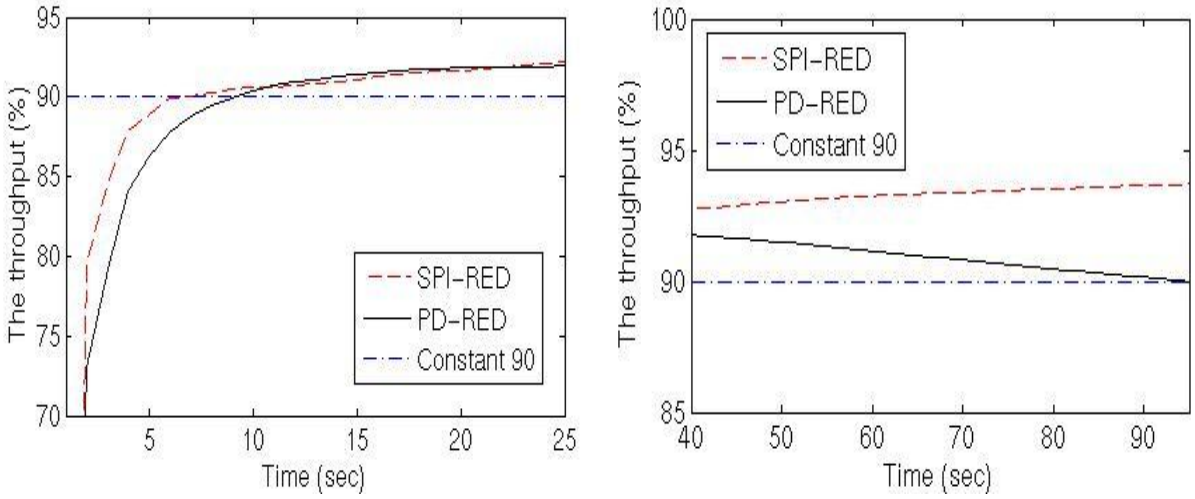


Figure 7.12: The throughput for PD-RED and SPI-RED



(a) Part 1 (b) Part 2

Figure 7.13: The throughput for PD-RED and SPI-RED: Part 1 and Part 2. Part 1 is the higher and left corner in Figure 7.12; Part 2 is the higher and right corner in Figure 7.12.

7.6 QoS of failure detectors influenced by AQM

7.6.1 System structure

In each process, there is a failure detector module that can monitor the other processes and report the other processes' status. Then, the failure detector can provide such process status information for distributed applications. The QoS metrics of a failure detector mainly include detection time, mistake rate and query accuracy probability. These three metrics determine whether a failure detector is good or not, while they are influenced by communication environment used by failure detector. In detail, a good failure detector can provide service to distributed application with good QoS, i.e., short detection time, low mistake rate and high query accuracy probability. A perfect failure detector can detect the other processes with no mistake and very short detection time. Generally, a failure detector monitors the other processes by sending messages periodically (heartbeat messages) through a communication network. The specific system structure is shown in Figure 8.1. In this figure, active queue management (AQM) scheme used in router is an effective network congestion avoidance method for network.

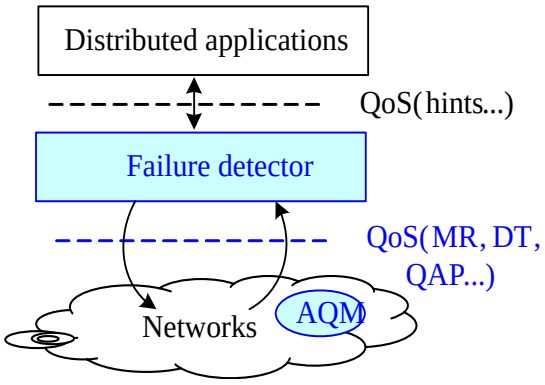


Figure 7.14: The relation structure model of failure detector and active queue management.

How AQM affects the performance of failure detector will be discussed in the following. Generally speaking, if the communication network is unreliable, i.e., lots of heartbeat messages will be lost, then failure detector will suspect wrongly the processes with high probability. Thus, very high mistake rate and low accuracy probability will get for failure detector. There are many important factors in communication networks, which have great effect on QoS of failure detector, such as probability of message loss, change of network topology due to node failure, and dynamic and unpredictable message delay. Especially in wide area distributed systems,due to many applications communicate each other using the same network; it is easy to bring the congestion of network in bottleneck router. The

Table 7.5: QoS dynamic comparison when congestion becomes severe

FD scheme	Detection time (DT)	Mistake rate (MR)	Query accuracy probability (QAP)
TAM FD	Become larger	Become larger	Become small, the worst case is zero
ED FD	Become larger, but there is a maximal value after that there is no data	Become larger, but there is a maximal value after that there is no data	Become small, the worst case is zero
Kappa FD	Become larger, but there is a maximal value after that MR/QAP reach theoretical maximum values	Become larger	Become small, the worst case is zero
Self-tuning FD	Slightly become large while still satisfying the target QoS of users	Slightly become large while still satisfying the target QoS of users	Slightly become small while still satisfying the target QoS of users

congestion of network can lead to lots of messages lost. Therefore, the performance of communication networks also affects quality of service of failure detector. Furthermore, how to avoid such congestion in bottleneck router becomes a very important issue.

AQM schemes have a certain effect on the QoS of FDs. In Table 7.5, it presents the effect of AQM on QoS of failure detectors. AQM is an effective congestion avoidance method of network and congestion level of network has a certain effect on QoS of failure detector. This table gives the effect of congestion level of network on QoS of TAM FD, ED FD, Kappa FD, and Self-tuning FD (when the target QoS can be satisfied). From this table, we can find: when the congestion becomes severe, detection time of TAM FD, ED FD and Kappa FD becomes larger; mistake rate has the same tread as detection time; query accuracy probability will become small. That is because: when the network congestion becomes severe, a lot of message will be lost and the network delay is larger. For TAM FD, the safety margin is dynamically adjusted based on the network delay and the related parameters α and β. When a lot of messages are lost, the value of β and network delay will become larger. The safety margin of TAM FD will become larger and larger, thus the detection time will become larger. High mistake rate and low query accuracy probability is because lots of message loss due to network congestion. For ED FD, when the network congestion becomes severe, the network delay becomes larger and lots of messages are lost. Due to message loss, the mistake rate of ED FD will increase. Because of the longer network delay, the average value of inter-arrival times is increased and the query accuracy probability will reduce. Thus, the value of will reduce; based on the function of ED FD, under the same threshold of suspicion, the detection time will increase. For Kappa FD, it is also an instance of accrual failure detector like ED FD and ϕ FD. Also, Kappa FD has the similar performance change when network congestion becomes severe. The difference between ED FD and Kappa FD is that: for ED FD, detection time becomes larger, but when detection time is large enough there is a maximal value after that there is no data; for Kappa FD, detection time also becomes larger, but when detection time is large enough there is a maximal value after that MR/QAP reach

theoretical optimal values.

The effect of network congestion on QoS of self-tuning FD is different from TAM FD, ED FD and kappa FD. For self-tuning FD, detection time, mistake rate and query accurate probability have slightly change. That is because, when the network congestion becomes severe, the network delay and message loss increases. But self-FD can tune its parameters based on the network condition change in order to satisfy the target QoS of users. Even though the network environment change has certain effect on QoS of FD, this effect is not as much as that on TAM FD, ED FD and Kappa FD.

7.6.2 Why improving QoS of network will improve QoS of failure detector

Based on a general large wide-area failure detectors distributed network system model with a large number of monitored components (may be called only large wide-area system in below) for dynamic heartbeat streams and on the system stability, it is necessary to design an algorithm to regulate the sending rate of heartbeat messages, which are limited by the bottleneck router, where the control parameters can be designed to ensure the stability of the control loop in terms of adjusting sending rate of heartbeat streams.

Stelling et al [49] proposed a failure detection service for the Globus toolkit, which has been designed to use existing fabric components, including vendor-supplied protocols and interfaces [94]. This approach solves the scalability of the failure detection, while it does not solve the message explosion and system dynamism. Van Renesse et al [50] distinguished two variations of gossip-style protocols. One is named basic gossiping and the other is named multi-level gossiping. To adapt it to a large-scale network, a variant of the basic gossiping protocol called multi-level gossiping protocol is proposed. Gossip-style protocols can address the problems of message explosion, message loss and dynamism. Unfortunately, there are drawbacks in gossip-style protocols, such as this protocol does not work well when a large percentage of components crash or become partitioned away. Then, a failure detector may spend long time to detect crashed components by gossip messages. This protocol is quite simple but is less efficient than approaches based on hierarchical, tree-based protocols.

Most of the proposed protocols were developed for local area networks and were not efficient in the context of a wide area distributed system consisting of many components and characterized by a high level of asynchrony, long message delay, high probability of message loss and which topology might change as a result of a reconfiguration. The large number of components and their large wide-area distribution system as well as the dynamic structure of the system increase the risk of failures. Thus, providing a scalable and generic failure detection service as a basic QoS is of primary importance in wide area distributed systems.

In the Figure 7.15, the architecture of the proposed failure detector service has two layers: the lower layer includes local monitors and the upper layer includes data collectors. The local monitor is responsible for monitoring the host on which it runs as well as selected processes on that host. It periodically sends heartbeat messages to data collectors including information on the monitored components. The data collectors receive heartbeats from local monitors, identifies failed components, and notifies applications about relevant events concerning monitored components.

This approach improves the failure detection time in a large wide-area system. Thus,

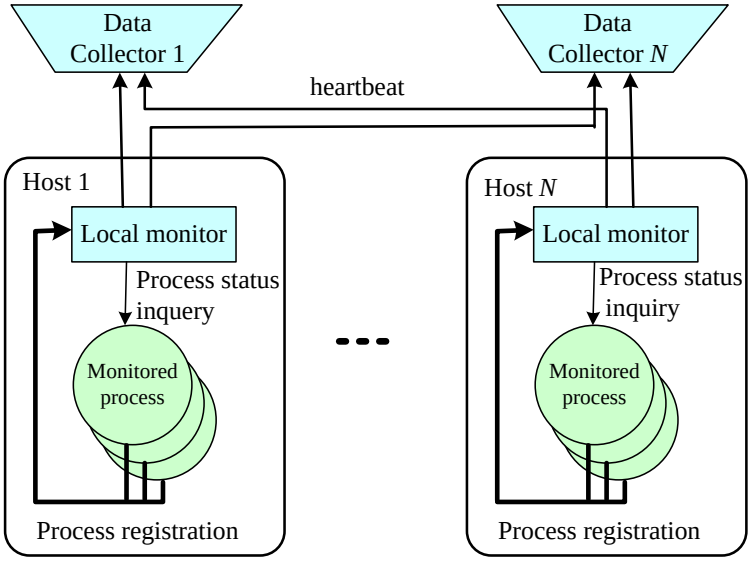


Figure 7.15: The architecture of the wide-area failure detection service [49, 99].

it solves the scalability problem. However, it has drawbacks. In this approach, each local monitor broadcasts heartbeats to all data collectors. Therefore, this approach probably does not solve the message explosion problem. A large wide-area system may change its topology by component leaving/joining at runtime but the proposed architecture is static and does not adapt well to such changes in system topology (a dynamism problem).

7.6.3 Theoretical network communication model for the wide-area failure detection service

In general, we improve the architecture to the large-scale distributed network systems model for dynamic heartbeat stream in Figure 7.16.

In this model, the dynamic heartbeat streams coming from the hosts are a large number of streams, which get together to the router connected with the data collector, then it is easily congested in the bottleneck router. Control algorithm is located at the data collectors. We are interested in finding the change of the sending rate from the hosts based on heartbeat streams requested by bottleneck router in a specific time. Note that the buffer in the bottleneck router can be optionally set for temporary storage of recent transactions from the transaction heartbeat streams.

And the considered heartbeat streams service is described as follows: Time is slotted with the duration $[n, n+1)$, equals to T. The associated data is transferred by a fixed size packet.

The host generates forward control packet (FCP) that is passed by the middle routers including the control information and finally is received by the data collectors. Based on the control information, the data collectors send heartbeat messages and backward

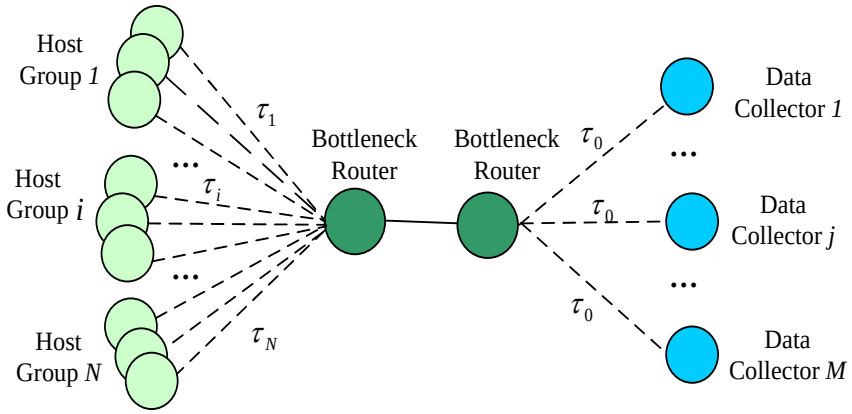


Figure 7.16: Large wide-area network system model for dynamic heartbeat streams.

control packet (BCP) into the network.

The middle routers between the data collector and the hosts transfer FCP and the heartbeat streams to its downstream, and transfer BCP to its upstream.

After the bottleneck router receives the relevant information, the required probability of packets dropping is computed.

Unless otherwise specified, the following notations pertain to the remainder of the paper. τ_0: Normalized link forward delay from the data collector to the bottleneck router; τ_i: Normalized link forward delay from the bottleneck router to the hosts group i, $(1 \leq i \leq N)$.

In the next part, we present a novel AQM scheme combined with control theory that can help in solving or optimizing some problems in networks. Furthermore, this chapter discusses the design and analysis of implementing a scalable service for such large wide-area distributed systems, so that it avoids the congestion in bottleneck routers. We further show how a controller is designed as a AQM scheme in the router and is analyzed from the theoretical aspects. Simulation results show the efficiency of our scheme in terms of high utilization of the bottleneck link, fast response and good stability of buffer occupancy as well as of the controlled sending rates.

7.7 Conclusion

In this chapter, we proposed a packet dropping scheme, called SPI-RED, to improve the performance of RED. We have analysed the average queue length stability of SPI-RED, and have given guidelines for selecting control gains. This method can also be applied to the other variants of RED. Based on the stability conditions and control gain selection method, extensive simulation results by ns2 demonstrate that the SPI-RED scheme outperforms other state-of-the-art AQM algorithms in terms of robustness, drop probability and stability.

Chapter 8

Conclusion and Future Work

8.1 Failure detector

We first explore the relative failure detection, which is an important issue for supporting dependability in distributed systems, and often is an important performance bottleneck in the event of node failure. In this field, we analyze the existing failure detections, and then develop four different schemes (Tuning adaptive margin failure detection; Exponential distribution failure detection; Kappa failure detection; Self-tuning failure detection) to ensure acceptable quality of service in unpredictable network environments.

Self-tuning failure detection: For the former failure detection scheme, all of them can not actively tune their parameters by themselves to satisfy the requirement of users in dynamic networks. To the question, we present a self-tuning failure detection based on Chen failure detection, called selftuning failure detection. Furthermore, a lot of experimental results demonstrate that our sheme is effective. And we are sure that this idea also can apply into other failure detection to achieve self-tuning requirement.

Tuning adaptive margin failure detection: It is an optimization the adaptation of [30] due to its tuning safety margin based on the network conditions. Tuning adaptive margin failure detection significantly improves quality of service in the aggressive range, especially when the network is unstable. Furthermore, we explore the effect of memory usage on the performance of adaptive failure detectors. The experimental results over several kinds of networks (Cluster, WiFi, Wired local area network, and Wide area network) show that the properties of the existing adaptive failure detections; and demonstrate that the optimization is reasonable and acceptable, especially in aggressive range and in unstable networks; and present the effect of memory size on the overall quality of service of each adaptive failure detection.

Exponential distribution failure detection: Observing from lots of experimental statistical results, we find it is not a good assumption that the Phi failure detection [18] uses the normal distribution to estimate the arrival time of the coming heartbeat, especially in large scale distributed network or unstable networks. Therefore, here we develop an optimization over Phi failure detection based on exponential distribution, called exponential distribution failure detection. This significantly improves quality of service, especially in the design of real systems. Extensive experiments have been carried out based on several kinds of networks (Cluster, WiFi, Wired local area network, and Wide area network). The experimental results have shown the properties of the existing adaptive failure detections, and demonstrated that the presented exponential distribution

Table 8.1: Property comparison for our TAM FD, ED FD, Kappa FD, and self-tuning FD

Failure detector	Suitable deployment environments	Unsuitable deployment environments	Choice of parameters
TAM FD	dynamic networks (WAN)	very short delay (Cluster)	match by hand
ED FD	aggressive range	conservative range	match by hand
Kappa FD	aggressive & conservative range		match by hand
self-tuning FD	large delay variability (WAN)	small delay variability (Cluster)	self-tuning by itself

failure detection outperforms the existing failure detections in the aggressive range.

Kappa failure detection: This part analyzes a failure detection implementation, called Kappa failure detection, which gives a real value for a suspicion level to each process. It has the formal properties of accrual failure detection. While the traditional failure detection is the binary information (trust vs. suspect). Aim at the practical shortcomings of previous state-of-the-art implementations, our scheme allows for gradual settings between an aggressive behavior and a conservative one. At last, we demonstrate our scheme with an extensive performance based on a variety of environments.

Applications of our TAM FD, ED FD, Kappa FD, and self-tuning FD: This part analyzes the different applications based on the different properties of the four failure detectors. In Table 8.1, there are simple comparison about the four failure detectors.

For TAM FD, it uses a tuning adaptive margin to adapt the dynamic networks. Therefore, there are prominently good performance in dynamic networks (i.e., large RTT and large variability of transfer delay with much message loss, for example, WAN). While in the very short delay case (for example, Cluster), the performance of TAM FD is not better than that of ϕ FD in the aggressive range. It is suitable for the applications, which are not sensitive to either DT and MR/QAP, while, they need to balance the DT and MR/QAP dynamically. Choosing one parameter is with the cost of the other one. For example, it is suitable for the scientific computing applications.

For ED FD, it is improved from ϕ FD, which is an aggressive scheme. ED FD uses the exponential distribution instead of the normal distribution for the inter-arrival time of heartbeats. Therefore, it is more aggressive than ϕ FD. If the applications require short detection time (how fast), low mistake rate (how well), and high accuracy query probability (how well), in this case ED FD is a good choice. So in the Cluster case, ED FD obtains the best performance than the other three schemes in an aggressive range. While if the applications require low mistake rate and high accuracy query probability, and have not very high requirement about the detection time, then Chen FD is a better choice than ED FD.

For Kappa FD, it is a development of an instance of accrual failure detector, and useful for both aggressive failure detection and conservative failure detection. For example, Kappa FD is suitable for the database replication, which is very sensitive to wrong suspicious and is not strong to DT. So they benefit from failure detection with very small MR, even the MR is 0.

For self-tuning FD, it adjusts the next freshness point to get different QoS to satisfy the target QoS gradually. Therefore, it is useful for the dynamic networks (for example, WAN), and gets outstanding performance. For the very short delay case (for Cluster), it is also ok, while the performance is not very outstanding. Because in this case, ED FD has got perfect results. Furthermore, for a target QoS from users, only self-tuning FD can adjust its parameters by itself, other three failure detectors give a list QoS services, and the choose the corresponding parameters by hand to match the QoS requirement. For example, self-tuning FD is suitable for the users, who are fast moving. It changes the environment very quick and many times. So self-tuning FD can adjust the next freshness point to adapt the change of environment.

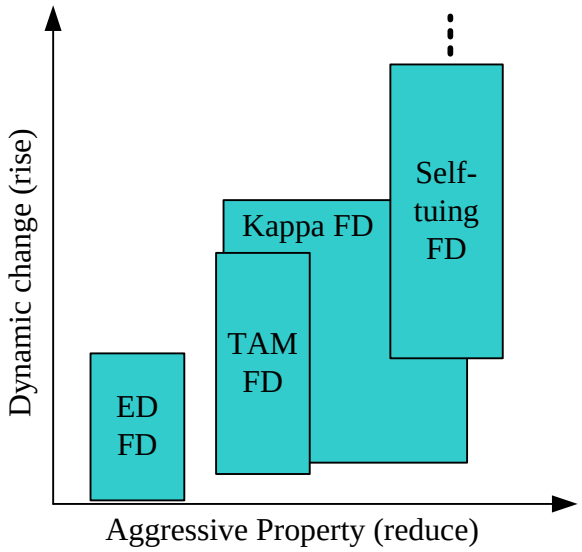


Figure 8.1: The relation structure model of failure detectors.

In Fig. 8.1, it gives more relation about the proposed failure detectors. The aggressive property rises from left to right, and the dynamic change[1] reduces from below side to upside. It is clear that Ed FD is the most aggressive scheme in the four schemes. Kappa FD has a large area in both aggressive rang and conservative range. Self-tuning FD is used to a large dynamic case, and the dynamic change can become very large. By comparing the performance of different failure detectors by a lot of experiments, the users in actual applications can choose a suitable failure detector based on your analysis.

8.2 Active queue management

There are some relation between the failure detection and the active queue management scheme. Failure detection is generally based on distributed communication networks. Reversely, the performance (delay, throughput, rate of packets dropping, and so on) of communication networks also affects quality of service of failure detector. Generally speaking,

[1]We know, the network environment is dynamic and unpredictable, heartbeats have quite different delay to arrive receivers. Here we not only consider the standard deviation of the delay, but also consider many other important factors, such as the probability of message loss, the change of network topology, some burst flow, and so on.

if the communication network is unreliable, i.e., heart messages have large delay and lots of heartbeat messages will be lost, then failure detector will suspect wrongly the processes with high probability. Thus, very high mistake rate and low accuracy probability will get for failure detector. More theory analysis are shown in Chapter 7.6. Therefore, it becomes very necessary to improve the performance of communication network.

Failure detection is generally based on distributed communication networks. Reversely, the performance of communication networks also affects quality of service of failure detector. Therefore, it becomes very necessary to improve the performance of communication network. In order to make sure the network communication is effective, we explore the related active queue management schemes to support TCP flows in networks to ensure good actual measurements with failure detection.

In this part, we first present a self-tuning proportional and integral controller scheme based on average queue length of router. Then we prove the stability of the network system, and present an effective method for the control gain selection. After that, extensive simulations have been conducted with ns2. The simulation results have demonstrated that the proposed self-tuning proportional and integral controller algorithm outperforms the existing active queue management schemes in terms of drop probability and stability.

8.3 Future work

In future work, we would like to explore the QoS scalability, as interference from heavier network traffic (e.g., a scenario where most of the nodes in the networked system have active FDs), to see whether that will affect detection accuracy, detection time, etc. Also, we would explore their properties and relation in software engineering applications, then find or propose a reasonable FD in fault-tolerant distributed system, and apply the proposed FD into an actual fault-tolerant distributed system, specially, we are very interested in designing an self-tuning failure detector (SFD) in actual fault-tolerant distributed system.

For all the proposed failure detectors so far, the common point of them is that they all can detect the directly connected processes. While, for some processes that are not directly connected, i.e., they only can communicate each other by some middle processes, all failure detectors are not applicable. Therefore, an open question arises: how to design an indirect failure detector to make sure any two processes, even they are not connected directly, can detect each other effectively.

Furthermore, we also would like to explore in the following aspects.

(1) New schemes based on different architectures of failure detection;

(2) Build a pragmatic platform on failure detection;

(3) Application of failure detections in Ad hoc, Mobile network, or other environment;

Bibliography

[1] N. Sergent, X. Defago, and A. Schiper. Impact of a failure detection mechanism on the performance of consensus. In Proc. 8th IEEE Pacific Rim Symp. on Dependable Computing(PRDC-8), pages 137C145, Seoul, Korea, December 2001.

[2] N. Hayashibara, A. Cherif, and T. Katayama. Failure detectors for large-scale distributed systems. In Proc. 21st IEEE Symp. on Reliable Distributed Systems (SRDS-21), Intl. Workshop on Self-Repairing and Self-Configurable Distributed Systems (RCDS2002), pages 404C409, Osaka, Japan, October 2002.

[3] N. Hayashibara. Accrual failure detectors. Doctoral thesis, Japan Advanced Institute of Science and Technology, June, 2004.

[4] I. Gupta, T. D. Chandra, G. S. Goldszmidt. On scalable and efficient distributed failure detectors. In Proc. 20th ACM symp. on Principles of Distributed Computing, pages 170 - 179, Newport, Rhode Island, United States, Aug. 2001.

[5] M. K. Aguilera, W. Chen, and S. Toueg. Failure detection and consensus in the crash-recovery model. Distributed Computing, 13(2): 99-125, Springer-Verlag, 2000.

[6] P. Felber, X. Défago, R. Guerraoui, and P. Oser. Failure detectors as first class objects. In Proc. IEEE Intl. Symp. On Distributed Objects and Applications (DOA'99), pages 132-141, Edinburgh, Scotland, Sep. 1999.

[7] C. Delporte-Gallet, H. Fauconnier, R. Guerraoui. A realistic look at failure detectors. In Proc. Intl. Conf. on Dependable Systems and Networks (DSN'02), pages 345- 353, Washington DC, Jun. 2002.

[8] M. J. Fischer, N. A. Lynch, and M. D. Paterson. Impossibility of distributed consensus with one faulty process. Journal of ACM, 32(2): 374-382, Apr. 1985.

[9] T. D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. Journal of the ACM, 43(2): 225- 267, Mar. 1996.

[10] M. K. Aguilera, W. Chen, and S. Toueg. Using the heartbeat failure detector for quiescent reliable communication and consensus in partition-able networks. Theoretical Computer Science, 220(1): 3-30, Jun. 1999.

[11] M. Larrea, A. Fernandez, and S. Arevalo. Optimal implementation of the weakest failure detector for solving consensus. In Proceedings of the 19th Annual ACM Symposium on Principles of Distributed Computing, pages 334-334, NY, Jul. 2000.

[12] R. C. Nunes, I. Jansch-Porto. QoS timeout-based self-tuned failure detectors: the effects of the communication delay predictor and the safety margin. In Proc. 2004 International Conference on Dependable Systems and Networks (DSN 2004), pages 753-761, Florence, Italy, June 2004.

[13] L. Fabio, R. Macêdo. Adapting failure detectors to communication network load fluctuations using SNMP and artificial neural networks. In Proc. Second Latin-American Symposium on Dependable Computing (LADC 2005), pages 191-205, Salvador, Brazil, Oct. 2005.

[14] C. Fetzer, M. Raynal, and F. Tronel. An adaptive failure detection protocol. In Proc. IEEE the 8th Pacific Rim Symposium on Dependable Computing, pages 146- 153, Seoul, Korea, Dec. 2001.

[15] I. Sotoma, E. R. M. Madeira. Adaptation - algorithm to adaptive fault monitoring and their implementation on CORBA. In Proc. 3th Intl. Symp. on Distributed Objects and Applications (DOA'01), pages 219-228, Rome, Italy, Sep. 2001.

[16] M. Bertier, O. Marin, P. Sens. Implementation and performance evaluation of an adaptable failure detector. In Proc. Intl. Conf. on Dependable Systems and Networks (DSN'02), pages 354-363, Washington DC, USA, Jun. 2002.

[17] M. Bertier, O. Marin, P. Sens. Performance analysis of a hierarchical failure detector. In Proc. Dependable Systems and Networks (DSN'03), pages 635-644, San Fra., USA, Jun. 2003.

[18] N. Hayashibara, X. Défago, R. Yared, and T. Katayama. The *phi* accrual failure detector. In Proc. 23rd IEEE Intl. Symp. on Reliable Distributed Systems (SRDS'04), pages 66-78, Florianpolis, Brazil, Oct. 2004.

[19] X. Défago, P. Urban, N. Hayashibara, T. Katayama. Definition and specification of accrual failure detectors. In Proc. Intl. Conf. on Dependable Systems and Networks (DSN'05), pages 206 - 215, Yokohama, Japan, Jun. 2005.

[20] R. Macêdo. Implementing failure detection through the use of a self-tuned time connectivity indicator. TR, RT008/98, Laboratŕio de Sistemas Distribudos - LaSiD, Salvador-Brazil, Aug. 1998.

[21] P. Felber. The CORBA object group service - a service approach to object groups in CORBA. PhD thesis, Départment D'informatique, Lausanne, EPFL, Swizerland, 1998.

[22] A. Basu, B. Charron-Bost, and S. Toueg. Solving problems in the presence of process crashes and lossy links. TR96-1609, Cornell University, USA, Sep. 1996.

[23] V. Jacobson. Congestion Avoidance and Control. In Proc. of ACM SIGCOMM'88, pages 314–329, Stanford, CA, USA, Aug. 1988.

[24] L. Falai and A. Bondavalli. Experimental evaluation of the QoS of failure detectors on wide area network. In Proc. of the Int. Conf. on Dependable Systems and Networks (DSN'05), Yokohama, Japan, pages 624-633, Jun. 2005.

[25] R. C. Nunes, I. Jansch-Porto. Modeling communication delays in distributed systems using time series. In Proc. 21st IEEE Symp. on Reliable Distributed Systems (SRDS'02), pages 268-273, Suita, Japan, Oct. 2002.

[26] L. M. R. Sampaio, Francisco V. Brasileiro, Walfredo Cirne, Jorge C. A. Figueiredo. How bad are wrong suspicions toward adaptive distributed protocols. In Proc. IEEE Intl. Conf. on Dependable Systems and Networks (DSN 2003), pages 551-560, San Francisco, CA, USA, Jun. 2003.

[27] http://ddsg.jaist.ac.jp/en/jst/data.html.

[28] N. Hayashibara and M. Takizawa. Performance Analysis of the ϕ Failure Detector with its Tunable Parameters. In Proc. of 17th International Conf. on Database and Expert Systems Applications, pages 20-24, Krakow, Poland, Oct. 2006.

[29] R. Guerraoui, M. Larrea, and A. Schiper. Non-blocking atomic commitment with an unreliable failure detector. Symposium on Reliable Distributed Systems, pages 41–50, 1995.

[30] W. Chen, S. Toueg, and M. K. Aguilera. On the quality of service of failure detectors. IEEE Transaction on Computers, 51(5):561–580, May 2002.

[31] A. Basu, B. Charron-Bost, and S. Toueg. Simulating Reliable Links with Unreliable Links in the Presence of Process Crashes. In Proc. Workshop on Distributed Algorithms (WDAG 1996), pages 105–122, Bologna, Italy, 1996.

[32] B. Charron-Bost, X. Défago, and A. Schiper. Broadcasting messages in fault-tolerant distributed systems: the benefit of handling input-triggered and output-triggered suspicions differently. In Proc. 21st IEEE Intl. Symp. on Reliable Distributed Systems (SRDS'02), pages 244–249, Osaka, Japan, Oct. 2002.

[33] X. Défago, A. Schiper, and N. Sergent. Semi-passive replication. In Proc. 17th IEEE Intl. Symp. Reliable Distributed Systems (SRDS'98), pages 43–50, West Lafayette, IN, USA, Oct. 1998.

[34] P. Urbán, I. Shnayderman, and A. Schiper. Comparison of failure detectors and group membership: Performance study of two atomic broadcast algorithms. In Proc. IEEE Intl. Conf. on Dependable Systems and Networks (DSN'03), pages 645–654, San Francisco, CA, USA, June 2003.

[35] M. Müller. Performance evaluation of a failure detector using SNMP. Semester project report, École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland, Feb. 2004.

[36] A. Bondavalli, S. Chiaradonna, F. Di Giandomenico, and F. Grandoni. Threshold-based mechanisms to discriminate transient from intermittent faults. IEEE Transaction on Parallel & Distributed Systems, 49(3):230-245, March 2000.

[37] F. Chu. Reducing Ω to $\Diamond W$. Information Processing Letters, 67(6):289-293, September 1998.

[38] F. J. N. Cosquer, L. E. T. Rodrigues, and P. Veríssimo. Using tailored failure suspectors to support distributed cooperative applications. In Proc. 7th IASTED Intl. Conf. on Parallel and Distributed Computing and Systems (PDCS 2005), pages 352-356, Washington, DC, USA, October 1995.

[39] J. Dunagan, N. J. A. Harvey, M. B. Jones, D. Kostic, M. Theimer, and A. Wolman. FUSE: Lightweight guaranteed distributed failure notification. In Proc. 6th Symp. on Operating Systems Design and Implementation (OSDI 2004), San Francisco, CA, USA, December 2004.

[40] C. Dwork, N. A. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. J. ACM, 35(2):288-323, April 1988.

[41] R. Friedman. Fuzzy group membership. In A. Schiper, A. Shvartsman, H. Weatherspoon, and B. Y. Zhao, editors, Future Directions in Distributed Computing, volume 2584 of LNCS, pages 114-118. Springer-Verlag, January 2003. Position paper.

[42] L. Lamport. The part-time parliament. ACM Transactions on Computer Systems, 16(2):133-169, May 1998.

[43] A. Mostéfaoui, E. Mourgaya, and M. Raynal. Asynchronous implementation of failure detectors. In Proc. IEEE Intl. Conf. on Dependable Systems and Networks (DSN 2003), pages 351–360, San Francisco, CA, USA, June 2003.

[44] A. Mostéfaoui, D. Powell, and M. Raynal. A hybrid approach for building eventually accurate failure detectors. In Proc. 10th IEEE Pacific Rim Intl. Symp. on Dependable Computing (PRDC), pages 57–65, Papeete, Tahiti, March 2004.

[45] A. Mostéfaoui, M. Raynal, and C. Travers. Crash-resilient time-free eventual leadership. In Proc. 23nd IEEE Intl. Symp. on Reliable Distributed Systems (SRDS 2004), Florianópolis, Brazil, October 2004.

[46] 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 Proc. IEEE Intl. Conf. on Dependable Systems and Networks (DSN 2004), pages 753-761, Florence, Italy, June 2004.

[47] M. Raynal. A short introduction to failure detectors for asynchronous distributed systems. ACM Sigact News, Distributed Computing Column, 36(1):53-70, 2005.

[48] I. Sotoma and E. R. M. Madeira. ADAPTATION-algorithms to adaptive fault monitoring and their implementation on CORBA. In Proc. 3rd Intl. Symp. on Distributed-Objects and Applications (DOA 2001), pages 219-228, Rome, Italy, September 2001. IEEE Computer Society Press.

[49] P. Stelling, I. Foster, C. Kesselman, C. Lee, and G. von Laszewski. A fault detection service for wide area distributed computations. In Proc. 7th IEEE Symp. on High Performance Distributed Computing, pages 268-278, July 1998.

[50] R. van Renesse, Y. Minsky, and M. Hayden. A gossip-style failure detection service. In N. Davies, K. Raymond, and J. Seitz, editors, Middleware 1998, pages 55-70, The Lake District, UK, 1998.

[51] M. Wiesmann, P. Urbán, and X. Défago. An SNMP based failure detection service. In Proc. 25th IEEE Intl. Symp. on Reliable Distributed Systems (SRDS 2006), pages 365-374, Leeds, UK, October 2006.

[52] S. Athuraliya, S. H. Low, V. H. Li, and Q. Yin. REM: active queue management. IEEE Networking, Vol. 15, pp. 48-53, May 2001.

[53] M. C. Choy, D. Srinivasan, R. L. Cheu. Cooperative, hybrid agent architecture for real-time traffic signal control. IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and humans, Vol. 33, No. 5, pp. 597-607, September 2003.

[54] S. Huang, W. Ren, S. C. Chan. Design and performance evaluation of mixed manual and automated control traffic. IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and humans, Vol. 30, No. 6, pp. 661-673, Nov. 2000.

[55] S. Lee, C. Hou. A neural-fuzzy system for congestion control in ATM networks. IEEE Transactions on systems, man and cybernetics-Part B: cybernetics, Vol. 30, No. 1, pp. 2-9, February 2000.

[56] J. Sun, K. Ko, G. Chen, S. Chan, and M. Zukerman. PD-RED: to improve the performance of RED. IEEE Communications Letters, Vol. 7, No. 8, pp. 406-408, August 2003.

[57] V. Misra, W. Gong, and D. F. Towsley. Fluid-based Analysis of a Network of AQM Routers Supporting TCP Flows with an Application to RED. In Proceedings of ACM SIGCOMM 2000, Sweden, pp. 151-160, August 28- September 1, 2000.

[58] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. IEEE/ACM Transactions on Networking, Vol. 1, pp. 397-413, August 1993.

[59] C. N. Long, J. Wu and X. P. Guan. Local stability of REM algorithm with time-varying delays. IEEE Communications Letters, Vol. 7, pp. 142-144, March 2003.

[60] C. V. Hollot, V. Misra, D. Towsley, and W. B. Gong. A Control Theoretic Analysis of RED. In Proceedings of IEEE INFOCOM 2001, Anchorage, Alaska, Vol. 3, pp. 1510-1519, April 22-26, 2001.

[61] M. Christiansen, K. Jeffay, D. Ott, and F. D. Smith. Tuning RED for Web Traffic. In Proceedings of ACM SIGCOMM 2000, Sweden, pp. 139-150, August 28-September 1, 2000.

[62] S. Floyd and V. Jacobson. On traffic phase effects in packet-switched gateways. Internetworking: Research and Experience, Vol. 3, No. 3, pp. 115-156, September 1992.

[63] V. Firoiu and M. Borden. A Study of Active Queue Management for Congestion Control. In Proceedings of IEEE INFOCOM 2000, Tel-Aviv, Israel, pp. 1435-1444, March 26-30, 2000.

[64] G. Lannaccone, M. May, and C. Diot. Aggregate traffic performance with active queue management and drop from tail. ACM SIGCOMM Computer Communication Review, Vol. 31, No. 3, pp. 4-13, July 2001.

[65] M. May, J. Bolot, C. Diot, and B. Lyles. Reasons not to deploy RED. In Proceedings of the 7th Internet Workshop on Quality of Service (IWQoS'99), London, U.K., pp. 260-262, June 1-4, 1999.

[66] J. Aweya, M. Ouellette, and D. Y. Montuno. A Control Theoretic Approach to Active Queue Management. Computer Networks, Vol. 36, No. 2-3, pp. 203-235, 2001.

[67] S. Floyd, R. Gummadi and S. Shenker. Adaptive RED: An algorithm for increasing the robustness of RED's active queue management. Berkeley, CA, Technical Report, 2001. Available at: http://www.icir.org/floyd/red.html.

[68] J. Aweya, M. Ouellette, D. Y. Montuno and A. Chapman. An Optimization-oriented View of Random Early Detection. Computer Communications, Vol. 24, pp. 1170-1187, 2001.

[69] C. V. Hollot, V. Misra, D. Towsley and W. B. Gong. Analysis and design of controllers for AQM routers supporting TCP flows. IEEE Transactions on Automatic Control, Vol. 47, pp. 945-959, June 2002.

[70] C. V. Hollot, V. Maisra, D. Towsley and W. Gong. On designing improved controllers for AQM routers supporting TCP flows. In Proceedings of IEEE INFOCOM 2001, Anchorage, Alaska, pp. 1726-1734, April 22-26, 2001. Available at: http://www.ieee-infocom.org/2001/paper/792.pdf.

[71] X. Deng, S. Yi, G. Kesidis, and C. R. Das. A Control Theoretic Approach for Designing Adaptive Active Queue Management Schemes. In Proceedings of IEEE GlOBECOM 2003, San Francisco, USA, Vol. 5, pp. 2947-2951, December 1-5, 2003.

[72] K. Zhou and J. C. Doyle. Essentials of Robust Control. Prentice Hall, NY, 1998.

[73] W. Fang, K. G. Shin, D. D. Kandlur, and D. Saha. The BLUE active queue management algorithms. IEEE/ACM Transactions on Networking, Vol. 10, No. 4, pp. 513-528, August 2002.

[74] H. Lim, K. Park, E. Park and C. Choi. Proportional-Integral Active Queue Management with an Anti-Windup Compensator. In Proceedings of 2002 Conference on Information Sciences and Systems, Princeton University, March 20-22, 2002. Available at: http://csl.snu.ac.kr/publication/paper/ciss02.pdf.

[75] A. Rogers, E. David, N. R. Jennings. Self-organized routing for wireless microsensor networks. IEEE Transactions on Systems, Man and Cybernetics, Part A: SYSTEMS AND HUMANS, Vol. 35, No. 3, pp. 349-359, May 2005.

[76] W. Kamen, B. S. Heck. Fundamentals of signals and systems using the web and matlab. 2th edition, prentice Hall, INC, pp. 439-443, 2002.

[77] P. Karn, and C. Partridge. Improving round-trip time estimates in reliable transport Protocols. ACM Computer Communication Review, Vol. 25, No. 1, pp. 66-74, January 1995.

[78] H. Zhang, C. V. Hollot, D. Towsley, and V. Misra. A self-tuning structure for adaptation in TCP/AQM networks. ACM SIGMETRICS Performance Evaluation Review, Vol. 31, No. 1, pp. 302-303, June 2003.

[79] T. J. Ott, T. V. Lakshman, and L. Wong. SRED: Stabilized RED. In Proceedings of IEEE INFOCOM 1999, Vol. 3, New York, pp. 1346-1355, March 21-25, 1999.

[80] S. H. Low, F. Paganini, J. Wang, and J. C. Doyle. Linear stability of TCP/RED and a scalable control. Computer Networks Journal, Vol. 43, No. 5, pp. 633-647, Dec. 2003.

[81] USC/ISI, Los Angeles, CA. The NS simulator and the documentation. Available at: http://www.isi.edu/nsnam/ns/.

[82] W. Feng, D. D. Kandlur, D. Saha, and K. G. Shin. A self-configuring RED Gateway. In Proceedings of IEEE INFOCOMM 1999, New York, NY, Vol. 3, pp. 1320-1328, March 21-25, 1999.

[83] C. Wang, B. Li, Y. T. Hou, K. Sohraby, Y. Lin. LRED: A Robust Active Queue Management Scheme Based on Packet Loss Ratio. In Proceedings of IEEE INFOCOM 2004, Hong Kong, March 7-11, 2004. Available at: http://www.ieee-infocom.org/2004/Papers/01_1.PDF.

[84] S. Kunnivur and R. Srikant. Analysis and design of an adaptive virtual queue (AVQ) algorithm for active queue management. In Proceedings of ACM SIGCOMM 2001, UC San Diego, CA, USA, pp. 123-134, August 27-31, 2001.

[85] S. Kunniyur and R. Srikant. End-to-end congestion control: Utility functions, random losses and ECN marks. In Proceedings of INFOCOM 2000, Vol. 3, pp. 1323-1332, Tel-Aviv, Israel, March 26-30, 2000.

[86] D. Katabi and C. Blake. A note on the stability requirements of adaptive virtual queue. LCS Document Number: MIT-LCS-TM-626, 2-13-2002. Available at: http://www.lcs.mit.edu/publications/pubs/pdf/MIT-LCS-TM-626.pdf.

[87] Y. Gao and J. C. Hou. A state feedback control approach to stabilizing queues for ECN-enabled TCP flows. In Proceedings of IEEE INFOCOM 2003, Vol. 3, pp. 2301-2311, San Francisco, CA, March 30 - April 2, 2003.

[88] Y. Gao, G. He and J. Hou. On Leveraging Traffic Predictability in Active Queue Management. In Proceedings of IEEE INFOCOM 2002, New York, USA, June 23-27, 2002. Available at: http://lion.cs.uiuc.edu/courses/cs497hou/PAQM.pdf.

[89] C. Caserri, M. Meo. A New Approach to model the stationary behavior of TCP connections. In Proceedings of IEEE INFOCOM 2000, Vol. 1, pp. 367-375, Tel-Aviv, Israel, March 26-30, 2000.

[90] X. Défago, N. Xiong, Y. Yang, and N. Hayashibara. Pragmatic Accrual Failure Detection with Kappa-FD. Technical report, JAIST, Japan, Nov. 2007.

[91] D. Lea. Concurent Programming in Java. Addison-Wesley, 1997.

[92] Kanaka Juvva Raj Rajkumar. A Real-Time Push-Pull Communications Model for Distributed Real-Time and Multimedia Systems. CMU-CS-99-107, January, 1999.

[93] Kanaka Juvva. End-to-End Predictability in Real-Time Push-Pull Communications. Proceedings of the Third IEEE International Symposium on Object-Oriented Real-Time Distributed Computing,pp 278,2000.

[94] I. Foster, C. Kesselman, and S. Tuecke, "The anatomy of the grid," Int'l Journal of Supercomputer Applications, 2001.

[95] Requirements for Internet Hosts-Communication Layers, R. Braden, ed., RFC 1122, Oct. 1989.

[96] G. F. Pfister, In Search of Clusters, second ed. Prentice Hall, 1998.

[97] R. van Renesse, K.P. Birman, and S. Maffeis, "Horus: A Flexible Group Communication System," Comm. ACM, Apr. 1996, vol. 39, no. 4, pp. 76-83.

[98] M. G. Hayden, "The Ensemble System," PhD thesis, Dept. of Computer Science, Cornell Univ., Jan. 1998.

[99] I. Foster, C. Kesselman, and S. Tuecke, "The anatomy of the grid," Intl. Journal of High Performance Computing Applications, 15(3):200-222, 2001.

Publications

Conference papers:

[1] Y. He, N. Xiong, X. Défago, Y. Yang and J. He, "A Single-pass Online Data Mining Algorithm Combined with Control Theory with Limited Memory in Dynamic Data Streams," In Proc. of 4th International Conference on Grid and Cooperative Computing (GCC 2005), pp. 1119-1130, Beijing, China, November 30-December 3, 2005.

[2] N. Xiong, X. Défago, Y. He and Y. Yang, "A Resource-based Server Performance Control for Grid Computing Systems," In Proc. of IFIP International Conference on Network and Parallel Computing 2005 (NPC 2005), pp. 56-64, Beijing, China, Nov. 30-Dec. 2, 2005.

[3] N. Xiong, Y. Yang and X. Défago, "LRC-RED: A Self-tuning Robust and Adaptive AQM Scheme," In Proc. of International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT 2005), pp. 655-659, Dalian, China, December, 2005.

[4] N. Xiong, Y. Yang, Y. He, "On the Quality of Service of Failure Detectors Based on Control Theory," In Proc. of IEEE 20th International Conference on Advanced Information Networking and Applications (AINA 2006), pp. 75-80, Vienna, Austria, April, 2006.

[5] N. Xiong, X. Défago, X. Jia, Y. Yang, Y. He, "Design and Analysis of a Self-tuning Proportional and Integral Controller for Active Queue Management Routers to Support TCP Flows," In Proc. of 25th Annual Conference on Computer Communications (INFOCOM 2006), Barcelona, Spain, April, 2006.

[6] N. Xiong, X. Défago, Y. Yang, Y. He, and J. He, "On Control Gain Selection in PI-RED," In Proc. of IEEE International Conference On Networking, Sensing and Control (ICNSC 2006), pp. 516-522, Ft. Lauderdale, Florida, USA, April, 2006.

[7] N. Xiong, Y. Yang, J. He, and Y. He, "On Designing QoS for Congestion Control Service Using Neural Network Predictive Techniques," In Proc. of IEEE International Conference On Granular Computing (IEEE-GrC 2006), pp. 299-304, Atlanta, USA, May, 2006.

[8] N. Xiong, Y. Yang, X. Défago, "Comparative Analysis of QoS and Memory Usage of Adaptive Failure Detectors," In Proc. of the 13th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC'07), Melbourne, Victoria, Australia, December 17-19, 2007.

Journal paper:

[9] N. Xiong, L. T. Yang, Y. Yang, and X. Défago, Y. He, "A Novel Numerical Algorithm Based on Self-Tuning Controller to Support TCP Flows," Appear to Special Issue in the refereed IMACS Journal: Mathematics and Computers in Simulation, March 2007.

Technical report:

[10] N. Xiong, X. Défago, "ED FD: Improving the Phi Accrual Failure Detector," Technical report, IS-RR-2007-007, JAIST, Japan, Apr. 27, 2007.

[11] X. Défago, N. Xiong, Y. Yang, and N. Hayashibara, "Pragmatic Accrual Failure Detection with Kappa-FD," Technical report, JAIST, Japan, Nov. 2007.